Reihe Informatik
TR-2006-013

Unnesting SQL Queries in the Presence of Disjunction

M. Brantner[1] N. May G. Moerkotte

University of Mannheim
Database Research Group
B6, 29
68131 Mannheim, Germany
msb|norman|moer@db.informatik.uni-mannheim.de

[1]This work was supported by the Deutsche Forschungsgemeinschaft under grant MO 507/10-1.

Abstract

Optimizing nested queries is an intricate problem. It becomes even harder if in a nested query the linking predicate or the correlation predicate occurs disjunctively. We present the first unnesting strategy that can effectively deal with such queries.

The starting point of our approach is to translate SQL into the relational algebra extended by bypass operators. Then we present for the first time unnesting equivalences which are valid for algebraic expressions containing bypass operators. Applying these to the translated queries results in our effective unnesting strategy for nested SQL queries with disjunction. With an extensive experimental study (including three commercial DBMSs), we demonstrate the possible performance gains of our approach.

1 Introduction

Nested queries easily become a performance bottleneck because in many cases, they demand a nested-loop evaluation. For conjunctive predicates this problem has been addressed successfully, e.g. [19, 25]. However, current unnesting techniques fail in the presence of disjunctive predicates. Despite the fact that disjunctions occurring inside nested queries are common in practice, we are not aware of any publication which treats unnesting nested queries which contain disjunctions, i.e. the linking or correlation predicate occur in a disjunction. For example, when asked about disjunctions in connection with nested queries, César A. Galindo-Legaria from the Microsoft SQL Server Group said: "We are running into it quite often. Actually, we have done something about it but would like to do more."

Key Idea. Let us consider an example for an analytical query. Assume we are interested in all European suppliers that deliver a certain part with minimum supply costs or have a minimal amount of this part available on stock. In SQL, this query can be formulated as follows:

```
SELECT  s_acctbal, s_name, n_name, p_partkey,
        p_mfgr, s_address, s_phone, s_comment
FROM    part, supplier, partsupp, nation, region
WHERE   p_partkey = ps_partkey
        AND s_suppkey = ps_suppkey AND p_size = 15
        AND p_type LIKE '%BRASS'
        AND s_n_key = n_n_key
        AND n_r_key = r_r_key AND r_name = 'EUROPE'
        AND (ps_supplycost = (
        SELECT  min(ps_supplycost)
        FROM    partsupp, supplier, nation, region
        WHERE   p_partkey = ps_partkey
                s_suppkey = ps_suppkey
                AND s_n_key = n_n_key
                AND n_r_key = r_r_key
                AND r_name = 'EUROPE')
        OR ps_availqty > 2000)
        ORDER BY s_acctbal desc, n_name,
        s_name, p_partkey
```

This query is very similar to TPC-H Query 2. Hence, we refer to it as Query 2d. It exhibits two key components: (1) it features a nested, correlated subquery, and (2) it contains a disjunction. Our unnesting strategy is capable of optimizing nested queries whose linking or correlation predicates occur disjunctively. The key idea is that the nested query block needs to be evaluated only for those tuples of the outer query block that do not pass the cheap and simple predicate `ps_availqty > 2000`. For those tuples, we are currently restricted to an inefficient nested-loop evaluation. However, our novel unnesting technique allows to employ more efficient evaluation algorithms. Consequently, our approach exploits both the short-cut evaluation of the disjunction and the power of unnesting nested queries.

Our Approach. The starting point of our approach is to translate SQL into the relational algebra extended with bypass operators [17]. Then, we apply our novel unnesting equivalences which can cope with disjunctions on a large variety of nested queries. As a result, nested query blocks are removed, and the resulting queries are much more efficient to evaluate.

Applying unnesting at the algebraic level has mainly three advantages. (1) It is possible to give rigorous correctness proofs for the unnesting equivalences. (2) Unnesting techniques stated as algebraic equivalences are query language independent as long as the query language is translatable into the algebra. (3) Unnesting equivalences can be used during plan generation. This allows to apply them in a cost-based manner. The latter is especially important in our case since some unnesting strategies do not always result in better plans.

Contributions. The main contributions of our paper are:

- We present equivalences for unnesting algebraic expressions with bypass operators to handle disjunctive linking and correlation predicates.
- We show how they can be used to effectively unnest SQL queries with scalar subqueries featuring an aggregation function in the `select` clause where the linking predicate involves $\theta \in \{=, \neq, <, \leq, >, \geq\}$.
- Our techniques can be applied not only to queries with exactly one nested block (simple queries) but also to queries whose nesting has a linear or even a tree structure.
- We provide experimental results demonstrating the performance improvements that can be achieved by our approach.
- We present how our approach can be applied to quantified table subqueries with the operators EXISTS, NOT EXISTS, IN, and NOT IN.
- We consider non-equality predicates as correlation predicates.
- We cover nested queries in the `where` clause and the `select` clause.

Limitations. As a current limitation, we restrict ourselves to queries exhibiting direct correlation: the correlation predicate only refers to attributes of the current block and the direct outer block.

Further, we do exclude linking predicates with linking operators θ `SOME/ANY`, or θ `ALL` with $\theta \in \{<, \leq, >, \geq\}$ from our discussion. Using aggregate functions that are aware of NULL values, we can still optimize these queries by turning these quantifiers into the aggregate functions $\texttt{min}_{\text{NULL}}$ or $\texttt{max}_{\text{NULL}}$.

Structure of this Paper. The remainder of the paper is organized as follows. Section 2 briefly introduces preliminaries. Section 4 contains our unnesting techniques for scalar subqueries. After introducing these approaches, we show their effectiveness with an experimental study (Section 5). At the end, we summarize related work in Section 6 and conclude the paper with Section 7.

2 Preliminaries

2.1 Terminology

A *query block* is a `select-from-where` expression. A query containing a query block nested in another query block is called a *nested query*. The containing query block is called *outer* query block, and the contained block is called *inner* query block. An inner query block is also called *nested* query block. Let p be a predicate occurring in the `where` clause of an inner query block. If p refers to attributes defined in the outer query block and to attributes defined in the inner query block, p is called a *correlation predicate*, and the inner query block is called *correlated*. A predicate q in the `where` clause of the outer query block which contains the inner query block as an argument is called *connection predicate*. The operator used in the connection predicate is called *connection operator*. Connection predicates are also called *linking predicates* [3].

If a linking predicate occurs in a disjunction as, for example, in the introductionary query, this is called *disjunctive linking*. Analogously, if the correlation predicate occurs in a disjunction, this is called *disjunctive correlation*.

2.2 Classification

Kim introduced four types of nested query blocks [19]: *A*, *N*, *JA*, and *J*. Let us refer to a nested query block as *B*. If *B* contains an aggregate function, *B* is of type *A* or *JA* and is called *scalar* subquery. *B* must return a single column. If *B* contains a correlation predicate, it is of type *J* or *JA*. A nested query block *B* that neither has an aggregate function nor a correlation predicate is of type *N*. Query blocks with an aggregation function but no correlation predicate are of type *A*. Nested query blocks of type *N* or *J* are called *table* subqueries. They are connected to their outer query block using the *positive linking operators* `EXISTS`, `SOME/ANY`, and `IN` or *negative linking operators* `NOT EXSTS`, `ALL`, and `NOT IN`. We cover these cases in our technical report [1] only.

While Kim concentrates on classifying single nested query blocks, Muralikrishna classifies queries according to the nesting structure [22]. He subdivided queries with more than one nested block into linear and tree queries: A *Linear (Nested) Query* is a query where at most one block is nested within any block. *A Tree (Nested) Query* is a query with at least one block which has two or more blocks nested within it at the same level. We complete this classification and call a query with exactly one nested block a *Simple (Nested) Query*.

2.3 Algebra

The domain underlying the relational algebra is sets of tuples. The core algebra contains the following operators: union ($\cup$), intersection ($\cap$), set-difference ($\setminus$), projection (Π), renaming operator (ρ), selection (σ), theta-join ($\bowtie$), semijoin ($\ltimes$), antisemijoin ($\triangleright$), and the grouping operator (Γ) [13]. We denote disjoint union by $\dot{\cup}$.

For the purpose of this paper, we extend the core algebra by four operators: a binary grouping operator (Γ) [5, 28], a leftouterjoin ($\mathbb{N}^{g:f(\emptyset)}$) [5, 7], a numbering operator (ν), and a map operator (χ). Implementations for binary grouping operators (Γ) can be found in [21]. The leftouterjoin ($\mathbb{N}^{g:f(\emptyset)}$) is required to address the "count bug" [18], i.e. losing a tuple due to an empty group. Therefore, the function f assigns meaningful values to empty groups. The numbering operator (ν) characterizes each tuple with a unique deterministic number (e.g. a physical tuple identifier). We mainly use the map operator to apply a function to each tuple. Both are required for the unnesting techniques introduced in Section 4.

Figure 1 summarizes the formal definition of the four operators. As a final important extension of our algebra, we allow subscripts to contain algebraic expressions. In our case, such subscripts result from translating nested query blocks in the `where` clause, i.e. algebraic operators appear in selection predicates.

In order to effectively deal with disjunction, we need algebraic operators that split their output into two streams: a positive and a negative one. For example, a selection produces a *positive stream* containing all those tuples for which the selection predicate evaluates to true; the *negative stream* contains the remaining tuples. These operators are called *bypass* operators [17]. To denote the positive and negative streams of a bypass operator, we use the superscripts $+$ and $-$, respectively.

For this paper we need a bypass selection ($\sigma^{\pm}$), a bypass join ($\bowtie^{\pm}$), a bypass semijoin ($\ltimes^{\pm}$) and a bypass antisemijoin ($\triangleright^{\pm}$). Their definitions are given in Figure 1.

Although the algebra is based on sets of tuples, our approach is also applicable for an algebra on bags. However, the focus of this paper is on sets. Our proofs in the Appendix B elaborate on the applicability of our techniques on bags.

3 Unnesting Table Subqueries

We now present our detailed unnesting techniques along the lines of the classification introduced in Section 2.2. As table subqueries (i.e. types N & J) are less demanding, we start out with them. As

Non-standard operators:

$$
\begin{aligned}
e_1 \Gamma_{g;A_1\theta A_2;f}(e_2) &:= \{x.A_1 \circ [g:G] | x \in e_1 \wedge \\
&\quad G = f(\{y | y \in e_2 \wedge x.A_1 \theta y.A_2\})\} \\
e_1 \bowtie^{g:f(\emptyset)}_p e_2 &:= e_1 \bowtie_p e_2 \cup \{x \circ z | x \in e_1 \wedge \\
&\quad \nexists y \in e_2 : p(x,z) \wedge \mathcal{A}(z) = \mathcal{A}(e_2) \wedge \\
&\quad g \in \mathcal{A}(e_2) \wedge \forall a \in (\mathcal{A}(e_2) \setminus g) : \\
&\quad (z.a : \text{NULL} \wedge z.g : f(\emptyset))\} \\
\nu_A(e) &:= \{t_i \circ [A:i] | e = \{t_1, \ldots, t_n\}\} \\
\chi_{a:e_2}(e_1) &:= \{x \circ [a : e_2(x)] | x \in e_1\}
\end{aligned}
$$

Bypass operators:

$$
\begin{aligned}
\sigma^+_p(e) &:= \{x | x \in e \wedge p(x)\} \\
\sigma^-_p(e) &:= e \setminus \sigma^+_p(e) \stackrel{*}{=} \{x | x \in e \wedge \neg p(x)\} \\
e_1 \bowtie^+_p e_2 &:= \{x \circ y | x \in e_1 \wedge y \in e_2 \wedge p(x,y)\} \\
e_1 \bowtie^-_p e_2 &:= (e_1 \times e_2) \setminus (e_1 \bowtie^+_p e_2) \\
&\stackrel{*}{=} \{x \circ y | x \in e_1 \wedge y \in e_2 \wedge \not p(x,y)\} \\
e_1 \ltimes^+_p e_2 &:= \{x | x \in e_1 \wedge \exists y \in e_2 \wedge p(x,y)\} \\
e_1 \ltimes^-_p e_2 &:= e_1 \setminus (e_1 \ltimes^+_p e_2) \\
&\stackrel{*}{=} \{x | x \in e_1 \wedge \nexists y \in e_2 \wedge p(x,y)\} \\
e_1 \rhd^+_p e_2 &:= \{x | x \in e_1 \wedge \nexists y \in e_2 \wedge p(x,y)\} \\
e_1 \rhd^-_p e_2 &:= e_1 \setminus (e_1 \rhd^+_p e_2) \\
&\stackrel{*}{=} \{x | x \in e_1 \wedge \exists y \in e_2 \wedge p(x,y)\}
\end{aligned}
$$

*: only valid for two-valued logic (cf. [17] for details). $[\cdot]$ denotes tuple construction. $\circ$ denotes tuple concatenation. $\mathcal{A}(\mathrm{R})$ is the set of attributes of relation R. $\mathcal{F}(\mathrm{e})$ is the set of free attributes that occur freely in expression e.

Figure 1: Operators of the algebra

type N queries can be unnested by applying the unnesting techniques for type J queries, we deal with them later in Sec. 3.3.

This section is organized as follows. First, we discuss the basic idea of our approach by means of two queries. Second, we present the general solution in the form of unnesting equivalences. On their left-hand side, they have a selection whose predicate contains disjunctively a quantified algebraic expression. On their right-hand side they introduce a bypass operator. Then we move on to more advanced issues.

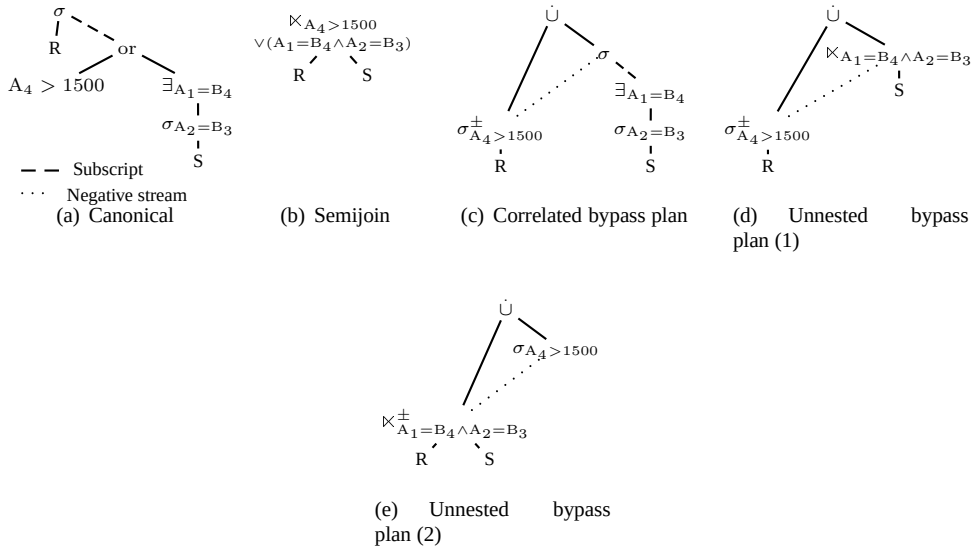
3.1 Disjunctive Linking

The first example query exhibits a nested query block whose disjunctively occurring linking predicate uses the linking operator `IN`. The nested block is of type J, as it has a correlation predicate in its `where` clause.

```
SELECT  DISTINCT *
FROM    R
WHERE   R.A1 IN (SELECT S.B4
                 FROM S
                 WHERE R.A2 = S.B3)
        OR R.A4 > 1500;
```

Q1

Due to the existential nature of the `IN` operator, the algebraic expression resulting from the translation of the query has an existential quantifier subscripted with the linking predicate. The argument of this quantifier itself is again an algebraic expression. As the existential quantifier occurs in a selection operator, the nesting of the query blocks in the query is reflected by a nesting of algebraic expressions, i.e. the subscript of an algebraic operator again contains an algebraic expression.



(a) Canonical (b) Semijoin (c) Correlated bypass plan (d) Unnested bypass plan (1)

(e) Unnested bypass plan (2)

Figure 2: Unnesting strategy for Q1 (sketch)

Translating the query into a nested algebraic expression yields the following:

$$\sigma_{(\exists_{\mathrm{A_1=B_4}}(\sigma_{\mathrm{A_2=B_3}}(\mathrm{S})))\vee \mathrm{A_4>1500}}(\mathrm{R}).$$

Fig. 2(a) shows the more readable tree form of this expression. Evaluating the selection predicate, which contains the inner query block, for every tuple produced by the outer query block (R) does not look very efficient.

To avoid this nested-loop evaluation, we would like to unnest the subquery. In the conjunctive case, nested queries are usually unnested by applying semijoins. Recast into the algebraic framework, this amounts to applying the following equivalence:

$$\sigma_{\exists_{\mathrm{A_1=B_1}}(\sigma_{\mathrm{A_2=B_2}}(\mathrm{S}))}(\mathrm{R}) \equiv \mathrm{R} \ltimes_{\mathrm{A_1=B_1}\wedge\mathrm{A_2=B_2}} \mathrm{S}. \quad (1)$$

Let us see what happens if we apply this traditional technique to our translated query. The resulting expression (called semijoin plan) is shown in Fig. 2(b). The problem is that efficient implementations of joins and semijoins only exist for equijoins whereas in our case the (semi)join condition contains a disjunction at the top level. Implementations other than a simple nested-loop evaluation are beyond reach. Thus, we are again stuck with a nested-loop evaluation.

Let us take a closer look at the query. Assume that a tuple from R satisfies `R.A4 > 1500`. Then, we do not have to check `R.A1 IN` ... for it: it qualifies independently of the outcome of this check. Further, if a tuple from R does not satisfy `R.A4 > 1500`, it must satisfy `R.A1 IN` ... in order to qualify. Thus, it does make sense to split the tuple stream produced by scanning R into two independent streams: one containing those tuples satisfying `R.A4 > 1500` and one with the remaining tuples. The latter then needs to be filtered by `R.A1 IN` .... Finally, as the two streams are disjoint, a disjoint union ($\dot\cup$) on these two streams suffices to produce the final result. Bypass operators capture exactly this kind of reasoning. This is why we want to use them for unnesting. Let us introduce a bypass selection with predicate `R.A4 > 1500`. Fig. 2(c) shows the result. The positive stream of the bypass selection (denoted by a solid line) directly contributes to the final result whereas the negative stream (denoted by dots) is filtered by a selection with the algebraic equivalent of `R.A1 IN` .... This equivalent, $\exists_{\mathrm{A_1=B_4}}(\sigma_{\mathrm{A_2=B_3}}(\mathrm{S}))$, is the filter predicate of a top-level selection and itself contains an algebraic expression (especially a scan of S). Hence, we still have a rather inefficient nested algebraic expression demanding a nested-loop evaluation. However, we are prepared for the final, performance-improving step.

We now introduce a semijoin to unnest the query. Although the details are given in the next subsection, we still would like to give the result:

$$\begin{aligned} \mathrm{e} &= \mathrm{e}_1 \,\dot\cup\, \mathrm{e}_2 \\ \mathrm{e}_1 &= \sigma^{+}_{\mathrm{A}_4>1500}(\mathrm{R}) \\ \mathrm{e}_2 &= (\sigma^{-}_{\mathrm{A}_4>1500}(\mathrm{R})) \ltimes_{\mathrm{A}_1=\mathrm{B}_4 \wedge \mathrm{A}_2=\mathrm{B}_3} (\mathrm{S}). \end{aligned}$$

Fig. 2(d) shows this expression in a more readable form. The semijoin now operates on the negative stream of the bypass selection and the scan of S. Since its condition is now a conjunction of two equality predicates, it can be evaluated very efficiently. We verify this claim in our experiments in Section 5.

