Universität Stuttgart

Fakultät Informatik, Elektrotechnik und Informationstechnik

Universitätsstraße 38
70569 Stuttgart
Germany

Report No. 2005/08

Laws for Rewriting Queries Containing Division Operators

Ralf Rantzau and Christoph Mangold

October 2005

CR-Klassifikation: H.2.4

Laws for Rewriting Queries Containing Division Operators*

Ralf Rantzau
IBM Almaden Research Center
650 Harry Road, San Jose, CA 95120, USA
rrantzau@acm.org

Christoph Mangold
Universität Stuttgart
Universitätsstraße 38, 70569 Stuttgart, Germany
mangold@informatik.uni-stuttgart.de

Abstract

Relational division, also known as small divide, *is a derived operator of the relational algebra that realizes a many-to-one set containment test, where a set is represented as a group of tuples: Small divide discovers which sets in a dividend relation contain all elements of the set stored in a divisor relation. The great divide operator extends small divide by realizing many-to-many set containment tests. It is also similar to the set containment join operator for schemas that are not in first normal form.*

Neither small nor great divide has been implemented in commercial relational database systems although the operators solve important problems and many efficient algorithms for them exist. We present algebraic laws that allow rewriting expressions containing small or great divide, illustrate their importance for query optimization, and discuss the use of great divide for frequent itemset discovery, an important data mining primitive.

A recent theoretic result shows that small divide must *be implemented by special purpose algorithms and not be simulated by pure relational algebra expressions to achieve efficiency. Consequently, an efficient implementation requires that the optimizer treats small divide as a first-class operator and possesses powerful algebraic laws for query rewriting.*

1 Introduction

In this section, we motivate our work, give an intuition of the small and great divide operators, and outline the paper.

1.1 Problem Statement and Main Results

The division operator can be used to answer queries involving universal quantification like "Find the suppliers that supply *all* blue parts." Division is a derived operator like join, that is, it can be expressed by the basic algebra operators projection, selection, Cartesian product (sometimes called cross-product), union, and difference (see Appendix A for operator definitions). However, several algorithms exist that realize its behavior more efficiently than an execution plan based on the basic operators [14]. More importantly, recent theoretic work has demonstrated that division must be implemented as a stand-alone operator to achieve efficiency [25].

The small divide operator has two input relations, the dividend and the divisor. The dividend is composed of zero or more groups of tuples and each group is matched against all tuples of the divisor relation. The great divide is a natural extension of small divide, where the divisor can be composed of zero or more groups of tuples like the dividend. It tests each divisor group against each dividend group.

What is the role of algebraic laws for query optimization? Before a query is executed by the query execution engine of a relational database management system (RDBMS), the query optimizer rewrites the algebraic representation of the query according to transformation rules. Typically, one type of transformation rules is based on algebraic laws and the other maps logical operators to a physical operators. For instance, the logical operator join is mapped to the physical operator hash-join.

An algebraic law is a logical equivalence between two different representations of an algebraic expression. Both representations describe the same set of tuples for every possible database content. Together with heuristics and/or cost estimations, the optimizer applies transformation rules to subexpressions of the query such that the entire query can be evaluated with the minimal resource consumption or the shortest response time. Algebraic laws for the basic operators of the relational algebra are discussed, for example, in [13, 24]. The implementation of transformation rules (rewrite rules) in a commercial RDBMS are described, for example, in [26, 31]. Frameworks for building query optimizers, like Cascades [15] and XXL [3], allow to study the code that is required to realize transformation rules in an

*This report has the same content as paper [35]. However, only this report contains the proofs of laws and theorems as an appendix, which were omitted in the paper due to space restrictions.

a	b
1	1
1	4
2	1
2	2
2	3
2	4
3	1
3	3
3	4

(a) r_1 (dividend)

b
1
3

(b) r_2 (divisor)

a
2
3

(c) r_3 (quotient)

Figure 1. Division: $r_1 \div r_2 = r_3$

RDBMS.

To the best of our knowledge, no commercial RDBMS has an implementation of relational division. One reason is that there is no keyword in the SQL standard that would allow to express universal quantification (that is, the all-quantifier) intuitively. Another reason is that set containment tests are not considered as important as the existential element test that is realized by the join operator. However, special applications like frequent itemset discovery could be processed efficiently and formulated more intuitively if division would be a first-class operator. Suppose that an RDBMS offers one or more efficient implementations of division, that is, physical division operators like hash-division or merge-sort division [16, 36]. Since division is a derived operator, an optimizer could replace the division operator by an expression that simulates the operator and apply transformation rules on the basic operators in the expression. In addition, it should also be able to apply rewrite rules to the division operator directly since efficient implementations are available in the query execution engine.

The algebraic laws presented in this paper either preserve the division operator (it occurs in the both expression of the equivalence) or produce some non-trivial rewrite result that may improve efficiency of the computation in an RDBMS. Note that there are an infinite number of equivalent expressions for any given algebraic expression. We have tried to distill effective and interesting laws for rule based optimizers.

No previous work has covered the rewriting of queries involving division or generalized division although data-intensive applications like frequent itemset discovery would benefit from a division syntax in SQL and an efficient implementation of the operator in a query execution engine.

1.2 Outline

The remainder of this paper is organized as follows. In the following section, we discuss several definitions for the small and great divide, which are used in the proofs of the laws. In Section 3, we motivate the potential of the great divide for an important data mining primitive. In Section 4, we suggest a hypothetical SQL syntax extension for the operators before we present the algebraic laws in Section 5. Section 6 discusses related work. We conclude the paper in Section 7. Appendix A gives an overview of the operators used in this paper. Appendix B contains all proofs. We decided to present the proofs in sufficient detail to make them easy to comprehend.

2 The Division Operator

We will discuss the original division operator as well as a generalization of it, which was given three different names in previous work. After this section, we will refer to the two operators as small divide and great divide for the rest of this paper.

2.1 The Small Divide

Let $R_1(A \cup B)$ and $R_2(B)$ be relation schemas, where $A = \{a_1, \ldots, a_m\}$ and $B = \{b_1, \ldots, b_n\}$ are nonempty disjoint sets of attributes. Let $r_1(R_1)$ and $r_2(R_2)$ be relations on these schemas. We call r_1 the *dividend*, r_2 the *divisor*, and r_3 the *quotient* of the division operation $r_1 \div r_2 = r_3$. The schema of r_3 is $R_3(A)$. Figure 1 illustrates example input and output relations of the division operator.

The original definition of the division operator was given by Codd [10], formulated as a query in tuple relational calculus:

DEFINITION 1 (CODD'S DIVISION): $r_1 \div r_2 = \{t \mid t = t_1.A \land t_1 \in r_1 \land r_2 \subseteq i_{r_1}(t)\}$, *where* $i_{r_1}(x)$ *is called the* image set of x under r_1 *and is defined by* $i_{r_1}(x) = \{y \mid (x, y) \in r_1\}$

In this calculus expression, the term $t = t_1.A$ means that a tuple in the result (quotient) consists of the attribute values for A of the dividend tuple t_1.

In the following, we give two further equivalent definitions of division, provided by Healy and Maier in [27] using relational algebra.[1] We use Codd's, Healy's, and Maier's definitions for the proofs of our algebraic laws.

DEFINITION 2 (HEALY'S DIVISION): $r_1 \div r_2 = \pi_A(r_1) - \pi_A((\pi_A(r_1) \times r_2) - r_1)$

[1] Another algebraic definition given in the literature is $r_1 \div r_2 = ((r_1 \ltimes r_2) ⟕ r_2)\overline{\ltimes} r_2$ [9], where semi-join ($\ltimes$), anti-semi-join ($\overline{\ltimes}$), and left outer join (⟕) are used. An indirect approach based on counting was discussed in [16], where ${}_G\gamma_F(r_1)$ is the grouping operator [13], G is a list of r_1's attributes and F is a list of aggregation functions applied to an attribute of r_1: $r_1 \div r_2 = \pi_A\left({}_A\gamma_{\text{count}(B)\to c}(r_1 \ltimes r_2) \bowtie \gamma_{\text{count}(B)\to c}(r_2)\right)$. A definition in tuple relational calculus is $r_1 \div r_2 = \{t \mid \forall t_2 \in r_2 \exists t_1 \in r_1 : t = t_1.A \land t_1.B = t_2.B\}$ [11]. A definition mixing tuple relational calculus with relational algebra is $r_1 \div r_2 = \{t \in \pi_A(r_1) \mid (t) \bowtie r_2 \subseteq r_1\}$ [1].

DEFINITION 3 (MAIER'S DIVISION): $r_1 \div r_2 = \bigcap_{t \in r_2} \pi_A \left(\sigma_{B=t} \left(r_1\right)\right)$

In [11], the basic division operator was called *small divide* to distinguish it from a generalization of it, called great divide, to be discussed next.

2.2 The Great Divide

Before we discuss three equivalent definitions of an extended division operator, we briefly consider another operator related to them: the set containment join. Let $R_1(A \cup B_1)$, $R_2(B_2 \cup C)$, and $R_3(A \cup B_1 \cup B_2 \cup C)$ be relation schemas, where $A = \{a_1, \ldots, a_m\}$, $B_1 = \{b_1\}$, $B_2 = \{b_2\}$, and $C = \{c_1, \ldots, c_p\}$ are attribute sets, A and C are disjoint and may be empty, B_1 and B_2 are disjoint and nonempty, A and B_1 are disjoint, and B_2 and C are disjoint. Note that the sets B_1 and B_2 consist of a single *set-valued* attribute, respectively. Let $r_1(R_1)$, $r_2(R_2)$, and $r_3(R_3)$ be relations on these schemas. The *set containment join* $r_1 \bowtie_{b_1 \supseteq b_2} r_2 = r_3$ is a join between the set-valued attributes b_1 and b_2, where we ask for the combinations of tuples $t_1 \in r_1$ and $t_2 \in r_2$ such that set $t_1.b_1$ contains all elements of set $t_2.b_2$. Several efficient algorithms and strategies for realizing this operator in an RDBMS have been proposed [19, 29, 30, 32, 33].

We have recently suggested a generalization of division that we called *set containment division*, denoted by $\div_1^*$, because of its similarity to the set containment join [36]. Let $R_1(A \cup B)$, $R_2(B \cup C)$, and $R_3(A \cup C)$ be relation schemas, where $A = \{a_1, \ldots, a_m\}$, $B = \{b_1, \ldots, b_n\}$, and $C = \{c_1, \ldots, c_p\}$ are nonempty sets of attributes, A and B are disjoint, and B and C are disjoint. Let $r_1(R_1)$, $r_2(R_2)$, and $r_3(R_3)$ be relations on these schemas. Although we define a new operator, we continue to use the terms dividend, divisor, and quotient for the relations r_1, r_2, and r_3, respectively. The dividend relation r_1 has the same schema as for the small divide. However, the divisor relation r_2 has additional attributes C. The set containment division operator is defined as follows:

DEFINITION 4 (SET CONTAINMENT DIVISION): $r_1 \div_1^* r_2 = \bigcup_{t \in \pi_C(r_2)} \left(r_1 \div \pi_B \left(\sigma_{C=t} \left(r_2\right)\right)\right) \times (t)$

The idea is to iterate over the groups defined by the attributes $r_2.C$. Each group is a separate divisor for a division with dividend r_1. We "attach" the divisor group value to the resulting quotient tuples by a Cartesian product between each quotient group and a one-tuple relation (t).

a	b
1	1
1	4
2	1
2	2
2	3
2	4
3	1
3	3
3	4

(a) r_1 (dividend)

b	c
1	1
2	1
4	1
1	2
3	2

(b) r_2 (divisor)

a	c
2	1
2	2
3	2

(c) r_3 (quotient)

Figure 2. Generalized division: $r_1 \div^* r_2 = r_3$

a	b_1
1	$\{1,4\}$
2	$\{1,2,3,4\}$
3	$\{1,3,4\}$

(a) r_1

b_2	c
$\{1,2,4\}$	1
$\{1,3\}$	2

(b) r_2

a	b_1	b_2	c
2	$\{1,2,3,4\}$	$\{1,2,4\}$	1
2	$\{1,2,3,4\}$	$\{1,3\}$	2
3	$\{1,3,4\}$	$\{1,3\}$	2

(c) r_3

Figure 3. Set containment join: $r_1 \bowtie_{b_1 \supseteq b_2} r_2 = r_3$

The similarity between set containment division and set containment join can be seen by comparing Figures 2 and 3. Despite the similarity of the output, the operators have some subtle differences:

1. The input relations of set containment join are not in first normal form due to the set-valued attributes.

2. Set containment division does not preserve the "join" attributes in B.

3. Set containment join allows empty sets as join attribute values whereas set containment division does not have the notion of an empty set.

4. The attribute sets A and C of the set containment join may be empty.

Despite these differences, the operators both solve the same problem—to find those pairs of sets (s_1, s_2) from two collections of sets where $s_1 \supseteq s_2$.

In 1982, Robert Demolombe suggested a *generalized division* operator, denoted by $\div_2^*$, that is equivalent (see Theorem 1 below) to set containment division [12]. Besides a definition of the operator in tuple relational calculus and predicate calculus, he gives an algebraic definition:

DEFINITION 5 (GENERALIZED DIVISION): $r_1 \div_2^* r_2 = \left(\pi_A\left(r_1\right) \times \pi_C\left(r_2\right)\right) - \pi_{A \cup C}\left(\left(\pi_A\left(r_1\right) \times r_2\right) - \left(r_1 \times \pi_C\left(r_2\right)\right)\right)$

In 1988, Stephen Todd suggested—presumably independent from Demolombe—a generalized division operator but he did not publish it himself. However, it has been discussed by Darwen and Date [11], where it was called *great divide*, denoted by $\div_3^*$. A definition in relational algebra is given by the following expression:

DEFINITION 6 (GREAT DIVIDE): $r_1 \div_3^* r_2 = \left(\pi_A\left(r_1\right) \times \pi_C\left(r_2\right)\right) - \pi_{A \cup C}\left(\left(\pi_A\left(r_1\right) \times r_2\right) - \left(r_1 \bowtie r_2\right)\right)$

Definition 6 differs only slightly from Definition 5 of generalized division. It uses a join instead of a Cartesian product. Darwen and Date write that great divide degenerates to small divide, as specified in Definition 2, if $C = \emptyset$ [11]. We prove in Appendix B the following theorem:

THEOREM 1: *Set containment division ($\div_1^*$), generalized division ($\div_2^*$), and great divide ($\div_3^*$) are equivalent operators.*

The three definitions have been suggested independently. However, while the publications on generalized division [12] and great divide [11] solely focus on the relationship between the *logical* operator and the basic division operator, our previous work on the set containment division operator [34, 36] put its emphasis on algorithms that implement *physical* operators and investigated applications for this operator. In the rest of the paper, we will use Demolombe's term *generalized division* and use the symbol $\div^*$ for the operator.

3 Frequent Itemset Discovery: An Application of Great Divide

Frequent itemset discovery is an important data mining subtask of association rule discovery algorithms [2]. It searches for combinations of elements that occur more frequently in a large amount of sets, called *transactions*, than a user-defined threshold, called *minimum support*. Most frequent itemset discovery algorithms such as *Apriori* proceed iteratively. In the kth iteration, the algorithm computes all frequent itemsets of size k. The first iteration simply counts the frequency of each item in the transactions, filters out those that have insufficient support, and adds the frequent ones to the result. Each of the following iterations is two-phase. In the *candidate generation phase* of the kth iteration, the algorithm computes a superset of the frequent itemsets of size k, called candidate k-itemsets. In the *support counting phase*, the candidate k-itemsets are probed against the transactions to check how many times a candidate is contained in a transaction. The itemsets that occur more frequently than the minimum support are added to the result.

