Optimizing multiset relational algebra queries using weak-equivalent rewrite rules

Jelle Hellings[1], Yuqing Wu[2], Dirk Van Gucht[3], and Marc Gyssens[4]

[1] McMaster University, 1280 Main St. W., Hamilton, ON L8S 4L7, Canada
[2] Pomona College, 185 E 6th St., Claremont, CA 91711, USA
[3] Indiana University, 919 E 10th St., Bloomington, IN 47408, USA
[4] Hasselt University, Martelarenlaan 42, 3500, Hasselt, Belgium

Abstract. Relational query languages rely heavily on costly join operations to combine tuples from multiple tables into a single resulting tuple. In many cases, the cost of query evaluation can be reduced by *manually optimizing* (parts of) queries to use cheaper semi-joins instead of joins. Unfortunately, existing database products can only apply such optimizations *automatically* in rather limited cases.
To improve on this situation, we propose a framework for automatic query optimization via *weak-equivalent rewrite rules* for a multiset relational algebra (that serves as a faithful formalization of core SQL). The weak-equivalent rewrite rules we propose aim at replacing joins by semi-joins. To further maximize their usability, these rewrite rules do so by only providing "weak guarantees" on the evaluation results of rewritten queries. We show that, in the context of certain operators, these weak-equivalent rewrite rules still provide strong guarantees on the final evaluation results of the rewritten queries.

Keywords: Query optimization · Relational algebra · Multiset semantics · Semi-joins

1 Introduction

To combine tables, SQL relies on join operations that are costly to evaluate. To reduce the high costs of joins, a significant portion of query optimization and query planning is aimed at evaluating joins as efficient as possible. Still, it is well-known that some complex join-based SQL queries can be further optimized manually by rewriting these queries into queries that involve semi-joins implicitly. As an illustration, we consider the following example involving a graph represented as a binary relation. The SQL query

```
SELECT DISTINCT S.nfrom
FROM edges S, edges R, edges T, edges U
WHERE S.nto = R.nfrom AND R.nto = T.nfrom AND T.nto = U.nfrom;
```

computes the sources of paths of length four. A straightforward way to evaluate this query is to compute the joins, then project the join result onto $S.nfrom$,

and, finally, remove duplicates. The cost of this straightforward evaluation is very high: in the worst-case, the join result is quartic in size with respect to the size of the number of edges (the size of the *Edges* relation). The typical way to manually optimize this query is to rewrite it as follows:

```
SELECT DISTINCT nfrom FROM edges
WHERE nto IN (SELECT nfrom FROM edges
              WHERE nto IN (SELECT nfrom FROM edges
                            WHERE nto IN (SELECT nfrom
                                          FROM edges)));
```

In most relational database systems, the `WHERE ... IN ...` clauses in the rewritten query are evaluated using a semi-join algorithm. In doing so, this query can be evaluated in linear time with respect to the number of edges, which is a significant improvement. E.g., when evaluated on a randomly generated database with 75 000 rows (each row a single edge) managed by PostgreSQL 14.1, the original query could not finish in a reasonable amount of time, whereas the manually rewritten semi-join style query finished in 90 ms.

We believe that query optimizers should not require users to manually rewrite queries to enforce particular evaluation strategies: manual rewriting goes against the advantages of using high-level declarative languages such as SQL. Instead, we want query optimizers to be able to recognize situations in which semi-join rewriting is appropriate, and apply such optimizing rewritings automatically.

Traditional approaches towards query optimization for SQL and the relational algebra usually employ two basic steps [12]. First, the query involved is rewritten. The rewrite rules used in these rewrites guarantee *strong-equivalence*: the original subquery and the rewritten subquery always evaluate to the same result. Examples of such rules are the well-known push-down rules for selection and projection, which can reduce the size of intermediate query results significantly. Second, the order of execution of the operations, appropriate algorithms to perform each operation, and data access methods are chosen to evaluate the rewritten query.

Unfortunately, requiring strong-equivalence imposes a severe restriction on the rewrite rules that can be considered. As a consequence, there are significant limitations to query optimization using traditional query rewriting: often, these rewrite rules only manipulate the order of operations. More lucrative rewritings, such as replacing expensive join operations by semi-joins, are not considered because such rewrites cannot guarantee strong-equivalence.

To improve on this situation, we propose the concept of *weak-equivalence*, which is a relaxation of strong-equivalence. Weak-equivalent rewrite rules only guarantee that the original subquery and the rewritten subquery evaluate to the same result up to duplicate elimination (with respect to the attributes of interest). The rewrite rules we propose are aimed at eliminating joins in favor of semi-joins and eliminating the need for deduplication altogether. To illustrate the benefits of weak-equivalent rewrites, we present two examples.

As a first example, consider a university database containing the relation *Course*, with attributes *id* and *name*, and the relation *Enroll*, with, among its

attributes, *cid* (the course id). Other attributes of *Enroll* refer to students. Now consider the task of rewriting the relational algebra query

$$\pi_{C.name}(q) \text{ with } q = \rho_C(Course) \bowtie_{C.id=E.cid} \rho_E(Enroll)$$

that computes the set of names of courses in which students are effectively enrolled. As the end result of this query is a projection on *only* the *name* attribute of *Course* (which we assume to be a key), any rewrite of the subquery q can forgo any of the other attributes. E.g., although subquery q yields a completely different result than subquery $q' = \rho_C(Course) \ltimes_{C.id=E.cid} \rho_E(Enroll)$, their projection onto $C.name$ is identical.

As a second example, consider a sales database containing the relation *Customer*, with among its attributes *cname*, the relation *Product*, with among its attributes *pname* and *type*, and the relation *Bought*, with among its attributes *cname* and *pname*. We refer to Figure 1 for an instance of this sales database. Now consider the following query:

```
SELECT DISTINCT C.cname, P.type
FROM customer C, bought B, product P
WHERE C.cname = B.cname AND B.pname = P.pname AND
      P.type = 'food';
```

which our rules can rewrite into the following query:

```
SELECT cname, 'food' AS type FROM customer WHERE cname IN (
    SELECT cname FROM bought WHERE pname IN (
        SELECT pname FROM product WHERE type = 'food'));
```

Observe that the rewritten query does not perform any joins, even though information from several relations is combined and returned. Moreover, the need for deduplication is eliminated, as the available key on *cname* guarantees that no duplicates are possible in the resultant query. In this particular example, the original query joins table *Bought* with two tables on their primary keys. Hence, all intermediate results remain small if query evaluation chooses a proper join order. Still, even in this case, the rewritten query evaluates 15%–20% faster on a randomly generated database with 500 customers, 24 077 products, and 100 000 sale records in *Bought*.

The latter example also illustrates that the applicability of weak-equivalent rewrite rules may depend on structural properties, such as keys, of the input relations and of the intermediate query results. Therefore, we also propose techniques to derive these properties from the available schema information.

The automatic use of semi-join algorithms in query evaluation has already been studied and applied before. In these approaches, however, semi-joins are typically only employed as a preprocessing step for joins. E.g., in distributed databases, semi-joins are used as an intermediate step to reduce the size of relations before joining them and, as a consequence, reducing the communication overhead of distributing (intermediate) relations to other computational nodes [3]. The semi-join has a similar role in the well-known acyclic multi-join algorithm

Customer	
cname	*age*
Alice	19
Bob	20
Eve	21

Product	
pname	*type*
apple	food
apple	fruit
banana	fruit
car	non-food

Bought		
cname	*pname*	*price*
Alice	apple	0.35
Alice	apple	0.35
Bob	apple	0.45
Bob	banana	0.50
Eve	car	10000

Fig. 1. A database instance for a sales database that has customers, a categorization of products according to their types, and transaction information for each sale.

of Yannakakis [13]. We take a different approach: using semi-joins, we aim at eliminating join operations altogether instead of merely reducing their cost.