Remark. It is important to recognize that commuting the bypass selection with the semijoin (see Fig. 2(e)) is also feasible. This enables further optimization potential. Assume that the second predicate is expensive to evaluate. Then it may be cheaper to perform the semijoin first. This situation is recognized by comparing ranks of the predicates: the one with the lower rank should be evaluated first [26]. For a predicate p the rank ($\mathrm{rank}(p)$) is defined as $\frac{s-1}{c}$, where s is the selectivity of predicate p and c is the cost required to evaluate p.

3.2 Disjunctive Correlation

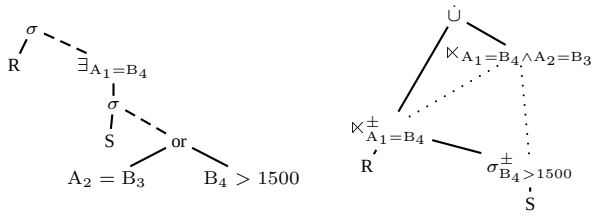
The following query contains a disjunctively occurring correlation predicate, i.e. disjunctive correlation:

```
SELECT  DISTINCT *
FROM    R
WHERE   R.A1 IN (SELECT S.B4
                 FROM S
                 WHERE R.A2 = S.B3
                 OR S.B4 > 1500)
```

Q2

Fig. 3(a) depicts the canonical translation of this query. Note that we cannot unnest this query with the technique of the first example, because the `where` clause of the nested query contains a disjunction with two predicates, one of which is the correlation predicate.

Consider a tuple r of R. If there exists a tuple s in S such that s passes the tests $s.B_4 > 1500$ and $r.A_1 = s.B_4$, then r is contained in the result. This is expressed by the bypass semijoin in Fig 3(b). If no such tuple in S exists, r becomes part of the negative output of the bypass semijoin. For r to qualify, there must be a tuple s in S such that $r.A_1 = s.B_4$ and $r.A_2 = s.B_3$. As those tuples in S with $s.B_4 > 1500$ have been checked before, we only need to consider those $s \in S$ not having $s.B_4 > 1500$. Thus, we have to perform a semijoin on the negative output of the bypass semijoin and the negative output of the selection (see again Fig 3(b)).



(a) Canonical (b) Unnested bypass plan

Figure 3: Unnesting strategy for Q2 (sketch)

$$\sigma_{\exists_{\mathrm{A}_1=\mathrm{B}_1}(\mathrm{S})\vee p}(\mathrm{R}) \equiv e_1 \dot{\cup} e_2 \quad (2)$$
$$e_1 := \sigma_p^+(R)$$
$$e_2 := (\sigma_p^-(R)) \ltimes_{\mathrm{A}_1=\mathrm{B}_1} \mathrm{S}$$
$$\sigma_{\exists_{\mathrm{A}_1=\mathrm{B}_1}(\mathrm{S})\vee p}(\mathrm{R}) \equiv e_1 \dot{\cup} e_2 \quad (3)$$
$$e_1 := \mathrm{R} \ltimes^+_{\mathrm{A}_1=\mathrm{B}_1} \mathrm{S}$$
$$e_2 := \sigma_p(\mathrm{R} \ltimes^-_{\mathrm{A}_1=\mathrm{B}_1} \mathrm{S})$$
$$\sigma_{\nexists_{\mathrm{A}_1=\mathrm{B}_1}(\mathrm{S})\vee p}(\mathrm{R}) \equiv e_1 \dot{\cup} e_2 \quad (4)$$
$$e_1 := \sigma_p^+(R)$$
$$e_2 := (\sigma_p^-(R)) \triangleright_{\mathrm{A}_1=\mathrm{B}_1} \mathrm{S}$$
$$\sigma_{\nexists_{\mathrm{A}_1=\mathrm{B}_1}(\mathrm{S})\vee p}(\mathrm{R}) \equiv e_1 \dot{\cup} e_2 \quad (5)$$
$$e_1 := \mathrm{R} \triangleright^+_{\mathrm{A}_1=\mathrm{B}_1} (\mathrm{S})$$
$$e_2 := \sigma_p(\mathrm{R} \triangleright^-_{\mathrm{A}_1=\mathrm{B}_1} \mathrm{S})$$

Figure 4: Equivalences for disjunctive N queries

3.3 Equivalences

After having worked out the general idea by means of two examples, we now introduce our novel equivalences. They are shown in Figure 4 for type N queries and Figure 5 for type J queries. We provide proofs of these equivalences in the appendix B. The equivalences allow us, for example, to formally derive the unnested plans presented before. On their left-hand side, they contain a selection with a predicate that results from the translation of a nested type N or type J query block. On their right-hand side, they have an unnested algebraic expression with bypass operators.

We first discuss the equivalences for disjunctive linking, then those for disjunctive correlation.

For each equivalence which unnests a type J query with disjunctive linking we have an unnesting equivalence for the corresponding type N query. As a result of the missing correlation predicate the unnested query only contains a simple join condition. For type N queries we need no unnesting equivalences for disjunctive correlation as they have no correlation predicate.

3.3.1 Disjunctive Linking

We split the discussion into two parts, one for positive and one for negative linking predicates.

Positive linking predicates Eqvs. 6 and 7 have been implicitly applied to our sample query Q1. The former yields the plan shown in Fig. 2(d), the latter the one in Fig. 2(e). Both unnest table subqueries exhibiting a positive linking predicate that occurs in a disjunction. The former employs the bypass technique to a disjunctively occurring subquery and unnests the subquery in the negative stream of a bypass selection. The positive stream contains all tuples that match a lower-ranked predicate p. A final union merges both streams without having to eliminate duplicates. The latter equivalence uses the same idea, but the subquery is evaluated first and the evaluation of a higher-ranked (expensive) predicate p is postponed into the negative stream.

Negative linking predicates Eqvs. 8 and 9 unnest queries with a negative linking predicate of the form `NOT IN` and `NOT EXISTS`, which occurs disjunctively. The first equivalence takes advantage of an antisemijoin in the negative stream to unnest the subquery. Note that in order to evaluate the correlation predicate it also becomes a join predicate of the antisemijoin. The second equivalence uses a bypass antisemijoin and postpones the evaluation of a higher-ranked predicate p into the negative stream.

$$\sigma_{\exists_{A_1=B_1}(\sigma_{A_2=B_2}(S))\vee p}(R) \equiv e_1 \dot{\cup} e_2 \quad (6)$$
$$e_1 := \sigma_p^+(R)$$
$$e_2 := (\sigma_p^-(R)) \ltimes_{A_1=B_1 \wedge A_2=B_2} S$$
$$\sigma_{\exists_{A_1=B_1}(\sigma_{A_2=B_2}(S))\vee p}(R) \equiv e_1 \dot{\cup} e_2 \quad (7)$$
$$e_1 := R \ltimes^+_{A_1=B_1 \wedge A_2=B_2} S$$
$$e_2 := \sigma_p(R \ltimes^-_{A_1=B_1 \wedge A_2=B_2} S)$$
$$\sigma_{\nexists_{A_1=B_1}(\sigma_{A_2=B_2}(S))\vee p}(R) \equiv e_1 \dot{\cup} e_2 \quad (8)$$
$$e_1 := \sigma_p^+(R)$$
$$e_2 := (\sigma_p^-(R)) \triangleright_{A_1=B_1 \wedge A_2=B_2} (S)$$
$$\sigma_{\nexists_{A_1=B_1}(\sigma_{A_2=B_2}(S))\vee p}(R) \equiv e_1 \dot{\cup} e_2 \quad (9)$$
$$e_1 := R \triangleright^+_{A_1=B_1 \wedge A_2=B_2} S$$
$$e_2 := \sigma_p(R \triangleright^-_{A_1=B_1 \wedge A_2=B_2} S)$$
$$\sigma_{\exists_{A_1=B_1}(\sigma_{A_2=B_2 \vee p}(S))}(R) \equiv e_1 \dot{\cup} e_2 \quad (10)$$
$$e_1 := R \ltimes^+_{A_1=B_1} e_3$$
$$e_2 := (R \ltimes^-_{A_1=B_1} e_3) \ltimes_{A_1=B_1 \wedge A_2=B_2} (\sigma_p^-(S))$$
$$e_3 := \sigma_p^+(S)$$
$$\sigma_{\nexists_{A_1=B_1}(\sigma_{A_2=B_2 \vee p}(S))}(R) \equiv e_1 \dot{\cup} e_2 \quad (11)$$
$$e_1 := R \triangleright^+_{A_1=B_1} e_3$$
$$e_2 := (R \triangleright^-_{A_1=B_1} e_3) \triangleright_{A_1=B_1 \wedge A_2=B_2} \sigma_p^-(S)$$
$$e_3 := \sigma_p^+(S)$$

Figure 5: Equivalences for disjunctive J queries

3.3.2 Disjunctive Correlation

Eqvs. 10 and 11 unnest queries with a disjunctive correlation predicate.

Positive linking predicates Eqv. 10 is used for unnesting queries whose linking operator is `IN` or `EXISTS`. The core benefit of this equivalence results from the clever filtering of tuples in R. First, the linking predicate is only checked for tuples of S that match the cheaper predicate p. Only the remaining tuples of R are checked for matches that pass the correlating predicate.

Negative linking predicates Eqv. 11 handles the linking operators `NOT EXISTS` and `NOT IN`. It applies the same strategy as explained in the previous equivalence that handles positive linking predicates. However, not that now an antisemijoin replaces the semijoin to check for the negative linking operator.

In both equivalences the predicate p can be a simple predicate or a nested query itself.

3.4 Completeness of Equivalences

It is important to understand that the equivalences suffice to unnest all nested queries with linking predicate `IN` or `EXISTS` and their negated counterparts. The reason is that the translation of these

queries exactly results in the patterns on the left-hand side of our equivalences. When no correlation predicate exists these equivalences can still be used where the constant TRUE replaces the predicate. Similarly, we can ignore the subscript of the quantifier for queries with linking operator EXISTS.

3.5 Tree Queries

Obviously, the equivalences introduced in the last subsection are capable of unnesting simple nested queries, i.e. those containing just a single nested block. It might be less obvious that they also allow us to unnest tree and linear queries. In this and the following subsection, we demonstrate that this is indeed the case.

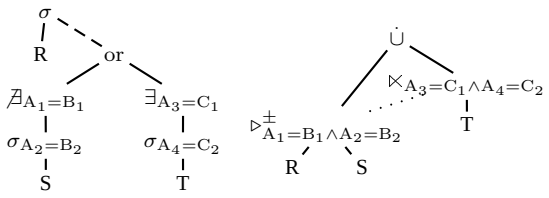
Let us start with the following tree query:

```
SELECT  DISTINCT *
FROM    R
WHERE   A1 NOT IN (SELECT B1
                   FROM    S
                   WHERE   A2 = B2)
        OR
        A3 IN   (SELECT C1
                 FROM    T
                 WHERE   A4 = C2)
```

Q3

In this query, we have two nested query blocks on the same level, one using IN and the other using NOT IN. Their linking predicates are connected by a disjunction. Both are correlated (i.e. of type J).

We briefly demonstrate that Eqvs. 9 and 1 enable us to unnest this query. The canonical translation of the query is given in Fig. 6(a). We can unnest this query by first applying Eqv. 9. This introduces a bypass antisemijoin whose join predicates are the linking and correlation predicates. Then we apply Eqv. 1 in the negative stream. The result is shown in Fig. 6(b).



(a) Canonical (b) Unnested bypass plan

Figure 6: Unnesting strategy for Q3 (sketch)

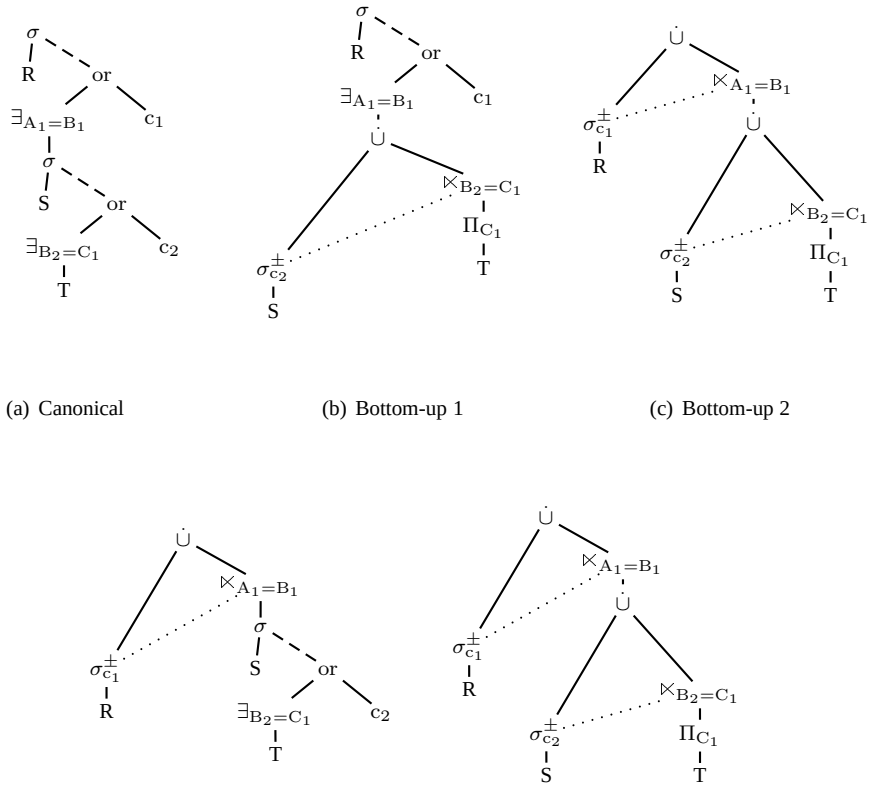
3.6 Linear Queries

The above strategies also work for linear queries with type N nested query blocks, i.e. we can choose to apply the equivalences, in a bottom-up or a top-down fashion. Fig. 7 compares both strategies for the following query which is a linear query.

```
SELECT  A1
FROM    R
WHERE   A1 IN   (SELECT B1
                 FROM   S
                 WHERE  B2 IN (SELECT C1
                               FROM T)
                        OR c2)
        OR c1
```

Q4

Next, we demonstrate how to unnest a linear query whose linking predicate occurs in a disjunction. Consider the following query containing two subqueries of type J:



(a) Canonical (b) Bottom-up 1 (c) Bottom-up 2

(d) Top-down 1 (e) Top-down 2

Figure 7: Unnesting strategy for Q4

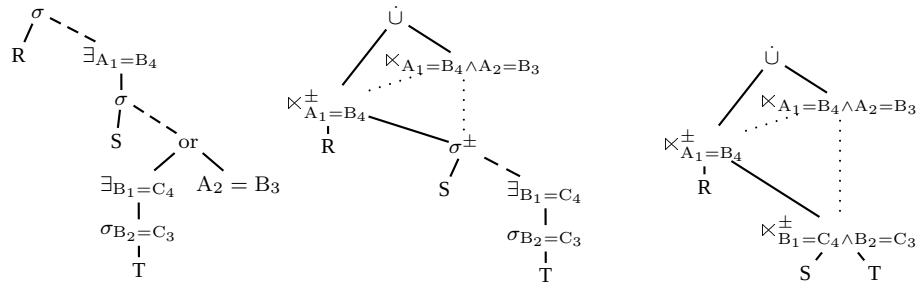
```
SELECT A1
FROM  R
WHERE A1 IN (SELECT B4
             FROM    S
             WHERE   A2 = B3
             OR      B1 IN (SELECT C4
                            FROM T
                            WHERE B2 = C3))
```

Q5

The deepest nested query block is connected to its outer query block by a disjunction. We depict our top-down unnesting strategy in Figure 8. Subfigure 8(a) contains the canonical translation. Applying Eqv. 10 for positive linking operators, as already shown for Query Q2, yields the intermediate plan from Fig. 8(b). Although the middle query block is already unnested, we would like to unnest the deepest nested block, too. This can finally be done applying Eqv. 1. Fig. 8(c) shows the final result.

3.7 Duplicate Handling

Our unnesting equivalences are defined for an algebra over sets of tuples. We now briefly argue (see appendix B for the proofs) that all of the equivalences presented in Fig. 5 are also correct for an algebra over multisets. This is necessary because by default SQL queries do not remove duplicates, i.e. they yield multisets of tuples.



(a) Canonical (b) Unnested bypass plan (1) (c) Unnested bypass plan (2)

Figure 8: Unnesting strategy for Q5 (sketch)

For our equivalences for table queries, two issues have to be considered. The first is the bypass technique. Bypassing does not cause any problems because it splits its input into two disjoint multisets, i.e. equal tuples go the same way. The final (disjoint) union merges both inputs without duplicate elimination. Hence, no duplicates are falsely eliminated. Further, no new (false) duplicates are introduced as long as there are no expressions producing duplicates in any of the two streams.

Hence, the second source of possible problems are the operators that are applied in the streams. All equivalences from Fig. 5 employ a semijoin or an antisemijoin. For both of them implementations are conceivable which adhere to the selection-like semantics, i.e. they neither wrongly eliminate nor generate duplicates. Hence, it is safe to apply our unnesting techniques to multisets.

4 Unnesting Scalar Subqueries

Unnesting scalar queries is difficult and error-prone. Particularly, empty groups and duplicates [18] have been sources of errors. As a new challenge, we now support disjunctive linking and correlation.

This section is organized as follows. First, we discuss the basic idea of our approach by means of two simple queries. Second, we present the general solution in the form of unnesting equivalences. On their left-hand side, they have a selection whose predicate contains disjunctively a nested algebraic expression with a top-level aggregation. On their right-hand side they introduce a bypass operator. After the discussion of our equivalences, we move on to more advanced issues, i.e. unnesting of linear queries, tree queries, and a discussion of duplicate handling.

Scalar subqueries of type A are easy to handle. Their result can be computed independently of the outer query, and materialization costs are negligible. Thus, it suffices to materialize the computed result. As their treatment is so simple, we do not discuss them any further but concentrate on the more challenging type JA queries.

4.1 Disjunctive Linking

In the following query, the subquery is of type JA, as it contains a predicate which refers to the attribute A_2 defined in the outer block and the attribute B_2 defined in the inner block:

```
SELECT  DISTINCT *
FROM    R
WHERE   A1 =  (SELECT COUNT(DISTINCT *)
               FROM    S
               WHERE   A2 = B2)
        OR A4 > 1500
```

Q6

The linking predicate compares the attribute A_1 with the result of the aggregation (i.e. count) in the inner query's `select` clause. Moreover, the linking predicate occurs in a disjunction. Translation into the algebra yields the following expression:

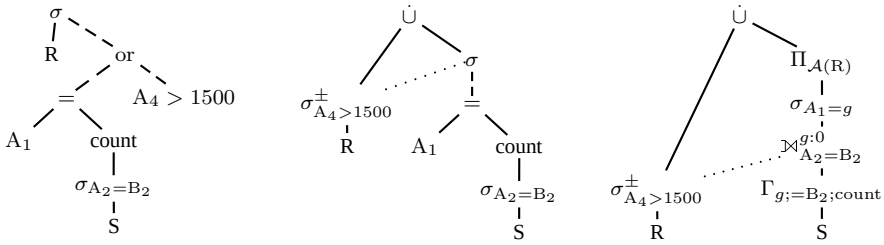
$$\sigma_{A_1=\mathrm{count}(\sigma_{A_2=B_2}(S))\vee A_4>1500}(R).$$

Fig. 9(a) presents this canonical evaluation plan in a more readable form.