Suppose, we want to discover frequent itemsets using an RDBMS. Let us focus on the support counting phase. For instance, given a table of transactions $\textit{transactions}(\textit{tid}, \textit{item})$ and a table of candidate itemsets $\textit{candidates}(\textit{itemset}, \textit{item})$, where *itemset* is a set identifier and *item* is an item identifier. A query-based frequent itemset discovery algorithm can compute a *quotient* table containing value pairs $(\textit{transactions.tid}, \textit{candidates.itemset})$ such that the item values belonging to *candidates.itemset* are contained in the set of items belonging to *transactions.tid*. This test is exactly the behavior of the great divide operator: $\textit{quotient} = \textit{transactions} \div^* \textit{candidates}$. Note that this computation does not require the candidate itemsets to have the same size k. The frequent itemsets can then be found by grouping the quotient table on *itemset*, counting the *tid* values per group, and discarding the groups with insufficient support.[2]

4 Embedding the Operators into SQL

In this section, we present a straightforward hypothetical syntax for the small and great divide operator in SQL and illustrate how these operators can be used for real queries. We will use a more straightforward example problem domain for the queries than in the previous section, namely the suppliers and parts scenario from database textbooks.

In the SQL standard [22], a production rule is defined for *table references*, which occur in the FROM clause of a query expression. We extend this clause by a nonterminal $\langle \textit{quotient} \rangle$ as follows:[3]

```
<table reference> ::= <table factor> |
                      <joined table> |
                      <quotient>
```

Without going into every detail of the SQL standard, this rule states that a table can be a base table, derived table, named query, etc., or the result of a join expression or the result of a division operation. We specify the following rule for expressions involving the small and great divide operators:

```
<quotient> ::= <table reference>
               DIVIDE BY
               <table reference>
               ON <search condition>
```

We illustrate the syntax using an example using a supplier-parts database with a table *supplies*(*s#*, *p#*) that lists the parts (*p#*) supplied by each supplier (*s#*) and a table *parts*(*p#*, *color*). The following query delivers for each color the suppliers who supply all parts with that color:[4]

```
Q1: SELECT s#, color
    FROM   supplies AS s DIVIDE BY parts AS p
           ON s.p# = p.p#
```

Note that we do not distinguish between the small and great divide on the language level. The great divide is a natural generalization of the small divide and can always be

[2] The idea of using a "vertical" representation for itemsets in the same way as for transactions that we just described was discussed in [36]. It is different from all SQL-based approaches of frequent itemset discovery in the literature as, for example, in [21, 37, 39].

[3] shown in extended Backus Normal Form (BNF) as in [23].

[4] We actually ask only for those suppliers who supply at least one part, that is, those *s#* values in a *suppliers*(*s#*, ...) table, where there exists a tuple in the *supplies* table with that *s#* value. This is a slight semantic difference between set containment join and great divide, as mentioned in Section 2.2.

used on the implementation/execution level. The ⟨*quotient*⟩ construct is equivalent to a *small* divide if *all* divisor attributes appear in the join condition of the ON clause as a conjunction[5] of equi-joins. An example use of small divide is the query "Find the suppliers that supply all blue parts" that was mentioned in Section 1.1, which can be formulated as follows:

```
Q2: SELECT s#
    FROM   supplies AS s DIVIDE BY (
             SELECT p#
             FROM   parts
             WHERE  color = 'blue') AS p
           ON s.p# = p.p#
```

Concerning the power of the suggested SQL syntax, one could allow a more general join condition than equi-joins between columns in the ON clause. However, the result of such a query would have a semantics that is completely different from small or great divide. We suggest to disallow this case. If such a different behavior is required, a user can still formulate the problem using other, basic operators of the SQL syntax.

We contrast query Q_1 with an equivalent query that simulates the universal quantification by two "NOT EXISTS" clauses, applying the mathematical equivalence between $\forall x \exists y : p(x, y)$ and $\neg\exists x \neg\exists y : p(x, y)$, where p is a predicate involving variables x and y:

```
Q3: SELECT DISTINCT s#, color
    FROM   supplies AS s1, parts AS p1
    WHERE  NOT EXISTS (
             SELECT *
             FROM   parts AS p2
             WHERE  p2.color = p1.color AND
                    NOT EXISTS (
                      SELECT *
                      FROM   supplies AS s2
                      WHERE  s2.p# = p2.p# AND
                             s2.s# = s1.s#))
```

A direct translation of this query asks for each supplier and color whether there is no part of the same color that is not supplied by the supplier. We use the keyword DISTINCT in the outermost SELECT clause to remove duplicates from the result. Otherwise, we would get the same (*s#*, *color*) value combination as many times as there are parts of the same color in *parts*.

Clearly, the query using a special syntax for the set containment problem is more concise and hence (likely) less error-prone to formulate than the query based on existential quantifications. Furthermore, it is not simple to devise a query-rewriting algorithm for a query optimizer that is able to detect those existential quantification constructs that can be replaced by a (great) divide operator. Only if the appropriate joins between inner and outer query are present does the query solve a real set containment problem.

[5]For tables r_1 and r_2 with schemas $R_1(a, b, c)$ and $R_2(b, c)$, respectively, we would use a query like `SELECT a FROM r1 DIVIDE BY r2 ON r1.b = r2.b AND r1.c = r2.c`.

5 Algebraic Laws

Some of the algebraic laws discussed in this section are based on the notion of a *partitioned* relation. We use the following notations for partitions:

- r'_i and r''_i denote nonempty *horizontal* partitions of relation r_i such that $r'_i \cup r''_i = r_i$, where $i \in \{1, 2\}$, that is, we define a decomposition of r_i's *tuples*. The two partitions may actually be different relations. We just express by this notation that two relations have the same schema.

- r^*_i and r^{**}_i denote relations that conform to the schemas of the *vertical* partitions R^*_i and R^{**}_i of R_i, respectively, such that $R^*_i \cup R^{**}_i = R_i$, where $i \in \{1, 2\}$. Hence, we define a decomposition of R_i's *attributes*.

For the laws that follow, we will indicate when we require partitions to be disjoint or not. The proofs of the laws and theorems can be found in Appendix B.

Before we present the laws, we state two theorems that emphasize that this binary operator is clearly asymmetric.

THEOREM 2: *Small divide is non-commutative, that is,* $r_1 \div r_2 \neq r_2 \div r_1$ *for relations* r_1 *and* r_2.

THEOREM 3: *Small divide is non-associative, that is,* $r_1 \div (r_2 \div r_3) \neq (r_1 \div r_2) \div r_3$ *for nonempty relations* r_1, r_2, *and* r_3.

5.1 Algebraic Laws for the Small Divide

5.1.1 Union

When the *divisor* r_2 is decomposed into horizontal partitions then one can divide by these divisors separately:

LAW 1: $r_1 \div (r'_2 \cup r''_2) = (r_1 \ltimes (r_1 \div r'_2)) \div r''_2$.

This law holds also for overlapping divisor partitions, as illustrated in the example in Figure 4. In this example, the r'_2 and r''_2 have one tuple in common with value $b = 3$. The resulting relation r_3 is the same if the table (a) is divided by the union of tables (c) and (d) compared to dividing (f) by (d).

It can help an RDBMS to employ pipeline parallelism as follows. Suppose, r_1 is grouped on A. We can employ efficient group-preserving algorithms for the inner small divide $r_1 \div r'_2$ as well as the semi-join and deliver the result as the dividend to the outer small divide, which can be realized by a group-preserving algorithm itself.

a	b
1	1
1	4
2	1
2	2
2	3
2	4
3	1
3	3
3	4
4	1
4	3

(a) r_1

b
1
3
4

(b) r_2

b
1
3

(c) r_2'

b
3
4

(d) r_2''

a
2
3
4

(e) $r_1 \div r_2'$

a	b
2	1
2	2
2	3
2	4
3	1
3	3
3	4
4	1
4	3

(f) $r_1 \ltimes (r_1 \div r_2')$

a
2
3

(g) r_3

Figure 4. An example for Law 1

a	b
1	1
1	2
1	3

(a) r_1'

a	b
1	2
1	4

(b) r_1''

b
1
4

(c) r_2

Figure 5. An example where the precondition of Law 2 is not fulfilled

When we decompose the *dividend* horizontally instead of the divisor, we must take care of the situation sketched in Figure 5. There is a quotient candidate value ($a = 1$) whose tuples are dispersed across the dividend relations but none of the groups contains *all* values of the divisor. However, the union of the groups does. In other words, $r_1' \div r_2 = \emptyset$ and $r_1'' \div r_2 = \emptyset$ but $(r_1' \cup r_1'') \div r_2 \neq \emptyset$. We have to exclude this situation in the precondition of Law 2. Formally, the following precondition must hold:

$$\begin{aligned} c_1(r_1', r_1'') \equiv \forall a \in \pi_A(r_1') \cap \pi_A(r_1'') : \\ r_2 \subseteq \pi_B\left(\sigma_{A=a}\left(r_1'\right)\right) \vee \\ r_2 \subseteq \pi_B\left(\sigma_{A=a}\left(r_1''\right)\right) \vee \\ r_2 \not\subseteq \pi_B\left(\sigma_{A=a}\left(r_1'\right) \cup \sigma_{A=a}\left(r_1''\right)\right) \end{aligned}$$

LAW 2: If condition $c_1\left(r_1', r_1''\right)$ is true then $\left(r_1' \cup r_1''\right) \div r_2 = \left(r_1' \div r_2\right) \cup \left(r_1'' \div r_2\right)$.

Since testing condition c_1 can be expensive, an RDBMS may use a stricter condition c_2 that is easier to check:

$$c_2(r_1', r_1'') \equiv \pi_A\left(r_1'\right) \cap \pi_A\left(r_1''\right) = \emptyset.$$

It can be shown easily that for any relations $r_1 = r_1' \cup r_1''$ and r_2 as defined before, if c_2 holds then also c_1 holds. By using condition c_2 instead of c_1 with Law 2, an RDBMS can parallelize a query execution with degree 2 as follows. Suppose that the query execution engine can access the data in table r_1 via an index on A. We can employ two parallel scans on table r_1: one that starts with the lowest value of A and scans the leaves of the index in ascending order of A and another that starts with the highest value of A and retrieves data in descending order of A. Both scans stop as soon as they encounter the same value for A. Exactly one of them has to process the entire last group. Higher degrees of parallelism can be achieved by partitioning r_1 into $n > 2$ partitions.

5.1.2 Selection

Let $p(X)$ denote a predicate involving only elements of a set of attributes X. Since only r_1 contains the attribute set A, we can state the following "selection push-down" law:

LAW 3: $\sigma_{p(A)}\left(r_1 \div r_2\right) = \sigma_{p(A)}\left(r_1\right) \div r_2$.

For a predicate that involves only attributes in B, the following "replicate-selection" law holds:

LAW 4: $r_1 \div \sigma_{p(B)}\left(r_2\right) = \sigma_{p(B)}\left(r_1\right) \div \sigma_{p(B)}\left(r_2\right)$.

As a third example of selection conditions, we will now analyze the case where there's a restriction specified on dividend attributes in B, only.

EXAMPLE 1:

$$\begin{aligned} \sigma_{p(B)}\left(r_1\right) \div r_2 = \left(\sigma_{p(B)}\left(r_1\right) \div \sigma_{p(B)}\left(r_2\right)\right) - \\ \pi_A\left(\pi_A\left(r_1\right) \times \sigma_{\neg p(B)}\left(r_2\right)\right). \end{aligned}$$

This expression is very similar to Law 4. We only have to take care of the situation where $\sigma_{\neg p(B)}\left(r_2\right) \neq \emptyset$. In this case, the expression $\sigma_{p(B)}\left(r_1\right) \div r_2$ is equal to the empty set because no dividend tuple has a value of B that can match a tuple in $\sigma_{\neg p(B)}\left(r_2\right)$. Hence, if $\sigma_{\neg p(B)}\left(r_2\right)$ contains at least one tuple, we can enforce that the result relation be empty by simply removing all A values in r_1 from the quotient candidates in $\sigma_{p(B)}\left(r_1\right) \div \sigma_{p(B)}\left(r_2\right)$. The Cartesian product is merely used to "switch" $\pi_A\left(r_1\right)$ "on or off."[6]

Figure 6 illustrates the example and exhibits the intermediate results in detail. The predicate on the B columns is defined as $b < 3$. Note that the result tables (e) and (i) are both empty since table (h) is nonempty.

To make our argumentation clearer, we could rewrite our expression as follows: Since our equivalence represents a rather extreme case, we do not state it as a law but leave it as an example. □

[6] Of course, it would suffice to combine $\pi_A(r_1)$ with only a single tuple of $\sigma_{\neg p(B)}\left(r_2\right)$ by the Cartesian product.

a	b
1	1
1	4
2	1
2	2
2	3
2	4
3	1
3	3
3	4
4	1
4	3

(a) r_1

a	b
1	1
2	1
2	2
3	1
4	1

(b) $\sigma_{b<3}(r_1)$

b
1
3
4

(c) r_2

b
1

(d) $\sigma_{b<3}(r_2)$

a

(e) $\sigma_{b<3}(r_1) \div r_2$

a
1
2
3
4

(f) $\sigma_{b<3}(r_1) \div \sigma_{b<3}(r_2)$

a	b
1	3
1	4
2	3
2	4
3	3
3	4
4	3
4	4

(g) $\pi_a(r_1) \times \sigma_{b\geq 3}(r_2)$

a
1
2
3
4

(h) $\pi_a(\pi_a(r_1) \times \sigma_{b\geq 3}(r_2))$

a

(i) $(\sigma_{b<3}(r_1) \div \sigma_{b<3}(r_2)) - \pi_a(\pi_a(r_1) \times \sigma_{b\geq 3}(r_2))$

Figure 6. An illustration for Example 1

5.1.3 Intersection

We can push small divide into intersections of dividend relations.

LAW 5: $(r_1' \cap r_1'') \div r_2 = (r_1' \div r_2) \cap (r_1'' \div r_2)$.

5.1.4 Difference

The following law can be used when we perform two restricted scans over the *same* dividend relation where both restrictions are defined *only* on the attributes in A. For example, $r_1' = \sigma_{a>10}(r_1)$ and $r_1'' = \sigma_{a>20}(r_1)$. In this case, we can push small divide into a difference of the dividend relations:

LAW 6: If $r_1' = \sigma_{p'(A)}(r_1) \supseteq \sigma_{p''(A)}(r_1) = r_1''$ then $(r_1' - r_1'') \div r_2 = (r_1' \div r_2) - (r_1'' \div r_2)$.

For a similar law, we require as precondition that $\pi_A(r_1')$ and $\pi_A(r_1'')$ are disjoint.[7]

LAW 7: If $\pi_A(r_1') \cap \pi_A(r_1'') = \emptyset$ then $(r_1' \div r_2) - (r_1'' \div r_2) = r_1' \div r_2$.

Clearly, this law can save a lot of resources of an RDBMS if the computation of $r_1'' \div r_2$ would be expensive. For example, suppose that A consists of a single integer attribute with values $[1..10^6]$ and the query is $(\sigma_{a\leq 10}(r_1) \div r_2) - (\sigma_{a>10}(r_1) \div r_2)$. Computing only the first part of the difference is inexpensive.