The usage of weak-equivalent rewrite rules for query optimization is inspired by the projection-equivalent rewriting techniques for graph query languages on binary relations proposed by Hellings et al [6, 7]. In the current paper, we not only adapt these projection-equivalent rewriting techniques to the setting of relational algebra, but we also extend them to effectively deal with multiset semantics [2, 4, 5, 9–11]. The latter is essential, because we want to use our weak-equivalent rewriting rules to optimize SQL queries, and SQL has multiset semantics. Furthermore, we integrate rewrite techniques based on derived structural knowledge on the schema and knowledge derived from selection conditions, which are both not applicable in the setting of binary relations.

Finally, we note that there have been previous SQL-inspired studies for rewriting and optimizing relational algebra with multiset semantics, e.g., [5, 11]. These studies do not cover the main optimizations explored in this work, however.

2 Preliminaries

We consider disjoint infinitely enumerable sets $\mathfrak{U}$ and $\mathfrak{N}$ of *constants* and *names*, respectively. A *relation schema* is a finite set of names, which are used as attributes. Let $\mathrm{A} \subseteq \mathfrak{N}$ be a relation schema. A *tuple over* A is a function $\mathtt{t} : \mathrm{A} \to \mathfrak{U}$, a *relation over* $\mathrm{A} \subseteq \mathfrak{N}$ is a set of tuples over A, and a *multiset relation over* A is a pair $\langle \mathscr{R}; \tau \rangle$, in which $\mathscr{R}$ is a relation over A and $\tau : \mathscr{R} \to \mathbb{N}^+$ is a function mapping tuples $\mathtt{t} \in \mathscr{R}$ to the number of copies of $\mathtt{t}$ in the multiset relation. We say that $\langle \mathscr{R}; \tau \rangle$ is a *set relation* if, for every $\mathtt{t} \in \mathscr{R}$, we have $\tau(\mathtt{t}) = 1$. We write $(\mathtt{t} : n) \in \langle \mathscr{R}; \tau \rangle$ for *tuple-count pair* $(\mathtt{t} : n)$ to indicate $\mathtt{t} \in \mathscr{R}$ with $\tau(\mathtt{t}) = n$. We write $\mathtt{t} \in \langle \mathscr{R}; \tau \rangle$ to indicate $\mathtt{t} \in \mathscr{R}$ and we write $\mathtt{t} \notin \langle \mathscr{R}; \tau \rangle$ to indicate $\mathtt{t} \notin \mathscr{R}$.

Let $\mathtt{t}$, $\mathtt{t}_1$, and $\mathtt{t}_2$ be tuples over the relation schema A and let $\mathrm{B} \subseteq \mathrm{A}$. The *restriction of* $\mathtt{t}$ *to* B, denoted by $\mathtt{t}|_{\mathrm{B}}$, is defined by $\mathtt{t}|_{\mathrm{B}} = \{a \mapsto \mathtt{t}(a) \mid a \in \mathrm{B}\}$. Tuples $\mathtt{t}_1$ and $\mathtt{t}_2$ *agree on* B, denoted by $\mathtt{t}_1 \equiv_{\mathrm{B}} \mathtt{t}_2$, if $\mathtt{t}_1|_{\mathrm{B}} = \mathtt{t}_2|_{\mathrm{B}}$. Let $\langle \mathscr{R}; \tau \rangle$ be a multiset relation over relation schema A and let $\mathrm{K} \subseteq \mathrm{A}$. We say that K is a *key* of $\langle \mathscr{R}; \tau \rangle$ if, for every pair of tuples $\mathtt{t}_1, \mathtt{t}_2 \in \langle \mathscr{R}; \tau \rangle$ with $\mathtt{t}_1 \equiv_{\mathrm{K}} \mathtt{t}_2$, we have $\mathtt{t}_1 = \mathtt{t}_2$.

A *database schema* is a 4-tuple $\mathfrak{D} = (\mathbf{N}, \mathbf{A}, \mathbf{K}, \mathbf{S})$, in which $\mathbf{N} \subseteq \mathfrak{N}$ is a set of *relation names*, $\mathbf{A}$ is a function mapping each relation name to a relation schema,

$\mathbf{K}$ is a function mapping each relation name to a set of sets of attributes of the corresponding relation schema, and $\mathbf{S}$ is a function mapping relation names to booleans. A *database instance* over schema $\mathfrak{D} = (\mathbf{N}, \mathbf{A}, \mathbf{K}, \mathbf{S})$ is a function $\mathfrak{I}$ mapping each name $R \in \mathbf{N}$ into a multiset relation. This multiset relation $\mathfrak{I}(R)$ has schema $\mathbf{A}(R)$, keys $\mathbf{K}(R)$, and must be a set relation if $\mathbf{S}(R) = \texttt{true}$.

Let $\mathtt{t}_1$ and $\mathtt{t}_2$ be tuples over relation schemas A_1 and A_2, respectively, such that $\mathtt{t}_1 \equiv_{\mathrm{A}_1 \cap \mathrm{A}_2} \mathtt{t}_2$. The *concatenation of* $\mathtt{t}_1$ *and* $\mathtt{t}_2$, denoted by $\mathtt{t}_1 \cdot \mathtt{t}_2$, is defined by $\mathtt{t}_1 \cdot \mathtt{t}_2 = \mathtt{t}_1 \cup \mathtt{t}_2$.[5] Notice that $\mathtt{t}_1 \cdot \mathtt{t}_2$ is a tuple over $\mathrm{A}_1 \cup \mathrm{A}_2$. Let A be a relation schema. A *condition on* A is either an expression of the form $a_1 = a_2$ or of the form $a = \mathtt{u}$, $a, a_1, a_2 \in \mathrm{A}$ and $\mathtt{u} \in \mathfrak{U}$. A tuple $\mathtt{t}$ over A satisfies $a_1 = a_2$ if $\mathtt{t}(a_1) = \mathtt{t}(a_2)$ and satisfies $a = \mathtt{u}$ if $\mathtt{t}(a) = \mathtt{u}$. If E is a set of conditions over A, then tuple $\mathtt{t}$ *satisfies* E if $\mathtt{t}$ satisfies each condition in E. By $\mathrm{attrs}(E)$, we denote the set of attributes used in conditions in E.

Example 1. The database schema for the sales database of Figure 1 consist of the named multiset relations *Customer* with schema $\{cname, age\}$, *Product* with schema $\{pname, type\}$, and *Bought* with schema $\{cname, pname, price\}$. *Customer* and *Product* are set relations with key $\{cname\}$ and trivial key $\{pname, type\}$, respectively. *Bought* is not a set relation and only has the trivial key $\{cname, pname, price\}$.

3 Multiset relational algebra

In this work, we study the multiset relational algebra. To distinguish a traditional set-based relational algebra operator $\oplus$ from its multiset relational algebra counterpart introduced below, we annotate the latter as $\dot{\oplus}$.