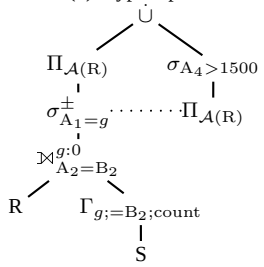
For the evaluation of this query, the inner query has to be evaluated for every tuple produced by the outer query block, i.e. in nested loops. Obviously, this is not very efficient. In order to unnest type JA queries in the conjunctive case, it is common practice to apply grouping on the correlation attributes of the inner query to perform the aggregation. Then, an outerjoin is performed to accomplish the match with the tuples from the outer query block with the grouped and aggregated result. The following equivalence captures this procedure:

$$\begin{aligned} & \sigma_{\mathrm{A_1}\theta\mathrm{f}(\sigma_{\mathrm{A_2}=\mathrm{B_2}}(\mathrm{S}))}(\mathrm{R}) \\ \equiv\ & \Pi_{\mathcal{A}(R)}(\sigma_{\mathrm{A_1}\theta\mathrm{g}}(\mathrm{R} \ltimes^{g:f(\emptyset)}_{\mathrm{A_2}=\mathrm{B_2}} (\Gamma_{g;=\mathrm{B_2};\mathrm{f}}(\mathrm{S})))). \end{aligned} \tag{12}$$

If the predicate in the outer query block was a conjunction, we could apply this equivalence without hesitation. However, if we apply this equivalence to the translation of the query, the resulting plan contains an outerjoin with a disjunctive join predicate. In this case, the only known implementation is the rather inefficient nested-loop implementation.



(a) Canonical (b) Bypass plan (c) Unnested bypass plan (1)



(d) Unnested bypass plan (2)

Figure 9: Unnesting strategy for Q6 (sketch)

Let us take a closer look at the query. Assume that a tuple from R satisfies `A4 > 1500`. Then, we do not have to check `A1 =` ... for it: it qualifies independently of the outcome of this check. Further, if a tuple from R does not satisfy `A4 > 1500`, it must satisfy `A1 =` ... in order to qualify. Thus, it does make sense to split the tuple stream produced by scanning R into two independent streams: one containing those tuples satisfying `A4 > 1500` and one with the remaining tuples. The latter then needs to be filtered by `A1 =` .... Finally, as the two streams are disjoint, a disjoint union ($\dot\cup$) on them suffices to produce the final result. Bypass operators capture exactly this kind of reasoning. This is why we want to use them for unnesting. Let us therefore introduce a bypass selection with predicate `A4 > 1500`. The following algebraic expression results from this:

$$\begin{aligned} \mathrm{e} &= \mathrm{e}_1 \dot\cup \mathrm{e}_2 \\ \mathrm{e}_1 &= \sigma^{+}_{\mathrm{A_4}>1500}(\mathrm{R}) \\ \mathrm{e}_2 &= \sigma_{\mathrm{A_1}=\mathrm{count}(\sigma_{\mathrm{A_2}=\mathrm{B_2}}(\mathrm{S}))}(\sigma^{-}_{\mathrm{A_4}>1500}(\mathrm{R})). \end{aligned}$$

Fig. 9(b) shows the more readable result. The positive stream of the bypass selection (denoted by a solid line) directly contributes to the final result. In addition, the negative stream (denoted by dots) is filtered by a selection with the algebraic equivalent of `A1 =` ....

With this expression as a starting point, we can derive the unnested bypass plan shown in Fig. 9(c). Those tuples of R that satisfy the predicate $\mathrm{A}_4 > 1500$ directly contribute to the result. Only for the remaining tuples, we need to check the condition expressed by $\mathrm{A}_1 = \ldots$. This check is represented in the plan by the same trick used to unnest conjunctively nested queries. In a first step, we group by the linking attribute B_2 of the inner query and calculate the aggregate. Then, we perform an outerjoin. For those tuples of R that do not find a join partner, the default handling of the outerjoin assures correctness. Last, we evaluate the linking predicate. It has been rewritten since the aggregation result has been materialized in the attribute g. A final projection on the attributes of R guarantees the same schema in the positive as well as the negative stream before unioning the two streams.

Remark. It is important to recognize that commuting the bypass selection with the selection in the negative stream (see Fig. 9(d)) is also feasible. This enables further optimization potential. Assume that the second predicate is expensive to evaluate. Then it may be cheaper to perform the selection first. This situation is recognized by comparing ranks of the predicates: the one with the lower rank should be evaluated first [26]. For a predicate p the rank ($\mathrm{rank}(p)$) is defined as $\frac{s-1}{c}$, where s is the selectivity of predicate p and c is the cost required to evaluate p.

That is, if the predicate $\mathrm{A}_4 > 1500$ was a very expensive one, we could evaluate the subquery first. In this case, the selection checking the linking predicate turns into a bypass selection, and the predicate $\mathrm{A}_4 > 1500$ is evaluated only in the bypass selection's negative stream. A projection on the attributes of R in both streams ensures the final schema.

4.2 Disjunctive Correlation

Not only the linking predicate can occur in a disjunction. The following query contains a disjunctively occurring correlation predicate, i.e. disjunctive correlation:

```
SELECT  DISTINCT *
FROM    R
WHERE   A1 =  (SELECT COUNT(DISTINCT *)
               FROM    S
               WHERE   A2 = B2
                       OR B4 > 1500)
```

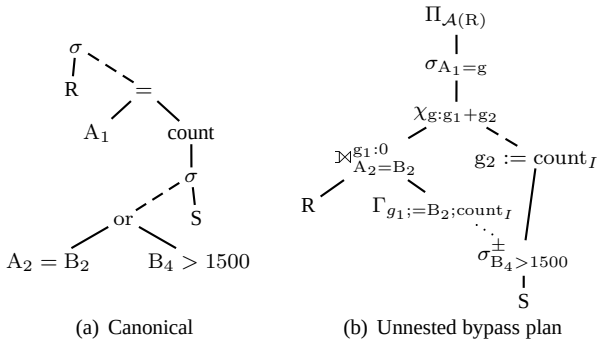
Q7

The aggregation function in the `select` clause of the nested query combines all tuples that pass for the correlation predicate $\mathrm{A}_2 = \mathrm{B}_2$ or the simple predicate $\mathrm{B}_4 > 1500$.

Similar to the canonical translation of Query Q6, but with the disjunction in the selection predicate of the nested selection, the canonical translation gives us (see also Figure 10(a))

$$\sigma_{\mathrm{A}_1 = \mathrm{count}(\sigma_{\mathrm{A}_2 = \mathrm{B}_2 \vee \mathrm{B}_4 > 1500}(\mathrm{S}))}(\mathrm{R}).$$

Unnesting is not possible with any of the existing techniques. For the following, we refer to the plan in Fig. 10(b). The general idea to unnest this query is based on two facts: (1) the aggregation function (count) is decomposable [6], and (2) the predicate $\mathrm{B}_4 > 1500$ can be evaluated independently of the outer query. This allows us to calculate the total count of the inner query from adding up the counts calculated for two disjoint subsets. Take a look at the bottom of the plan in Fig. 10(b). In the positive stream of the bypass selection (denoted by a solid line), we count all tuples from relation S that satisfy the predicate $\mathrm{B}_4 > 1500$. Those tuples of S that do not satisfy $\mathrm{B}_4 > 1500$ go into the negative stream. Here, they have to pass the correlation predicate before they contribute to the total count. Hence, we group them and evaluate the count function for each group. Analogously to the general unnesting strategy (see Eqv. 12), we apply an outerjoin to perform the match with the outer relation R and — in order to avoid the count bug — assign 0 to the attribute g_1 for those tuples from R that do not have a join partner. At the end, we need a map to add up the separately calculated values for g_1 and g_2 to give the total count g. The subsequent selection with predicate $\mathrm{A}_1 = \mathrm{g}$ checks the linking predicate. The final projection assures that the result only contains attributes from R.



(a) Canonical (b) Unnested bypass plan

Figure 10: Unnesting strategy for Q7 (sketch)

4.3 Equivalences

Before we can present our unnesting rewrites for scalar queries of type JA, we need to define *decomposability* of aggregate functions [21]. Let X, Y, and Z be sets with $X = Y \dot\cup Z$ and $Y \cap Z = \emptyset$. A scalar aggregate function $f : X \to \mathcal{N}$ is *decomposable* if there exist functions

$$\begin{aligned} f_I : X &\to \mathcal{N}' \\ f_O : \mathcal{N}', \mathcal{N}' &\to \mathcal{N} \end{aligned}$$

with $f(X) = f_O(f_I(Y), f_I(Z))$.

The SQL aggregation functions used most often are decomposable:

$$\begin{aligned} \text{count(X)} &\equiv \text{count}_I\text{(Y) + count}_I\text{(Z)} \\ \text{sum(X)} &\equiv \text{sum}_I\text{(Y) + sum}_I\text{(Z)} \\ \text{avg(X)} &\equiv \tfrac{\text{sum}_I(Y)+\text{sum}_I(Z))}{\text{count}_I(Y)+\text{count}_I(Z)} \\ \text{min(X)} &\equiv \text{min}_O\text{(min}_I\text{(Y),min}_I\text{(Z))} \\ \text{max(X)} &\equiv \text{max}_O\text{(max}_I\text{(Y),max}_I\text{(Z)).} \end{aligned}$$

The discussion of our equivalences (see Fig. 11) is split into two halves. In the first half, we discuss unnesting equivalences for queries with disjunctive linking. In the second half, we advance to unnesting equivalences for queries with disjunctive correlation. The proofs for these equivalences can be found in the appendix B.

4.3.1 Disjunctive Linking

In the equivalences, let f be an aggregation function.

Equivalences 13 and 15 are used to unnest scalar queries whose linking predicate occurs disjunctively. The former postpones the evaluation of the unnested subquery into the negative stream of a bypass selection. Basically, the unnesting technique is adapted from Eqv. 12. The idea of this equivalence has already been explained using Query Q6. Fig. 9(c) depicts this strategy.

The latter equivalence is used for first evaluating the unnested subquery, i.e. the linking predicate, and postpone the evaluation of the second predicate into the negative stream of the bypass selection. Fig. 9(d) visualizes this strategy.

Equivalences 14 and 16 These equivalences use the binary grouping operator instead of a sequence of unary grouping and leftouterjoin. They support an arbitrary correlation predicate $\text{A}_2\theta_2\ \text{B}_2$. For their validity they require the correlation predicate A_2 to be a key of R. This is necessary to be able to preserve the other attributes of R, e.g. A_1 or the free attribute of p. They are subsequently used in the expression.

$$\sigma_{\mathrm{p}\vee\mathrm{A}_1\theta(\mathrm{f}(\sigma_{\mathrm{A}_2=\mathrm{B}_2}(\mathrm{S})))}(\mathrm{R}) \equiv e_1 \dot{\cup} e_2 \quad (13)$$
$$e_1 := \sigma_p^+(\mathrm{R})$$
$$e_2 := \Pi_{\mathcal{A}(R)}(\sigma_{\mathrm{g}\theta\mathrm{A}_1}((\sigma_p^-(\mathrm{R})) \bowtie^{\mathrm{g:f}(\emptyset)}_{\mathrm{A}_2=\mathrm{B}_2} (\Gamma_{\mathrm{g};=\mathrm{B}_2;\mathrm{f}}(\mathrm{S}))))$$
$$\Pi_{A_1,A_2}(\sigma_{p\vee\mathrm{A}_1\theta_1(\mathrm{f}(\sigma_{\mathrm{A}_2\theta_2\mathrm{B}_2}(\mathrm{S})))}(\mathrm{R})) \equiv \Pi_{A_1,A_2}(e_1 \dot{\cup} e_2) \quad (14)$$
$$e_1 := \sigma_p^+(\mathrm{R})$$
$$e_2 := \Pi_{\mathcal{A}(R)}(\sigma_{\mathrm{g}\theta_1\mathrm{A}_1}((\sigma_p^-(\mathrm{R}))\Gamma_{\mathrm{g};\mathrm{A}_2\theta_2\mathrm{B}_2;\mathrm{f}}(\mathrm{S})))$$
$$\sigma_{\mathrm{p}\vee\mathrm{A}_1\theta(\mathrm{f}(\sigma_{\mathrm{A}_2=\mathrm{B}_2}(\mathrm{S})))}(\mathrm{R}) \equiv \Pi_{\mathcal{A}(R)}(e_1 \dot{\cup} e_2) \quad (15)$$
$$e_1 := \sigma^+_{\mathrm{g}\theta\mathrm{A}_1}((\mathrm{R}) \bowtie^{\mathrm{g:f}(\emptyset)}_{\mathrm{A}_2=\mathrm{B}_2} (\Gamma_{\mathrm{g};=\mathrm{B}_2;\mathrm{f}}(\mathrm{S})))$$
$$e_2 := \sigma_p(\sigma^-_{\mathrm{g}\theta\mathrm{A}_1}((\mathrm{R}) \bowtie^{\mathrm{g:f}(\emptyset)}_{\mathrm{A}_2=\mathrm{B}_2} (\Gamma_{\mathrm{g};=\mathrm{B}_2;\mathrm{f}}(\mathrm{S}))))$$
$$\Pi_{A_1,A_2}(\sigma_{\mathrm{p}\vee\mathrm{A}_1\theta_1(\mathrm{f}(\sigma_{\mathrm{A}_2\theta_2\mathrm{B}_2}(\mathrm{S})))}(\mathrm{R})) \equiv \Pi_{A_1,A_2}(e_1 \dot{\cup} e_2) \quad (16)$$
$$e_1 := \sigma^+_{\mathrm{g}\theta_1\mathrm{A}_1}((\mathrm{R})\Gamma_{\mathrm{g};\mathrm{A}_2\theta_2\mathrm{B}_2;\mathrm{f}}(\mathrm{S}))$$
$$e_2 := \sigma_p(\sigma^-_{\mathrm{g}\theta_1\mathrm{A}_1}((\mathrm{R})\Gamma_{\mathrm{g};\mathrm{A}_2\theta_2\mathrm{B}_2;\mathrm{f}}(\mathrm{S})))$$
$$\sigma_{\mathrm{A}_1\theta\mathrm{f}(\sigma_{\mathrm{A}_2=\mathrm{B}_2\vee p}(\mathrm{S}))}(\mathrm{R}) \equiv \Pi_{\mathcal{A}(R)}(\sigma_{\mathrm{A}_1\theta\mathrm{g}}(\chi_{\mathrm{g:f}_O(\mathrm{g}_1,\mathrm{e}_2)}(\mathrm{e}_1))) \quad (17)$$
$$e_1 := R \bowtie^{\mathrm{g}_1:\mathrm{f}_I(\emptyset)}_{\mathrm{A}_2=\mathrm{B}_2} (\Gamma_{\mathrm{g}_1;=\mathrm{B}_2;\mathrm{f}_I}(\sigma_p^-(\mathrm{S})))$$
$$e_2 := \mathrm{f}_I(\sigma_p^+(\mathrm{S}))$$
$$\Pi_{A_1,A_2}(\sigma_{A_1\theta_1 f(\sigma_{A_2\theta_2 B_2\vee p}(S))}(R)) = \Pi_{A_1,A_2}(\sigma_{A_1\theta_1 g}(\chi_{g:f_O(g_1,e_2)}(e_1))) \quad (18)$$
$$e_1 := (R)\Gamma_{g_1;A_2\theta_2 B_2;f_I}(\sigma_p^-(S))$$
$$e_2 := f_I(\sigma_p^+(S))$$
$$\sigma_{\mathrm{A}_1\theta\mathrm{f}(\sigma_{\mathrm{A}_2=\mathrm{B}_2\vee p}(\mathrm{S}))}(\mathrm{R}) \equiv \Pi_{\mathcal{A}(R)}(\sigma_{\mathrm{A}_1\theta\mathrm{g}}((R')\Gamma_{\mathrm{g};t1=t1';\mathrm{f}}(\rho_{\mathrm{t}1'\leftarrow\mathrm{t}1}(e_1 \dot{\cup} e_2)))) \quad (19)$$
$$R' := \nu_{t1}(\mathrm{R})$$
$$e_1 := R' \bowtie^+_{\mathrm{A}_2=\mathrm{B}_2} \mathrm{S}$$
$$e_2 := \sigma_p(R' \bowtie^-_{\mathrm{A}_2=\mathrm{B}_2} \mathrm{S})$$

Figure 11: Equivalences for disjunctive queries of type JA

4.3.2 Disjunctive Correlation

Equivalence 17 handles queries whose correlation predicate occurs in a disjunction. Its limitation is that the predicate expression p must not be a subquery itself. Moreover, this equivalence requires the aggregation function to be decomposable and the correlation predicate to be an equality predicate. Fig. 10 illustrates the idea of this equivalence for the query from Section 4.2. Its main idea is to generate partial, intermediate results, which are then combined by the subsequent map operator.

Equivalence 18 This equivalence uses the binary grouping operator instead of a sequence of unary grouping and leftouterjoin. It supports an arbitrary correlation predicate $\mathrm{A}_2\theta_2\ \mathrm{B}_2$. It requires the correlation predicate A_2 to be a key of R. This is necessary to be able to preserve the other attributes of R, e.g. A_1 or the free attribute of p.

Equivalence 19 in contrast, is more generally applicable. There are no restrictions on the aggregate function. In addition, predicate p may contain a nested query. The bypass join generates one positive stream for those tuples satisfying the correlation predicate and a complementary negative one where p is checked. Beforehand, we need to introduce a numbering operator ν, which enables us to correctly reassemble the results during the binary grouping.

4.4 Completeness of Equivalences

Our equivalences handle all cases of scalar subqueries with disjunctive linking and correlation. Thereby, the linking predicate can consist of an arbitrary linking operator ($\{=, \neq, <, \leq, >, \geq\}$).

Let us make sure that the canonical translation of a scalar subquery always leads to a pattern that matches the left-hand side of one of our equivalences. In this situation, the canonical translation results in an aggregate function call f as top-level member of a selection predicate, which is part of the linking predicate. In Eqv. 13 and 15 this corresponds to disjunctive linking. The argument of the aggregation function is again a selection checking for the correlation predicate, which in Eqv. 17 and 19 occurs in a disjunction. Remember that the former is a special case of the latter, where p must not be a subquery itself and the aggregation function f must be decomposable.