[7] This is not the weakest precondition. For the the law to hold, it would suffice to require that $\forall a \in \sigma_{A=a}(\pi_A(r_1') \cup \pi_A(r_1''))$: $r_2 \subseteq \sigma_{A=a}(\pi_A(r_1')) \vee r_2 \subseteq \sigma_{A=a}(\pi_A(r_1'')) \vee r_2 \not\subseteq \sigma_{A=a}(\pi_A(r_1') \cup \pi_A(r_1''))$. However, we prove the law only for the stronger precondition $\pi_A(r_1') \cap \pi_A(r_1'') = \emptyset$.

a_1
1
2

(a) r_1^*

a_2	b
1	1
1	2
1	3
2	1
2	3
3	2
3	3

(b) r_1^{**}

b
2
3

(c) r_2

a_1	a_2	b
1	1	1
1	1	2
1	1	3
1	2	1
1	2	3
1	3	2
1	3	3
2	1	1
2	1	2
2	1	3
2	2	1
2	2	3
2	3	2
2	3	3

(d) $r_1^* \times r_1^{**}$

a_2
1
3

(e) $r_1^{**} \div r_2$

a_1	a_2
1	1
1	3
2	1
2	3

(f) r_3

Figure 7. An example for Law 8

5.1.5 Cartesian Product

Let A_1 and A_2 be disjoint subsets of the attribute set A such that $A_1 \cup A_2 = A$. Let r_1^* be a relation with schema $R_1^*(A_1)$ and r_1^{**} be a relation with schema $R_1^{**}(A_2 \cup B)$. As usual, let $R_2(B)$ be the schema of the divisor r_2. Then it suffices to apply the small divide only to some of the attributes of the dividend:

LAW 8: $(r_1^* \times r_1^{**}) \div r_2 = r_1^* \times (r_1^{**} \div r_2)$.

Figure 7 illustrates Law 8 with an example. The law can help when the query optimizer finds that a predicate θ of a theta-join $\bowtie_\theta$ is always true since $\bowtie_{true} \equiv \times$.

Let B_1 and B_2 be disjoint nonempty subsets of the attribute set B such that $B_1 \cup B_2 = B$. Let r_1^* be a relation with schema $R_1^*(A \cup B_1)$ and r_1^{**} be a relation with schema $R_1^{**}(B_2)$. Again, let $R_2(B)$ be the schema of the divisor r_2. Then, we can state the following

LAW 9: If $\pi_{B_2}(r_2) \subseteq r_1^{**}$ then $(r_1^* \times r_1^{**}) \div r_2 = r_1^* \div \pi_{B_1}(r_2)$.

Figure 8 illustrates Law 9 with an example. All intermediate relations are shown. Note that the Cartesian product (d) does not necessarily have to be materialized by an RDBMS provided that the implementation of the subsequent small divide can cope with pipelined input. The same holds for the Cartesian product on the left hand side of Law 8 that was illustrated in 7(d).

EXAMPLE 2: With the help of Law 9 we can prove that $(r_1 \times s) \div (r_2 \times s) = r_1 \div r_2$. Let $B = B_1 \cup B_2$. We

a	b_1
1	1
1	2
1	3
2	2
2	3
3	1
3	3
3	4

(a) r_1^*

b_2
1
2

(b) r_1^{**}

b_1	b_2
1	2
3	1
3	2

(c) r_2

a	b_1	b_2
1	1	1
1	1	2
1	2	1
1	2	2
1	3	1
1	3	2
2	2	1
2	2	2
2	3	1
2	3	2
3	1	1
3	1	2
3	3	1
3	3	2
3	4	1
3	4	2

(d) $r_1^* \times r_1^{**}$

b_1
1
3

(e) $\pi_{B_1}(r_2)$

b_2
1
2

(f) $\pi_{R_1^{**}}(r_2)$

a
1
3

(g) r_3

Figure 8. An example for Law 9

have $R_1^*(A \cup B_1)$, $R_1^{**}(B_2)$, $R_2^*(B_1)$, $R_2^{**}(B_2)$ and thus $R_1(A \cup B_1 \cup B_2)$ as the dividend schema and $R_2(B_1 \cup B_2)$ as the divisor schema. We define $s = r_1^{**} = r_2^{**}$. The condition $r_1^{**} \subseteq \pi_{R_1^{**}}(r_2)$ is fulfilled since $r_1^{**} = r_2^{**} = \pi_{R_2^{**}}(r_2) = \pi_{R_1^{**}}(r_2)$. Hence, we have

$$
\begin{aligned}
& (r_1^* \times s) \div (r_2^* \times s) \\
&= (r_1^* \times r_1^{**}) \div (r_2^* \times r_2^{**}) \qquad \text{(Definition of } s\text{)} \\
&= (r_1^* \times r_1^{**}) \div r_2 \qquad \text{(Definition of } R_2\text{)} \\
&= r_1^* \div \pi_{B_1}(r_2) \qquad \text{(Law 9)} \\
&= r_1^* \div r_2^* \qquad \text{(Definition of } R_2\text{)}
\end{aligned}
$$

5.1.6 Join

Join, like small divide, is a derived operator. When a small divide operator occurs together with a join operator in an expression, it may be beneficial for the execution strategy of an RDBMS to rewrite the join operator and subsequently apply algebraic laws to rewrite the result in combination with small divide. The laws involving the selection operator in Section 5.1.2 as well as the laws concerning the Cartesian product in Section 5.1.5 can be used to rewrite expressions involving join and small divide, since $r \bowtie_\theta s = \sigma_\theta(r \times s)$, where $\bowtie_\theta$ is a theta-join with the condition θ. The following example illustrates such a rewrite.

EXAMPLE 3: Let r_1^*, r_1^{**}, and r_2 be relations with schemas $R_1^*(a, b_1)$, $R_1^{**}(b_2)$, and $R_2(b_1, b_2)$, respectively. Furthermore, let $r_1^{**}.b_2$ be a unique attribute and let $r_2.b_2$ be a foreign key that references r_1^{**}, that is, $\pi_{b_2}(r_2) \subseteq r_1^{**}$. Suppose, we want to compute relation $r_3 = (r_1^* \bowtie_{b_1 < b_2} r_1^{**}) \div$

a	b_1
1	1
1	2
1	3
2	2
2	3
3	1
3	3
3	4

(a) r_1^*

b_2
1
2
4

(b) r_1^{**}

b_1	b_2
1	4
3	4

(c) r_2

a	b_1	b_2
1	1	2
1	1	4
1	2	4
1	3	4
2	2	4
2	3	4
3	1	2
3	1	4
3	3	4

(d) $r_1^* \bowtie_{b_1 < b_2} r_1^{**}$

b_1
1
3

(e) $\pi_{b_1}(\sigma_{b_1 < b_2}(r_2))$

a
1
3

(f) r_3

Figure 9. An illustration of Example 3

r_2. We can derive the following expressions:

$$
\begin{aligned}
r_3 &= (r_1^* \bowtie_{b_1 < b_2} r_1^{**}) \div r_2 \\
&= \sigma_{b_1 < b_2}(r_1^* \times r_1^{**}) \div r_2 \ \text{(Definition of theta-join)} \\
&= (\sigma_{b_1 < b_2}(r_1^* \times r_1^{**}) \div \sigma_{b_1 < b_2}(r_2)) - \\
&\quad \pi_a(\pi_a(r_1^* \times r_1^{**}) \times \sigma_{b_1 \geq b_2}(r_2)) \quad \text{(Example 1)} \\
&= ((r_1^* \times r_1^{**}) \div \sigma_{b_1 < b_2}(r_2)) - \\
&\quad \pi_a(\pi_a(r_1^* \times r_1^{**}) \times \sigma_{b_1 \geq b_2}(r_2)) \qquad \text{(Law 4)} \\
&= (r_1^* \div \pi_{b_1}(\sigma_{b_1 < b_2}(r_2))) - \\
&\quad \pi_a(\pi_a(r_1^* \times r_1^{**}) \times \sigma_{b_1 \geq b_2}(r_2)) \qquad \text{(Law 9)} \\
&= (r_1^* \div \pi_{b_1}(\sigma_{b_1 < b_2}(r_2))) - \\
&\quad \pi_a(\pi_a(r_1^*) \times \sigma_{b_1 \geq b_2}(r_2)) \\
&\qquad \text{(since } a \in R_1^* \text{ but } a \notin R_1^{**}\text{)}
\end{aligned}
$$

Note that the term $\pi_a(r_1^*) \times \sigma_{b_1 \geq b_2}(r_2)$ is merely used to test if $\sigma_{b_1 \geq b_2}(r_2)$ contains at least one tuple. If yes, r_3 is an empty relation because $\pi_a(r_1^*)$ represents all a values in r_1^* and removing these values from the quotient $r_1^* \div \pi_{b_1}(\sigma_{b_1 < b_2}(r_2))$ would leave no tuples. Otherwise, r_3 is simply $r_1^* \div \pi_{b_1}(\sigma_{b_1 < b_2}(r_2))$. Figure 9 sketches some intermediate results that occur during the computation of our example expression.

An RDBMS might be able to execute a plan based on this expression more efficiently than a plan based on the original expression because no join between r_1^* and r_1^{**} is required. Such a situation occurs, for instance, when there is no index available on $r_1^*.b_1$ and no index on $r_1^{**}.b_2$, but when there are two indexes defined on the columns b_1 and b_2 of table r_2, respectively. □

Let us focus on a special type of join: the semi-join. Let r_3 be a relation with schema $R_3(A)$. Then we can state the following

LAW 10: $(r_1 \div r_2) \ltimes r_3 = (r_1 \ltimes r_3) \div r_2$.

This law can help an RDBMS if r_3 has few tuples and r_1 and r_2 have many tuples. It may be cheaper to keep r_3 in memory and to compute the semi-join in one scan over r_1,

a	x
1	1
1	2
1	3
2	1
2	3
3	1
3	3
3	4

(a) r_0

a	b
1	6
2	4
3	8

(b) $r_1 = {}_a\gamma_{\text{sum}(x)\to b}(r_0)$

b
4

(c) r_2

a	b
2	4

(d) $r_1 \ltimes r_2$

a
2

(e) $\pi_A(r_1 \ltimes r_2)$

Figure 10. An example for Law 11

x	b
1	1
1	2
1	3
2	1
2	3
3	1
3	3
3	4

(a) r_0

a	b
6	1
1	2
6	3
3	4

(b) $r_1 = {}_b\gamma_{\text{sum}(x)\to a}(r_0)$

b
1
3

(c) r_2

a	b
6	1
6	3

(d) $r_1 \ltimes r_2$

a
6

(e) $\pi_A(r_1 \ltimes r_2)$

Figure 11. An example for Law 12

especially if the join is highly selective and removes many tuples from r_1. Then, the small divide of the join result with r_2 is likely to be cheap.

5.1.7 Grouping

We consider two special cases involving the grouping operator. Concerning the first special case, let r_0 be a relation with schema $R_0(A \cup X)$ for some nonempty attribute set X. Let $r_1 = {}_A\gamma_{f(X)\to B}(r_0)$, where f is an aggregate function and its result is assigned to the attributes in B.[8] In other words, each quotient candidate group of the dividend consists of a single tuple. Hence, in order to find a quotient, the divisor cannot have more than one tuple. For this special case, we can formulate

LAW 11: $$r_1 \div r_2 = \begin{cases} r_1 \\ \quad \text{if } \sigma_{c=0}\left(\gamma_{\text{count}(B)\to c}\left(r_2\right)\right) \neq \emptyset, \\ \pi_A\left(r_1 \ltimes r_2\right) \\ \quad \text{if } \sigma_{c=1}\left(\gamma_{\text{count}(B)\to c}\left(r_2\right)\right) \neq \emptyset, \\ \quad \text{and} \\ \emptyset \text{ otherwise.} \end{cases}$$

Figure 10 illustrates an example for this law. Here, the aggregation operator computes the sum of the x values for each group of b in table r_0. This value is used as the new attribute a in r_1. Since each group formed defined by b has a single tuple the table (e) constitutes the result.

Now, let us consider another special case. Let r_0 be a relation with schema $R_0(X \cup B)$ for some nonempty attribute set X. Let $r_1 = {}_B\gamma_{f(X)\to A}(r_0)$, where f is an aggregate function and its result is assigned to the attributes in A.[8] In other words, each divisor attribute value B of the dividend occurs in a single tuple, that is, the groups defined by B have size one. Furthermore, let $r_2.B$ be a foreign key referencing $r_1.B$, that is, $r_2.B \subseteq \pi_B\left(r_1\right)$.

[8] The assignment $f(X) \to B$ is a simplification. In general, f is a list of aggregate functions $f_1, \ldots, f_n$, where $n = |B|$, such that $f(X) = (f_1(e_1(X)), \ldots, f_n(e_n(X))) = (b_1, \ldots, b_n) = B$ and $e_i(X)$ is an arithmetic expression using attributes of X, for example, $e_5 = 7x_3 - \sqrt{x_5}$. The set X may have any number of attributes, it need not be equal to B.

Hence, there can be at most one dividend tuple for each B value. We simply have to check if $\pi_A(r_1 \ltimes r_2)$ contains a single value. If it does, then this value is the quotient. Otherwise, there is no quotient.

LAW 12: $$r_1 \div r_2 = \begin{cases} \pi_A\left(r_1 \ltimes r_2\right) \\ \quad \text{if } \sigma_{c=1}\left(\gamma_{\text{count}(A)\to c}\left(\right.\right. \\ \qquad \left.\left.\pi_A\left(r_1 \ltimes r_2\right)\right)\right) \neq \emptyset, \text{ and} \\ \emptyset \text{ otherwise.} \end{cases}$$

Figure 11 illustrates an example for this law. Since table (e) contains a single tuple, this table also constitutes the quotient.

The two laws involving the grouping operator can improve the query execution time considerably because the small divide operation is replaced by a single join operation and a projection on the join result. However, since Laws 11 and 12 have rather restrictive prerequisites, we believe that their implementation is beneficial only in special purpose RDBMS.

5.2 Algebraic Laws for the Great Divide

We have identified several laws for the great divide operator $\div^*$. In the following, we show some of the laws that we consider as important.

5.2.1 Union

When the *divisor* r_2 is decomposed into horizontal partitions then one can divide by these divisors separately:

LAW 13: If $\pi_C\left(r_2'\right) \cap \pi_C\left(r_2''\right) = \emptyset$ then $r_1 \div^* \left(r_2' \cup r_2''\right) = \left(r_1 \div^* r_2'\right) \cup \left(r_1 \div^* r_2''\right)$.

This law allows to parallelize the execution of a query. Suppose that the dividend r_1 is replicated on n nodes of a query execution engine and that the divisor is equally distributed according to a hash function on $r_2.C$ across the nodes. Then it is possible to reduce the execution time to $\frac{1}{n}$ of the original time provided that the great divide execution is considerably more expensive than the final union/merge operator plus the cost for data shipping to and from the nodes.

5.2.2 Selection

The following law is the same as Law 3 for the small divide operator.

LAW 14: $\sigma_{p(A)}\,(r_1 \div^* r_2) = \sigma_{p(A)}\,(r_1) \div^* r_2$.

A similar "predicate push-down" law holds for attribute C of the divisor relation:

LAW 15: $\sigma_{p(C)}\,(r_1 \div^* r_2) = r_1 \div^* \sigma_{p(C)}\,(r_2)$.

The following law is the same as Law 4 for the small divide:

LAW 16: $r_1 \div^* \sigma_{p(B)}\,(r_2) = \sigma_{p(B)}\,(r_1) \div^* \sigma_{p(B)}\,(r_2)$.

5.2.3 Cartesian Product

The following law is the same as Law 8 for the small divide. It is useful for expressions involving joins when combined with Laws 15 and 16.

LAW 17: $(r_1^* \times r_1^{**}) \div^* r_2 = r_1^* \times (r_1^{**} \div^* r_2)$.

5.2.4 Join

The following example illustrates how an expression involving great divide and theta-join can be rewritten using the laws discussed before.