Let $\mathfrak{D} = (\mathbf{N}, \mathbf{A}, \mathbf{K}, \mathbf{S})$ be a database schema and let $\mathfrak{I}$ be a database instance over $\mathfrak{D}$. If e is a multiset relational algebra expression over $\mathfrak{D}$, which we will formally define next, then we write $\mathcal{S}(e; \mathfrak{D})$ to denote the schema of the multiset relation obtained by evaluating e on an instance over $\mathfrak{D}$, and we write $[\![e]\!]_{\mathfrak{I}}$ to denote the evaluation of e on instance $\mathfrak{I}$. The *standard* relational algebra expressions over $\mathfrak{D}$ are built from the following operators:

- *Multiset relation.* If $R \in \mathbf{N}$ is a relation name, then R is an expression with $\mathcal{S}(R; \mathfrak{D}) = \mathbf{A}(R)$ and $[\![R]\!]_{\mathfrak{I}} = \mathfrak{I}(R)$.
- *Selection.*[6] If e is an expression and E is a set of conditions on $\mathcal{S}(e; \mathfrak{D})$, then $\dot{\sigma}_E(e)$ is an expression with $\mathcal{S}(\dot{\sigma}_E(e); \mathfrak{D}) = \mathcal{S}(e; \mathfrak{D})$ and $[\![\dot{\sigma}_E(e)]\!]_{\mathfrak{I}} = \{(\mathtt{t} : n) \in [\![e]\!]_{\mathfrak{I}} \mid (\mathtt{t} \text{ satisfies } E)\}$.
- *Projection.* If e is an expression and $\mathrm{B} \subseteq \mathcal{S}(e; \mathfrak{D})$, then $\dot{\pi}_{\mathrm{B}}(e)$ is an expression with $\mathcal{S}(\dot{\pi}_{\mathrm{B}}(e); \mathfrak{D}) = \mathrm{B}$ and $[\![\dot{\pi}_{\mathrm{B}}(e)]\!]_{\mathfrak{I}} = \{(\mathtt{t}|_{\mathrm{B}} : \mathrm{count}(\mathtt{t}|_{\mathrm{B}}, e)) \mid \mathtt{t} \in [\![e]\!]_{\mathfrak{I}}\}$, in which $\mathrm{count}(\mathtt{t}|_{\mathrm{B}}, e) = \sum_{((\mathtt{s}:m) \in [\![e]\!]_{\mathfrak{I}}) \wedge (\mathtt{s} \equiv_{\mathrm{B}} \mathtt{t})} m$.

[5] Every occurrence of the operators $\cup$, $\cap$, and $-$ in this paper is to be interpreted using standard set semantics.

[6] We only consider conjunctions of conditions in the selection operator because more general boolean combinations of conditions do not provide additional opportunities for rewriting in our framework.

– *Renaming.* If e is an expression, $\mathrm{B} \subseteq \mathfrak{N}$, and $f : \mathcal{S}(e;\mathfrak{D}) \to \mathrm{B}$ is a bijection, then $\dot{\rho}_f(e)$ is an expression with $\mathcal{S}(\dot{\rho}_f(e);\mathfrak{D}) = \mathrm{B}$ and $[\![\dot{\rho}_f(e)]\!]_{\mathfrak{I}} = \{(\mathrm{rename}(\mathtt{t}, f) : m) \mid (\mathtt{t} : m) \in [\![e]\!]_{\mathfrak{I}}\}$, in which $\mathrm{rename}(\mathtt{t}, f) = \{f(a) \mapsto \mathtt{t}(a) \mid a \in \mathcal{S}(e;\mathfrak{D})\}$.
– *Deduplication.* If e is an expression, then $\dot{\delta}(e)$ is an expression with $\mathcal{S}(\dot{\delta}(e);\mathfrak{D}) = \mathbf{A}(e)$ and $[\![\dot{\delta}(e)]\!]_{\mathfrak{I}} = \{(\mathtt{t} : 1) \mid (\mathtt{t} : n) \in [\![e]\!]_{\mathfrak{I}}\}$.
– *Union, intersection, and difference.*[7] If e_1 and e_2 are expressions such that $\mathrm{A} = \mathcal{S}(e_1;\mathfrak{D}) = \mathcal{S}(e_2;\mathfrak{D})$, then, for $\oplus \in \{\cup, \cap, -\}$, $e_1 \mathbin{\dot{\oplus}} e_2$ is an expression with $\mathcal{S}(e_1 \mathbin{\dot{\oplus}} e_2;\mathfrak{D}) = \mathrm{A}$ and

$$
\begin{aligned}
[\![e_1 \mathbin{\dot{\cup}} e_2]\!]_{\mathfrak{I}} = {} & \{(\mathtt{t} : n_1 + n_2) \mid (\mathtt{t} : n_1) \in [\![e_1]\!]_{\mathfrak{I}} \wedge (\mathtt{t} : n_2) \in [\![e_2]\!]_{\mathfrak{I}}\} \cup \\
& \{(\mathtt{t} : n) \in [\![e_1]\!]_{\mathfrak{I}} \mid \mathtt{t} \notin [\![e_2]\!]_{\mathfrak{I}}\} \cup \{(\mathtt{t} : n) \in [\![e_2]\!]_{\mathfrak{I}} \mid \mathtt{t} \notin [\![e_1]\!]_{\mathfrak{I}}\}; \\
[\![e_1 \mathbin{\dot{\cap}} e_2]\!]_{\mathfrak{I}} = {} & \{(\mathtt{t} : \min(n_1, n_2)) \mid (\mathtt{t} : n_1) \in [\![e_1]\!]_{\mathfrak{I}} \wedge (\mathtt{t} : n_2) \in [\![e_2]\!]_{\mathfrak{I}}\}; \\
[\![e_1 \mathbin{\dot{-}} e_2]\!]_{\mathfrak{I}} = {} & \{(\mathtt{t} : n) \in [\![e_1]\!]_{\mathfrak{I}} \mid \mathtt{t} \notin [\![e_2]\!]_{\mathfrak{I}}\} \cup \\
& \{(\mathtt{t} : n_1 - n_2) \mid (n_1 > n_2) \wedge (\mathtt{t} : n_1) \in [\![e_1]\!]_{\mathfrak{I}} \wedge (\mathtt{t} : n_2) \in [\![e_2]\!]_{\mathfrak{I}}\}.
\end{aligned}
$$

– *θ-join and natural join.*[8] If e_1 and e_2 are expressions and E is a set of conditions on $\mathrm{A} = \mathcal{S}(e_1;\mathfrak{D}) \cup \mathcal{S}(e_2;\mathfrak{D})$, then $e_1 \mathbin{\dot{\bowtie}_E} e_2$ is an expression with $\mathcal{S}(e_1 \mathbin{\dot{\bowtie}_E} e_2;\mathfrak{D}) = \mathrm{A}$ and $[\![e_1 \mathbin{\dot{\bowtie}_E} e_2]\!]_{\mathfrak{I}}$ is defined by

$$
\begin{aligned}
\{(\mathtt{t}_1 \cdot \mathtt{t}_2 : n_1 \cdot n_2) \mid {} & (\mathtt{t}_1 : n_1) \in [\![e_1]\!]_{\mathfrak{I}} \wedge (\mathtt{t}_2 : n_2) \in [\![e_2]\!]_{\mathfrak{I}} \wedge \\
& \mathtt{t}_1 \equiv_{\mathcal{S}(e_1;\mathfrak{D}) \cap \mathcal{S}(e_2;\mathfrak{D})} \mathtt{t}_2 \wedge (\mathtt{t}_1 \cdot \mathtt{t}_2 \text{ satisfies } E)\}.
\end{aligned}
$$

If $E = \emptyset$, then we simply write $e_1 \mathbin{\dot{\bowtie}} e_2$ (the *natural join*).

Example 2. Consider the database instance $\mathfrak{I}$ of Example 1, which is visualized in Figure 1. The expression $e = \dot{\pi}_{age}(\dot{\sigma}_{type=\texttt{non-food}}(\mathit{Customer} \mathbin{\dot{\bowtie}} \mathit{Bought} \mathbin{\dot{\bowtie}} \mathit{Product}))$ returns the ages of people that bought non-food products: as Eve bought a car, we have $[\![e]\!]_{\mathfrak{I}} = \{(\mathit{age} \mapsto \mathtt{21}\!:\!1)\}$. If we change the condition $\mathit{type} = \texttt{non-food}$ into $\mathit{type} = \texttt{food}$, resulting in the expression e', then $[\![e']\!]_{\mathfrak{I}} = \{(\mathit{age} \mapsto \mathtt{19} : 2), (\mathit{age} \mapsto \mathtt{20} : 1)\}$, which includes Alice's age twice as she bought two apples. Observe that we have $[\![\dot{\delta}(e')]\!]_{\mathfrak{I}} = \{(\mathit{age} \mapsto \mathtt{19} : 1), (\mathit{age} \mapsto \mathtt{20} : 1)\}$.