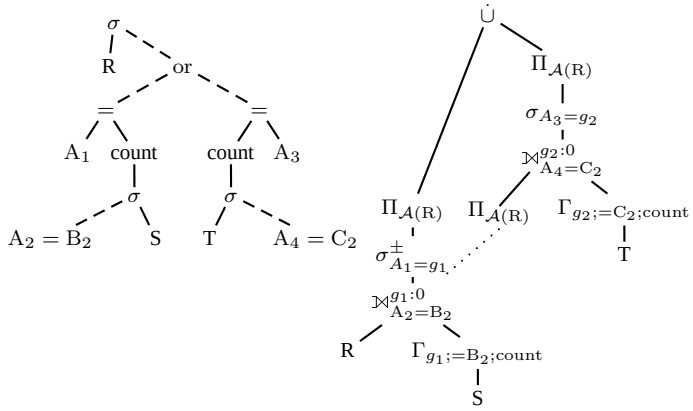
4.5 Tree Queries

Tree queries of type JA can be unnested quite easily by successive applications of our equivalences. Consider the following query:

```
SELECT  DISTINCT *
FROM    R
WHERE   A1 =(SELECT COUNT(DISTINCT *)
             FROM    S
             WHERE   A2 = B2)
        OR
        A3 = (SELECT COUNT(DISTINCT *)
              FROM T
              WHERE A4 = C2)
```

Q8

Figure 12 illustrates the canonical translation and the result of the following two steps. In a first step, we unnest the query using Eqv. 13 applied to the predicate, i.e. the subquery, having the lowest rank. In the second step, we have to make a choice: either we apply Eqv. 13 again, if there exists another subquery on the same level, or we apply Eqv. 12, if this is not the case. Because none of the subqueries contains a nested query, we apply Eqv. 12.



(a) Canonical (b) Unnested bypass plan

Figure 12: Unnesting strategy for Q8 (sketch)

4.6 Linear Queries

Linear JA queries are a special case of disjunctive correlation. The second predicate in the disjunction is again a linking predicate, as in the following query:

```
SELECT  DISTINCT *
FROM    R
WHERE   A1 =(SELECT COUNT(DISTINCT *)
             FROM    S
             WHERE   A2 = B2
             OR
             B3 = (SELECT COUNT(DISTINCT *)
                   FROM T
                   WHERE B4 = C2))
```

Q9

As you can see in Fig. 13, we start with the canonical translation and unnest in a top-down fashion. In a first step, we apply Eqv. 19. The result is shown in Fig. 13(b). From here, it becomes obvious that for the deepest nested expression Eqv. 12 can be applied which yields the final plan shown in Fig. 13(c).

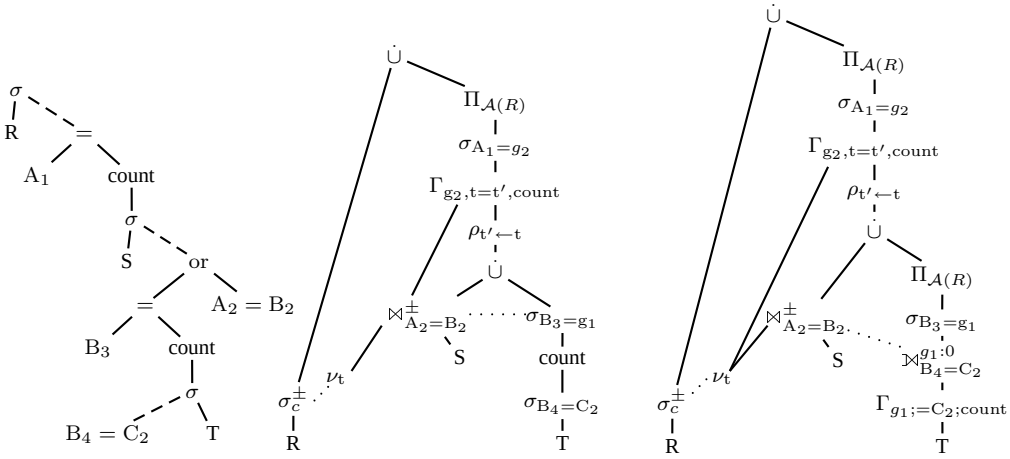


Figure 13: Unnesting strategy for Q9 (sketch)

4.7 Duplicate Handling

Let us make sure that all equivalences mentioned in this section are also correct when they are based on an algebra over multisets. The right-hand side of Equivalences 13, 15, and 17 contains an unary grouping on the nested query block's input, followed by a leftouterjoin. We observe that after grouping on the correlation attribute of the inner query, each value of the grouping attributes occurs exactly once. This key is later used as join attribute in the leftouterjoin. As a result, this join either finds exactly one matching tuple for each tuple resulting from the outer query block, or it keeps the outer block's tuple in order to preserve empty groups. Hence, the cardinality of the leftouterjoin is exactly the one of the outer relation R.

In Eqv. 17, we already ensured correctness of the duplicate semantic for expression e_1 above. The map operator does not influence duplicates, as it only computes the correct aggregate value.

The numbering operator ν in Eqv. 19 turns the multiset R into a set and thereby ensures correctness for multisets.

Each equivalence introduces one bypass operator. This operator, in the unnested plan, partitions its input into two disjoint sets. Thus, it neither creates duplicates nor discards any tuples. Moreover, the final union is defined for multisets, ensuring that duplicates are handled correctly.

5 Evaluation

To demonstrate the effectiveness of our unnesting techniques, we performed an extensive evaluation of the techniques presented in this paper. Specifically, we measured the execution times of the canonical and the unnested plans using our hybrid relational and XML DBMS Natix [10]. Additionally, we compared the resulting evaluation times with those measured for three major commercial database management systems, for anonymity reasons henceforth nicknamed DB 1, DB 2, and DB 3. For the same reason, we cannot present specific query evaluation plans for the commercial systems. However, the strategy can be predicted by comparing the evaluation times with the evaluation times resulting from the execution of the canonical plans in Natix.

After a brief description of the experimental setup, we present evaluations for simple table and scalar subqueries with disjunctive linking and correlation. Further, we also include linear and tree queries in our discussion.

5.1 Datasets

The evaluation is performed on several data sets based on two schemas: (1) the schema of the TPC-H benchmark [29] and (2) the schema RST used for the example queries from Section 4. The latter schema contains three tables (R, S, and T), each consisting of four columns $\mathrm{A}_i \in \mathcal{A}(\mathrm{R})$, $\mathrm{B}_i \in \mathcal{A}(\mathrm{S})$, and $\mathrm{C}_i \in \mathcal{A}(\mathrm{T})$ for $i = 1 \ldots 4$.

The data sets for the TPC-H benchmark are generated using the benchmark generator (dbgen) with scaling factors 0.01, 0.05, 0.5, 1, 5 and 10. This results in moderate database sizes of 11MB - 11GB.

For the independently scaled relations of the RST schema, we generated instances with scaling factors (sf) 1, 5, and 10. This led to 10.000, 50.000, and 100.000 rows and amounts to small tables of 178K, 1.1M, and 2.1M. In the evaluations below, SF1 denotes the scaling factor of the outer query block and SF2 the scaling factor of the inner query block. We did not use larger scaling factors because neither the canonical plans nor the commercial systems scaled well.

5.2 Settings

We used two comparable PCs with 1 GB of RAM each for the experiments. Not all commercial systems are available for the same operating system. We executed Natix and two of the commercial systems on one of the PCs running Linux 2.6.11. The other commercial system ran on the other PC under Windows XP. All queries were executed with a cold buffer. Further, for optimizing the queries using the commercial systems, we used the highest optimization level possible. However, we did not create any indexes.

Because of the necessity of using two different systems, the resulting evaluation times are not exactly comparable. However, the growth of the resulting evaluation times already demonstrates the effectiveness of our unnesting approaches.

5.3 Table Subqueries

5.3.1 Disjunctive Linking

In this section, we present a performance evaluation for Query Q1 on our synthetic schema and the Query 4d shown in the following on the TPC-H schema.

```
SELECT  o_orderpriority, count(*) as order_count
FROM    orders
WHERE   o_orderpriority = '1-URGENT'
        OR EXISTS (
          SELECT *
          FROM  lineitem
          WHERE l_commitdate < l_receiptdate
          AND   o_orderkey = l_orderkey)
GROUP BY o_orderpriority ORDER BY o_orderpriority
```

For the evaluation with Natix, we generated three alternative evaluation plans for both queries. For Query Q1 these are:

Canonical: A correlated plan, as depicted in Fig. 2(a).

Semijoin: An unnested plan, as shown in Fig. 2(b) using a nested-loop semijoin implementation.

Unnested: An unnested bypass plan, as given in Fig. 2(d).

For Query 4d we also generated this three alternatives. Their evaluation plans are depicted in Fig. 14.

The execution times (in seconds) of running these queries are summarized in Fig. 15. We aborted the evaluation of queries after six hours. These cases are denoted by n/a. The first six tables compare the runtimes for Query Q1 and the bottom table for Query 4d.

For both queries our unnested approach outperforms even the fastest commercial system (DB 2). For the largest scaling factors, our approach outperforms DB 2 by a factor of three or more. The remaining commercial systems show a performance similar to our canonical plan, i.e. a nested-loop approach. The results indicate that these commercial systems perform a rather naïve evaluation. The unacceptable evaluation times of these systems and the canonical plan underline the importance of unnesting nested queries. In general, the unnested plans of the two example queries finish up to four orders of magnitude faster than the naïve nested-loop evaluation.

5.3.2 Disjunctive Correlation

Besides the evaluation of type J queries with disjunctive linking, we performed an evaluation for queries with disjunctive correlation.

Therefore, we executed Query Q2 on our synthetic data set and generated two alternative query execution plans for Natix. The first alternative is based on the canonical translation (Fig. 3(a)) and the second is the unnested plan (Fig. 3(b)). Fig. 16 presents the results of this evaluation.

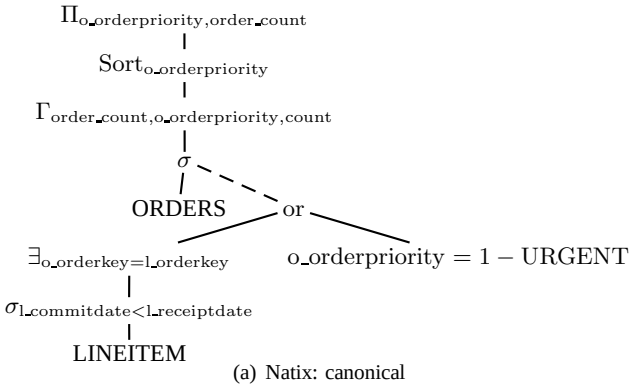
In the experiments, systems DB 1 and DB 2 perform as fast as our unnested plan. However, in contrast to the acceleration of nested queries, unnesting gives the cost-based optimizer more opportunities for reordering operators and selection of physical implementations. Finally, we point out the enormous performance gains realized compared to the naïve nested-loop evaluation chosen by system DB 3 and to our canonical plan.

5.3.3 Tree Table Subqueries

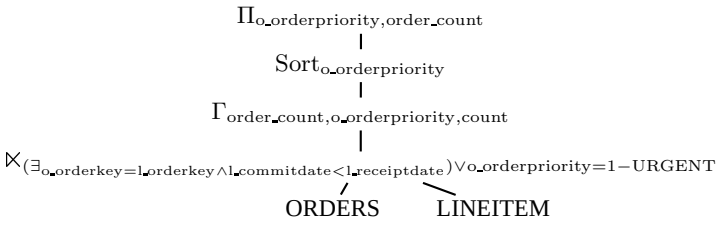
To evaluate queries with a tree structure, we executed the following query on the synthetic dataset.

```
SELECT DISTINCT *
FROM R
WHERE A1 IN (SELECT B4
                FROM S
                WHERE A2 = B3)                                    Q10
        OR
        A1 IN (SELECT C4
                FROM T
                WHERE A2 = C3)
```

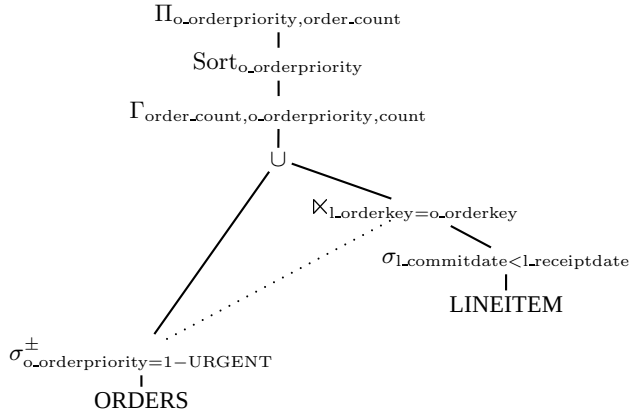
Query Q10 is a tree query, because it has two nested query blocks nested inside the top-level query block. Fig. 17 contains the results of our evaluation. In the evaluations SF1 denotes the scaling factor for the outer relation. SF2 denotes the scaling factor of both inner relations, i.e. S and T.



(a) Natix: canonical



(b) Natix: semijoin



(c) Natix: unnested

Figure 14: Query plan sketches for Query 4d

5.3.4 Linear Table Subqueries

Consider the following linear query and the according results of its evaluation (see Fig. 18). Again, SF1 denotes the scaling factor for the outer relation. SF2 denotes the scaling factor of both inner relations, i.e. S and T.

SF1 \ SF2	1	5	10
1	10.1	51.3	102
5	50.6	260	520
10	100	522	1043

(a) DB 1 (Q1)

SF1 \ SF2	1	5	10
1	0.21	0.28	0.17
5	0.86	0.83	0.89
10	1.75	1.84	1.86

(b) DB 2 (Q1)

SF1 \ SF2	1	5	10
1	7.78	41.4	83.3
5	33.8	175	363
10	66.1	342	663

(c) DB 3 (Q1)

SF1 \ SF2	1	5	10
1	10.8	53.1	104
5	45.5	228	452
10	86.2	431	852

(d) Natix: canonical (Q1)

SF1 \ SF2	1	5	10
1	4.0	4.05	4.0
5	4.01	4.03	3.96
10	4.01	4.02	3.97

(e) Natix: semijoin (Q1)

SF1 \ SF2	1	5	10
1	0.21	0.2	0.2
5	0.21	0.2	0.2
10	0.21	0.2	0.23

(f) Natix: unnested (Q1)

System/Technique \ TPC-H Scaling Factor	0.01	0.05	0.5	1	5	10
DB 1	84.0	3715	n/a	n/a	n/a	n/a
DB 2	0.08	1.83	24.7	43.9	290	616
DB 3	62.8	1742	n/a	n/a	n/a	n/a
Natix						
• canonical	79.7	3631	n/a	n/a	n/a	n/a
• semijoin	17.7	470	n/a	n/a	n/a	n/a
• unnested	0.19	0.48	3.67	15.6	79.3	189

(g) Query 4d

Figure 15: Evaluation for Q1 and 4d

SF1 \ SF2	1	5	10
1	0.06	0.17	0.24
5	0.08	0.35	0.56
10	0.11	0.38	0.70

(a) DB 1 (Q2)

SF1 \ SF2	1	5	10
1	0.13	0.27	0.38
5	0.11	0.61	0.95
10	0.13	0.63	1.23

(b) DB 2 (Q2)

SF1 \ SF2	1	5	10
1	8.78	35.9	87.1
5	46.9	220	400
10	94.7	469	885

(c) DB 3 (Q2)

SF1 \ SF2	1	5	10
1	11.9	48.3	72.9
5	66.3	278	661
10	163	633	1348

(d) Natix: canonical (Q2)

SF1 \ SF2	1	5	10
1	0.22	0.39	0.54
5	0.26	0.69	1.02
10	0.31	0.75	1.27

(e) Natix: unnested (Q2)

Figure 16: Evaluation for Q2

```
SELECT *
FROM R
WHERE R.A1 IN (SELECT S.A4
        FROM S
        WHERE R.A2 = S.A3
        OR S.A1 IN (SELECT T.A4
                FROM T
                WHERE S.A2 = T.A3))
```

Q11

Our unnested approach dominates all other approaches. The execution times of DB 1 and DB 3 in comparison to the execution times of our naïve evaluation plan are a sign of a nested-loop like evaluation of these commercial systems.

5.4 Scalar Subqueries

5.4.1 Disjunctive Linking

We now present our performance evaluation for simple queries with disjunctive linking and scalar subqueries. We selected Query Q6 and, based on the TPC-H schema, the introductionary Query 2d.

SF1 \ SF2	1	5	10
1	23.3	119	305
5	114	598	1524
10	231	1199	3008

(a) DB 1 (Q10)

SF1 \ SF2	1	5	10
1	0.34	0.14	0.16
5	0.13	0.19	0.27
10	0.16	0.22	0.34

(b) DB 2 (Q10)

SF1 \ SF2	1	5	10
1	17.2	91.9	179
5	84.3	496	920
10	166	944	1873

(c) DB 3 (Q10)

SF1 \ SF2	1	5	10
1	26.8	133	273
5	127	695	1323
10	266	1291	3003

(d) Natix: canonical (Q10)

SF1 \ SF2	1	5	10
1	0.23	0.68	2.04
5	0.27	1.23	3.21
10	0.41	1.97	4.66

(e) Natix: unnested (Q10)

Figure 17: Evaluation for Q10

SF1 \ SF2	1	5	10
1	8.71	191	902
5	8.36	497	1515
10	8.35	818	1995

(a) DB 1 (Q11)

SF1 \ SF2	1	5	10
1	0.19	0.75	1.64
5	0.31	0.91	1.73
10	0.53	1.14	1.94

(b) DB 2 (Q11)

SF1 \ SF2	1	5	10
1	25.6	418	1376
5	85.6	742	1974
10	149	1060	2925

(c) DB 3 (Q11)

SF1 \ SF2	1	5	10
1	24.2	348	1347
5	79.6	6574	1891
10	140	1057	2519

(d) Natix: canonical (Q11)

SF1 \ SF2	1	5	10
1	0.17	0.37	0.74
5	0.24	0.43	0.81
10	0.36	0.52	0.89

(e) Natix: unnested (Q11)

Figure 18: Evaluation for Q11

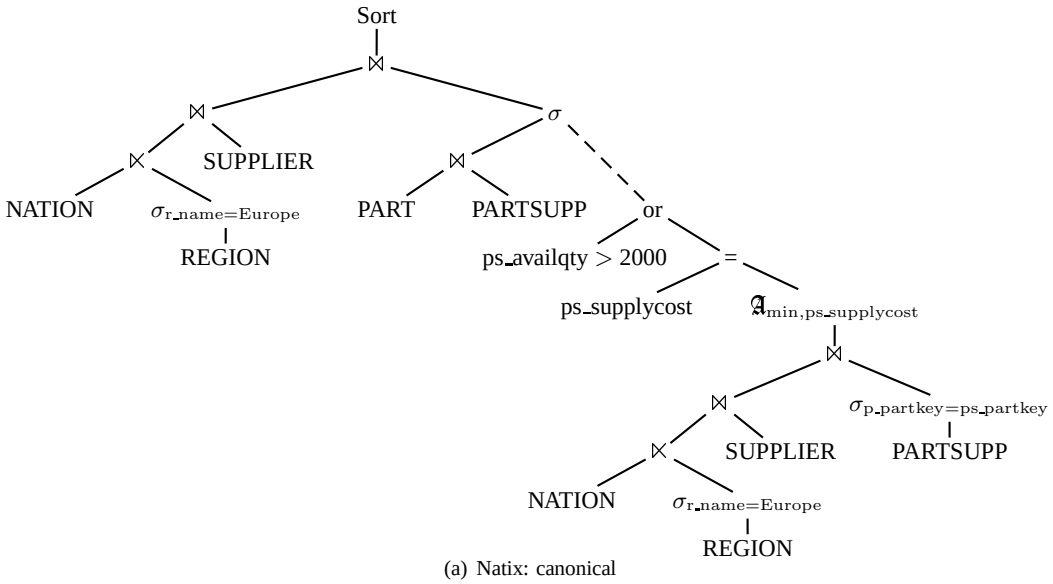
For both queries, we executed two query execution plans in Natix. The first plan implements a canonical translation, the second results from the application of Eqv. 13. Figures 9(a) and 9(c) illustrate these strategies for Query Q6. The plans for 2d are given in Fig. 19.