EXAMPLE 4: Let r_1^*, r_1^{**}, and r_2 be relations with schemas $R_1^*(a_1)$, $R_1^{**}(a_2, b_1)$, and $R_2(b_1, b_2)$, respectively. We can derive the following expressions:

$$
\begin{aligned}
& r_1^* \bowtie_{a_1=a_2} (r_1^{**} \div^* r_2) \\
&= \sigma_{a_1=a_2}\,(r_1^* \times (r_1^{**} \div^* r_2)) && \text{(Def. of theta-join)} \\
&= \sigma_{a_1=a_2}\,((r_1^* \times r_1^{**}) \div^* r_2) && \text{(Law 17)} \\
&= \sigma_{a_1=a_2}\,(r_1^* \times r_1^{**}) \div^* r_2 && \text{(Law 14)} \\
&= (r_1^* \bowtie_{a_1=a_2} r_1^{**}) \div^* r_2 && \text{(Definition of theta-join)}
\end{aligned}
$$

Suppose that an index is available on $r_1^*.a_1$ or on $r_1^{**}.a_2$. The join $r_1^* \bowtie_{a_1=a_2} r_1^{**}$ in the last expression can then be computed very efficiently. If this join has a high selectivity, it is possible that much fewer dividend groups of b values have to be tested against r_2 in the last expression compared to the first expression. $\square$

6 Related Work

An interesting theoretical result about the small divide operator has recently been published [25]. It justifies the efforts made by previous work on implementing small divide and set equality joins as efficient *special purpose operators*, which can achieve a time complexity of $O(n \log n)$ for algorithms based on sorting and counting. They prove that any expression of the small divide operator in the relational algebra with union, difference, projection, selection, and equi-joins, must produce intermediate results of quadratic size.[9]

Set containment join is considered an important operator for queries involving set-valued attributes [18, 20, 28, 30, 29, 32, 33, 41]. For example, set containment test operations have been used for optimizing a workload of continuous queries, in particular for checking if one query is a subquery of another. For instance, Chen and DeWitt [8] suggested an algorithm that re-groups continuous queries to maintain a close-to-optimal global query execution plan.

Another example of set containment joins is content-based retrieval using a search engine in document databases, where a huge set of documents is tested against a set of keywords that all have to appear in the document.

We have already discussed the area of data mining as another potential application area in Section 3.

The small divide operator has been studied in the context of *fuzzy relations*, for example, [6]. In a fuzzy relation, the tuples are weighted by a number between 0 and 1. One interpretation of an extended division operator for fuzzy relations, the *fuzzy quotient operator* [40], is based on one of several relaxed versions of the universal quantifier, called "almost all," which is realized by a so-called *ordered weighted average operator*. The fuzzy quotient operator produces those values of $a \in \pi_A(r_1)$, where for "almost all" elements $b \in \pi_B(r_2)$ the tuple $((a) \times (b))$ is in r_1 for some fuzzy relations r_1 and r_2 with schemas $R_1(A \cup B)$ and $R_2(B)$, respectively. Other interpretations of a "fuzzy" version for division are discussed, for example, in [5, 4].

Carlis proposed a generalization of the division operator, called *HAS* [7]. He argues that "division is misnamed" because there are more operators $\circ$ than division ($\div$) that fulfill the equation $(r_1 \times r_2) \circ r_2 = r_1$. He further claims that division is "hard to understand" because, among other arguments, "division is the only algebra operation that gives students any trouble." Finally, he writes that division is "insufficient" because it is not flexible enough, it allows only queries of the form "find the sets that contain *all* elements of a given set" but it does not help for queries asking for sets that contain, for example, at least five elements of a given set.

The HAS operator involves three relations: r_1 contains entities about which we want the answer if it qualifies in the result, r_2 contains entities that are used for the qualification, and r_3 contains the relationships between the entities in r_1 and r_2. For example, in the supplier-parts database

[9] Their main, more general, result is to show that any relational algebra expression that never produces intermediate results of quadratic size, will produce only intermediate results of linear size.

mentioned in Section 4, r_1 = *suppliers*, r_2 = *parts*, and r_3 = *supplies*. In addition, the HAS operator uses a combination of six "adverbs," called *associations*, to describe the qualification: *strictly more than*, *strictly less than*, *some of but not all plus something else*, *exactly*, *none of plus something else*, and *none at all*. There are $2^6 - 1 = 63$ possible combinations to choose between one and six associations for a specific HAS operator. Such a combination is considered as a disjunction of the participating associations.

We illustrate the algebra syntax used in [7] by showing how the small divide can be expressed by the HAS operator using one of the 63 association combinations: r_1 VIA r_3 HAS (*exactly* or *strictly more than*) OF r_2. The combination "*exactly* or *strictly more than*" is equivalent to the adverb "at least," typically used to describe division.

7 Conclusions

We have presented equivalences of the relational algebra for two important operators that realize a universal quantification, called small and great divide. The latter is a natural extension of the classic small divide operator that was introduced by Codd. The algebraic laws can serve as logical rewrite rules within the optimizer of an RDBMS that provides an implementation of small or great divide in the execution engine. To achieve efficiency for universal quantification queries, division operators *must* be implemented as first-class operators, as it was recently proven in [25].

Until today, relational division operators have not been implemented in any commercial RDBMS. However, with these operators, data-intensive data mining primitives like frequent itemset discovery or simple text searches using conjunctive queries can be formulated intuitively and be coupled more closely with an RDBMS. Hence, such "for-all" queries enjoy an optimization according to the current data characteristics and can be processed efficiently by these special-purpose operators. We do not claim that the laws presented in this paper constitute the only relevant ruleset. Nevertheless, we believe that several of our algebraic equivalences are necessary to enable an effective optimization of queries that use the small or great divide as a first-class operator.

Clearly, logical query rewriting is only one aspect of the query optimization problem. The mapping of logical operators to physical operators is another issue. We have recently implemented a collection of physical great divide operators into a Java query execution engine prototype based on the class library XXL [3]. A description of several great divide algorithms together with cost estimations based on input data characteristics (such as grouped or sorted columns in the dividend and divisor) was given in [36]. Future work will assess the effectiveness of the algebraic laws when implemented as transformation rules in a query optimizer. Besides such engineering problems, it is interesting to study further data-intensive applications with an intrinsic universal quantification problem besides frequent itemset discovery.

Acknowledgments

We thank Rakesh Agrawal for invaluable suggestions, Bernhard Mitschang for starting the project and guiding our work at Stuttgart, where much of this work was done, and Leonard Shapiro for the inspiration.

References

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[2] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *SIGMOD, Washington DC, USA*, pages 207–216, May 1993.

[3] J. V. d. Bercken, B. Blohsfeld, J.-P. Dittrich, J. Krämer, T. Schäfer, M. Schneider, and B. Seeger. XXL—A library approach to supporting efficient implementations of advanced database queries. In *VLDB, Rome, Italy*, pages 39–48, September 2001.

[4] P. Bosc. On the primitivity of the division of fuzzy relations. In *SAC, San Jose, California, USA*, pages 197–201, February–March 1997.

[5] P. Bosc, D. Dubois, O. Pivert, and H. Prade. Flexible queries in relational databases—the example of the division operator. *Theoretical Computer Science*, 171(1–2):281–302, 1997.

[6] B. P. Buckles and F. E. Petry. A fuzzy representation of data for relational databases. *Fuzzy Sets and Systems*, 7(3), May 1982. 213–226.

[7] J. V. Carlis. HAS, a relational algebra operator or divide is not enough to conquer. In *ICDE, Los Angeles, California, USA*, pages 254–261, February 1986.

[8] J. Chen and D. DeWitt. Dynamic re-grouping of continuous queries. In *VLDB, Hong Kong, China*, pages 430–441, August 2002.

[9] J. Claußen, A. Kemper, G. Moerkotte, and K. Peithner. Optimizing queries with universal quantification in object-oriented and object-relational databases. In *VLDB, Athens, Greece*, pages 286–295, August 1997.

[10] E. Codd. Relational completeness of database sublanguages. In R. Rustin, editor, *Courant Computer Science Symposium 6: Database Systems*, pages 65–98. Prentice-Hall, 1972.

[11] H. Darwen and C. Date. Into the great divide. In C. Date and H. Darwen, editors, *Relational Database: Writings 1989–1991*, pages 155–168. Addison-Wesley, Reading, Massachusetts, USA, 1992.

[12] R. Demolombe. Generalized division for relational algebraic language. *Information Processing Letters*, 14(4):174–178, 1982.

[13] H. Garcia-Molina, J. Ullman, and J. Widom. *Database Systems—The Complete Book*. Prentice-Hall, 2002.

[14] G. Graefe. Relational division: Four algorithms and their performance. In *ICDE, Los Angeles, California, USA*, pages 94–101, February 1989.

[15] G. Graefe. The Cascades framework for query optimization. *BTCDE*, 18(3):19–29, September 1995.

[16] G. Graefe and R. Cole. Fast algorithms for universal quantification in large databases. *TODS*, 20(2):187–236, 1995.

[17] T. Griffin and B. Kumar. Algebraic change propagation for semijoin and outerjoin queries. *SIGMOD Record*, 27(3):22–27, March 1998.

[18] S. Helmer. *Performance Enhancements for Advanced Database Management Systems*. PhD thesis, University of Mannheim, Germany, December 2000.

[19] S. Helmer and G. Moerkotte. Evaluation of main memory join algorithms for joins with set comparison join predicates. In *VLDB, Athens, Greece*, pages 386–395, August 1997.

[20] S. Helmer and G. Moerkotte. Compiling away set containment and intersection joins. Technical report, University of Mannheim, Germany, April 2002.

[21] M. Houtsma and A. Swami. Set-oriented data mining in relational databases. *DKE*, 17(3):245–262, December 1995.

[22] ISO/IEC. *Information Technology—Database Language—SQL—Part 2: Foundation (SQL/Foundation), Working Draft 9075-2:2003*, December 2002.

[23] ISO/IEC. *Information Technology—Database Language—SQL—Part 2: Framework (SQL/Framework), Working Draft 9075-2:2003*, December 2002.

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

[25] D. Leinders and J. V. den Bussche. On the complexity of division and set joins in the relational algebra. In *PODS, Baltimore, MD*, June 2005.

[26] G. M. Lohman. Grammar-like functional rules for representing query optimization alternatives. In *SIGMOD, Chicago, Illinois, USA*, pages 18–27, June 1988.

[27] D. Maier. *The Theory of Relational Databases*. Computer Science Press, 1983.

[28] N. Mamoulis. Efficient processing of joins on set-valued attributes. In *SIGMOD, San Diego, California, USA*, June 2003.

[29] S. Melnik and H. Garcia-Molina. Divide-and-conquer algorithm for computing set containment joins. In *EDBT, Prague, Czech Republic*, pages 427–444, March 2002.

[30] S. Melnik and H. Garcia-Molina. Adaptive algorithms for set containment joins. *TODS*, 28(1):56–99, March 2003.

[31] H. Pirahesh, J. M. Hellerstein, and W. Hasan. Extensible/rule based query rewrite optimization in Starburst. In *SIGMOD, San Diego, California, USA*, pages 39–48, June 1992.

[32] K. Ramasamy. *Efficient Storage and Query Processing of Set-valued Attributes*. PhD thesis, University of Wisconsin, Madison, Wisconsin, USA, 2002. 144 pages.

[33] K. Ramasamy, J. M. Patel, J. F. Naughton, and R. Kaushik. Set containment joins: The good, the bad and the ugly. In *VLDB, Cairo, Egypt*, pages 351–362, September 2000.

[34] R. Rantzau. Processing frequent itemset discovery queries by division and set containment join operators. In *DMKD, San Diego, California, USA*, June 2003.

[35] R. Rantzau and C. Mangold. Laws for rewriting queries containing division operators. In *Proceedings ICDE, Atlanta, Georgia, USA*, April 2006.

[36] R. Rantzau, L. Shapiro, B. Mitschang, and Q. Wang. Algorithms and applications for universal quantification in relational databases. *Information Systems*, 28(1):3–32, January 2003.

[37] S. Sarawagi, S. Thomas, and R. Agrawal. Integrating association rule mining with relational database systems: Alternatives and implications. In *SIGMOD, Seattle, Washington, USA*, pages 343–354, June 1998.

[38] A. Silberschatz, H. F. Korth, and S. Sudarshan. *Database System Concepts*. McGraw-Hill, fourth edition, 2001.

[39] S. Thomas and S. Chakravarthy. Performance evaluation and optimization of join queries for association rule mining. In *DaWaK, Florence, Italy*, pages 241–250, August–September 1999.

[40] R. R. Yager. Fuzzy quotient operators for fuzzy relational databases. In *IFES, Yokohama, Japan*, pages 289–296, November 1991.

[41] C. Zhang, J. Naughton, D. DeWitt, Q. Luo, and G. Lohman. On supporting containment queries in relational database management systems. In *SIGMOD, Santa Barbara, California, USA*, May 2001.

A Algebra Operators Used in this Paper

All of the operators in this paper, shown in the following table, have *set* semantics, that is, each input and output relation of an operator is a *set* of tuples. For a discussion of the difference between the set and bag/multi-set semantics of relational operators, see, for example, [13].

Operator	Name	Description
$\cup$	Set union	$r_1 \cup r_2 = \{t \mid t \in r_1 \vee t \in r_2\}$
$\cap$	Set intersection	$r_1 \cap r_2 = \{t \mid t \in r_1 \wedge t \in r_2\}$
$-$	Set difference	$r_1 - r_2 = \{t \mid t \in r_1 \wedge t \notin r_2\}$
$\times$	Cartesian product/ cross product	$r_1 \times r_2 = \{t_1 \circ t_2 \mid t_1 \in r_1 \wedge t_2 \in r_2\}$, where $\circ$ is the concatenation operator.
π_A	Projection	$\pi_A(r) = \{t.A \mid t \in r\}$, where A is a list of attributes, $\{A\}$ is the *set* of attributes in the *list* A, and $t.A$ is the concatenation of values from tuple t that appear in A.
σ_θ	Selection	$\sigma_\theta(r) = \{t \mid t \in r \wedge \theta(t)\}$, where θ is a condition.
$\bowtie_\theta$	Theta-join	$r_1 \bowtie_\theta r_2 = \sigma_\theta(r_1 \times r_2)$ and θ is a condition.
$\bowtie$	Natural join	$r_1 \bowtie r_2 = \pi_A(\sigma_\theta(r_1 \times r_2))$, where A is the set of attributes in the schema $R_1(r_1) \cup R_2(r_2)$, $\theta = \bigwedge_{i=1}^{n} r_1.a_i = r_2.a_i$, and $\{a_1, \ldots, a_n\}$ is the set of attributes appearing in the schema $R_1 \cap R_2$.
$\ltimes$	Left semi-join	$r_1 \ltimes r_2 = \pi_{[r_1]}(r_1 \bowtie r_2)$, where $[r_1]$ denotes the attributes of $R_1(r_1)$.
$\overline{\ltimes}$	Left anti-semi-join	$r_1 \overline{\ltimes} r_2 = r_1 - (r_1 \ltimes r_2)$.
$\sqsupset\!\bowtie$	Left outer join	$r_1 \sqsupset\!\bowtie r_2 = (r_1 \bowtie r_2) \cup \big((r_1 \overline{\ltimes} r_2) \times (\times_1^n (\mathit{NULL}))\big)$, where n is the number of attributes in schema $R_2(r_2)$ [17].
${}_G\gamma_F$	Grouping	G is a list of grouping attributes and F is a list of aggregation functions applied to some attribute values. Example: ${}_{a,d}\gamma_{\mathrm{sum}(b)\rightarrow total}(r_1)$ for some relation r_1 with schema $R_1(a,b,c,d)$ [13, 38].
$\div$	Small divide, Division	Example: $r_1 \div r_2 = r_3$ for some relations r_1, r_2, and r_3 with schemas $R_1(a,b,c)$, $R_2(b,c)$, and $R_3(a)$, respectively. Relation r_1 is called dividend, r_2 divisor, and r_3 quotient.
$\div^*$	Great divide, Generalized division, Set containment division	Example: $r_1 \div^* r_2 = r_3$ for some relations r_1, r_2, and r_3 with schemas $R_1(a,b,c)$, $R_2(b,c,d,e)$, and $R_3(a,d,e)$, respectively.
$\bowtie_\subseteq$	Set containment join	Example: $r_1 \bowtie_{b \subseteq c} r_2 = r_3$ for some relations r_1, r_2, and r_3 with schemas $R_1(a,b)$, $R_2(c,d,e)$, and $R_3(a,b,c,d,e)$, respectively, where b and c are set-valued attributes.