The extended multiset relational algebra. We aim at reducing the complexity of query evaluation by rewriting joins into semi-joins, which are not part of the standard relational algebra described above. *Extended* relational algebra expressions over $\mathfrak{D}$ are built from the standard relational algebra operators and the following additional ones:

– *θ-semi-join and semi-join.*[8] If e_1 and e_2 are expressions and E is a set of conditions on $\mathcal{S}(e_1;\mathfrak{D}) \cup \mathcal{S}(e_2;\mathfrak{D})$, then $e_1 \mathbin{\dot{\ltimes}_E} e_2$ is an expression with $\mathcal{S}(e_1 \mathbin{\dot{\ltimes}_E} e_2;\mathfrak{D}) = \mathcal{S}(e_1;\mathfrak{D})$ and $[\![e_1 \mathbin{\dot{\ltimes}_E} e_2]\!]_{\mathfrak{I}}$ is defined by

[7] The set operators we define have the same semantics as the `UNION ALL`, `INTERSECT ALL`, and `EXCEPT ALL` operators of standard SQL [8]. The semantics of `UNION`, `INTERSECT`, and `EXCEPT` can be obtained using deduplication.

[8] To simplify presentation, the θ-join and the θ-semi-join also perform equi-join on all attributes common to the multiset relations involved.

$$\{(\mathtt{t}_1 : n_1) \in [\![e_1]\!]_{\mathfrak{I}} \mid \exists \mathtt{t}_2\ (\mathtt{t}_2 \in [\![e_2]\!]_{\mathfrak{I}} \wedge \mathtt{t}_1 \equiv_{\mathcal{S}(e_1;\mathfrak{D})\cap\mathcal{S}(e_2;\mathfrak{D})} \mathtt{t}_2 \wedge (\mathtt{t}_1 \cdot \mathtt{t}_2 \text{ satisfies } E))\}.$$

If $E = \emptyset$, then we simply write $e_1 \ltimes e_2$ (the *semi-join*).

– *Max-union.*[9] If e_1 and e_2 are expressions such that $\text{A} = \mathcal{S}(e_1;\mathfrak{D}) = \mathcal{S}(e_2;\mathfrak{D})$, then $e_1 \dot\cup e_2$ is an expression with $\mathcal{S}(e_1 \dot\cup e_2;\mathfrak{D}) = \text{A}$ and

$$[\![e_1 \dot\cup e_2]\!]_{\mathfrak{I}} = \{(\mathtt{t} : \max(n_1, n_2)) \mid (\mathtt{t} : n_1) \in [\![e_1]\!]_{\mathfrak{I}} \wedge (\mathtt{t} : n_2) \in [\![e_2]\!]_{\mathfrak{I}}\} \cup \{(\mathtt{t} : n) \in [\![e_1]\!]_{\mathfrak{I}} \mid \mathtt{t} \notin [\![e_2]\!]_{\mathfrak{I}}\} \cup \{(\mathtt{t} : n) \in [\![e_2]\!]_{\mathfrak{I}} \mid \mathtt{t} \notin [\![e_1]\!]_{\mathfrak{I}}\};$$

– *Attribute introduction.*[10] If e is an expression, B is a set of attributes with $\text{B} \cap \mathcal{S}(e;\mathfrak{D}) = \emptyset$, and $f = \{b := x \mid b \in \text{B} \wedge x \in (\mathcal{S}(e;\mathfrak{D}) \cup \mathfrak{U})\}$ is a set of assignment-pairs, then $i_f(e)$ is an expression with $\mathcal{S}(i_f(e);\mathfrak{D}) = \mathcal{S}(e;\mathfrak{D}) \cup \text{B}$ and $[\![i_f(e)]\!]_{\mathfrak{I}} = \{(\mathtt{t} \cdot \{\text{B} \mapsto \text{value}(\mathtt{t}, x) \mid (b := x) \in f\} : m) \mid (\mathtt{t} : m) \in [\![e]\!]_{\mathfrak{I}}\}$, in which $\text{value}(\mathtt{t}, x) = \mathtt{t}(x)$ if $x \in \mathcal{S}(e;\mathfrak{D})$ and $\text{value}(\mathtt{t}, x) = x$ otherwise.

The extended relational algebra provides all the operators used in our framework and the following example illustrates the use of these operators:

Example 3. Consider the database instance $\mathfrak{I}$ of Examples 1 and 2. Let $e = \dot\sigma_{type=\mathtt{food}}(\mathit{Customer} \bowtie \mathit{Bought} \bowtie \mathit{Product})$. The query $\dot\delta(\dot\pi_{age}(e))$ is equivalent to $\dot\delta(\dot\pi_{age}(\mathit{Customer} \ltimes (\mathit{Bought} \ltimes_{type=\mathtt{food}} \mathit{Product})))$. The query $\dot\delta(\dot\pi_{cname,type}(e))$ is equivalent to $i_{type:=\mathtt{food}}(\dot\pi_{cname}(\mathit{Customer} \ltimes (\mathit{Bought} \ltimes_{type=\mathtt{food}} \mathit{Product})))$. Notice that we were able to eliminate joins altogether in these rewritings. In the latter rewriting, we were also able to eliminate deduplication, even though attributes from several relations are involved. This is because *cname* is a key of the set relation *Customer*.

4 Rewriting queries

Traditional rewrite rules for optimizing queries, such as the selection and projection push-down rewrite rules, guarantee *strong-equivalence*: the original subquery and the rewritten subquery always evaluate to the same result. Requiring this strong form of equivalence severely limits the optimizations one can perform in queries that involve projection and/or deduplication steps, as is illustrated in the following example:

Example 4. Consider the database instance $\mathfrak{I}$ of Examples 1–3. The query $e = \dot\delta(\dot\pi_{cname}(\mathit{Customer} \bowtie \mathit{Bought}))$ returns the names of customers that bought a product. This query is equivalent to $e' = \dot\delta(\dot\pi_{cname}(\mathit{Customer}) \ltimes \mathit{Bought})$. Observe that the subqueries $\mathit{Customer} \bowtie \mathit{Bought}$ and $\dot\pi_{cname}(\mathit{Customer} \bowtie \mathit{Bought})$ of e

[9] The max-union operators is inspired by the max-based multiset relation union [2].

[10] Attribute introduction is a restricted form of the operator commonly known as *generalized projection* or *extended projection* [1, 5].

are not equivalent to any subqueries of e', however. Hence, this rewriting cannot be achieved using strong-equivalent rewriting only. Finally, as $\{cname\}$ is a key of *Customer*, e and e' are also equivalent to $e'' = \dot{\pi}_{cname}(Customer) \ltimes Bought$.

To be able to discuss the full range of possible optimizations within the scope of projection and deduplication operations, we differentiate between the following notions of query equivalence:

Definition 1. *Let e_1 and e_2 be multiset relational algebra expressions over $\mathfrak{D}$ with $\mathrm{A} = \mathcal{S}(e_1; \mathfrak{D}) \cap \mathcal{S}(e_2; \mathfrak{D})$, and let $\mathrm{B} \subseteq \mathrm{A}$. We say that e_1 and e_2 are* strong-equivalent*, denoted by $e_1 \doteq e_2$, if, for every database instance $\mathfrak{I}$ over $\mathfrak{D}$, we have $[\![e_1]\!]_{\mathfrak{I}} = [\![e_2]\!]_{\mathfrak{I}}$. We say that e_1 and e_2 are* weak-equivalent*, denoted by $e_1 \mathrel{\dot{=}} e_2$, if, for every database instance $\mathfrak{I}$ over $\mathfrak{D}$, we have $[\![\dot{\delta}(e_1)]\!]_{\mathfrak{I}} = [\![\dot{\delta}(e_2)]\!]_{\mathfrak{I}}$. We say that e_1 and e_2 are* strong-B-equivalent*, denoted by $e_1 \doteq_{\mathrm{B}} e_2$, if, for every database instance $\mathfrak{I}$ over $\mathfrak{D}$, we have $[\![\dot{\pi}_{\mathrm{B}}(e_1)]\!]_{\mathfrak{I}} = [\![\dot{\pi}_{\mathrm{B}}(e_2)]\!]_{\mathfrak{I}}$. Finally, we say that e_1 and e_2 are* weak-B-equivalent*, denoted by $e_1 \mathrel{\dot{=}}_{\mathrm{B}} e_2$, if, for every database instance $\mathfrak{I}$ over $\mathfrak{D}$, we have $[\![\dot{\delta}(\dot{\pi}_{\mathrm{B}}(e_1))]\!]_{\mathfrak{I}} = [\![\dot{\delta}(\dot{\pi}_{\mathrm{B}}(e_2))]\!]_{\mathfrak{I}}$.*

While strong-equivalent expressions always yield the same result, weak-equivalent expressions yield the same result only up to duplicate elimination. For query optimization, the latter is often sufficient at the level of subqueries: structural properties, such as the presence of a key in one of the relations involved, may have as a side effect that any duplicates in subqueries are eliminated in the end result anyway.