Fig. 20 presents the results of our performance evaluation. We observe that our unnested approach excels all other approaches — for the RST as well as the TPC-H data set. In comparison with our canonical approach, the performance numbers of the commercial systems for the RST data set allow to deduce that these systems execute the nested query in a nested-loop like fashion. Only the commercial system DB 2 almost keeps up with our unnested approach. However, for the TPC-H data set our unnested approach even outperforms this system by one order of magnitude. The remaining commercial systems are surpassed by three to four orders of magnitude for the cases that finished within six hours.

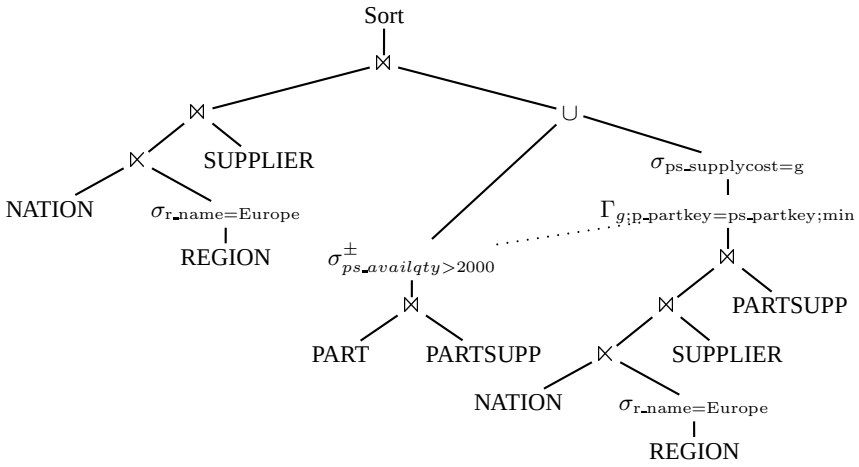
5.4.2 Disjunctive Correlation

Besides the evaluation of JA queries with disjunctive linking, we performed an evaluation for queries with disjunctive correlation. Therefore, we executed Query Q7 using the commercial systems on our synthetic data set and generated two alternative query execution plans for Natix. The first alternative is based on a canonical translation (see Fig. 10(a)). The second was derived by applying a strategy based on Eqv. 17 (see Fig. 10(b)).

Figure 21 presents our performance measurements for these plans. The assessments indicate that all commercial systems evaluate this query similarly to our canonical plan. For the moderate size of 2.1MB of the largest synthetic data set — scaling factor 10 for both the inner and outer query block —, our unnested approach outperforms the others by three to four orders of magnitude. Moreover, evaluation times up to half an hour for 2.1MB data seem unacceptable to us.



(a) Natix: canonical



(b) Natix: unnested

Figure 19: Query plan sketches for Query 2d

5.4.3 Tree Scalar Subqueries

To evaluate tree queries, we executed the following query on the synthetic dataset. Fig. 22 presents the results.

```
SELECT *
FROM R
WHERE A1 = (SELECT COUNT(*)
                FROM S
                WHERE A2 = B3)                                    Q12
       OR
       A2 = (SELECT COUNT(*)
                FROM T
                WHERE A4 = C4)
```

SF1 \ SF2	1	5	10
1	10.6	55.7	111
5	49.4	259	520
10	98.3	515	1029

(a) DB 1 (Q6)

SF1 \ SF2	1	5	10
1	0.19	0.33	0.52
5	0.92	1.17	1.30
10	1.95	2.13	2.52

(b) DB 2 (Q6)

SF1 \ SF2	1	5	10
1	5.06	25.1	50.1
5	25.7	144	267
10	49.8	259	558

(c) DB 3 (Q6)

SF1 \ SF2	1	5	10
1	10.9	54.9	109
5	46.8	235	474
10	88.5	450	899

(d) Natix: canonical (Q6)

SF1 \ SF2	1	5	10
1	0.2	0.24	0.3
5	0.78	0.87	0.98
10	1.6	1.65	1.74

(e) Natix: unnested (Q6)

System/Technique \ Factor	0.01	0.05	0.5	1	5	10
DB 1	0.14	0.36	52.5	123	n/a	n/a
DB 2	0.10	2.00	29.0	67.0	328	766
DB 3	0.27	0.57	48.7	234	n/a	n/a
Natix						
• canonical	79.7	3631	n/a	n/a	n/a	n/a
• unnested	0.14	0.19	0.82	1.49	23.1	49.5

(f) Query 2d

Figure 20: Evaluation for Q6 and 2d

SF1 \ SF2	1	5	10
1	16.7	90.3	184
5	82.7	445	905
10	165	892	1803

(a) DB 1 (Q7)

SF1 \ SF2	1	5	10
1	8.55	46.3	95.5
5	42.9	235	479
10	85.7	466	971

(b) DB 2 (Q7)

SF1 \ SF2	1	5	10
1	11.6	59.7	120
5	71.4	378	737
10	143	753	1519

(c) DB 3 (Q7)

SF1 \ SF2	1	5	10
1	16.0	98.6	208
5	79.8	470	897
10	166	1237	1768

(d) Natix: canonical (Q7)

SF1 \ SF2	1	5	10
1	0.12	0.14	0.15
5	0.22	0.24	0.26
10	0.38	0.41	0.42

(e) Natix: unnested (Q7)

Figure 21: Evaluation for Q7

SF1 \ SF2	1	5	10
1	31.8	190	539
5	225	1039	2435
10	395	1976	4874

(a) DB 1 (Q12)

SF1 \ SF2	1	5	10
1	0.14	0.47	0.92
5	0.36	0.72	1.17
10	0.66	1.02	1.58

(b) DB 2 (Q12)

SF1 \ SF2	1	5	10
1	25.6	136	274
5	147	717	1391
10	289	1446	2874

(c) DB 3 (Q12)

SF1 \ SF2	1	5	10
1	25.8	136	262
5	136	689	1342
10	257	1384	2693

(d) Natix: canonical (Q12)

SF1 \ SF2	1	5	10
1	0.19	0.29	0.45
5	0.4	0.58	0.79
10	0.81	0.96	1.23

(e) Natix: unnested (Q12)

Figure 22: Evaluation for Q12

5.4.4 Linear Scalar Subqueries

Consider the following linear query and the according results of its evaluation. In the evaluations SF1 denotes the scaling factor for the outer relation. SF2 denotes the scaling factor of both inner

relations, i.e. S and T.

```
SELECT *
FROM R
WHERE R.A1 = (SELECT count(S.B4)
        FROM S
        WHERE R.A2 = S.B3
        OR S.B1 = (SELECT count(T.C4)
                FROM T
                WHERE S.B2 = T.C3))
        OR R.A4 < 1500
```

Q13

SF1 \ SF2	1	5	10
1	n/a	n/a	n/a
5	n/a	n/a	n/a
10	n/a	n/a	n/a

(a) DB 1 (Q13)

SF1 \ SF2	1	5	10
1	152	784	1602
5	744	3884	9237
10	1517	7923	n/a

(b) DB 2 (Q13)

SF1 \ SF2	1	5	10
1	n/a	n/a	n/a
5	n/a	n/a	n/a
10	n/a	n/a	n/a

(c) DB 3 (Q13)

SF1 \ SF2	1	5	10
1	n/a	n/a	n/a
5	n/a	n/a	n/a
10	n/a	n/a	n/a

(d) Natix: canonical (Q13)

SF1 \ SF2	1	5	10
1	n/a	n/a	n/a
5	n/a	n/a	n/a
10	n/a	n/a	n/a

(e) Natix: unnested (Q13)

Figure 23: Evaluation for Q13

6 Related Work

Apart from introducing a classification for nested queries, Kim [19] was the first to rephrase nested SQL queries to contain joins or grouping. However, the validity of these rewrites depends on important restrictions. They mainly concern empty results for the inner query block, NULL values, and duplicate handling. Subsequent research found more unnesting techniques for SQL [8, 11, 12, 18, 25], OQL [5, 9, 27, 28], and XQuery [20].

Strategies for the evaluation of nested queries are discussed in [14, 15].

Several optimization techniques for queries containing disjunctive predicates have been proposed [2, 16, 17].

However, to the best of our knowledge, nobody has investigated unnesting nested queries with disjunctive linking or disjunctive correlation so far. Our approach for unnesting nested queries in these cases utilize the bypass technique introduced in [17].

Because bypass operators have two output streams, which are unioned later, the resulting expression forms a directed acyclic graph (DAG). Strategies for implementing bypass operators and query evaluation engines that support DAG-structured query plans can be found in [17, 23, 24]. Especially for a thorough discussion of the generation and evaluation of DAG-structured query plans, we refer to [23].

7 Conclusion and Outlook

7.1 Conclusion

We believe that nested queries containing disjunctive predicates have not yet attracted the attention they deserve. In our experimental study, we have shown that evaluating nested queries in a nested-loop-like fashion leads to unacceptable performance. With our novel unnesting strategy, we are able to remedy this situation and to substantially improve query execution times. Although most runtime systems and optimizers do not incorporate bypass plans, it is possible to transfer bypass plans into

plans without bypass operators. This can, for example, be done by tagging every tuple whether it belongs to the positive or negative stream.

7.2 Outlook

Encouraged by these results, we plan to enhance our unnesting approach to handle more sophisticated cases. These include, for example, (1) unnesting queries whose linking *and* correlation predicate occurs in a disjunction, (2) optimizing nested disjunctive queries in the `from` clause, (3) handling all linking operators, i.e. θ `ALL` and θ `SOME/ANY` for $\theta \in \{<, \leq, >, \geq\}$, and finally (4) unnesting queries featuring correlation predicates that refer to attributes defined in a non-adjacent query block (indirect correlation).

Since our unnesting technique creates DAG-structured algebraic expressions, we rely on effective optimization techniques for generating and executing DAG-structured query plans. The algebraic expressions produced by our techniques are quite demanding and, hence, might trigger further research in this direction.

Acknowledgments We would like to thank Simone Seeger for her comments on the manuscript and Uwe Steinel for the administration of the commercial DBMSs.

References

[1] M. Brantner, N. May, and G. Moerkotte. Unnesting SQL queries in the presence of disjunction. Technical report, University of Mannheim, March 2006. http://db.informatik.uni-mannheim.de/publications/TR-06-001.pdf.

[2] F. Bry. Towards an efficient evaluation of general queries: quantifier and disjunction processing revisited. In *Proceedings of ACM SIGMOD Conference, Oregon, USA*, pages 193–204, 1989.

[3] B. Cao and A. Badia. A nested relational approach to processing SQL subqueries. In *Proceedings of ACM SIGMOD Conference, Baltimore, USA*, pages 191–202, 2005.

[4] J. Claussen, A. Kemper, G. Moerkotte, K. Peithner, and M. Steinbrunn. Optimization and evaluation of disjunctive queries. *IEEE Transactions on Knowledge and Data Engineering*, 12(2):238–260, 2000.

[5] S. Cluet and G. Moerkotte. Classification and optimization of nested queries in object bases. Technical Report 95-6, RWTH Aachen, 1995.

[6] S. Cluet and G. Moerkotte. Efficient evaluation of aggregates on bulk types. In *Proceedings of 5th DBPL, Gubbio, Umbria, Italy*, 1995.

[7] C. J. Date. The outer join. In *Proceedings of ICOD, Cambridge, England*, pages 76–106, 1983.

[8] U. Dayal. Of nests and trees: A unified approach to processing queries that contain nested subqueries, aggregates, and quantifiers. In *Proceedings of the VLDB Conference, Brighton, England*, pages 197–208, 1987.

[9] L. Fegaras. Query unnesting in object-oriented databases. In *Proceedings ACM SIGMOD Conference, Seattle, USA*, pages 49–60, 1998.

[10] T. Fiebig, S. Helmer, C-C. Kanne, G. Moerkotte, J. Neumann, R. Schiele, and T. Westmann. Anatomy of a native XML base management system. *VLDB J.*, 11(4):292–314, 2002.

[11] C. Galindo-Legaria and M. Joshi. Orthogonal optimization of subqueries and aggregation. In *Proceedings of ACM SIGMOD Conference, Santa Barbara, USA*, pages 571–581, 2001.

[12] R. A. Ganski and H. K. T. Wong. Optimization of nested SQL queries revisited. In *Proceedings of the ACM SIGMOD, San Francisco, USA*, pages 23–33, 1987.

[13] H. Garcia-Molina, J. D. Ullman, and J. Widom. *Database Systems: The Complete Book.* Prentice Hall, 2002. 0-13-098043-9.

[14] G. Graefe. Executing nested queries. In *BTW 2003, Datenbanksysteme für Business, Technologie und Web, Leipzig*, pages 58–77, 2003.

[15] R. Guravannavar, H. S. Ramanujam, and S. Sudarshan. Optimizing nested queries with parameter sort orders. In *Proceedings of the VLDB Conference, Trondheim, Norway*, pages 481–492, 2005.

[16] M. Jarke and J. Koch. Query optimization in database systems. *ACM Computing Surveys*, 16(2):111–152, June 1984.

[17] A. Kemper, G. Moerkotte, K. Peithner, and M. Steinbrunn. Optimizing disjunctive queries with expensive predicates. *Proceedings of ACM SIGMOD Conference*, 23(2):336–347, June 1994.

[18] W. Kiessling. SQL-like and Quel-like correlation queries with aggregates revisited. ERL/UCB Memo 84/75, University of Berkeley, 1984.

[19] W. Kim. On optimizing an SQL-like nested query. *ACM Transactions on Database Systems*, 7(3):443–469, September 1982.

[20] N. May, S. Helmer, and G. Moerkotte. Nested queries and quantifiers in an ordered context. In *Proceedings of the ICDE Conference, Boston, USA*, pages 239–250, 2004.

[21] N. May and G. Moerkotte. Main memory implementations for binary grouping. In *Proceedings of the XML Database Symposium, Trondheim, Norway*, pages 162–176, London, UK, 2005. Springer-Verlag.

[22] M. Muralikrishna. Improved unnesting algorithms for join aggregate sql queries. In *Proceedings of VLDB Conference, Vancouver, Canada*, pages 91–102, 1992.

[23] T. Neumann. *Efficient Generation and Execution of DAG-Structured Query Graphs*. PhD thesis, University of Mannheim, 2005.

[24] P. Roy. Optimization of DAG-structured query evaluation plans. Master's thesis, Indian Institute of Technology, Bombay, 1998.

[25] P. Seshadri, H. Pirahesh, and T. Y. Cliff Leung. Complex query decorrelation. In *Proceedings of the ICDE Conference, New Orleans, USA*, pages 450–458, 1996.

[26] J. R. Slagle. An efficient algorithm for finding certain minimum-cost procedures for making binary decisions. *J. ACM*, 11(3):253–264, 1964.

[27] H. J. Steenhagen. *Optimization of Object Query Languages.* PhD thesis, Department of Computer Science, University of Twente., 1995.

[28] H. J. Steenhagen, P. M. G. Apers, H. M. Blanken, and R. A. de By. From nested-loop to join queries in oodb. In *Proceedings of the VLDB Conference, Santiago de Chile, Chile*, pages 618–629, 1994.

[29] Transaction Processing Performance Council TPC. TPC benchmark H (decision support). Standard Specification Version 2.3.0, Transaction Processing Performance Council, 2006. http://www.tpc.org/.

A Algebra for Sets

All evaluation techniques in this paper are discussed based on algebraic equivalences. To make the paper self-contained, we give some basic notations and the definitions of the traditional algebra operators used throughout the paper. In a second step we define the extended bypass operators which will be used for decorrelating disjunctive queries.

We support the set of well-known algebraic operators, e.g. selection, joins, or grouping. Their formal definitions, which is already given in the literature [13], follows after introducing notations used in their definition.

Each algebra expression e produces a set of tuples with identical attributes denoted by $\mathcal{A}(e)$. The set of free variables of an expression e is denoted by $\mathcal{F}(e)$. Single tuples are constructed by using the standard $[\cdot]$ brackets. The projection of a tuple t on a set of attributes A is denoted by $t|_A$. We also define $t|_{\overline{A}} := t|_{A(t)\setminus A}$. The concatenation of tuples is denoted by $\circ$. We write $a : b$ to assign the name a to the value b (e.g. when creating a new attribute a). We write R to get all tuples contained in relation R.

As algebra expressions produce sets of tuples, we can use the regular set operations $\cup$, $\cap$ and $\setminus$ to construct new expressions. Note that that the $\cup$ operator has to perform duplicate elimination, as it produces a set. If it is clear that no duplicates can occur, we write $\dot\cup$ for a union without duplicate elimination.

$$\begin{aligned} e_1 \cup e_2 &:= \{x | x \in e_1 \vee x \in e_2\} \\ e_1 \cap e_2 &:= \{x | x \in e_1 \wedge x \in e_2\} \\ e_1 \setminus e_2 &:= \{x | x \in e_1 \wedge x \notin e_2\} \end{aligned}$$

A projection on all tuples of an expression e is defined as

$$\Pi_A(e) \quad := \quad \{x|_A | x \in e\}$$

A selection σ of the set of tuples resulting from expression e using a predicate p is defined as follows:

$$\sigma_p(e) \quad := \quad \{x | x \in e \wedge p(x)\}$$

Several join operators are supported:

$$\begin{aligned}
e_1 \bowtie_p e_2 &:= \{x \circ y | x \in e_1 \wedge y \in e_2 \wedge p(x,y)\} \\
e_1 \ltimes_p e_2 &:= \{x | x \in e_1 \wedge \exists y \in e_2 \wedge p(x,y)\} \\
e_1 \triangleright_p e_2 &:= \{x | x \in e_1 \wedge \nexists y \in e_2 \wedge p(x,y)\} \\
e_1 \bowtie_p^{g:f(\emptyset)} e_2 &= e_1 \bowtie_p e_2 \cup \{x \circ z | x \in e_1 \wedge \\
&\quad \nexists y \in e_2 : p(x,y) \wedge \mathcal{A}(z) = \mathcal{A}(e_2) \wedge g \in \mathcal{A}(e_2) \wedge \\
&\quad \forall a \in (\mathcal{A}(e_2) \setminus g) : (z.a : \text{NULL} \wedge z.g : f(\emptyset))\}
\end{aligned}$$

The d-join is a join between two sets, where the evaluation of the second set e_2 depends on a tuple x of the first set ($e_2(x)$). Its definition is as follows:

$$e_1 < e_2 > \quad := \quad \{x \circ y | x \in e_1 \wedge y \in e_2(x)\}$$

Sometimes we will also use $e_1 \underrightarrow{\bowtie} e_2$ instead of $e_1 < e_2 >$.

For explicit grouping operations and decorrelation purposes we define two grouping operators. The unary grouping is defined as

$$\Gamma_{g;\theta A;f}(e) \quad := \quad \{x_{|A} \circ [g : G] | x \in e \wedge G = f(\{y | y \in e \wedge y.A \theta x.A\})\}$$

and the binary grouping as

$$e_1 \Gamma_{g;A_1 \theta A_2;f}(e_2) \quad := \quad \{x_{|A_1} \circ [g : G] | x \in e_1 \wedge G = f(\{y | y \in e_2 \wedge x.A_1 \theta y.A_2\})\}$$

Finally, for the translation of nested queries we need the following map operator:

$$\chi_{a:e_2}(e_1) \quad := \quad \{x \circ [a : e_2(x)] | x \in e_1\}$$

To translate the linking predicates `EXISTS` and `IN`, we employ the internal function $\exists_p(e)$; for the negated form we use $\nexists_p(e)$. Both return true if there exists (does not exist) at least one element in their argument e that satisfies the predicate p. We write $\exists$ and $\nexists$ if the predicate is true, i.e. there is no linking predicate (e.g. `EXISTS` and `NOT EXISTS`). Further, for each of the common SQL aggregation functions `SUM`, `COUNT`, `MAX`, `MIN`, and `AVG` we have a corresponding internal function.