B Proofs

B.1 Lemmas

LEMMA 1: *Let X, Y, and Z be sets. Then, $X - Y = X - Z \Leftrightarrow X \cap Y = X \cap Z$.*

PROOF (LEMMA 1): We prove the lemma by deriving implications for both directions of the equivalence. First, we show that $X - Y = X - Z \Rightarrow X \cap Y = X \cap Z$:

$$\begin{aligned}
& t \in (X \cap Y) \\
\Leftrightarrow\ & t \in X \wedge t \in Y \\
\Leftrightarrow\ & (t \in X \wedge t \notin X) \vee (t \in X \wedge t \in Y) \\
\Leftrightarrow\ & t \in X \wedge (t \notin X \vee t \in Y) \\
\Leftrightarrow\ & t \in X \wedge \neg(t \in X \wedge t \notin Y) \\
\Leftrightarrow\ & t \in X \wedge \neg(t \in (X - Y)) \\
\Leftrightarrow\ & t \in X \wedge \neg(t \in (X - Z)) \text{ due to our assumption} \\
\Leftrightarrow\ & t \in X \wedge \neg(t \in X \wedge t \notin Z) \\
\Leftrightarrow\ & t \in X \wedge (t \notin X \vee t \in Z) \\
\Leftrightarrow\ & (t \in X \wedge t \notin X) \vee (t \in X \wedge t \in Z) \\
\Leftrightarrow\ & t \in X \wedge t \in Z \\
\Leftrightarrow\ & t \in (X \cap Z)
\end{aligned}$$

Next, we show that $X \cap Y = X \cap Z \Rightarrow X - Y = X - Z$:

$$\begin{aligned}
& t \in (X - Y) \\
\Leftrightarrow\ & t \in X \wedge t \notin Y \\
\Leftrightarrow\ & (t \in X \wedge t \notin Y) \vee (t \in X \wedge t \notin X) \\
\Leftrightarrow\ & (t \in X \vee (t \in X \wedge t \in X)) \wedge \\
& (t \notin Y \vee (t \in X \wedge t \notin X)) \\
\Leftrightarrow\ & t \in X \wedge (t \notin Y \vee (t \in X \wedge t \notin X)) \\
\Leftrightarrow\ & t \in X \wedge (t \in X \vee t \notin Y) \wedge (t \notin X \vee t \notin Y) \\
\Leftrightarrow\ & t \in X \wedge (t \notin X \vee t \notin Y) \\
\Leftrightarrow\ & t \in X \wedge \neg(t \in X \wedge t \in Y) \\
\Leftrightarrow\ & t \in X \wedge \neg(t \in (X \cap Y)) \\
\Leftrightarrow\ & t \in X \wedge \neg(t \in (X \cap Z)) \text{ due to our assumption} \\
\Leftrightarrow\ & t \in X \wedge \neg(t \in X \wedge t \in Z) \\
\Leftrightarrow\ & t \in X \wedge (t \notin X \vee t \notin Z) \\
\Leftrightarrow\ & (t \in X \wedge t \notin X) \vee (t \in X \wedge t \notin Z) \\
\Leftrightarrow\ & t \in X \wedge t \notin Z \\
\Leftrightarrow\ & t \in (X - Z)
\end{aligned}$$

□

LEMMA 2: *Let X_1, X_2, Y_1 and Y_2 be sets. If $X_1 \cap X_2 = \emptyset$ and $X_i \supseteq Y_i$ for $i \in \{1, 2\}$ then $(X_1 - Y_1) \cup (X_2 - Y_2) = (X_1 \cup X_2) - (Y_1 \cup Y_2)$.*

PROOF (LEMMA 2):

$$\begin{aligned}
& t \in (X_1 - Y_1) \cup (X_2 - Y_2) \\
\Leftrightarrow\ & (t \in X_1 \wedge t \notin Y_1) \vee (t \in X_2 \wedge t \notin Y_2) \\
\Leftrightarrow\ & (t \in X_1 \vee t \in X_2) \wedge (t \in X_1 \vee t \notin Y_2) \wedge \\
& (t \notin Y_1 \vee t \in X_2) \wedge (t \notin Y_1 \vee t \notin Y_2) \\
\Leftrightarrow\ & (t \in X_1 \vee t \in X_2) \wedge \neg(t \notin X_1 \wedge t \in Y_2) \wedge \\
& \neg(t \in Y_1 \wedge t \notin X_2) \wedge \neg(t \in Y_1 \wedge t \in Y_2) \\
\Leftrightarrow\ & t \in (X_1 \cup X_2) \wedge t \notin (Y_2 - X_1) \wedge \\
& t \notin (Y_1 - X_2) \wedge t \notin (Y_1 \cap Y_2) \\
\Leftrightarrow\ & t \in (X_1 \cup X_2) \wedge \\
& \neg(t \in (Y_2 - X_1) \vee t \in (Y_1 - X_2) \vee t \in (Y_1 \cap Y_2)) \\
\Leftrightarrow\ & t \in (X_1 \cup X_2) \wedge \\
& t \notin ((Y_2 - X_1) \cup (Y_1 - X_2) \cup (Y_1 \cap Y_2))
\end{aligned}$$

We find that

$$(Y_2 - X_1) = Y_2 \text{ since } Y_2 \subseteq X_2 \text{ and } X_2 \cap X_1 = \emptyset$$

and that

$$(Y_1 - X_2) = Y_1 \text{ since } Y_1 \subseteq X_1 \text{ and } X_1 \cap X_2 = \emptyset.$$

Hence, we have

$$\begin{aligned}
& (Y_2 - X_1) \cup (Y_1 - X_2) \cup (Y_1 \cap Y_2) \\
=\ & Y_2 \cup Y_1 \cup (Y_1 \cap Y_2) \\
=\ & Y_2 \cup Y_1.
\end{aligned}$$

Thus, we finally find that

$$\begin{aligned}
& t \in (X_1 \cup X_2) \wedge \\
& t \notin ((Y_2 - X_1) \cup (Y_1 - X_2) \cup (Y_1 \cap Y_2)) \\
\Leftrightarrow\ & t \in (X_1 \cup X_2) \wedge t \notin (Y_1 \cup Y_2) \\
\Leftrightarrow\ & t \in (X_1 \cup X_2) - (Y_1 \cup Y_2).
\end{aligned}$$

□

LEMMA 3: *Set containment division ($\div_1^*$) and great divide ($\div_3^*$) are equivalent operators.*

PROOF (LEMMA 3): In the following, we will show the equivalence of the relational algebra expressions of set containment division in Definition 4 and of great divide used in Definition 6. Let r_1 be a dividend relation and r_2 a divisor relation with schemas $R_1(A \cup B)$ and $R_2(B \cup C)$, respectively, as defined in Section 2.2. Let $\{C_1, \ldots, C_k\}$ be the set of (distinct) tuples in $\pi_C(r_2)$. If the divisor is non-empty then $k \geq 1$. We use the following algebraic law as a proposition:

(P1) $\pi_A(r_1 \cup r_2) = \pi_A(\pi_A(r_1) \cup \pi_A(r_2))$ for any relations r_1 and r_2 with the same schema $R(A \cup X)$, where A is a nonempty set of attributes and attribute set X may be empty or not.

Let us start with expression e, the definition of set containment division:

$$\begin{aligned} e &= r_1 \div_1^* r_2 \\ &= \bigcup_{t \in \pi_C(r_2)} \left(r_1 \div \pi_B\left(\sigma_{C=t}\left(r_2\right)\right)\right) \times (t) \end{aligned}$$

We replace the division operator by Definition 2:

$$\begin{aligned} e = &\bigcup_{t \in \pi_C(r_2)} (\pi_A(r_1) - \\ &\quad \pi_A((\pi_A(r_1) \times \pi_B(\sigma_{C=t}(r_2))) - r_1)) \times (t) \\ = &\left(\bigcup_{t \in \pi_C(r_2)} \pi_A(r_1) \times (t)\right) - \\ &\left(\bigcup_{t \in \pi_C(r_2)} (\pi_A((\pi_A(r_1) \times \right. \\ &\quad \pi_B(\sigma_{C=t}(r_2))) - r_1)) \times (t)) \\ = &\underbrace{(\pi_A(r_1) \times \pi_C(r_2))}_{e_0} - \left(\bigcup_{t \in \pi_C(r_2)} (\pi_A((\pi_A(r_1) \times \right. \\ &\quad \pi_B(\sigma_{C=t}(r_2))) - r_1)) \times (t)) \\ = &\, e_0 - \left(\bigcup_{1 \le i \le k} (\pi_A((\pi_A(r_1) \times \right. \\ &\quad \pi_B(\sigma_{C=C_i}(r_2))) - r_1)) \times (C_i)) \\ = &\, e_0 - \pi_{A \cup C}\left(\bigcup_{1 \le i \le k} \pi_{A \cup C}(((\pi_A(r_1) \times \right. \\ &\quad \pi_B(\sigma_{C=C_i}(r_2))) \times (C_i)) - (r_1 \times (C_i)))) \\ = &\, e_0 - \pi_{A \cup C}\left(\bigcup_{1 \le i \le k} \pi_{A \cup C}((\pi_A(r_1) \times \right. \\ &\quad \sigma_{C=C_i}(r_2)) - (r_1 \times (C_i)))) \end{aligned}$$

$$\begin{aligned} = &\, e_0 - \pi_{A \cup C}\left(\bigcup_{1 \le i \le k} \pi_{A \cup C}(\right. \\ &\quad \underbrace{\pi_{A \cup r_2.B \cup C}(\pi_A(r_1) \times \sigma_{C=C_i}(r_2))}_{e_i^1} - \\ &\quad \left.\left.\underbrace{\pi_{A \cup r1.B \cup C}(r_1 \times (C_i))}_{e_i^2}\right)\right) \end{aligned}$$

Next, let us take a look at expression $\tilde{e}$ representing Todd's great divide:

$$\begin{aligned} \tilde{e} &= r_1 \div_3^* r_2 \\ &= \underbrace{(\pi_A(r_1) \times \pi_C(r_2))}_{\tilde{e}_0} - \\ &\quad \pi_{A \cup C}((\pi_A(r_1) \times r_2) - (r_1 \bowtie r_2)) \\ &= \tilde{e}_0 - \pi_{A \cup C}\left(\left(\pi_A(r_1) \times \bigcup_{1 \le i \le k} \sigma_{C=C_i}(r_2)\right) - \right. \\ &\quad \left.\pi_{A \cup r_2.B \cup C}\left(r_1 \bowtie_{r_1.B=r_2.B} \bigcup_{1 \le i \le k} \sigma_{C=C_i}(r_2)\right)\right) \\ &= \tilde{e}_0 - \pi_{A \cup C}\left(\left(\bigcup_{1 \le i \le k} \pi_A(r_1) \times \sigma_{C=C_i}(r_2)\right) - \right. \\ &\quad \left.\pi_{A \cup r_2.B \cup C}\left(\bigcup_{1 \le i \le k} r_1 \bowtie_{r_1.B=r_2.B} \sigma_{C=C_i}(r_2)\right)\right) \end{aligned}$$

Using Lemma 2 we get

$$\begin{aligned} \tilde{e} = \tilde{e}_0 - \pi_{A \cup C}&\left(\bigcup_{1 \le i \le k} \pi_{A \cup r_2.B \cup C}(\pi_A(r_1) \times \right. \\ &\quad \sigma_{C=C_i}(r_2)) - \\ &\quad \left.\pi_{A \cup r_2.B \cup C}(r_1 \bowtie_{r_1.B=r_2.B} \sigma_{C=C_i}(r_2))\right) \end{aligned}$$

Using proposition P1 we get

$$\begin{aligned} \tilde{e} = \tilde{e}_0 - \pi_{A \cup C}&\left(\bigcup_{1 \le i \le k} \pi_{A \cup C}(\right. \\ &\quad \underbrace{\pi_{A \cup r_2.B \cup C}(\pi_A(r_1) \times \sigma_{C=C_i}(r_2))}_{\tilde{e}_i^1} - \\ &\quad \left.\left.\underbrace{\pi_{A \cup r_2.B \cup C}(r_1 \bowtie_{r_1.B=r_2.B} \sigma_{C=C_i}(r_2))}_{\tilde{e}_i^2}\right)\right) \end{aligned}$$

We see that expressions e and $\tilde{e}$ differ only in the subexpressions e_i^2 and $\tilde{e}_i^2$, respectively. We are now going to show that $e_i^1 - e_i^2 = \tilde{e}_i^1 - \tilde{e}_i^2$. Then we know that $e = \tilde{e}$, that is, set containment division and great divide are equivalent.