Example 5. Consider the queries of Example 4. We have $e \doteq e' \doteq e''$, $e \doteq_{\{cname\}} Customer \ltimes Bought$, $Customer \bowtie Bought \mathrel{\dot{=}}_{\{cname,age\}} Customer \ltimes Bought$, and $\dot{\pi}_{cname}(Customer \bowtie Bought) \mathrel{\dot{=}}_{\{cname\}} \dot{\pi}_{cname}(Customer) \ltimes Bought$.

Examples 4 and 5 not only show the relevance of non-strong-equivalent rewriting rules, they also show that further optimizations are possible if the expressions evaluate to set relations or satisfy certain keys. Therefore, to facilitate the discussion, we extend the definition of set relations and keys to expressions. Let e be an expression over $\mathfrak{D}$ with $\mathrm{A} = \mathcal{S}(e; \mathfrak{D})$. We say that e is a *set relation* if, for every database instance $\mathfrak{I}$ over $\mathfrak{D}$, $[\![e]\!]_{\mathfrak{I}}$ is a set relation and we say that $\mathrm{B} \subseteq \mathrm{A}$ is a *key* of e if, for every database instance $\mathfrak{I}$ over $\mathfrak{D}$, B is a key of $[\![e]\!]_{\mathfrak{I}}$.

The following simple rules can be derived by straightforwardly applying Definition 1:

Proposition 1. *Let e, e_1, and e_2 be expressions over $\mathfrak{D}$ with $\mathrm{A} = \mathcal{S}(e; \mathfrak{D}) = \mathcal{S}(e_1; \mathfrak{D}) \cap \mathcal{S}(e_2; \mathfrak{D})$ and $\mathrm{B} \subseteq \mathrm{A}$. Let $\mathrel{\hat{=}}$ be either $\doteq$ or $\mathrel{\dot{=}}$. We have:*

(i) *if $e_1 \mathrel{\hat{=}} e_2$, then $e_1 \mathrel{\hat{=}}_{\mathrm{B}} e_2$;*
(ii) *if $e_1 \mathrel{\hat{=}}_{\mathrm{A}} e_2$, then $e_1 \mathrel{\hat{=}} e_2$;*
(iii) *if $\mathrm{C} \subseteq \mathrm{B}$ and $e_1 \mathrel{\hat{=}}_{\mathrm{B}} e_2$, then $e_1 \mathrel{\hat{=}}_{\mathrm{C}} e_2$;*
(iv) *if $e_1 \mathrel{\hat{=}}_{\mathrm{B}} e_2$, then $\dot{\pi}_{\mathrm{B}}(e_1) \mathrel{\hat{=}} \dot{\pi}_{\mathrm{B}}(e_2)$;*
(v) *if $e_1 \doteq_{\mathrm{B}} e_2$, then $e_1 \mathrel{\dot{=}}_{\mathrm{B}} e_2$;*
(vi) *if $e_1 \mathrel{\dot{=}} e_2$, then $\dot{\delta}(e_1) \doteq \dot{\delta}(e_2)$;*

(vii) *if* $e_1 \mathrel{\tilde{=}}_{\mathrm{B}} e_2$*,* e_1 *and* e_2 *are set relations, and* B *is a key of* e_1 *and* e_2*, then* $e_1 \doteq_{\mathrm{B}} e_2$*; and*
(viii) $e \doteq_{\mathrm{B}} \dot{\pi}_{\mathrm{B}}(e)$ *and* $e \mathrel{\tilde{=}} \dot{\delta}(e)$*.*

Proposition 1.*(iii)* allows us to restrict the scope of strong-B-equivalences and weak-B-equivalences to subsets of B.

In the presence of conditions, as enforced by selections, θ-joins, or θ-semi-joins, we can also extend the scope of strong-B-equivalences and weak-B-equivalences. E.g., if $\mathtt{u} \in \mathfrak{U}$ is a constant and e_1 and e_2 are expressions over $\mathfrak{D}$ with $a, a', b \in \mathcal{S}(e_1; \mathfrak{D}) \cap \mathcal{S}(e_2; \mathfrak{D})$ and $e_1 \mathrel{\tilde{=}}_{\{a\}} e_2$, then $\dot{\sigma}_{a=a',b=\mathtt{u}}(e_1) \mathrel{\tilde{=}}_{\{a,a',b\}} \dot{\sigma}_{a=a',b=\mathtt{u}}(e_2)$. Next, we develop the framework for these scope extensions, for which we first introduce the closure of a set of attributes under a set of conditions:

Definition 2. *Let* A *be a relation schema, let* $\mathrm{B} \subseteq \mathrm{A}$*, and let* E *be a set of conditions over* A*. The* closure of B under E*, denoted by* $\mathcal{C}(\mathrm{B}; E)$*, is the smallest superset of* B *such that, for every condition* $(v = w) \in E$ *or* $(w = v) \in E$*,* $v \in \mathcal{C}(\mathrm{B}; E)$ *if and only if* $w \in \mathcal{C}(\mathrm{B}; E)$*.*

If $a \in \mathrm{A}$, then we write $\mathcal{C}(a; E)$ for $\mathcal{C}(\{a\}; E)$.

Notice that, besides attributes, $\mathcal{C}(\mathrm{B}; E)$ may also contain constants. E.g., if $(b = \mathtt{u}) \in E$ for $b \in \mathrm{B}$ and $\mathtt{u} \in \mathfrak{U}$, then $\mathtt{u} \in \mathcal{C}(\mathrm{B}; E)$. We denote $\mathcal{C}(\mathrm{B}; E) \cap \mathrm{A}$, the set of attributes in $\mathcal{C}(\mathrm{B}; E)$, by $\mathrm{attrs}(\mathrm{B}; E)$; and we denote $\mathcal{C}(\mathrm{B}; E) - \mathrm{A}$, the set of constants in $\mathcal{C}(\mathrm{B}; E)$, by $\mathrm{consts}(\mathrm{B}; E)$.