A.1 Bypass Operators

In order to ease the formal description for the bypass operators, their definitions come in two halves. The definition of each operator is divided into the part yielding the positive stream — denoted with superscript "+" — and the half yielding the negative stream — denoted by superscript "-":

We define the bypass selection ($\sigma^{\pm}$) as follows:

$$\begin{aligned}\sigma_p^+(e) &:= \{x|x \in e \wedge p(x)\}\\ \sigma_p^-(e) &:= e - \sigma_p^+(e) \stackrel{*}{=} \{x|x \in e \wedge \neg p(x)\}\end{aligned}$$

We define the bypass join ($\bowtie^{\pm}$) as follows:

$$\begin{aligned}e_1 \bowtie_p^+ e_2 &:= \{x \circ y|x \in e_1 \wedge y \in e_2 \wedge p(x,y)\}\\ e_1 \bowtie_p^- e_2 &:= (e_1 \times e_2) - (e_1 \bowtie_p^+ e_2) \stackrel{*}{=} \{x \circ y|x \in e_1 \wedge y \in e_2 \wedge \neg p(x,y)\}\end{aligned}$$

We define the bypass semijoin ($\ltimes^{\pm}$) as follows:

$$\begin{aligned}e_1 \ltimes_p^+ e_2 &:= \{x|x \in e_1 \wedge \exists y \in e_2 \wedge p(x,y)\}\\ e_1 \ltimes_p^- e_2 &:= e_1 - (e_1 \ltimes_p^+ e_2) \stackrel{*}{=} \{x|x \in e_1 \wedge \neg\exists y \in e_2 \wedge p(x,y)\}\end{aligned}$$

The bypass anti-join is defined as:

$$\begin{aligned}e_1 \triangleright_p^+ e_2 &:= \{x|x \in e_1 \wedge \neg\exists y \in e_2 \wedge p(x,y)\}\\ e_1 \triangleright_p^- e_2 &:= e_1 - (e_1 \triangleright_p^+ e_2) \stackrel{*}{=} \{x|x \in e_1 \wedge \exists y \in e_2 \wedge p(x,y)\}\end{aligned}$$

In the definitions of the negative streams the equalities marked with * are only valid for two-valued logic (cf. [4] for details).

B Proofs

For the following proofs let *lhs* denote the left-hand side and *rhs* the right-hand side of an equivalence.

B.1 Proof of Equivalence 2 and 6

We proof only Equivalence 6 repeated below. The correctness of Equivalence 2 follows immediately when we replace the correlation predicate $\mathrm{A}_2 = \mathrm{B}_2$ by the constant TRUE.

$$\begin{aligned}\sigma_{\exists_{\mathrm{A}_1=\mathrm{B}_1}(\sigma_{\mathrm{A}_2=\mathrm{B}_2}(\mathrm{S}))\vee p}(\mathrm{R}) &\equiv e_1 \dot\cup e_2\\ e_1 &:= \sigma_p^+(\mathrm{R})\\ e_2 &:= (\sigma_p^-(\mathrm{R})) \ltimes_{\mathrm{A}_1=\mathrm{B}_1\wedge\mathrm{A}_2=\mathrm{B}_2} \mathrm{S}\end{aligned}$$

if $\mathcal{F}(p) \subseteq \mathcal{A}(R)$, $\mathcal{A}(R) \cap \mathcal{A}(S) = \emptyset$, $A_1 \in \mathcal{A}(R)$, $B_1 \in \mathcal{A}(S)$, (for Eqv. 6 in addition $A_2 \in \mathcal{A}(R), B_2 \in \mathcal{A}(S)$)

$$\begin{aligned}\mathrm{rhs} &= \sigma_p^+(R) \dot\cup ((\sigma_p^-(R)) \ltimes_{A_1=B_1\wedge A_2=B_2} (S))\\ &= \{r|r \in R \wedge p\} \dot\cup \{r|r \in \{x|x \in R \wedge \neg p\} \wedge \exists s \in S \wedge r.A_1 = s.B_1 \wedge r.A_2 = s.B_2\}\\ &= \{r|r \in R \wedge p\} \dot\cup \{r|r \in R \wedge \neg p \wedge \exists s \in S \wedge r.A_1 = s.B_1 \wedge r.A_2 = s.B_2\}\\ &= \{r|r \in R \wedge (p \vee (\neg p \wedge \exists s \in S \wedge r.A_1 = s.B_1 \wedge r.A_2 = s.B_2))\}\\ &= \{r|r \in R \wedge (p \vee (\neg p \wedge \exists s \in \{t|t \in S \wedge r.A_2 = t.B_2\} \wedge r.A_1 = s.B_1))\}\\ &= \sigma_{p\vee\exists_{A_1=B_1}(\sigma_{A_2=B_2}(S))}(R)\\ &= \mathrm{lhs}\end{aligned}$$

Note that predicates p and $\neg p$ partition the tuples of R into two disjoint sets. We explicitly use this property in the bypass selection to avoid the creation of duplicates. Together with the short-circuit evaluation of the disjunction duplicate semantics are preserved. For this reason and because all remaining operators preserve duplicates this equivalence holds for bags.

B.2 Proof of Equivalence 3 and 7

We proof only Equivalence 7 repeated below. The correctness of Equivalence 3 follows immediately when we replace the correlation predicate $\mathrm{A}_2 = \mathrm{B}_2$ by the constant TRUE.

$$\begin{aligned}\sigma_{\exists_{\mathrm{A}_1=\mathrm{B}_1}(\sigma_{\mathrm{A}_2=\mathrm{B}_2}(\mathrm{S}))\vee p}(\mathrm{R}) &\equiv e_1 \dot\cup e_2\\ e_1 &:= \mathrm{R} \ltimes^{+}_{\mathrm{A}_1=\mathrm{B}_1\wedge\mathrm{A}_2=\mathrm{B}_2} \mathrm{S}\\ e_2 &:= \sigma_p(\mathrm{R} \ltimes^{-}_{\mathrm{A}_1=\mathrm{B}_1\wedge\mathrm{A}_2=\mathrm{B}_2} \mathrm{S})\end{aligned}$$

if $\mathcal{F}(p) \subseteq \mathcal{A}(R)$, $\mathcal{A}(R) \cap \mathcal{A}(S) = \emptyset$, $A_1 \in \mathcal{A}(R)$, $B_1 \in \mathcal{A}(S)$, (for Eqv. 7 in addition $A_2 \in \mathcal{A}(R), B_2 \in \mathcal{A}(S)$)

$$\begin{aligned}\mathrm{rhs} &= (R \ltimes^{+}_{A_1=B_1\wedge A_2=B_2} S) \dot\cup (\sigma_p(R \ltimes^{-}_{A_1=B_1\wedge A_2=B_2} S))\\ &= \{r|r \in R \wedge \exists s \in S \wedge r.A_1 = s.B_1 \wedge r.A_2 = s.B_2\} \dot\cup\\ &\quad \{r|r \in R \wedge p \wedge \neg\exists s \in S \wedge r.A_1 = s.B_1 \wedge r.A_2 = s.B_2\}\\ &= \{r|r \in R \wedge ((\exists s \in S \wedge r.A_1 = s.B_1 \wedge r.A_2 = s.B_2) \vee\\ &\qquad (p \wedge \neg\exists s \in S \wedge r.A_1 = s.B_1 \wedge r.A_2 = s.B_2))\}\\ &= \{r|r \in R \wedge (p \vee (\exists s \in S \wedge r.A_1 = s.B_1 \wedge r.A_2 = s.B_2))\}\\ &= \{r|r \in R \wedge (p \vee (\exists s \in \{t|t \in S \wedge r.A_2 = t.B_2\} \wedge r.A_1 = s.B_1))\}\\ &= \sigma_{p\vee\exists_{A_1=B_1}(\sigma_{A_2=B_2}(S))}(R)\\ &= \mathrm{lhs}\end{aligned}$$

Again, we rely on a disjoint partitioning of R and short-circuit evaluation of the disjunction in the last step. The correctness for multisets follows for the same reasons as for equivalences 2 and 6.

B.3 Proof of Equivalence 4 and 8

We proof only Equivalence 8 repeated below. The correctness of Equivalence 4 follows immediately when we replace the correlation predicate $\mathrm{A}_2 = \mathrm{B}_2$ by the constant TRUE.

$$\begin{aligned}\sigma_{\nexists_{\mathrm{A}_1=\mathrm{B}_1}(\sigma_{\mathrm{A}_2=\mathrm{B}_2}(\mathrm{S}))\vee p}(\mathrm{R}) &\equiv e_1 \dot\cup e_2\\ e_1 &:= \sigma^{+}_p(\mathrm{R})\\ e_2 &:= (\sigma^{-}_p(\mathrm{R})) \triangleright_{\mathrm{A}_1=\mathrm{B}_1\wedge\mathrm{A}_2=\mathrm{B}_2} \mathrm{S}\end{aligned}$$

if $\mathcal{F}(p) \subseteq \mathcal{A}(R)$, $\mathcal{A}(R) \cap \mathcal{A}(S) = \emptyset$, $A_1 \in \mathcal{A}(R)$, $B_1 \in \mathcal{A}(S)$, (for Eqv. 8 in addition $A_2 \in \mathcal{A}(R), B_2 \in \mathcal{A}(S)$)

$$\begin{aligned}\mathrm{rhs} &= \sigma^{+}_p(R) \dot\cup ((\sigma^{-}_p(R)) \triangleright_{A_1=B_1\wedge A_2=B_2} (S))\\ &= \{r|r \in R \wedge p\} \dot\cup \{r|r \in \{x|x \in R \wedge \neg p\} \wedge \neg\exists s \in S \wedge r.A_1 = s.B_1 \wedge r.A_2 = s.B_2\}\\ &= \{r|r \in R \wedge p\} \dot\cup \{r|r \in R \wedge \neg p \wedge \neg\exists s \in S \wedge r.A_1 = s.B_1 \wedge r.A_2 = s.B_2\}\\ &= \{r|r \in R \wedge (p \vee (\neg p \wedge \neg\exists s \in S \wedge r.A_1 = s.B_1 \wedge r.A_2 = s.B_2))\}\\ &= \{r|r \in R \wedge (p \vee (\neg p \wedge \neg\exists s \in \{t|t \in S \wedge r.A_2 = t.B_2\} \wedge r.A_1 = s.B_1))\}\\ &= \sigma_{p\vee\nexists_{A_1=B_1}(\sigma_{A_2=B_2}(S))}(R)\\ &= \mathrm{lhs}\end{aligned}$$

Again, we rely on a disjoint partitioning of R and short-circuit evaluation of the disjunction in the last step. The correctness for multisets follows for the same reasons as for Equivalences 2 and 6.

B.4 Proof of Equivalence 5 and 9

We proof only Equivalence 9 repeated below. The correctness of Equivalence 5 follows immediately when we replace the correlation predicate $\mathrm{A}_2 = \mathrm{B}_2$ by the constant TRUE.

$$\begin{aligned}
\sigma_{\nexists_{\mathrm{A}_1=\mathrm{B}_1}(\sigma_{\mathrm{A}_2=\mathrm{B}_2}(\mathrm{S}))\vee p}(\mathrm{R}) &\equiv e_1 \dot{\cup} e_2 \\
e_1 &:= \mathrm{R} \triangleright^{+}_{\mathrm{A}_1=\mathrm{B}_1 \wedge \mathrm{A}_2=\mathrm{B}_2} \mathrm{S} \\
e_2 &:= \sigma_p(\mathrm{R} \triangleright^{-}_{\mathrm{A}_1=\mathrm{B}_1 \wedge \mathrm{A}_2=\mathrm{B}_2} \mathrm{S})
\end{aligned}$$

if $\mathcal{F}(p) \subseteq \mathcal{A}(R)$, $\mathcal{A}(R) \cap \mathcal{A}(S) = \emptyset$, $A_1 \in \mathcal{A}(R)$, $B_1 \in \mathcal{A}(S)$, (for Eqv. 9 in addition $A_2 \in \mathcal{A}(R), B_2 \in \mathcal{A}(S)$)

$$\begin{aligned}
\mathrm{rhs} &= (R \triangleright^{+}_{A_1=B_1 \wedge A_2=B_2} S) \dot{\cup} (\sigma_p(R \triangleright^{-}_{A_1=B_1 \wedge A_2=B_2} S)) \\
&= \{r | r \in R \wedge \neg\exists s \in S \wedge r.A_1 = s.B_1 \wedge r.A_2 = s.B_2\} \dot{\cup} \\
&\quad \{r | r \in R \wedge p \wedge \exists s \in S \wedge r.A_1 = s.B_1 \wedge r.A_2 = s.B_2\} \\
&= \{r | r \in R \wedge ((\neg\exists s \in S \wedge r.A_1 = s.B_1 \wedge r.A_2 = s.B_2) \vee \\
&\qquad (p \wedge \exists s \in S \wedge r.A_1 = s.B_1 \wedge r.A_2 = s.B_2))\} \\
&= \{r | r \in R \wedge (p \vee (\neg\exists s \in \{t | t \in S \wedge r.A_2 = t.B_2\} \wedge r.A_1 = s.B_1))\} \\
&= \sigma_{p \vee \nexists_{A_1=B_1}(\sigma_{A_2=B_2}(S))}(R) \\
&= \mathrm{lhs}
\end{aligned}$$

Again, we rely on a disjoint partitioning of R and short-circuit evaluation of the disjunction in the last step. The correctness for multisets follows for the same reasons as for Equivalences 2 and 6.

B.5 Proof of Equivalence 10

$$\begin{aligned}
\sigma_{\exists_{\mathrm{A}_1=\mathrm{B}_1}(\sigma_{\mathrm{A}_2=\mathrm{B}_2 \vee p}(\mathrm{S}))}(\mathrm{R}) &\equiv e_1 \dot{\cup} e_2 \\
e_1 &:= \mathrm{R} \ltimes^{+}_{\mathrm{A}_1=\mathrm{B}_1} e_3 \\
e_2 &:= (\mathrm{R} \ltimes^{-}_{\mathrm{A}_1=\mathrm{B}_1} e_3) \ltimes_{\mathrm{A}_1=\mathrm{B}_1 \wedge \mathrm{A}_2=\mathrm{B}_2} (\sigma^{-}_p(\mathrm{S})) \\
e_3 &:= \sigma^{+}_p(\mathrm{S})
\end{aligned}$$

if $\mathcal{F}(p) \subseteq \mathcal{A}(S)$, $\mathcal{A}(R) \cap \mathcal{A}(S) = \emptyset$, $A_1, A_2 \in \mathcal{A}(R)$, $B_1, B_2 \in \mathcal{A}(S)$

$$
\begin{aligned}
\text{rhs} &= (R \ltimes^{+}_{A_1=B_1} (\sigma^{+}_{p}(S))) \dot\cup \\
&\quad ((R \ltimes^{-}_{A_1=B_1} (\sigma^{+}_{p}(S))) \ltimes_{A_1=B_1 \wedge A_2=B_2} (\sigma^{-}_{p}(S))) \\
&= \{r | r \in R \wedge \exists s \in \{t | t \in S \wedge p\} \wedge r.A_1 = s.B_1\} \dot\cup \\
&\quad \{t | t \in \{r | r \in (R \setminus \{x | x \in R \wedge \exists s \in \{t | t \in S \wedge p\} \wedge x.A_1 = s.B_1\}) \wedge \\
&\qquad \exists s \in \{y | y \in S \wedge \neg p\} \wedge t.A_1 = s.B_1 \wedge t.A_2 = s.B_2\} \\
&= \{r | r \in R \wedge \exists s \in S \wedge r.A_1 = s.B_1 \wedge p\} \dot\cup \\
&\quad \{t | t \in \{r | r \in R \setminus \{x | x \in R \wedge \exists s \in S \wedge x.A_1 = s.B_1 \wedge p\}\} \wedge \\
&\qquad \exists s \in S \wedge \neg p \wedge t.A_1 = s.B_1 \wedge t.A_2 = s.B_2\} \\
&= \{r | r \in R \wedge \exists s \in S \wedge r.A_1 = s.B_1 \wedge p\} \dot\cup \\
&\quad \{r | r \in (R - \{x | x \in R \wedge \exists s \in S \wedge x.A_1 = s.B_1 \wedge p\}) \wedge \\
&\qquad \exists s \in S \wedge \neg p \wedge r.A_1 = s.B_1 \wedge r.A_2 = s.B_2\} \\
&= \{r | ((r \in R \wedge \exists s \in S \wedge r.A_1 = s.B_1 \wedge p) \vee \\
&\qquad (r \in (R - \{x | x \in R \wedge \exists s \in S \wedge x.A_1 = s.B_1 \wedge p\}) \wedge \\
&\qquad \exists s \in S \wedge \neg p \wedge r.A_1 = s.B_1 \wedge r.A_2 = s.B_2))\} \\
&\overset{*}{=} \{r | r \in R \wedge ((\exists s \in S \wedge r.A_1 = s.B_1 \wedge p) \vee (\exists s \in S \wedge \neg p \wedge r.A_1 = s.B_1 \wedge r.A_2 = s.B_2))\} \\
&= \{r | r \in R \wedge \exists s \in S \wedge ((p \wedge r.A_1 = s.B_1) \vee (\neg p \wedge r.A_1 = s.B_1 \wedge r.A_2 = s.B_2))\} \\
&= \{r | r \in R \wedge \exists s \in S \wedge r.A_1 = s.B_1 \wedge (p \vee (\neg p \wedge r.A_2 = s.B_2))\} \\
&= \{r | r \in R \wedge \exists s \in \{t | t \in S \wedge (p \vee (\neg p \wedge r.A_2 = t.B_2))\} \wedge r.A_1 = s.B_1\} \\
&= \sigma_{\exists_{A_1=B_1}(\sigma_{A_2=B_2 \vee p}(S))}(R) \\
&= \text{lhs}
\end{aligned}
$$

In the last step we assume short-circuit evaluation of the disjunction. In the step marked * we rely on the more general definition of the bypass semijoin where the negative stream is defined by set difference.

$$
\begin{aligned}
e_1 \ltimes^{+}_{p} e_2 &:= \{x | x \in e_1 \wedge \exists y \in e_2 \wedge p\} \\
e_1 \ltimes^{-}_{p} e_2 &:= e_1 - (e_1 \ltimes^{+}_{p} e_2)
\end{aligned}
$$

For the correctness of this equivalence for multisets it is important to realize that the bypass semijoin partitions the tuples of R into two disjoint streams. Either stream is correct under multiset semantics. Finally the outermost union merges both streams without changing the number of duplicates.