Instead of showing that $e_i^1 - e_i^2 = \tilde{e}_i^1 - \tilde{e}_i^2$, we prove the equivalent statement $e_i^1 \cap e_i^2 = \tilde{e}_i^1 \cap \tilde{e}_i^2$. These statements are equivalent because of Lemma 1. Based on this lemma, we can derive the following expressions:

$$
\begin{aligned}
& \tilde{e}_i^1 \cap \tilde{e}_i^2 \\
&= \pi_{A \cup r_2.B \cup C}\left(\pi_A\left(r_1\right) \times \sigma_{C=C_i}\left(r_2\right)\right) \cap \\
&\quad \pi_{A \cup r_2.B \cup C}\left(r_1 \ltimes_{r_1.B=r_2.B} \sigma_{C=C_i}\left(r_2\right)\right) \\
&= \pi_{A \cup r_2.B \cup C}\left(\pi_A\left(r_1\right) \times \sigma_{C=C_i}\left(r_2\right)\right) \cap \\
&\quad \pi_{A \cup r_2.B \cup C}\left(\sigma_{r_1.B=r_2.B}\left(r_1 \times \sigma_{C=C_i}\left(r_2\right)\right)\right) \\
&= \pi_{A \cup r_2.B \cup C}\left(r_1 \times \sigma_{C=C_i}\left(r_2\right)\right) \cap \\
&\quad \pi_{A \cup r_2.B \cup C}\left(\sigma_{r_1.B=r_2.B}\left(r_1 \times \sigma_{C=C_i}\left(r_2\right)\right)\right) \\
&= \pi_{A \cup r_2.B \cup C}\left(\sigma_{r_1.B=r_2.B}\left(r_1 \times \sigma_{C=C_i}\left(r_2\right)\right)\right) \\
&= \pi_{A \cup r_2.B \cup C}\left(r_1 \ltimes_{r_1.B=r_2.B} \sigma_{C=C_i}\left(r_2\right)\right) \\
&= \pi_{A \cup r_2.B \cup C}\left(\sigma_{r_1.B=r_2.B}\left(r_1 \times \sigma_{C=C_i}\left(r_2\right)\right)\right) \\
&= \pi_{A \cup r_2.B \cup C}\left(\left(r_1 \times \sigma_{C=C_i}\left(r_2\right)\right) \cap \right. \\
&\qquad \left. \pi_{A \cup r_1.B \cup C}\left(r_1 \times \sigma_{C=C_i}\left(r_2\right)\right)\right) \\
&= \pi_{A \cup r_2.B \cup C}\left(\pi_A\left(r_1\right) \times \sigma_{C=C_i}\left(r_2\right)\right) \cap \\
&\quad \pi_{A \cup r_1.B \cup C}\left(r_1 \times \sigma_{C=C_i}\left(r_2\right)\right) \\
&= \pi_{A \cup r_2.B \cup C}\left(\pi_A\left(r_1\right) \times \sigma_{C=C_i}\left(r_2\right)\right) \cap \\
&\quad \pi_{A \cup r_1.B \cup C}\left(r_1 \times \left(C_i\right)\right) \\
&= e_i^1 \cap e_i^2
\end{aligned}
$$

□

LEMMA 4: *Great divide ($\div_3^*$) and generalized division ($\div_2^*$) are equivalent operators.*

PROOF (LEMMA 4): In the following, we will show the equivalence of the relational algebra expressions of great divide used in Definition 6 and of generalized division in Definition 5. Let r_1 be a dividend relation and r_2 a divisor relation with schemas $R_1(A \cup B)$ and $R_2(B \cup C)$, respectively, as defined in Section 2.2. Let us review expression e, the definition of great divide:

$$
\begin{aligned}
e &= r_1 \div_3^* r_2 \\
&= \left(\pi_A\left(r_1\right) \times \pi_C\left(r_2\right)\right) - \\
&\quad \pi_{A \cup C}\left(\underbrace{\left(\pi_A\left(r_1\right) \times r_2\right)}_{e_1} - \underbrace{\left(r_1 \ltimes r_2\right)}_{e_2}\right)
\end{aligned}
$$

Now, we compare expression e to expression $\tilde{e}$, the definition of generalized division:

$$
\begin{aligned}
\tilde{e} &= r_1 \div_2^* r_2 \\
&= \left(\pi_A\left(r_1\right) \times \pi_C\left(r_2\right)\right) - \\
&\quad \pi_{A \cup C}\left(\underbrace{\left(\pi_A\left(r_1\right) \times r_2\right)}_{\tilde{e}_1} - \underbrace{\left(r_1 \times \pi_C\left(r_2\right)\right)}_{\tilde{e}_2}\right)
\end{aligned}
$$

We find that e and $\tilde{e}$ differ only in the expression e_2 and $\tilde{e}_2$, respectively. If we can show that $e_1 - e_2 = \tilde{e}_1 - \tilde{e}_2$, we have proved that $e = \tilde{e}_1$. Because of Lemma 1, it suffices to show that $e_1 \cap e_2 = \tilde{e}_1 \cap \tilde{e}_2$:

$$
\begin{aligned}
& e_1 \cap e_2 \\
&= \left(\pi_A\left(r_1\right) \times r_2\right) \cap \\
&\quad \left(r_1 \ltimes r_2\right) \\
&= \left(\pi_A\left(r_1\right) \times r_2\right) \cap \\
&\quad \pi_{A \cup r_2.B \cup C}\left(\sigma_{r_1.B=r_2.B}\left(r_1 \times r_2\right)\right) \\
&= \left(\pi_A\left(r_1\right) \times \pi_{B \cup C} r_2\right) \cap \\
&\quad \pi_{A \cup r_2.B \cup C}\left(\sigma_{r_1.B=r_2.B}\left(r_1 \times r_2\right)\right) \\
&= \pi_{A \cup r_2.B \cup C}\left(r_1 \times r_2\right) \cap \\
&\quad \pi_{A \cup r_2.B \cup C}\left(\sigma_{r_1.B=r_2.B}\left(r_1 \times r_2\right)\right) \\
&= \pi_{A \cup r_2.B \cup C}\left(\sigma_{r_1.B=r_2.B}\left(r_1 \times r_2\right)\right) \\
&\qquad \text{since } \sigma_{r_1.B=r_2.B}\left(r_1 \times r_2\right) \subseteq \left(r_1 \times r_2\right) \\
&= \pi_{A \cup r_2.B \cup C}\left(r_1 \times r_2\right) \cap \pi_{A \cup r_1.B \cup C}\left(r_1 \times r_2\right) \\
&= \left(\pi_A\left(r_1\right) \times r_2\right) \cap \left(r_1 \times \pi_C\left(r_2\right)\right) \\
&= \tilde{e}_1 \cap \tilde{e}_2
\end{aligned}
$$

□

B.2 Theorems

PROOF (THEOREM 1): Lemma 3 shows that set containment division ($\div_1^*$) and great divide ($\div_3^*$) are equivalent, and Lemma 4 shows that great divide ($\div_3^*$) and generalized division ($\div_2^*$) are equivalent. By transitivity we see that all three operators are equivalent. □

PROOF (THEOREM 2): Let $R_1(A_1)$, $R_1(A_2)$, and $R_1(A_3)$ be the schemas of relations r_1, r_2, and r_3 in the expression $r_1 \div r_2 = r_3$. According to the definition of division, the divisor has n attributes and the dividend has $m + n$ attributes, where $m > 0$ and $n > 0$. Since $m + n > n$, it is impossible to interchange r_1 and r_2, that is, $r_2 \div r_1$ is an invalid expression. □

PROOF (THEOREM 3): We show that if we assume that valid relation schemas exists for the the three relations we will arrive at a contradiction. Suppose, A_1, A_2, and A_3 are the attributes of the relations r_1, r_2, and r_3, respectively. If the two expressions are equivalent then the corresponding quotient relation schemas are the same. The relation schema of $r_1 \div (r_2 \div r_3)$ is defined by expression $e_1 = A_1 - (A_2 - A_3)$ and the schema of $(r_1 \div r_2) \div r_3$ is $e_2 = (A_1 - A_2) - A_3$. We try to show that $t \in e_1 \leftrightarrow t \in e_2$ is a tautology, that is, the expression is true for any value of tuple t. Since $t \in e_1 \leftrightarrow t \in e_2 = (t \in e_1 \rightarrow t \in e_2) \wedge (t \in$

$e_2 \rightarrow t \in e_1$), we can analyze each implication separately:

$$
\begin{aligned}
& t \in e_2 \rightarrow t \in e_1 \\
\Leftrightarrow\ & t \notin e_2 \vee t \in e_1 \\
\Leftrightarrow\ & t \notin ((A_1 - A_2) - A_3) \vee t \in (A_1 - (A_2 - A_3)) \\
\Leftrightarrow\ & (t \notin (A_1 - A_2) \vee t \in A_3) \vee \\
& (t \in A_1 \wedge t \notin (A_2 - A_3)) \\
\Leftrightarrow\ & (t \notin A_1 \vee t \in A_2 \vee t \in A_3) \vee \\
& (t \in A_1 \wedge (t \notin A_2 \vee t \in A_3)) \\
\Leftrightarrow\ & (t \notin A_1 \vee t \in A_2 \vee t \in A_3 \vee t \in A_1) \wedge \\
& (t \notin A_1 \vee t \in A_2 \vee t \in A_3 \vee t \notin A_2 \vee t \in A_3) \\
\Leftrightarrow\ & \textit{true} \wedge \textit{true} \\
\Leftrightarrow\ & \textit{true}
\end{aligned}
$$

Now, we analyze the opposite direction of the equivalence:

$$
\begin{aligned}
& t \in e_1 \rightarrow t \in e_2 \\
\Leftrightarrow\ & t \notin e_1 \vee t \in e_2 \\
\Leftrightarrow\ & t \notin (A_1 - (A_2 - A_3)) \vee t \in ((A_1 - A_2) - A_3) \\
\Leftrightarrow\ & t \notin A_1 \vee t \in (A_2 - A_3) \vee \\
& (t \in (A_1 - A_2) \wedge t \notin A_3) \\
\Leftrightarrow\ & t \notin A_1 \vee (t \in A_2 \wedge t \notin A_3) \vee \\
& (t \in A_1 \wedge t \notin A_2 \wedge t \notin A_3) \\
\Leftrightarrow\ & (t \notin A_1 \vee (t \in A_2 \wedge t \notin A_3) \vee t \in A_1) \wedge \\
& (t \notin A_1 \vee (t \in A_2 \wedge t \notin A_3) \vee t \notin A_2) \wedge \\
& (t \notin A_1 \vee (t \in A_2 \wedge t \notin A_3) \vee t \notin A_3) \\
\Leftrightarrow\ & \textit{true} \wedge (t \notin A_1 \vee (t \in A_2 \wedge t \notin A_3) \vee t \notin A_2) \wedge \\
& (t \notin A_1 \vee t \notin A_3) \\
\Leftrightarrow\ & t \notin A_1 \vee ((t \in A_2 \wedge t \notin A_3) \vee t \notin A_2 \vee t \notin A_3) \\
\Leftrightarrow\ & t \notin A_1 \vee t \notin A_2 \vee t \notin A_3 \\
\Leftrightarrow\ & t \notin A_1 \cap A_2 \cap A_3
\end{aligned}
$$

Since $t \in e_1 \leftrightarrow t \in e_2 = t \notin A_1 \cap A_2 \cap A_3 \neq \textit{true}$ for any value of t (for a value $t \in A_1 \cap A_2 \cap A_3$ it is *false*), we have found a contradiction to our assumption that the expression is a tautology. □

B.3 Laws

PROOF (LAW 1): Let

$$
\begin{aligned}
e &= r_1 \ltimes (r_1 \div r_2') \\
&= \{t \mid t \in r_1 \wedge t.A \in (r_1 \div r_2')\} \\
&= \{t \mid t \in r_1 \wedge t.A \in \{u \mid \exists u_1 : u = u_1.A \wedge \\
&\qquad u_1 \in r_1 \wedge r_2' \subseteq \{y \mid (u, y) \in r_1\}\}\} \\
&= \{t \mid t \in r_1 \wedge \exists u_1 : t.A = u_1.A \wedge \\
&\qquad u_1 \in r_1 \wedge r_2' \subseteq \{y \mid (t.A, y) \in r_1\}\}.
\end{aligned}
$$

Since $t \in r_1$ already implies $\exists u_1 : t.A = u_1.A \wedge u_1 \in r_1$, we have

$$
e = \{t \mid t \in r_1 \wedge r_2' \subseteq \{y \mid (t.A, y) \in r_1\}\}.
$$

Hence,

$$
\begin{aligned}
& (r_1 \ltimes (r_1 \div r_2')) \div r_2'' \\
&= \{s \mid \exists s_1 : s = s_1.A \wedge s_1 \in e \wedge r_2'' \subseteq \{z \mid (s, z) \in e\}\} \\
&= \{s \mid \exists s_1 : s = s_1.A \wedge \\
&\qquad s_1 \in \{t \mid t \in r_1 \wedge \\
&\qquad\quad r_2' \subseteq \{y \mid (t.A, y) \in r_1\}\} \wedge \\
&\qquad r_2'' \subseteq \{z \mid (s, z) \in \{t \mid t \in r_1 \wedge \\
&\qquad\quad r_2' \subseteq \{y \mid (t.A, y) \in r_1\}\}\} \\
&= \{s \mid \exists s_1 : s = s_1.A \wedge s_1 \in r_1 \wedge \\
&\qquad\quad r_2' \subseteq \{y \mid (s_1.A, y) \in r_1\}\} \wedge \\
&\qquad r_2'' \subseteq \{z \mid (s, z) \in r_1 \wedge \\
&\qquad\quad r_2' \subseteq \{y \mid ((s, z).A, y) \in r_1\}\}\} \\
&= \{s \mid \exists s_1 : s = s_1.A \wedge \\
&\qquad s_1 \in r_1 \wedge r_2' \subseteq \{y \mid (s, y) \in r_1\}\} \wedge \\
&\qquad r_2'' \subseteq \{z \mid (s, z) \in r_1 \wedge \\
&\qquad\quad r_2' \subseteq \{y \mid (s, y) \in r_1\}\}\} \\
&= \{s \mid \exists s_1 : s = s_1.A \wedge \\
&\qquad s_1 \in r_1 \wedge r_2' \subseteq \{y \mid (s, y) \in r_1\}\} \wedge \\
&\qquad r_2'' \subseteq \{z \mid (s, z) \in r_1\} \wedge \\
&\qquad\quad r_2'' \subseteq \{z \mid r_2' \subseteq \{y \mid (s, y) \in r_1\}\}\} \\
&= \{s \mid \exists s_1 : s = s_1.A \wedge \\
&\qquad s_1 \in r_1 \wedge r_2' \subseteq \{y \mid (s, y) \in r_1\}\} \wedge \\
&\qquad r_2'' \subseteq \{z \mid (s, z) \in r_1\}\} \\
&= \{s \mid \exists s_1 : s = s_1.A \wedge s_1 \\
&\qquad \in r_1 \wedge (r_2' \cup r_2'') \subseteq \{y \mid (s, y) \in r_1\}\} \\
&= r_1 \div (r_2' \cup r_2'')
\end{aligned}
$$

□

PROOF (LAW 2): We prove that if condition $c_1(r_1', r_1'')$ is *true* then $(r_1' \cup r_1'') \div r_2 = (r_1' \div r_2) \cup (r_1'' \div r_2)$.

We use the following algebraic laws as propositions, where we assume that relations r_1, r_2, s_1, and s_2 have the same schema:

(P1) $\sigma_\theta(r_1 \cup r_2) = \sigma_\theta(r_1) \cup \sigma_\theta(r_2)$ [13]

(P2) $\pi_A(r_1 \cup r_2) = \pi_A(r_1) \cup \pi_A(r_2)$, where A is any subset of r_1's and r_2's relation schemas.

$$
\begin{aligned}
&(r_1' \cup r_1'') \div r_2 \\
&= \bigcap_{t \in r_2} \pi_A\left(\sigma_{B=t}\left(r_1' \cup r_1''\right)\right) \qquad \text{(Definition 3)} \\
&= \bigcap_{t \in r_2} \pi_A\left(\sigma_{B=t}\left(r_1'\right) \cup \sigma_{B=t}\left(r_1''\right)\right) \qquad \text{(P1)} \\
&= \bigcap_{t \in r_2} \pi_A\left(\sigma_{B=t}\left(r_1'\right)\right) \cup \pi_A\left(\sigma_{B=t}\left(r_1''\right)\right) \qquad \text{(P2)} \\
&= \bigcap_{t \in r_2} \pi_A\left(\sigma_{B=t}\left(r_1'\right)\right) \cup \bigcap_{t \in r_2} \pi_A\left(\sigma_{B=t}\left(r_1''\right)\right) \\
&\qquad \text{(see below)} \\
&= (r_1' \div r_2) \cup (r_1'' \div r_2) \qquad \text{(Definition 3)}
\end{aligned}
$$

To show the missing step in the above transformation, we restrict ourselves to the case where r_2 contains two tuples, t_1 and t_2, only. This can easily be extended to the general case. Consider

$$
\begin{aligned}
&\bigcap_{t \in \{t_1, t_2\}} \pi_A\left(\sigma_{B=t}\left(r_1'\right)\right) \cup \pi_A\left(\sigma_{B=t}\left(r_1''\right)\right) \\
&= \left(\pi_A\left(\sigma_{B=t_1}\left(r_1'\right)\right) \cup \pi_A\left(\sigma_{B=t_1}\left(r_1''\right)\right)\right) \cap \\
&\quad \left(\pi_A\left(\sigma_{B=t_2}\left(r_1'\right)\right) \cup \pi_A\left(\sigma_{B=t_2}\left(r_1''\right)\right)\right) \\
&= \underbrace{\left(\pi_A\left(\sigma_{B=t_1}\left(r_1'\right)\right) \cap \pi_A\left(\sigma_{B=t_2}\left(r_1'\right)\right)\right)}_{S_{r'r'}} \cup \\
&\quad \underbrace{\left(\pi_A\left(\sigma_{B=t_1}\left(r_1''\right)\right) \cap \pi_A\left(\sigma_{B=t_2}\left(r_1''\right)\right)\right)}_{S_{r''r''}} \cup \\
&\quad \underbrace{\left(\pi_A\left(\sigma_{B=t_1}\left(r_1'\right)\right) \cap \pi_A\left(\sigma_{B=t_2}\left(r_1''\right)\right)\right)}_{S_{r'r''}} \cup \\
&\quad \underbrace{\left(\pi_A\left(\sigma_{B=t_1}\left(r_1''\right)\right) \cap \pi_A\left(\sigma_{B=t_2}\left(r_1'\right)\right)\right)}_{S_{r''r'}} .
\end{aligned}
$$

To show that this is equal to $\bigcap_{t \in \{t_1, t_2\}} \pi_A\left(\sigma_{B=t}\left(r_1'\right)\right) \cup \bigcap_{t \in \{t_1, t_2\}} \pi_A\left(\sigma_{B=t}\left(r_1''\right)\right)$, we need to argue why $S_{r'r''}$ and $S_{r''r'}$ are subsets of $S_{r'r'} \cup S_{r''r''}$. The basic idea is that $S_{r'r''}$ and $S_{r''r'}$ are sufficiently restricted by precondition c_1. In the following we will show with an indirect proof that $S_{r'r''}$ meets this requirement if precondition c_1 is true. The proof for $S_{r''r'}$ is analogous.