Intuitively, the values of a tuple for the attributes in $\mathrm{attrs}(\mathrm{B}; E)$ are uniquely determined by the values of that tuple for the attributes in B. There may be other attributes $c \notin \mathrm{attrs}(\mathrm{B}; E)$ that can only have a single value, however. E.g., if attribute c is constraint by a constant condition of the form $c = \mathtt{u}$, $\mathtt{u} \in \mathfrak{U}$. The values of a tuple for such attributes c are therefore trivially determined by the values of that tuple for the attributes in B. This observation leads to the following definition:

Definition 3. *Let* A *be a relation schema, let* $\mathrm{B} \subseteq \mathrm{A}$*, and let* E *be a set of conditions over* A*. The set of attributes determined by* B*, denoted by* $\det(\mathrm{B}; E)$*, is defined by* $\mathrm{attrs}(\mathrm{B}; E) \cup \{a \in \mathrm{A} \mid \mathrm{consts}(a; E) \neq \emptyset\}$*.*

The intuition given above behind the notions in Definitions 2 and 3 can be formalized as follows:

Lemma 1. *Let* t *be a tuple over a relation schema* A *and let* E *be a set of conditions over* A*. If* t *satisfies* E*, then, for every* $a \in \mathrm{A}$*, we have*

(i) $\mathsf{t}(a) = \mathsf{t}(b)$ *for every* $b \in \mathrm{attrs}(a; E)$*;*
(ii) $\mathsf{t}(a) = \mathtt{u}$ *for every* $\mathtt{u} \in \mathrm{consts}(a; E)$*;*
(iii) $|\mathrm{consts}(a; E)| \leq 1$*; and*
(iv) $\mathsf{t} \equiv_{\det(\mathrm{B}; E)} \mathsf{t}'$ *if* $\mathrm{B} \subseteq \mathrm{A}$*,* t' *is a tuple over* A*,* t' *satisfies* E*, and* $\mathsf{t} \equiv_{\mathrm{B}} \mathsf{t}'$*.*

Using Lemma 1, we prove the following rewrite rules for selection:

Theorem 1. *Let g and h be expressions over $\mathfrak{D}$ with $\mathrm{G} = \mathcal{S}(g;\mathfrak{D})$ and $\mathrm{H} = \mathcal{S}(h;\mathfrak{D})$, let E be a set of conditions over G, let $\mathrm{A} \subseteq (\mathrm{H} \cap \mathrm{attrs}(E))$, let $\mathrm{B} \subseteq (\det(\mathrm{A};E) - \mathrm{H})$, and let f be a set of assignment-pairs such that, for each $b \in \mathrm{B}$, there exists $(b := x) \in f$ with $x \in (\mathcal{C}(b;E) \cap (\mathrm{A} \cup \mathfrak{U}))$. We have*

(i) *if $\dot{\sigma}_E(g) \tilde{=}_{\mathrm{A}} h$, then $\dot{\sigma}_E(g) \tilde{=}_{\mathrm{A}\cup\mathrm{B}} i_f(h)$; and*
(ii) *if $\dot{\sigma}_E(g) \doteq_{\mathrm{A}} h$, then $\dot{\sigma}_E(g) \doteq_{\mathrm{A}\cup\mathrm{B}} i_f(h)$.*

Proof. We first prove *(i)*. Assume $\dot{\sigma}_E(g) \tilde{=}_{\mathrm{A}} h$ and let $\mathfrak{I}$ be a database instance over $\mathfrak{D}$. We prove $[\![\dot{\delta}(\dot{\pi}_{\mathrm{A}\cup\mathrm{B}}(\dot{\sigma}_E(g)))]\!]_{\mathfrak{I}} = [\![\dot{\delta}(\dot{\pi}_{\mathrm{A}\cup\mathrm{B}}(i_f(h)))]\!]_{\mathfrak{I}}$ by showing that there exist n and n' such that

$$(\mathtt{t} : n) \in [\![\dot{\pi}_{\mathrm{A}\cup\mathrm{B}}(\dot{\sigma}_E(g))]\!]_{\mathfrak{I}} \iff (\mathtt{t} : n') \in [\![\dot{\pi}_{\mathrm{A}\cup\mathrm{B}}(i_f(h))]\!]_{\mathfrak{I}}.$$

Assume $(\mathtt{t} : n) \in [\![\dot{\pi}_{\mathrm{A}\cup\mathrm{B}}(\dot{\sigma}_E(g))]\!]_{\mathfrak{I}}$, and let $(\mathtt{t}_1 : p_1), \ldots, (\mathtt{t}_j : p_j) \in [\![\dot{\sigma}_E(g)]\!]_{\mathfrak{I}}$ be all tuple-count pairs such that, for every i, $1 \leq i \leq j$, $\mathtt{t}_i \equiv_{\mathrm{A}\cup\mathrm{B}} \mathtt{t}$. By construction, we have $n = p_1 + \cdots + p_j$. By Lemma 1.*(iv)* and $\mathrm{B} \subseteq \det(\mathrm{A};E)$, no other $\mathtt{t}'$ exists with $\mathtt{t}' \notin \{\mathtt{t}_1, \ldots, \mathtt{t}_j\}$, $(\mathtt{t}' : p') \in [\![\dot{\sigma}_E(g)]\!]_{\mathfrak{I}}$ and $\mathtt{t}' \equiv_{\mathrm{A}} \mathtt{t}$ as this would imply $\mathtt{t}' \equiv_{\mathrm{A}\cup\mathrm{B}} \mathtt{t}$. By $\dot{\sigma}_E(g) \tilde{=}_{\mathrm{A}} h$, we have $(\mathtt{t}|_{\mathrm{A}} : q) \in [\![\dot{\pi}_{\mathrm{A}}(h)]\!]_{\mathfrak{I}}$. Let $(\mathtt{s}_1 : q_1), \ldots, (\mathtt{s}_k : q_k) \in [\![h]\!]_{\mathfrak{I}}$ with, for every i, $1 \leq i \leq k$, $\mathtt{s}_i \equiv_{\mathrm{A}} \mathtt{t}$. By construction, we have $q = q_1 + \cdots + q_k$. We prove that, for all $1 \leq i \leq k$, $\mathtt{t} \equiv_{\mathrm{B}} \mathtt{s}_i$. Consider $b \in \mathrm{B}$ with $(b := x) \in f$:

1. If $x \in \mathrm{attrs}(b;E)$, then $x \in \mathrm{A}$ and, by Lemma 1.*(i)*, we have $\mathtt{t}(b) = \mathtt{t}(x)$. By $x \in \mathrm{A}$ and $\mathtt{t} \equiv_{\mathrm{A}} \mathtt{s}_i$, we have $\mathtt{t}(b) = \mathtt{t}(x) = \mathtt{s}_i(x)$. From the semantics of $i_{b:=x}$, it follows that $\mathtt{t}(b) = \mathtt{t}(x) = \mathtt{s}_i(x) = \mathtt{s}_i(b)$.
2. If $x \in \mathrm{consts}(b;E)$, then, by Lemma 1.*(iii)*, there exists a constant $\mathtt{u} \in \mathfrak{U}$ such that $\mathrm{consts}(b;E) = \{\mathtt{u}\}$. By Lemma 1.*(ii)*, we have $\mathtt{t}(x) = \mathtt{u}$. From the semantics of $i_{b:=\mathtt{u}}$, it follows that $\mathtt{t}(b) = \mathtt{u} = \mathtt{s}_i(b)$.

We conclude that $(\mathtt{t} : q) \in [\![\dot{\pi}_{\mathrm{A}\cup\mathrm{B}}(i_f(h))]\!]_{\mathfrak{I}}$ and $n' = q$. To prove *(ii)*, we bootstrap the proof of *(i)*. By $\dot{\sigma}_E(g) \doteq_{\mathrm{A}} h$, we have $n = q$. Hence, we have $n = n'$ and $(\mathtt{t} : n) \in [\![\dot{\pi}_{\mathrm{A}\cup\mathrm{B}}(\dot{\sigma}_E(g))]\!]_{\mathfrak{I}}$ if and only if $(\mathtt{t} : n) \in [\![\dot{\pi}_{\mathrm{A}\cup\mathrm{B}}(i_f(h))]\!]_{\mathfrak{I}}$. □

In the presence of constant conditions, Theorem 1 can be applied to joins to rewrite them into semi-joins combined with attribute introduction. An example of such rewrites is exhibited in Example 3.

Applications of Theorem 1 involve *attribute introduction*, which makes query results larger. Hence, it is best to delay this operation by pulling the operator up. Proposition 2 presents rewrite rules to do so.