B.6 Proof of Equivalence 11

$$
\begin{aligned}
\sigma_{\nexists_{\mathrm{A_1=B_1}}(\sigma_{\mathrm{A_2=B_2 \vee p}}(\mathrm{S}))}(\mathrm{R}) &\equiv e_1 \dot\cup e_2 \\
e_1 &:= \mathrm{R} \triangleright^{+}_{\mathrm{A_1=B_1}} e_3 \\
e_2 &:= (\mathrm{R} \triangleright^{-}_{\mathrm{A_1=B_1}} e_3) \triangleright_{\mathrm{A_1=B_1 \wedge A_2=B_2}} (\sigma^{-}_{p}(\mathrm{S})) \\
e_3 &:= \sigma^{+}_{p}(\mathrm{S})
\end{aligned}
$$

if $\mathcal{F}(p) \subseteq \mathcal{A}(S)$, $\mathcal{A}(R) \cap \mathcal{A}(S) = \emptyset$, $A_1, A_2 \in \mathcal{A}(R)$, $B_1, B_2 \in \mathcal{A}(S)$

$$
\begin{aligned}
\text{rhs} &= (R \triangleright^{+}_{A_1=B_1} (\sigma^{+}_{p}(S))) \dot\cup \\
&\quad ((R \triangleright^{-}_{A_1=B_1} (\sigma^{+}_{p}(S))) \triangleright_{A_1=B_1 \wedge A_2=B_2} (\sigma^{-}_{p}(S))) \\
&= \{r | r \in R \wedge \neg\exists s \in \{t | t \in S \wedge p\} \wedge r.A_1 = s.B_1\} \dot\cup \\
&\quad \{t | t \in \{r | r \in (R \setminus \{x | x \in R \wedge \neg\exists s \{t | t \in S \wedge p\} \wedge x.A_1 = s.B_1\}) \wedge \\
&\qquad \neg\exists s \in \{y | y \in S \wedge \neg p\} \wedge t.A_1 = s.B_1 \wedge t.A_2 = s.B_2\} \\
&= \{r | r \in R \wedge \neg\exists s \in S \wedge r.A_1 = s.B_1 \wedge p\} \dot\cup \\
&\quad \{t | t \in \{r | r \in R \setminus \{x | x \in R \wedge \neg\exists s \in S \wedge x.A_1 = s.B_1 \wedge p\}\} \wedge \\
&\qquad \neg\exists s \in S \wedge \neg p \wedge t.A_1 = s.B_1 \wedge t.A_2 = s.B_2\} \\
&= \{r | r \in R \wedge \neg\exists s \in S \wedge r.A_1 = s.B_1 \wedge p\} \dot\cup \\
&\quad \{r | r \in (R - \{x | x \in R \wedge \neg\exists s \in S \wedge x.A_1 = s.B_1 \wedge p\}) \wedge \\
&\qquad \neg\exists s \in S \wedge \neg p \wedge r.A_1 = s.B_1 \wedge r.A_2 = s.B_2\} \\
&= \{r | ((r \in R \wedge \neg\exists s \in S \wedge r.A_1 = s.B_1 \wedge p) \vee \\
&\qquad (r \in (R - \{x | x \in R \wedge \neg\exists s \in S \wedge x.A_1 = s.B_1 \wedge p\}) \wedge \\
&\qquad \neg\exists s \in S \wedge \neg p \wedge r.A_1 = s.B_1 \wedge r.A_2 = s.B_2))\} \\
&\overset{*}{=} \{r | r \in R \wedge ((\neg\exists s \in S \wedge r.A_1 = s.B_1 \wedge p) \vee (\neg\exists s \in S \wedge \neg p \wedge r.A_1 = s.B_1 \wedge r.A_2 = s.B_2))\} \\
&= \{r | r \in R \wedge ((\{s | s \in S \wedge r.A_1 = s.B_1 \wedge p\} = \emptyset) \vee \\
&\quad (\{s | s \in S \wedge \neg p \wedge r.A_1 = s.B_1 \wedge r.A_2 = s.B_2\} = \emptyset))\} \\
&= \{r | r \in R \wedge (\{s | s \in S \wedge ((r.A_1 = s.B_1 \wedge p) \vee (\neg p \wedge r.A_1 = s.B_1 \wedge r.A_2 = s.B_2))\} = \emptyset)\} \\
&= \{r | r \in R \wedge \neg\exists s \in S \wedge ((p \wedge r.A_1 = s.B_1) \vee (\neg p \wedge r.A_1 = s.B_1 \wedge r.A_2 = s.B_2))\} \\
&= \{r | r \in R \wedge \neg\exists s \in S \wedge r.A_1 = s.B_1 \wedge (p \vee (\neg p \wedge r.A_2 = s.B_2))\} \\
&= \{r | r \in R \wedge \neg\exists s \in \{t | t \in S \wedge (p \vee (\neg p \wedge r.A_2 = t.B_2))\} \wedge r.A_1 = s.B_1\} \\
&= \sigma_{\neg\exists_{A_1=B_1}(\sigma_{A_2=B_2 \vee p}(S))}(R) \\
&= \text{lhs}
\end{aligned}
$$

In the last step we assume short-circuit evaluation of the disjunction. In the step marked * we rely on the more general definition of the bypass antijoin where the negative stream is defined by set difference.

$$
\begin{aligned}
e_1 \triangleright^{+}_{p} e_2 &:= \{x | x \in e_1 \wedge \neg\exists y \in e_2 \wedge p\} \\
e_1 \triangleright^{-}_{p} e_2 &:= e_1 - (e_1 \triangleright^{+}_{p} e_2)
\end{aligned}
$$

For the correctness of this equivalence for multisets it is important to realize that the bypass antijoin partitions the tuples of R into two disjoint streams. Either stream is correct under multiset semantics. Finally the outermost union merges both streams without changing the number of duplicates.

B.7 Proof of Equivalence 20

$$
\sigma_{A_1 \theta_1 f(S)}(R) = \sigma_{A_1 \theta_1 g}(\chi_{g:f(S)}(R)) \tag{20}
$$

We observe that the equivalence holds if and only if the argument to the selections is equivalent. To see this we rewrite the lhs as follows:

$$
\begin{aligned}
\text{lhs} &= \sigma_{A_1\theta f(S)}(R) \\
&= \{r | r \in R \wedge A_1\theta f(\{s | s \in S\})\} \\
&= \{t_{|\mathcal{A}(R)} | t \in \{r \circ [g : f(\{s | s \in S\})] | r \in R\} \wedge t.g\theta t.A_1\} \\
&= \sigma_{A_1\theta g}(\chi_{g:f(S)}(R)) \\
&= \text{rhs}
\end{aligned}
$$

As a result, all proofs presented next are also valid when the result of the nested scalar query block is returned with the tuples of the outer query block. Hence, these equivalences are also valid for disjunctive correlation of nested scalar queries in the select clause.

B.8 Proof of Equivalence 13

$$
\begin{aligned}
\sigma_{p \vee A_1\theta f(\sigma_{A_2=B_2}(S))}(R) &= e_1 \dot{\cup} e_2 \\
e_1 &:= \sigma_p^+(R) \\
e_2 &:= \Pi_{\mathcal{A}(R)}(\sigma_{g\theta A_1}((\sigma_p^-(R)) \bowtie^{g:f(\emptyset)}_{A_2=B_2} (\Gamma_{g;=B_2;f}(S))))
\end{aligned}
$$

if $\mathcal{F}(p) \subseteq \mathcal{A}(R)$, $\mathcal{A}(R) \cap \mathcal{A}(S) = \emptyset$, A_1,$A_2 \in \mathcal{A}(R)$, $B_2 \in \mathcal{A}(S)$, $g \notin \mathcal{A}(R) \cup \mathcal{A}(S)$

We define $R' := \sigma_p^-(R) = \{r | r \in R \wedge \neg p\}$ and
$S' := \Gamma_{g;=B_2;f}(S) = \{s_{|B_2} \circ [g : G] | s \in S \wedge G = f(\{y | y \in S \wedge y.B_2 = s.B_2\})\}$.

$$
\begin{aligned}
\text{rhs} &= \sigma_p^+(R) \dot{\cup} \Pi_{\mathcal{A}(R)}(\sigma_{g\theta A_1}((\sigma_p^-(R)) \bowtie^{g:f(\emptyset)}_{A_2=B_2} (\Gamma_{g;=B_2;f}(S)))) \\
&= \{r | r \in R \wedge p\} \dot{\cup} \\
&\quad \{r | r \in R' \wedge s \in S' \wedge r.A_2 = s.B_2 \wedge s.g\theta r.A_1\} \cup \\
&\quad \{r | r \in R' \wedge \neg\exists s \in S' : r.A_2 = s.B_2 \wedge \mathcal{A}(z) = \mathcal{A}(S') \wedge g \in \mathcal{A}(S') \wedge \\
&\qquad \forall a \in \mathcal{A}(S) \setminus g : (z.a : \text{NULL}) \wedge z.g : f(\emptyset) \wedge z.g\theta r.A_1\} \\
&= \{r | r \in R \wedge (p \vee (\neg p \wedge ((s \in S' \wedge r.A_2 = s.B_2 \wedge s.g\theta r.A_1) \vee \\
&\qquad (\neg\exists s \in S' : r.A_2 = s.B_2 \wedge \mathcal{A}(z) = \mathcal{A}(S') \wedge g \in \mathcal{A}(S') \wedge \\
&\qquad \forall a \in \mathcal{A}(S) \setminus g : (z.a : \text{NULL}) \wedge z.g : f(\emptyset) \wedge z.g\theta r.A_1))))\} \\
&= \{r | r \in R \wedge (p \vee (\neg p \wedge \\
&\qquad ((s \in \{t_{|B_2} \circ [g : G] | t \in S \wedge G = f(\{y | y \in S \wedge y.B_2 = t.B_2\})\} \wedge r.A_2 = s.B_2 \wedge s.g\theta r.A_1) \vee \\
&\qquad (\neg\exists s \in \{t_{|B_2} \circ [g : G] | t \in S \wedge G = f(\{y | y \in S \wedge y.B_2 = t.B_2\})\} : r.A_2 = s.B_2 \wedge \\
&\qquad\quad \mathcal{A}(z) = \mathcal{A}(S') \wedge g \in \mathcal{A}(S') \wedge \forall a \in \mathcal{A}(S) \setminus g : (z.a : \text{NULL}) \wedge z.g : f(\emptyset) \wedge z.g\theta r.A_1))))\} \\
&= \{r | r \in R \wedge (p \vee (\neg p \wedge ((s \in S \wedge G = f(\{y | y \in S \wedge y.B_2 = s.B_2\}) \wedge r.A_2 = s.B_2 \wedge G\theta r.A_1) \vee \\
&\qquad (\neg\exists s \in S : r.A_2 = s.B_2 \wedge \mathcal{A}(z) = \mathcal{A}(S) \wedge \forall a \in \mathcal{A}(S) : (z.a : \text{NULL}) \wedge G = f(\emptyset) \wedge G\theta r.A_1))))\} \\
&= \{r | r \in R \wedge (p \vee (\neg p \wedge G = f(\{y | y \in S \wedge r.A_2 = y.B_2\}) \wedge G\theta r.A_1))\} \\
&= \sigma_{p \vee A_1\theta f(\sigma_{A_2=B_2}(S))}(R) \\
&= \text{lhs}
\end{aligned}
$$

For the correctness of this equivalence over multisets we note that the bypass selection partitions the tuples in R into two disjoint sets. The leftouterjoin finds at most one match with the tuples of expression $\Gamma_{g;=B_2;f}S$ because the grouping operator creates a key on the join attribute B_2. Additionally, the leftouterjoin returns by definition at least one tuple for each tuple in R and thus handles the case for empty groups. Hence, duplicates in R are handled properly.

The result of the aggregation is computed correctly by the unary grouping operator. It simply precomputes the value of the aggregate function f for each value of attribute B_2 in S, i.e. for each non-empty group.

The proof of equivalence 14 for the special case of equality predicates follows directly from the proof of the following Lemma. Note however, that we require A_2 in R to be a key. Otherwise, as the lemma tells us, attribute A_1 and other attributes of R are not available after binary grouping.

Lemma:

$$\Pi_{A_2 \cup g}(R \bowtie^{f(\emptyset)}_{A_2=B_2} (\Gamma_{g;=B_2;f}(S))) \quad \equiv \quad (R)\Gamma_{g;A_2=B_2;f}(S) \tag{21}$$

Proof:

$$\begin{aligned}
\text{lhs} \;=\; & \{r_{|A_2} \circ s_{|g} | r \in R \wedge s \in \{t \circ [g:G] | t \in S \wedge \\
& \quad G = f(\{y | y \in S \wedge t.B_2 = y.B_2\})\} \wedge r.A_2 = s.B_2\} \cup \\
& \{r_{|A_2} \circ z_{|g} | r \in R \wedge \neg\exists y \in S : r.A_2 = y.B_2 \wedge \mathcal{A}(z) = \mathcal{A}(S) \wedge g \in \mathcal{A}(S) \wedge \\
& \quad \forall a \in (\mathcal{A}(S) \setminus g) : (z.a : \text{NULL} \wedge z.g : f(\emptyset))\} \\
=\; & \{r_{|A_2} \circ [g:G] | r \in R \wedge s \in S \wedge r.A_2 = s.B_2 \wedge G = f(\{y | y \in S \wedge s.B_2 = y.B_2\})\} \cup \\
& \{r_{|A_2} \circ [g:G] | r \in R \wedge (\neg\exists s \in S : r.A_2 = s.B_2) \wedge G = f(\emptyset)\} \\
=\; & \{r_{|A_2} \circ [g:G] | r \in R \wedge ((s \in S \wedge r.A_2 = s.B_2 \wedge G = f(\{y | y \in S \wedge s.B_2 = y.B_2\})) \vee \\
& \quad (G = f(\emptyset) \wedge \neg\exists s \in S : r.A_2 = s.B_2))\} \\
=\; & \{r_{|A_2} \circ [g:G] | r \in R \wedge G = f(\{y | y \in S \wedge r.A_2 = y.B_2\})\} \\
=\; & (R)\Gamma_{g;A_2=B_2;f}(S) \\
=\; & \text{rhs}
\end{aligned}$$

B.9 Proof of Equivalence 14

$$\begin{aligned}
\Pi_{A_1,A_2}(\sigma_{p \vee A_1\theta_1 f(\sigma_{A_2\theta_2 B_2}(S))}(R)) \;&=\; \Pi_{A_1,A_2}(e_1 \dot\cup e_2) \\
e_1 \;&:=\; \sigma^+_p(R) \\
e_2 \;&:=\; \sigma_{g\theta_1 A_1}(\sigma^-_p(R)\Gamma_{g;A_2\theta_2 B_2;f}(S))
\end{aligned}$$

if $\mathcal{F}(p) \subseteq \mathcal{A}(R)$, $\mathcal{A}(R) \cap \mathcal{A}(S) = \emptyset$, A_1,$A_2 \in \mathcal{A}(R)$, $B_2 \in \mathcal{A}(S)$, $g \notin \mathcal{A}(R) \cup \mathcal{A}(S)$ and the functional dependency $A_2 \rightarrow A_1$ holds.

$$\begin{aligned}
\text{rhs} \;=\; & \Pi_{A_1,A_2}(\sigma^+_p(R) \dot\cup (\Pi_{\mathcal{A}(R)}(\sigma_{g\theta_1 A_1}(\sigma^-_p(R)\Gamma_{g;A_2\theta_2 B_2;f}(S))))) \\
=\; & \{r_{|A_1,A_2} | r \in R \wedge p\} \dot\cup \{t_{|A_1,A_2} | t \in \{r \circ [g:G] | r \in R \wedge G\theta_1 A_1 \wedge \neg p \wedge \\
& \quad G = f(\{y | y \in S \wedge r.A_2\theta_2 y.B_2\})\}\} \\
=\; & \{r_{|A_1,A_2} | r \in R \wedge (p \vee (\neg p \wedge G\theta_1 A_1 \wedge G = f(\{y | y \in S \wedge r.A_2\theta_2 y.B_2\})))\} \\
=\; & \{r_{|A_1,A_2} | r \in R \wedge (p \vee (\neg p \wedge G\theta_1 A_1 G = f(\{y | y \in S \wedge r.A_2\theta_2 y.B_2\})))\} \\
=\; & \Pi_{A_1,A_2}(\sigma_{p \vee A_1\theta_1 f(\sigma_{A_2\theta_2 B_2}(S))}(R)) \\
=\; & \text{lhs}
\end{aligned}$$

For the correctness of this equivalence over multisets we note that the bypass selection partitions the tuples in R into two disjoint sets. The returns exactly one tuple per input tuple of $\sigma^-_p(R)$ extended with the result of the aggregate function f. Thereby empty groups are initialized and handled properly. The final union merges the disjoint subsets of the bypass selection establishing the correct final result.

B.10 Proof of Equivalence 15

$$\begin{aligned}\sigma_{\mathrm{p}\vee\mathrm{A}_1\theta(\mathrm{f}(\sigma_{\mathrm{A}_2=\mathrm{B}_2}(\mathrm{S})))}(\mathrm{R}) &\equiv \Pi_{\mathcal{A}(R)}(e_1 \dot\cup e_2)\\ e_1 &:= \sigma^+_{\mathrm{g}\theta\mathrm{A}_1}((\mathrm{R}) \ltimes^{\mathrm{g:f}(\emptyset)}_{\mathrm{A}_2=\mathrm{B}_2} (\Gamma_{\mathrm{g};=\mathrm{B}_2;\mathrm{f}}(\mathrm{S})))\\ e_2 &:= \sigma_p(\sigma^-_{\mathrm{g}\theta\mathrm{A}_1}((\mathrm{R}) \ltimes^{\mathrm{g:f}(\emptyset)}_{\mathrm{A}_2=\mathrm{B}_2} (\Gamma_{\mathrm{g};=\mathrm{B}_2;\mathrm{f}}(\mathrm{S}))))\end{aligned}$$

if $\mathcal{F}(p) \subseteq \mathcal{A}(R)$, $\mathcal{A}(R) \cap \mathcal{A}(S) = \emptyset$, A_1,$A_2 \in \mathcal{A}(R)$, $B_2 \in \mathcal{A}(S)$, $g \notin \mathcal{A}(R) \cup \mathcal{A}(S)$
First, we simplify the following subexpression X that occurs both in e_1 and e_2:

$$\begin{aligned}X &= \Pi_{\mathcal{A}(R)\cup g}((\mathrm{R}) \ltimes^{\mathrm{g:f}(\emptyset)}_{\mathrm{A}_2=\mathrm{B}_2} (\Gamma_{\mathrm{g};=\mathrm{B}_2;\mathrm{f}}(\mathrm{S})))\\ &= \{r \circ s_{|g} | r \in R \wedge s \in \{t \circ [g:G] | t \in S \wedge\\ &\quad G = f(\{y | y \in S \wedge t.B_2 = y.B_2\})\} \wedge r.A_2 = s.B_2\} \cup\\ &\quad \{r \circ z_{|g} | r \in R \wedge \neg\exists y \in S : r.A_2 = y.B_2 \wedge \mathcal{A}(z) = \mathcal{A}(S) \wedge g \in \mathcal{A}(S) \wedge\\ &\quad \forall a \in (\mathcal{A}(S) \setminus g) : (z.a : \mathrm{NULL} \wedge z.g : f(\emptyset))\}\\ &= \{r \circ [g:G] | r \in R \wedge s \in S \wedge r.A_2 = s.B_2 \wedge G = f(\{y | y \in S \wedge s.B_2 = y.B_2\})\} \cup\\ &\quad \{r \circ [g:G] | r \in R \wedge \neg\exists s \in S : r.A_2 = s.B_2 \wedge G = f(\emptyset)\}\\ &= \{r \circ [g:G] | r \in R \wedge ((s \in S \wedge r.A_2 = s.B_2 \wedge G = f(\{y | y \in S \wedge s.B_2 = y.B_2\})) \vee\\ &\quad (G = f(\emptyset) \wedge \neg\exists s \in S : r.A_2 = s.B_2))\}\\ &= \{r \circ [g:G] | r \in R \wedge G = f(\{y | y \in S \wedge r.A_2 = y.B_2\})\}\end{aligned}$$

Now we use X to complete the proof:

$$\begin{aligned}\mathrm{rhs} &= \Pi_{\mathcal{A}(R)}((\sigma^+_{g\theta A_1}(X)) \dot\cup (\sigma_p(\sigma^-_{g\theta A_1}(X))))\\ &= \{x_{|\mathcal{A}(R)} | x \in X \wedge x.g\theta x.A_1\} \dot\cup \{x_{|\mathcal{A}(R)} | x \in X \wedge \neg(x.g\theta x.A_1) \wedge p\}\\ &= \{x_{|\mathcal{A}(R)} | x \in X \wedge ((x.g\theta x.A_1) \vee (\neg(x.g\theta x.A_1) \wedge p))\}\\ &= \{x_{|\mathcal{A}(R)} | x \in \{r \circ [g:G] | r \in R \wedge G = f(\{y | y \in S \wedge r.A_2 = y.B_2\})\}\\ &\qquad \wedge((x.g\theta x.A_1) \vee (\neg(x.g\theta x.A_1) \wedge p))\}\\ &= \{r | r \in R \wedge G = f(\{y | y \in S \wedge r.A_2 = y.B_2\}) \wedge ((G\theta r.A_1) \vee (\neg(G\theta r.A_1) \wedge p))\}\\ &= \sigma_{p\vee A_1\theta f(\sigma_{A_2=B_2}(S))}(R)\\ &= \mathrm{lhs}\end{aligned}$$

The correctness of this equivalence over multisets follows by the same argumentation as for Equivalence 13.