Assume that $S_{r'r''} \not\subseteq S_{r'r'} \cup S_{r''r''}$. Remember that we are still in the case where $r_2 = \{t_1, t_2\}$. Hence,

$$
\begin{aligned}
&\exists t : t \in S_{r'r''} \wedge t \notin S_{r'r'} \wedge t \notin S_{r''r''} \\
\Leftrightarrow\ &\exists t : t \in \pi_A\left(\sigma_{B=t_1}\left(r_1'\right)\right) \cap \pi_A\left(\sigma_{B=t_2}\left(r_1''\right)\right) \wedge \\
&t \notin \left(\pi_A\left(\sigma_{B=t_1}\left(r_1'\right)\right) \cap \pi_A\left(\sigma_{B=t_2}\left(r_1'\right)\right)\right) \wedge \\
&t \notin \left(\pi_A\left(\sigma_{B=t_1}\left(r_1''\right)\right) \cap \pi_A\left(\sigma_{B=t_2}\left(r_1''\right)\right)\right) \\
\Leftrightarrow\ &\exists t : t.t_1 \in r_1' \wedge t.t_2 \in r_1'' \wedge \\
&\neg\left(t.t_1 \in r_1' \wedge t.t_2 \in r_1'\right) \wedge \\
&\neg\left(t.t_1 \in r_1'' \wedge t.t_2 \in r_1''\right) \\
\Rightarrow\ &\exists t : r_2 \subseteq \pi_B\left(\sigma_{A=t}\left(r_1'\right) \cup \sigma_{A=t}\left(r_1''\right)\right) \wedge \\
&\neg\left(r_2 \subseteq \pi_B\left(\sigma_{A=t}\left(r_1'\right)\right)\right) \wedge \\
&\neg\left(r_2 \subseteq \pi_B\left(\sigma_{A=t}\left(r_1''\right)\right)\right) \\
\Leftrightarrow\ &\neg c_1(r_1', r_1'') \text{ for } a = t.
\end{aligned}
$$

□

PROOF (LAW 3): We use the following algebraic laws given in [13] as propositions:

(P1) $\sigma_\theta(r_1 - r_2) = \sigma_\theta(r_1) - \sigma_\theta(r_2)$,

(P2) $\pi_X(\sigma_\theta(r)) = \pi_X(\sigma_\theta(\pi_Y(r)))$, where Y contains X and the attributes mentioned in condition θ, in particular, $\sigma_{p(A)}(\pi_A(r_1)) = \pi_A(\sigma_{p(A)}(\pi_A(r_1))) = \pi_A(\sigma_{p(A)}(r_1))$, and

(P3) $\sigma_\theta(r_1 \times r_2) = \sigma_\theta(r_1) \times r_2$ if θ restricts attributes of r_1, only.

$$
\begin{aligned}
&\sigma_{p(A)}(r_1 \div r_2) \\
&= \sigma_{p(A)}\left(\pi_A\left(r_1\right) - \pi_A\left(\left(\pi_A\left(r_1\right) \times r_2\right) - r_1\right)\right) \\
&\qquad \text{(Definition 2)} \\
&= \sigma_{p(A)}\left(\pi_A\left(r_1\right)\right) - \sigma_{p(A)}\left(\pi_A\left(\left(\pi_A\left(r_1\right) \times r_2\right) - r_1\right)\right) \\
&\qquad \text{(P1)} \\
&= \pi_A\left(\sigma_{p(A)}\left(r_1\right)\right) - \\
&\quad \pi_A\left(\sigma_{p(A)}\left(\pi_A\left(\left(\pi_A\left(r_1\right) \times r_2\right) - r_1\right)\right)\right) \text{ (P2)} \\
&= \pi_A\left(\sigma_{p(A)}\left(r_1\right)\right) - \\
&\quad \pi_A\left(\sigma_{p(A)}\left(\pi_A\left(\left(\pi_A\left(r_1\right) \times r_2\right)\right) - \sigma_{p(A)}\left(r_1\right)\right)\right) \\
&\qquad \text{(P1)} \\
&= \pi_A\left(\sigma_{p(A)}\left(r_1\right)\right) - \\
&\quad \pi_A\left(\pi_A\left(\sigma_{p(A)}\left(\left(\pi_A\left(r_1\right) \times r_2\right)\right) - \sigma_{p(A)}\left(r_1\right)\right)\right) \\
&\qquad \text{(P2)} \\
&= \pi_A\left(\sigma_{p(A)}\left(r_1\right)\right) - \\
&\quad \pi_A\left(\pi_A\left(\left(\sigma_{p(A)}\left(\pi_A\left(r_1\right)\right) \times r_2\right) - \sigma_{p(A)}\left(r_1\right)\right)\right) \\
&\qquad \text{(P3)}
\end{aligned}
$$

$$
\begin{aligned}
&= \pi_A\left(\sigma_{p(A)}\left(r_1\right)\right) - \\
&\quad \pi_A\left(\pi_A\left(\left(\pi_A\left(\sigma_{p(A)}\left(r_1\right)\right) \times r_2\right) - \sigma_{p(A)}\left(r_1\right)\right)\right) \\
&\qquad \text{(P2)} \\
&= \sigma_{p(A)}(r_1) \div r_2 \qquad \text{(Definition 2)}
\end{aligned}
$$

□

PROOF (LAW 4):

$$
\begin{aligned}
& r_1 \div \sigma_{p(B)}\left(r_2\right) \\
&= \sigma_{p(B)\vee\neg p(B)}\left(r_1\right) \div \sigma_{p(B)}\left(r_2\right) \\
&= \left(\sigma_{p(B)}\left(r_1\right) \cup \sigma_{\neg p(B)}\left(r_1\right)\right) \div \sigma_{p(B)}\left(r_2\right) \\
&= \left(\sigma_{p(B)}\left(r_1\right) \div \sigma_{p(B)}\left(r_2\right)\right) \cup \\
&\quad \left(\sigma_{\neg p(B)}\left(r_1\right) \div \sigma_{p(B)}\left(r_2\right)\right) \qquad \text{(Law 2)} \\
&= \left(\sigma_{p(B)}\left(r_1\right) \div \sigma_{p(B)}\left(r_2\right)\right) \cup \emptyset \\
&= \sigma_{p(B)}\left(r_1\right) \div \sigma_{p(B)}\left(r_2\right)
\end{aligned}
$$

We can apply Law 2 to the expression in line 2 because the law's precondition $c_2(\sigma_{p(B)}(r_1), \sigma_{\neg p(B)}(r_1))$ (and, of course, also c_1) is obviously fulfilled. □

PROOF (LAW 5): According to Codd's definition of division in tuple relational calculus (Definition 1), we can derive the following equivalences:

$$
\begin{aligned}
& (r_1' \div r_2) \cap (r_1'' \div r_2) \\
&= \{t \mid t = t_1.A \wedge t_1 \in r_1' \wedge r_2 \subseteq \{y \mid (t,y) \in r_1'\}\} \cap \\
&\quad \{t \mid t = t_1.A \wedge t_1 \in r_1'' \wedge r_2 \subseteq \{y \mid (t,y) \in r_1''\}\} \\
&= \{t \mid t = t_1.A \wedge t_1 \in r_1' \wedge t_1 \in r_1'' \wedge \\
&\qquad r_2 \subseteq \{y \mid (t,y) \in r_1'\} \wedge r_2 \subseteq \{y \mid (t,y) \in r_1''\}\} \\
&= \{t \mid t = t_1.A \wedge t_1 \in r_1' \wedge t_1 \in r_1'' \wedge \\
&\qquad r_2 \subseteq \{y \mid (t,y) \in r_1'\} \cap \{y \mid (t,y) \in r_1''\}\} \\
&= \{t \mid t = t_1.A \wedge t_1 \in r_1' \wedge t_1 \in r_1'' \wedge \\
&\qquad r_2 \subseteq \{y \mid (t,y) \in r_1' \wedge (t,y) \in r_1''\}\} \\
&= \{t \mid t = t_1.A \wedge t_1 \in r_1' \wedge t_1 \in r_1'' \wedge \\
&\qquad r_2 \subseteq \{y \mid (t,y) \in r_1' \cap r_1''\}\} \\
&= (r_1' \cap r_1'') \div r_2
\end{aligned}
$$

□

PROOF (LAW 6): With Codd's definition we get:

$$
\begin{aligned}
& (r_1' \div r_2) - (r_1'' \div r_2) \\
&= \{t \mid t = t_1.A \wedge\ t_1 \in r_1' \wedge \\
&\qquad r_2 \subseteq \{y \mid\ (t_1.A, y) \in r_1'\}\} - \\
&\quad \{t \mid t = t_1.A \wedge t_1 \in r_1'' \wedge \\
&\qquad r_2 \subseteq \{y \mid\ (t_1.A, y) \in r_1''\}\} \\
&= \{t \mid t = t_1.A \wedge t_1 \in r_1' \wedge \\
&\qquad r_2 \subseteq \{y \mid\ (t_1.A, y) \in r_1'\} \wedge \\
&\qquad (t_1 \notin r_1'' \vee r_2 \nsubseteq \{y \mid\ (t_1.A, y) \in r_1''\})\} \\
&= \{t \mid t = t_1.A \wedge \\
&\qquad (t_1 \in r_1' \wedge r_2 \subseteq \{y \mid\ (t_1.A, y) \in r_1'\} \wedge \\
&\qquad\quad t_1 \notin r_1'') \vee \\
&\qquad (t_1 \in r_1' \wedge r_2 \subseteq \{y \mid\ (t_1.A, y) \in r_1'\} \wedge \\
&\qquad\quad r_2 \nsubseteq \{y \mid\ (t_1.A, y) \in r_1''\})\} \\
&= \{t \mid t = t_1.A \wedge \\
&\qquad (t_1 \in r_1' - r_1'' \wedge r_2 \subseteq \{y \mid\ (t_1.A, y) \in r_1'\}) \vee \\
&\qquad (t_1 \in r_1' \wedge r_2 \subseteq \{y \mid\ (t_1.A, y) \in r_1' - r_1''\})\}
\end{aligned}
$$

From the precondition we get the following (since the predicate applies to attributes in A only):

$$
\begin{aligned}
& t_1 \in r_1' - r_1'' \wedge (t_1.A, y) \in r_1' \\
&\equiv (t_1.A, y) \in r_1' - r_1'' \qquad (1)
\end{aligned}
$$

$$
t_1 \in r_1' \wedge (t_1.A, y) \in r_1' - r_1'' \equiv t_1 \in r_1' - r_1'' \qquad (2)
$$

If we use these equivalences in the above equation we directly get:

$$
\begin{aligned}
&= \{t \mid t = t_1.A \wedge t_1 \in r_1' - r_1'' \wedge \\
&\qquad r_2 \subseteq \{y \mid\ (t_1.A, y) \in r_1' - r_1''\}\} \\
&= (r_1' - r_1'') \div r_2
\end{aligned}
$$

□

PROOF (LAW 7): Our assumption that $\pi_A\left(r_1'\right)$ and $\pi_A\left(r_1''\right)$ are disjoint is equivalent to $\pi_A\left(r_1'\right) \cap \pi_A\left(r_1''\right) = \emptyset$. Hence, $(r_1' - r_1'') \div r_2 = r_1' \div r_2$. Therefore, we can show that

$$
\begin{aligned}
& (r_1' \div r_2) - (r_1'' \div r_2) \\
&= \bigcap_{t \in r_2} \pi_A\left(\sigma_{B=t}\left(r_1'\right)\right) - \bigcap_{t \in r_2} \pi_A\left(\sigma_{B=t}\left(r_1''\right)\right) \\
&\qquad \text{(Definition 3)}
\end{aligned}
$$

$$= \bigcap_{t \in r_2} \pi_A \left(\sigma_{B=t}\left(r_1'\right)\right) \text{ since } \pi_A\left(r_1'\right) \cap \pi_A\left(r_1''\right) = \emptyset$$
$$= r_1' \div r_2$$

□

PROOF (LAW 8): We use the following algebraic laws as propositions:

(P1) $\sigma_\theta(r_1 \times r_2) = r_1 \times \sigma_\theta(r_2)$ for relations r_1 and r_2 with schemas $R_1(A)$ and $R_2(B)$, respectively, and θ contains only attributes in B.

(P2) $\pi_{B\cup C}(r_1 \times r_2) = \pi_B(r_1) \times \pi_C(r_2)$ for relations r_1 and r_2 with schemas $R_1(A \cup B)$ and $R_2(C \cup D)$, respectively.