Proposition 2. *Let e, e_1, and e_2 be expressions over $\mathfrak{D}$. We have*

(i) $\dot{\sigma}_E(i_f(e)) \doteq i_f(\dot{\sigma}_E(e))$ *if, for all $(b := x) \in f$, $b \notin \mathrm{attrs}(E)$;*
(ii) $\dot{\pi}_{\mathrm{B}}(i_f(e)) \doteq i_{f'}(\dot{\pi}_{\mathrm{B}'}(e))$ *if $f' \subseteq f$ and $\mathrm{B} = \mathrm{B}' \cup \{b \mid (b := x) \in f'\}$.*
(iii) $\dot{\rho}_g(i_f(e)) \doteq i_{f'}(\dot{\rho}_{g'}(e))$ *if $g' = \{a \mapsto g(a) \mid a \in \mathcal{S}(e;\mathfrak{D})\}$ and*

$$f' = \{g(b) := g(x) \mid (b := x) \in f \wedge x \in \mathcal{S}(e;\mathfrak{D})\} \cup \{g(b) := x \mid (b := x) \in f \wedge x \in \mathfrak{U}\}.$$

(iv) $\dot{\delta}(i_f(e)) \doteq i_f(\dot{\delta}(e))$;
(v) $i_f(e_1) \oplus i_f(e_2) \doteq i_f(e_1 \oplus e_2)$ *if* $\oplus \in \{\dot{\cup}, \dot{\cap}, \dot{-}\}$;
(vi) $i_f(e_1) \ltimes_E e_2 \doteq i_{f'}(e_1 \ltimes_{E'} e_2)$ *if* $f' = \{(b := x) \in f \mid b \notin \mathcal{S}(e_2; \mathfrak{D})\}$ *and* $E' = E \cup \{b = x \mid (b := x) \in f \wedge b \in \mathcal{S}(e_2; \mathfrak{D})\}$;
(vii) $i_f(e_1) \ltimes_E e_2 \doteq i_f(e_1 \ltimes_{E'} e_2)$ *if*

$$E' = E \cup \{b = x \mid (b := x) \in f \wedge b \in \mathcal{S}(e_2; \mathfrak{D})\};$$

(viii) $e_1 \ltimes_E i_f(e_2) \doteq e_1 \ltimes_{E'} e_2$ *if* $E' = E \cup \{b = x \mid (b := x) \in f \wedge b \in \mathcal{S}(e_1; \mathfrak{D})\}$;
(ix) $i_f(e_1) \dot{\sqcup} i_f(e_2) \doteq i_f(e_1 \dot{\sqcup} e_2)$; *and*
(x) $i_f(i_g(e)) \doteq i_{f \cup g}(e)$ *if, for all* $(b := x) \in f$, *we have* $x \in (\mathcal{S}(e; \mathfrak{D}) \cup \mathfrak{U})$.

Attribute introduction and renaming play complementary roles in the context of projection, as is illustrated by the following example:

Example 6. Consider the expression e with $\mathcal{S}(e; \mathfrak{D}) = \{a, b\}$ and consider the query $\dot{\pi}_{a,c,d}(i_{c:=a,d:=b}(e))$. As we do not need b after the projection, we can also rename b to d instead of introducing d. This alternative approach results in $\dot{\pi}_{a,c,d}(i_{c:=a}(\dot{\rho}_{a \mapsto a, b \mapsto d}(e))$. In this expression, we can easily push the attribute introduction through the projection, resulting in the expression $i_{c:=a}(\dot{\pi}_{a,d}(\dot{\rho}_{a \mapsto a, b \mapsto d}(e)))$.

The following rewrite rules can be used to apply the rewriting of Example 6:

Proposition 3. *Let* e *be an expression over* $\mathfrak{D}$ *and let* i_f *be an attribute introduction operator applicable to* e. *We have*

(i) *if* $(b := x) \in f$, $x \in \mathcal{S}(e; \mathfrak{D})$, *and* $c = b$ *for all* $(c := x) \in f$, *then*

$$i_f(e) \doteq_{(\mathcal{S}(e;\mathfrak{D}) \cup \{b\}) - \{x\}} i_{f \setminus \{b:=x\}}(\dot{\rho}_{\{x \mapsto b\} \cup \{a \mapsto a \mid a \in (\mathcal{S}(e;\mathfrak{D}) - \{x\})\}}(e));$$

(ii) *if* $(b_1 := x), (b_2 := x) \in f$, $b_1 \neq b_2$, *then* $i_f(e) \doteq i_{\{b_2 := b_1\}}(i_{f - \{b_2 := x\}}(e))$.

The rewrite rules of Proposition 2 that involve selections and attribute introductions put heavy restrictions on the sets of conditions involved. To alleviate these restrictions, we can use Proposition 3 and the well-known push-down rules for selection [2, 5, 10, 12], to push some attribute introductions through selections.

What we have done up to now is examining how selection conditions interact with the notions of strong-equivalence and weak-equivalence. Next, we will put these results to use. As explained in the Introduction, our focus is twofold:

1. eliminating joins in favor of semi-joins; and
2. eliminating deduplication.

These foci are covered in Sections 4.1 and 4.2, respectively. In Section 4.3, finally, we investigate how the other operators interact with strong-equivalence and weak-equivalence.

4.1 θ-joins and θ-semi-joins

Above, we explored how selection conditions interact with strong-equivalence and weak-equivalence. Since θ-joins and θ-semi-joins implicitly use selection conditions, the techniques developed also apply to θ-joins and θ-semi-joins, which are the focus of this subsection. First, we notice that we can use the following rules to change the conditions involved in selections, θ-joins, and θ-semi-joins.

Proposition 4. *Let E and E' be sets of conditions over the same set of attributes A. If we have $\mathcal{C}(a;E) = \mathcal{C}(a;E')$ for every $a \in \mathrm{A}$, then we have*

(i) $\dot{\sigma}_E(e) \doteq \dot{\sigma}_{E'}(e)$;
(ii) $e_1 \mathbin{\dot{\bowtie}}_E e_2 \doteq e_1 \mathbin{\dot{\bowtie}}_{E'} e_2$; *and*
(iii) $e_1 \mathbin{\dot{\ltimes}}_E e_2 \doteq e_1 \mathbin{\dot{\ltimes}}_{E'} e_2$.

The equivalence at the basis of semi-join-based query rewriting in the set-based relational algebra is $e_1 \ltimes e_2 = \pi_{\mathrm{A}_1}(e_1 \bowtie e_2)$ with A_1 the relation schema of the relation to which e_1 evaluates. Notice, however, that the equivalence $e_1 \mathbin{\dot{\ltimes}} e_2 = \dot{\pi}_{\mathrm{A}_1}(e_1 \mathbin{\dot{\bowtie}} e_2)$ does *not* hold, because the multiset semi-join does not take into account the number of occurrences of tuples in $[\![e_2]\!]_{\mathfrak{I}}$:[11]

Example 7. Consider the database instance $\mathfrak{I}$ of Example 1, which is visualized in Figure 1. We apply the queries $e_1 = \dot{\pi}_{\{cname,age\}}(\mathit{Customer} \mathbin{\dot{\bowtie}} \mathit{Bought})$ and $e_2 = \mathit{Customer} \mathbin{\dot{\ltimes}} \mathit{Bought}$. We have

$$[\![e_1]\!]_{\mathfrak{I}} = \{(\mathit{cname} \mapsto \texttt{Alice}, \mathit{age} \mapsto \texttt{19} : 2), \dots\};$$
$$[\![e_2]\!]_{\mathfrak{I}} = \{(\mathit{cname} \mapsto \texttt{Alice}, \mathit{age} \mapsto \texttt{19} : 1), \dots\}.$$

Even though the above rewriting of the projection of a join into the corresponding semi-join is not a strong-equivalent rewriting, we observe that it is still a weak-equivalent rewriting.