B.11 Proof of Equivalence 16

The proof of Equivalence 16 for θ_2 as equality predicate follows from Lemma 21. Again the A_2 must be a key of R.

For the general case we have:

$$\begin{aligned}\Pi_{A_1,A_2}(\sigma_{\mathrm{p}\vee\mathrm{A}_1\theta_1(\mathrm{f}(\sigma_{\mathrm{A}_2\theta_2\mathrm{B}_2}(\mathrm{S})))}(\mathrm{R})) &\equiv \Pi_{A_1,A_2}(e_1 \dot\cup e_2)\\ e_1 &:= \sigma^+_{\mathrm{g}\theta_1\mathrm{A}_1}((\mathrm{R})\Gamma_{g;\mathrm{A}_2\theta_2\mathrm{B}_2;f}(\mathrm{S}))\\ e_2 &:= \sigma_p(\sigma^-_{\mathrm{g}\theta_1\mathrm{A}_1}((\mathrm{R})\Gamma_{g;\mathrm{A}_2\theta_2\mathrm{B}_2;f}(\mathrm{S})))\end{aligned}$$

if $\mathcal{F}(p) \subseteq \mathcal{A}(R)$, $\mathcal{A}(R) \cap \mathcal{A}(S) = \emptyset$, A_1,$A_2 \in \mathcal{A}(R)$, $B_2 \in \mathcal{A}(S)$, $g \notin \mathcal{A}(R) \cup \mathcal{A}(S)$ and the functional dependency $A_2 \rightarrow A_1$ holds.

To simplify the following expression we define X as:

$$\begin{aligned} X &= (\mathrm{R})\Gamma_{g;\mathrm{A}_2\theta_2\mathrm{B}_2;f}(\mathrm{S}) \\ &= \{r_{|A_1,A_2} \circ [g:G] | r \in R \wedge G = f(\{s | s \in S \wedge r.A_2\theta_2 s.B_2\})\} \end{aligned}$$

$$\begin{aligned} \text{rhs} &= \Pi_{A_1,A_2}(\sigma^{+}_{\mathrm{g}\theta_1\mathrm{A}_1}((\mathrm{R})\Gamma_{g;\mathrm{A}_2\theta_2\mathrm{B}_2;f}(\mathrm{S})) \dot\cup \sigma_p(\sigma^{-}_{\mathrm{g}\theta_1\mathrm{A}_1}((\mathrm{R})\Gamma_{g;\mathrm{A}_2\theta_2\mathrm{B}_2;f}(\mathrm{S})))) \\ &= \{t_{A_1,A_2} | t \in (\{x | x \in X \wedge x.g\theta_1 x.A_1\} \dot\cup \{y | y \in X \wedge \neg(y.g\theta_1 y.A_1) \wedge p\})\} \\ &= \{t_{A_1,A_2} | t \in X \wedge x.g\theta_1 x.A_1 \vee p\} \\ &= \{t_{A_1,A_2} | t \in \{r_{|A_1,A_2} \circ [g:G] | r \in R \wedge G = f(\{s | s \in S \wedge r.A_2\theta_2 s.B_2\})\} \wedge (G\theta_1 x.A_1 \vee p)\} \\ &= \{r_{A_1,A_2} | r \in R \wedge G = f(\{s | s \in S \wedge r.A_2\theta_2 s.B_2\}) \wedge (G\theta_1 r.A_1 \vee p)\} \\ &= \Pi_{A_1,A_2}(\sigma_{p \vee A_1\theta_1 f(\sigma_{A_2\theta_2 B_2}(S))}(R)) \\ &= \text{lhs} \end{aligned}$$

The correctness of this equivalence for multisets follows by the same arguments as for Eqv.14.

B.12 Proof of Equivalence 17

$$\begin{aligned} \sigma_{A_1\theta f(\sigma_{A_2=B_2 \vee p}(S))}(R) &= \Pi_{\mathcal{A}(R)}(\sigma_{A_1\theta g}(\chi_{g:f_O(g_1,e_2)}(e_1))) \\ e_1 &:= R \,\mathbb{N}^{g_1:f_I(\emptyset)}_{A_2=B_2} (\Gamma_{g_1;=B_2;f_I}(\sigma^{-}_p(S))) \\ e_2 &:= f_I(\sigma^{+}_p(S)) \end{aligned}$$

if $\mathcal{F}(p) \subseteq \mathcal{A}(S)$, $\mathcal{A}(R) \cap \mathcal{A}(S) = \emptyset$, $A_i \in \mathcal{A}(R)$, $i = 1, 2$, $B_2 \in \mathcal{A}(S)$, $g, g_1 \notin \mathcal{A}(R) \cup \mathcal{A}(S)$, f is decomposable [21], i.e.

there are sets X, Y, and Z, with $X = Y \dot\cup Z$ and $Y \cap Z = \emptyset$. The scalar aggregate function $f : X \to \mathcal{N}$ is *decomposable* if there exist functions

$$\begin{aligned} f_I : X &\to \mathcal{N}' \\ f_O : \mathcal{N}', \mathcal{N}' &\to \mathcal{N} \end{aligned}$$

with $f(X) = f_O(f_I(Y), f_I(Z))$.

To simplify the expressions we define X.

$$\begin{aligned} X &= \Pi_{\mathcal{A}(R) \cup g_1}(R \,\mathbb{N}^{g_1:f_I(\emptyset)}_{A_2=B_2} (\Gamma_{g_1;=B_2;f_I}(\sigma^{-}_p(S)))) \\ &= \{r \circ s_{|g_1} | r \in R \wedge s \in \{t \circ [g_1:G] | t \in S \wedge \neg p \wedge \\ &\quad G = f_I(\{y | y \in S \wedge \neg p \wedge t.B_2 = y.B_2\})\} \wedge r.A_2 = s.B_2\} \cup \\ &\quad \{r \circ z_{|g_1} | r \in R \wedge \neg\exists y \in S : r.A_2 = y.B_2 \wedge \neg p \wedge \mathcal{A}(z) = \mathcal{A}(S) \wedge g_1 \in \mathcal{A}(S) \wedge \\ &\quad \forall a \in (\mathcal{A}(S) \setminus g_1) : (z.a : \mathrm{NULL} \wedge z.g_1 : f_I(\emptyset))\} \\ &= \{r \circ [g_1:G] | r \in R \wedge G = f_I(\{y | y \in S \wedge \neg p \wedge r.A_2 = y.B_2\})\} \cup \\ &\quad \{r \circ z_{|g_1} | r \in R \wedge \neg\exists y \in S : r.A_2 = y.B_2 \wedge \neg p \wedge \mathcal{A}(z) = \mathcal{A}(S) \wedge g_1 \in \mathcal{A}(S) \wedge \\ &\quad \forall a \in (\mathcal{A}(S) \setminus g_1) : (z.a : \mathrm{NULL} \wedge z.g_1 : f_I(\emptyset))\} \\ &= \{r \circ [g_1:G] | r \in R \wedge G = f_I(\{y | y \in S \wedge \neg p \wedge r.A_2 = y.B_2\})\} \cup \\ &\quad \{r \circ [g_1:G] | r \in R \wedge \neg\exists y \in S : r.A_2 = y.B_2 \wedge \neg p \wedge G = f_I(\emptyset))\} \\ &= \{r \circ [g_1:G] | r \in R \wedge G = f_I(\{y | y \in S \wedge \neg p \wedge r.A_2 = y.B_2\})\} \end{aligned}$$

Using Eqv. 20, this equivalence also holds without the final selection on both sides. Hence, we remove can it in the remainder of the proof.

$$
\begin{aligned}
\text{rhs} &= \Pi_{\mathcal{A}(R)\cup g}(\chi_{g:f_O(g_1,f_I(\sigma_p^+(S)))}(X)) \\
&= \{x \circ [g : f_O(g_1, f_I(\{y|y \in S \wedge p\}))]_{|\mathcal{A}(R)\cup g}|x \in X\} \\
&= \{x \circ [g : G]_{|\mathcal{A}(R)\cup g}|x \in \{r \circ [g_1 : G]|r \in R \wedge G = f_I(\{y|y \in S \wedge \neg p \wedge r.A_2 = y.B_2\})\} \wedge \\
&\quad G = f_O(g_1, f_I(\{y|y \in S \wedge p\}))\} \\
&= \{r \circ [g : G]|r \in R \wedge G = f_O(f_I(\{y|y \in S \wedge \neg p \wedge r.A_2 = y.B_2\}), f_I(\{y|y \in S \wedge p\}))\} \\
&= \{r \circ [g : f(\{y|y \in S \wedge (r.A_2 = y.B_2 \vee p)\})]|r \in R\} \\
&= \chi_{g:f(\sigma_{A_2=B_2\vee p}(S))}(R) \\
&= \text{lhs}
\end{aligned}
$$

To establish the correctness for multisets we observe that no tuple of R is duplicated. Hence, we only have to be careful that at most one tuple of S matches with every tuple of R.

In the first part of the proof tuples of $\sigma_p^-(S)$ are partitioned by the join predicate $A_2 = B_2$. Matching tuples in $\sigma_p^-(S)$ are combined to exactly one value — the result of the aggregation function f_I. This value produces exactly one output tuple for matching tuple in R. Tuples in R that do not find any matching tuple in $\sigma_p^-(S)$ are padded with the value of the aggregation function for an empty input.

In the second part no new tuples are produced or filtered. Thus, the correct number of result tuples are produced. When we investigate the correctness of the final aggregate value we note that the tuples of S are partitioned by predicate p (resp. $\neg p$). The latter were handled properly in the first part of the proof. The former are aggregated independently in the subscript of the map operator χ. When both partitions are combined according to the definition of decomposable aggregate functions the correct aggregate value is computed.

B.13 Proof of Equivalence 18

$$
\begin{aligned}
\Pi_{A_1,A_2}(\sigma_{A_1\theta_1 f(\sigma_{A_2\theta_2 B_2\vee p}(S))}(R)) &= \Pi_{A_1,A_2}(\sigma_{A_1\theta_1 g}(\chi_{g:f_O(g_1,e_2)}(e_1))) \\
e_1 &:= (R)\Gamma_{g_1;A_2\theta_2 B_2;f_I}(\sigma_p^-(S)) \\
e_2 &:= f_I(\sigma_p^+(S))
\end{aligned}
$$

if $\mathcal{F}(p) \subseteq \mathcal{A}(S)$, $\mathcal{A}(R) \cap \mathcal{A}(S) = \emptyset$, $A_i \in \mathcal{A}(R)$, $i = 1, 2$, $B_2 \in \mathcal{A}(S)$, $g, g_1 \notin \mathcal{A}(R) \cup \mathcal{A}(S)$, the functional dependency $A_2 \rightarrow A_1$ holds, and f is decomposable as discussed in the proof for Eqv. 17.

Using Eqv. 20, this equivalence also holds without the final selection on both sides. Hence, we remove can it in the remainder of the proof. In addition, we ignore the final projection as it is only needed to establish the same schema of the result tuples.

$$
\begin{aligned}
\text{rhs} &= \chi_{g:f_O(g_1,f_I(\sigma_p^+(S)))}((R)\Gamma_{g_1;A_2\theta_2 B_2;f_I}(\sigma_p^-(S))) \\
&= \{x \circ [g : f_O(x.g_1, f_I(\{y|y \in S \wedge p\}))]|x \in \{r \circ [g_1 : G]|r \in R \wedge \\
&\qquad G = f_I(\{y|y \in S \wedge \neg p \wedge r.A_2\theta_2 y.B_2\})\} \\
&= \{r \circ [g : f_O(f_I(\{y|y \in S \wedge \neg p \wedge r.A_2\theta_2 y.B_2\}), f_I(\{y|y \in S \wedge p\}))]|r \in R\} \\
&= \{r \circ [g : f(\{y|y \in S \wedge (r.A_2\theta_2 y.B_2 \vee p)\})]|r \in R\} \\
&= \chi_{g:f(\sigma_{A_2\theta_2 B_2\vee p}(S))}(R) \\
&= \text{lhs}
\end{aligned}
$$

To establish the correctness for multisets we observe that no tuple of R is duplicated. The binary grouping takes care that for every tuple of R a single group exists.

The bypass selection partitions the tuples of S into two disjoint sets. The binary grouping matches all tuples of $\sigma_p^-(S)$ to the groups established from R. Tuples of R that do not find a join partner are

properly initialized by the binary grouping. Hence exactly the same tuples of R – extended with the result of aggregation – are returned by the binary grouping operator.

In the map operator does not produce or filtere any tuples. Thus, the correct number of result tuples are produced. When we investigate the correctness of the final aggregate value we note that the tuples of S are partitioned by predicate p (resp. $\neg p$). The latter were handled properly by the binary grouping operator. The former are aggregated independently in the subscript of the map operator χ. When both partitions are combined according to the definition of decomposable aggregate functions the correct aggregate value is computed.

B.14 Proof of Equivalence 19

$$\begin{array}{rcl}
\sigma_{A_1\theta_1 \mathrm{f}(\sigma_{A_2\theta_2 B_2 \vee p}(S))}(R) & = & \Pi_{\mathcal{A}(R)}(\sigma_{A_1\theta_1 g}((R')\Gamma_{g;t1=t1';f}(\rho_{t1'\leftarrow t1}(e_1 \dot\cup e_2)))) \\
R' & := & \nu_{t1}(R) \\
e_1 & := & R' \bowtie^{+}_{A_2\theta_2 B_2} S \\
e_2 & := & \sigma_p(R' \bowtie^{-}_{A_2\theta_2 B_2} S)
\end{array}$$

if $\mathcal{F}(p) \subseteq \mathcal{A}(R) \cup \mathcal{A}(S)$, $\mathcal{A}(R) \cap \mathcal{A}(S) = \emptyset$, $A_i \in \mathcal{A}(R), i = 1, 2$, $B_2 \in \mathcal{A}(S)$, $g \notin \mathcal{A}(R) \cup \mathcal{A}(S)$

Let us point out that we assume the id $t1$ returned by the numbering operator is computed deterministically. For an ordered set, this might be the position of a tuple within the set, it might be a tuple identifier as it is commonly used to store tuples in relational databases, or simply the key attribute of a relation. We may only use that for $t \in R'$ and $s \in R'$ it holds that if $t.t1 = s.t1 \Rightarrow t = s$. Note that $t1$ is a key for the tuple it is generated for.

We use X as a shortcut:

$$\begin{array}{rcl}
X & := & \rho_{t'_1\leftarrow t_1}(e_1 \dot\cup e_2) \\
& = & \rho_{t'_1\leftarrow t_1}((R' \bowtie^{+}_{A_2\theta_2 B_2} S) \dot\cup (\sigma_p(R' \bowtie^{-}_{A_2\theta_2 B_2} S))) \\
& = & \{r \circ s_{t'_1\leftarrow t_1} | r \in R' \wedge s \in S \wedge r.A_2\theta_2 s.B_2\} \dot\cup \\
& & \{y_{t'_1\leftarrow t_1} | y \in \{r \circ s | r \in R' \wedge s \in S \wedge \neg(r.A_2\theta_2 s.B_2)\} \wedge p\} \\
& = & \{r \circ s_{t'_1\leftarrow t_1} | r \in R' \wedge s \in S \wedge r.A_2\theta_2 s.B_2\} \dot\cup \\
& & \{r \circ s_{t'_1\leftarrow t_1} | r \in R' \wedge s \in S \wedge \neg(r.A_2\theta_2 s.B_2) \wedge p\} \\
& = & \{r \circ s_{t'_1\leftarrow t_1} | r \in R' \wedge s \in S \wedge (r.A_2\theta_2 s.B_2 \vee (\neg(r.A_2\theta_2 s.B_2) \wedge p))\}
\end{array}$$

Thanks to Eqv. 20 we can safely remove the final selection on both sides. We also remove the final projection on the rhs because it is only required for syntactic reasons.

$$\begin{array}{rcl}
\mathrm{rhs} & = & (R')\Gamma_{g;t1=t1';f}(X) \\
& = & \{t \circ [g : G] | t \in R' \wedge G = f(\{y | y \in X \wedge t.t_1 = y.t'_1\})\} \\
& = & \{t \circ [g : G] | t \in R' \wedge G = f(\{y | y \in \{[s \circ r]_{t'_1\leftarrow t_1} | r \in R' \wedge s \in S \wedge \\
& & \quad (r.A_2\theta_2 s.B_2 \vee (\neg(r.A_2\theta_2 s.B_2) \wedge p))\} \wedge t.t_1 = y.t'_1\})\} \\
& \stackrel{*}{=} & \{t \circ [g : G] | t \in R \wedge G = f(\{s | s \in S \wedge (t.A_2\theta_2 s.B_2 \vee (\neg(t.A_2\theta_2 s.B_2) \wedge p))\})\} \\
& \stackrel{**}{=} & \{t \circ [g : G] | t \in R \wedge G = f(\{s | s \in S \wedge (t.A_2\theta_2 s.B_2 \vee p)\})]\} \\
& = & \{t \circ [g : f(\{s | s \in S \wedge (t.A_2\theta_2 s.B_2 \vee p)\})] | t \in R\} \\
& = & \chi_{g:f(\sigma_{A_2\theta_2 B_2 \vee p}(S))}(R) \\
& = & \mathrm{lhs}
\end{array}$$

In the step marked * we use the bijectivity of the numbering operator mentioned in the beginning of the proof. Hence we resume with R instead of R'. In the step marked ** we make use of short circuit evaluation of $\vee$.

Note that employing the numbering operator makes this equivalence applicable for either sets or multisets.