(P3) $(r_1 \times r_2) \cap (r_1 \times r_3) = r_1 \times (r_2 \cap r_3)$ for relations r_1, r_2, and r_3 with schemas $R_1(A)$, $R_2(B)$, and $R_3(B)$, respectively.

$$\begin{aligned}
&\left(r_1^* \times r_1^{**}\right) \div r_2 \\
&= \bigcap_{t \in r_2} \pi_{A_1 \cup A_2}\left(\sigma_{B=t}\left(r_1^* \times r_1^{**}\right)\right) && \text{(Definition 3)} \\
&= \bigcap_{t \in r_2} \pi_{A_1 \cup A_2}\left(r_1^* \times \sigma_{B=t}\left(r_1^{**}\right)\right) && \text{(P1)} \\
&= \bigcap_{t \in r_2} \pi_{A_1}\left(r_1^*\right) \times \pi_{A_2}\left(\sigma_{B=t}\left(r_1^{**}\right)\right) && \text{(P2)} \\
&= \pi_{A_1}\left(r_1^*\right) \times \bigcap_{t \in r_2} \pi_{A_2}\left(\sigma_{B=t}\left(r_1^{**}\right)\right) && \text{(P3)} \\
&= r_1^* \times \left(r_1^{**} \div r_2\right) && \text{(Definition 3)}
\end{aligned}$$

□

PROOF (LAW 9):

$$\begin{aligned}
&\left(r_1^* \times r_1^{**}\right) \div r_2 \\
&= \bigcap_{t \in r_2} \pi_A\left(\sigma_{B=t}\left(r_1^* \times r_1^{**}\right)\right) && \text{(Definition 3)} \\
&= \bigcap_{t \in r_2} \pi_A\left(\sigma_{B_1=t.B_1 \wedge B_2=t.B_2}\left(r_1^* \times r_1^{**}\right)\right) \\
&= \bigcap_{t \in r_2} \pi_A\left(\sigma_{B_1=t.B_1 \wedge \mathit{true}}\left(r_1^* \times r_1^{**}\right)\right) \\
&\qquad \text{(since } \forall t' \in r_2 : t'.B_2 \in r_1^{**}\text{)} \\
&= \bigcap_{t \in r_2} \pi_A\left(\sigma_{B_1=t.B_1}\left(r_1^* \times r_1^{**}\right)\right) \\
&= \bigcap_{t \in \pi_{B_1}(r_2)} \pi_A\left(\sigma_{B_1=t}\left(r_1^* \times r_1^{**}\right)\right) \\
&= \bigcap_{t \in \pi_{B_1}(r_2)} \pi_A\left(\sigma_{B_1=t}\left(r_1^*\right)\right) \\
&= r_1^* \div \pi_{B_1}\left(r_2\right) && \text{(Definition 3)}
\end{aligned}$$

□

PROOF (LAW 10):

$$\begin{aligned}
&\left(r_1 \div r_2\right) \ltimes r_3 \\
&= \pi_A\left(\left(r_1 \div r_2\right) \bowtie r_3\right) && \text{(Definition of semi-join)} \\
&= \pi_A\left(\sigma_{A=A}\left(\left(r_1 \div r_2\right) \times r_3\right)\right) \\
&\qquad \text{(Definition of natural join)} \\
&= \pi_A\left(\bigcup_{t \in r_3} \sigma_{A=t.A}\left(\left(r_1 \div r_2\right) \times (t)\right)\right) \\
&\qquad \text{(Definition of Cartesian product)} \\
&= \bigcup_{t \in r_3} \sigma_{A=t.A}\left(r_1 \div r_2\right) \\
&\qquad \text{(Remove duplicate attribute set } A\text{)} \\
&= \bigcup_{t \in r_3} \left(\sigma_{A=t.A}\left(r_1\right) \div r_2\right) && \text{(Law 3)} \\
&= \left(\bigcup_{t \in r_3} \sigma_{A=t.A}\left(r_1\right)\right) \div r_2 && \text{(Law 2)} \\
&= \left(r_1 \ltimes r_3\right) \div r_2 && \text{(Definition of semi-join)}
\end{aligned}$$

□

PROOF (LAW 11): As defined in Section 2.1, the schema of r_1 is $R_1(A \cup B)$ and the schema of r_2 is $R_2(B)$. We will show the three cases separately.

Case 1: $\sigma_{c=0}\left(\gamma_{\text{count}(B)\to c}\left(r_2\right)\right) \neq \emptyset$, that is, $r_2 = \emptyset$.
It follows from the definition that $r_1 \div r_2 = r_1$.

Case 2: $\sigma_{c=1}\left(\gamma_{\text{count}(B)\to c}\left(r_2\right)\right) \neq \emptyset$, that is, $|r_2| = 1$.
Assume that, w.l.o.g., $r_2 = \{t_2\}$. We have to show the following:

$$r_1 \div r_2 = \pi_A\left(r_1 \ltimes r_2\right)$$
$$\bigcap_{t \in r_2} \pi_A\left(\sigma_{B=t}\left(r_1\right)\right) = \pi_A\left(r_1 \ltimes r_2\right) \qquad \text{(Definition 3)}$$
$$\pi_A\left(\sigma_{B=t_2}\left(r_1\right)\right) = \pi_A\left(r_1 \ltimes r_2\right) \qquad \text{(with } r_2 = \{t_2\}\text{)}$$

Now, let us consider $\sigma_{B=t_2}(r)$:

$$\begin{aligned}
\sigma_{B=t_2}\left(r_1\right) &= \{t \in r_1 \,|\, t.B = t_2\} \\
&= \{t \in r_1 \,|\, t.B \in r_2\} \\
&= r_1 \ltimes r_2.
\end{aligned}$$

Hence, $\pi_A(\sigma_{B=t_2}(r_1)) = \pi_A(r_1 \ltimes r_2)$.

Case 3: $\sigma_{c>1}\left(\gamma_{\text{count}(B)\to c}(r_2)\right) \neq \emptyset$, that is, $|r_2| > 1$.

From the construction of r_1 as $r_1 = {}_A\gamma_{f(X)\to B}(r_0)$ we know that $\forall t_1, t_2 \in r_1 : t_1.A \neq t_2.A$. From the precondition of Case 3 we also know that $\exists t_1, t_2 \in r_2 : t_1 \neq t_2$. With Definition 3 of the division operator the claim can be shown by a simple indirect proof. □

PROOF (LAW 12): Let $e = \pi_A(r_1 \ltimes r_2) = \{t_a | \exists t_b \in r_2 : (t_a, t_b) \in r_1\}$. We have to show three cases:

Case 1: $|e| > 1$:

In this case there exist $t_{a_1}, t_{a_2} \in e$ with $t_{a_1} \neq t_{a_2}$. Hence, there also exist $t_{b_1}, t_{b_2} \in r_2$ with $t_{b_1} \neq t_{b_2}$ and $(t_{a_1}, t_{b_1}) \in r_1$ and $(t_{a_2}, t_{b_2}) \in r_1$. Considering the precondition of the law $\forall (t_{a_1}, t_{b_1}), (t_{a_2}, t_{b_2}) \in r_1 : t_{b_1} \neq t_{b_2}$ this implies that

(a) $(t_{a_1}, t_{b_2}) \notin r_1$ and

(b) $\forall t_{a_i} \neq t_{a_1} : (t_{a_i}, t_{b_1}) \notin r_1$.

Hence, there is no t_a such that $r_2 \subseteq \{y | (t_a, y) \in r_1\}$. It directly follows that $r_1 \div r_2 = \{t_a | \exists t_b \in r_2 : (t_a, t_b) \in r_1 \wedge r_2 \subseteq \{y | (t_a, y) \in r_1\}\} = \emptyset$.

Case 2: $|e| < 1$:

Consider

$$\begin{aligned}
& r_1 \div r_2 \\
&= \{t_a | \exists t_b \in r_2 : (t_a, t_b) \in r_1 \wedge \\
&\qquad r_2 \subseteq \{y | (t_a, y) \in r_1\}\} \\
&\subseteq \{t_a | \exists b \in r_2 : (t_a, t_b) \in r_1\} \\
&= \pi_A(r_1 \ltimes r_2) \\
&= \emptyset
\end{aligned}$$

Case 3: $|e| = 1$:

Because of the precondition that the divisor attribute set $r_2.B$ is a foreign key referencing r_1 we have $\pi_B(r_1) \supseteq r_2$. This implies $\forall t_b \in r_2\ \exists t_a : (t_a, t_b) \in r_1$.

With $|e| = |\{t_a | \exists t_b \in r_2 : (t_a, t_b) \in r_1\}| = 1$ we conclude that $\exists t_a : \forall t_b \in r_2\, (t_a, t_b) \in r_1$, which implies that $|r_1 \div r_2| = |\{t_a | \exists t_b \in r_2 : (t_a, t_b) \in r_1 \wedge r_2 \subseteq \{y | (t_a, y) \in r_1\}\}| \geq 1$.

In Case 2 we have shown that $r_1 \div r_2 \subseteq \pi_A(r_1 \ltimes r_2)$. From this it follows that $r_1 \div r_2 = \pi_A(r_1 \ltimes r_2)$. □

PROOF (LAW 13):

$$\begin{aligned}
& r_1 \div^* (r_2' \cup r_2'') \\
&= \bigcup_{t \in \pi_C(r_2' \cup r_2'')} (r_1 \div \pi_B(\sigma_{C=t}(r_2' \cup r_2''))) \times (t) \\
&\qquad \text{(Definition 4)} \\
&= \left(\bigcup_{t \in \pi_C(r_2')} (r_1 \div \pi_B(\sigma_{C=t}(r_2' \cup r_2''))) \times (t)\right) \cup \\
&\quad \left(\bigcup_{t \in \pi_C(r_2'')} (r_1 \div \pi_B(\sigma_{C=t}(r_2' \cup r_2''))) \times (t)\right)
\end{aligned}$$

From our assumption $\pi_C(r_2') \cap \pi_C(r_2'') = \emptyset$ it follows for all $t \in \pi_C(r_2')$ that $\sigma_{C=t}(r_2'') = \emptyset$ and hence $\sigma_{C=t}(r_2' \cup r_2'') = \sigma_{C=t}(r_2')$. Similarly,

$$\begin{aligned}
& \pi_C(r_2') \cap \pi_C(r_2'') = \emptyset \\
\Rightarrow\ & \forall t \in \pi_C(r_2'') : \sigma_{C=t}(r_2'') = \emptyset \\
\Leftrightarrow\ & \forall t \in \pi_C(r_2'') : \sigma_{C=t}(r_2' \cup r_2'') = \sigma_{C=t}(r_2').
\end{aligned}$$

Hence, we have

$$\begin{aligned}
& r_1 \div^* (r_2' \cup r_2'') \\
&= \left(\bigcup_{t \in \pi_C(r_2')} (r_1 \div \pi_B(\sigma_{C=t}(r_2'))) \times (t)\right) \cup \\
&\quad \left(\bigcup_{t \in \pi_C(r_2'')} (r_1 \div \pi_B(\sigma_{C=t}(r_2''))) \times (t)\right) \\
&= (r_1 \div^* r_2') \cup (r_1 \div^* r_2''). \qquad \text{(Definition 4)}
\end{aligned}$$

□

PROOF (LAW 14):

$$\begin{aligned}
& \sigma_{p(A)}(r_1 \div^* r_2) \\
&= \sigma_{p(A)}\left(\bigcup_{t \in \pi_C(r_2)} (r_1 \div \pi_B(\sigma_{C=t}(r_2))) \times (t)\right) \\
&\qquad \text{(Definition 4)} \\
&= \bigcup_{t \in \pi_C(r_2)} \sigma_{p(A)}((r_1 \div \pi_B(\sigma_{C=t}(r_2))) \times (t)) \\
&= \bigcup_{t \in \pi_C(r_2)} \left(\sigma_{p(A)}(r_1) \div \pi_B(\sigma_{C=t}(r_2))\right) \times (t) \\
&= \sigma_{p(A)}(r_1) \div^* r_2 \qquad \text{(Definition 4)}
\end{aligned}$$

□

PROOF (LAW 15):

$$
\begin{aligned}
&\sigma_{p(C)}\left(r_1 \div^* r_2\right) \\
&= \sigma_{p(C)}\left(\bigcup_{t \in \pi_C(r_2)} \left(r_1 \div \pi_B\left(\sigma_{C=t}\left(r_2\right)\right)\right) \times (t)\right) \\
&\qquad \text{(Definition 4)} \\
&= \bigcup_{t \in \sigma_{p(C)}(\pi_C(r_2))} \left(r_1 \div \pi_B\left(\sigma_{C=t}\left(r_2\right)\right)\right) \times (t) \\
&= \bigcup_{t \in \sigma_{p(C)}(\pi_C(r_2))} \left(r_1 \div \right. \\
&\qquad \left. \pi_B\left(\sigma_{C=t}\left(\sigma_{p(C)}\left(r_2\right)\right)\right)\right) \times (t) \\
&= \bigcup_{t \in \pi_C\left(\sigma_{p(C)}(r_2)\right)} \left(r_1 \div \right. \\
&\qquad \left. \pi_B\left(\sigma_{C=t}\left(\sigma_{p(C)}\left(r_2\right)\right)\right)\right) \times (t) \\
&= r_1 \div^* \sigma_{p(C)}\left(r_2\right) \qquad \text{(Definition 4)}
\end{aligned}
$$

□

PROOF (LAW 16):

$$
\begin{aligned}
&\sigma_{p(B)}\left(r_1\right) \div^* \sigma_{p(B)}\left(r_2\right) \\
&= \bigcup_{t \in \pi_C\left(\sigma_{p(B)}(r_2)\right)} \left(\sigma_{p(B)}\left(r_1\right) \div \right. \\
&\qquad \left. \pi_B\left(\sigma_{C=t}\left(\sigma_{p(B)}\left(r_2\right)\right)\right)\right) \times (t) \\
&\qquad \text{(Definition 4)} \\
&= \bigcup_{t \in \pi_C\left(\sigma_{p(B)}(r_2)\right)} \left(\sigma_{p(B)}\left(r_1\right) \div \right. \\
&\qquad \left. \pi_B\left(\sigma_{p(B)}\left(\sigma_{C=t}\left(r_2\right)\right)\right)\right) \times (t) \\
&= \bigcup_{t \in \pi_C\left(\sigma_{p(B)}(r_2)\right)} \left(\sigma_{p(B)}\left(r_1\right) \div \right. \\
&\qquad \left. \sigma_{p(B)}\left(\pi_B\left(\sigma_{C=t}\left(r_2\right)\right)\right)\right) \times (t) \\
&= \bigcup_{t \in \pi_C\left(\sigma_{p(B)}(r_2)\right)} \left(r_1 \div \right. \\
&\qquad \left. \sigma_{p(B)}\left(\pi_B\left(\sigma_{C=t}\left(r_2\right)\right)\right)\right) \times (t) \qquad \text{(Law 4)} \\
&= \bigcup_{t \in \pi_C\left(\sigma_{p(B)}(r_2)\right)} \left(r_1 \div \right. \\
&\qquad \left. \pi_B\left(\sigma_{p(B)}\left(\sigma_{C=t}\left(r_2\right)\right)\right)\right) \times (t) \\
&= \bigcup_{t \in \pi_C\left(\sigma_{p(B)}(r_2)\right)} \left(r_1 \div \right. \\
&\qquad \left. \pi_B\left(\sigma_{C=t}\left(\sigma_{p(B)}\left(r_2\right)\right)\right)\right) \times (t) \\
&= r_1 \div^* \sigma_{p(B)}\left(r_2\right) \qquad \text{(Definition 4)}
\end{aligned}
$$

□

PROOF (LAW 17): We use the following algebraic laws as propositions:

(P1) $\left(r_1 \times r_2\right) \times r_3 = r_1 \times \left(r_2 \times r_3\right)$

(P2) $r_1 \times \left(r_2 \cup r_3\right) = \left(r_1 \times r_2\right) \cup \left(r_1 \times r_3\right)$

$$
\begin{aligned}
&\left(r_1^* \times r_1^{**}\right) \div^* r_2 \\
&\qquad \text{(Definition 4)} \\
&= \bigcup_{t \in \pi_C(r_2)} \left(\left(r_1^* \times r_1^{**}\right) \div \pi_B\left(\sigma_{C=t}\left(r_2\right)\right)\right) \times (t) \\
&= \bigcup_{t \in \pi_C(r_2)} \left(r_1^* \times \left(r_1^{**} \div \pi_B\left(\sigma_{C=t}\left(r_2\right)\right)\right)\right) \times (t) \\
&\qquad \text{(Law 8)} \\
&= \bigcup_{t \in \pi_C(r_2)} r_1^* \times \left(\left(r_1^{**} \div \pi_B\left(\sigma_{C=t}\left(r_2\right)\right)\right) \times (t)\right) \\
&\qquad \text{(P1)} \\
&= r_1^* \times \bigcup_{t \in \pi_C(r_2)} \left(r_1^{**} \div \pi_B\left(\sigma_{C=t}\left(r_2\right)\right)\right) \times (t) \\
&\qquad \text{(P2)} \\
&= r_1^* \times \left(r_1^{**} \div^* r_2\right) \\
&\qquad \text{(Definition 4)}
\end{aligned}
$$

□