We now formalize rewrite rules involving joins and semi-joins:

Theorem 2. *Let g_1, g_2, h_1, and h_2 be expressions over $\mathfrak{D}$ with $\mathrm{G}_1 = \mathcal{S}(g_1;\mathfrak{D})$, $\mathrm{G}_2 = \mathcal{S}(g_2;\mathfrak{D})$, $\mathrm{H}_1 = \mathcal{S}(h_1;\mathfrak{D})$, $\mathrm{H}_2 = \mathcal{S}(h_2;\mathfrak{D})$, $\mathrm{A}_1 \subseteq (\mathrm{G}_1 \cap \mathrm{H}_1)$, $\mathrm{A}_2 \subseteq (\mathrm{G}_2 \cap \mathrm{H}_2)$, and $\mathrm{A}_1 \cap \mathrm{A}_2 = \mathrm{G}_1 \cap \mathrm{G}_2 = \mathrm{H}_1 \cap \mathrm{H}_2$, let E be a set of conditions over $\mathrm{A}_1 \cup \mathrm{A}_2$, let $\mathrm{B} \subseteq (\mathrm{A}_2 - \mathrm{A}_1)$, and let f be a set of assignment-pairs such that, for each $b \in \mathrm{B}$, there exists $(b := x) \in f$ with $x \in (\mathcal{C}(b;E) \cap (\mathrm{A}_1 \cup \mathfrak{U}))$. We have*

(i) $g_1 \mathbin{\dot{\bowtie}}_E g_2 \mathbin{\tilde{=}}_{\mathrm{A}_1 \cup \mathrm{A}_2} h_1 \mathbin{\dot{\bowtie}}_E h_2$ *if* $g_1 \mathbin{\tilde{=}}_{\mathrm{A}_1} h_1$ *and* $g_2 \mathbin{\tilde{=}}_{\mathrm{A}_2} h_2$;
(ii) $g_1 \mathbin{\dot{\bowtie}}_E g_2 \mathbin{\doteq}_{\mathrm{A}_1 \cup \mathrm{A}_2} h_1 \mathbin{\dot{\bowtie}}_E h_2$ *if* $g_1 \mathbin{\doteq}_{\mathrm{A}_1} h_1$ *and* $g_2 \mathbin{\doteq}_{\mathrm{A}_2} h_2$;
(iii) $g_1 \mathbin{\dot{\bowtie}}_E g_2 \mathbin{\tilde{=}}_{\mathrm{A}_1} h_1 \mathbin{\dot{\ltimes}}_E h_2$ *if* $g_1 \mathbin{\tilde{=}}_{\mathrm{A}_1} h_1$ *and* $g_2 \mathbin{\tilde{=}}_{\mathrm{A}_2} h_2$;
(iv) $g_1 \mathbin{\dot{\bowtie}}_E g_2 \mathbin{\doteq}_{\mathrm{A}_1} h_1 \mathbin{\dot{\ltimes}}_E h_2$ *if* $g_1 \mathbin{\doteq}_{\mathrm{A}_1} h_1$, $g_2 \mathbin{\tilde{=}}_{\mathrm{A}_2} h_2$, g_2 *is a set relation, and* g_2 *has a key* C *with* $\mathrm{C} \subseteq \det(\mathrm{A}_1 \cap \mathrm{A}_2; E)$;

[11] We could have defined a multiset semi-join operator that *does take* into account the number of occurrences of tuples in $[\![e_2]\!]_{\mathfrak{I}}$. With such a semi-join operator, we would no longer be able to sharply reduce the size of intermediate query results, however, and lose some potential to optimize query evaluation.

(v) $g_1 \ltimes_E g_2 \mathrel{\tilde{=}}_{\mathrm{A}_1 \cup \mathrm{B}} i_f(h_1 \ltimes_E h_2)$ *if* $g_1 \mathrel{\tilde{=}}_{\mathrm{A}_1} h_1$ *and* $g_2 \mathrel{\tilde{=}}_{\mathrm{A}_2} h_2$*; and*
(vi) $g_1 \ltimes_E g_2 \mathrel{\dot{=}}_{\mathrm{A}_1 \cup \mathrm{B}} i_f(h_1 \ltimes_E h_2)$ *if* $g_1 \mathrel{\dot{=}}_{\mathrm{A}_1} h_1$*,* $g_2 \mathrel{\tilde{=}}_{\mathrm{A}_2} h_2$*,* g_2 *is a set relation, and* g_2 *has a key* C *with* $\mathrm{C} \subseteq \det(\mathrm{A}_1 \cap \mathrm{A}_2; E)$.

Proof (Sketch). We prove *(i)*. Let $\mathfrak{I}$ be a database instance over $\mathfrak{D}$. We prove $[\![\dot{\delta}(\dot{\pi}_{\mathrm{A}_1 \cup \mathrm{A}_2}(g_1 \ltimes_E g_2))]\!]_{\mathfrak{I}} = [\![\dot{\delta}(\dot{\pi}_{\mathrm{A}_1 \cup \mathrm{A}_2}(h_1 \ltimes_E h_2))]\!]_{\mathfrak{I}}$ by showing that there exist n and n' such that

$$(\mathsf{t} : n) \in [\![\dot{\pi}_{\mathrm{A}_1 \cup \mathrm{A}_2}(g_1 \ltimes_E g_2)]\!]_{\mathfrak{I}} \iff (\mathsf{t} : n') \in [\![\dot{\pi}_{\mathrm{A}_1 \cup \mathrm{A}_2}(h_1 \ltimes_E h_2)]\!]_{\mathfrak{I}}.$$

Assume $(\mathsf{t} : n) \in [\![\dot{\pi}_{\mathrm{A}_1 \cup \mathrm{A}_2}(g_1 \ltimes_E g_2)]\!]_{\mathfrak{I}}$, and let $(\mathsf{r}_1 : p_1), \ldots, (\mathsf{r}_{k_1} : p_{k_1}) \in [\![g_1]\!]_{\mathfrak{I}}$ and $(\mathsf{s}_1 : q_1), \ldots, (\mathsf{s}_{k_2} : q_{k_2}) \in [\![g_2]\!]_{\mathfrak{I}}$ be all tuple-count pairs such that, for every i_1, $1 \leq i_1 \leq k_1$, $\mathsf{r}_{i_1} \equiv_{\mathrm{A}_1} \mathsf{t}$, and, for every i_2, $1 \leq i_2 \leq k_2$, $\mathsf{s}_{i_2} \equiv_{\mathrm{A}_2} \mathsf{t}$. Let $p = p_1 + \cdots + p_{k_1}$ and $q = q_1 + \cdots + q_{k_2}$. By construction, we have that, for every i_1 and i_2, $1 \leq i_1 \leq k_1$ and $1 \leq i_2 \leq k_2$, $(\mathsf{r}_{i_1} \cdot \mathsf{s}_{i_2} : p_{i_1} \cdot q_{i_2}) \in [\![g_1 \ltimes_E g_2]\!]_{\mathfrak{I}}$. Hence, $(\mathsf{t}|_{\mathrm{A}_1} : p) \in [\![\dot{\pi}_{\mathrm{A}_1}(g_1)]\!]_{\mathfrak{I}}$, $(\mathsf{t}|_{\mathrm{A}_2} : q) \in [\![\dot{\pi}_{\mathrm{A}_2}(g_2)]\!]_{\mathfrak{I}}$, and $n = p \cdot q$. By $g_1 \mathrel{\tilde{=}}_{\mathrm{A}_1} h_1$ and $g_2 \mathrel{\tilde{=}}_{\mathrm{A}_2} h_2$, we have $(\mathsf{t}|_{\mathrm{A}_1} : p') \in [\![\dot{\pi}_{\mathrm{A}_1}(h_1)]\!]_{\mathfrak{I}}$ and $(\mathsf{t}|_{\mathrm{A}_2} : q') \in [\![\dot{\pi}_{\mathrm{A}_2}(h_2)]\!]_{\mathfrak{I}}$. Let $(\mathsf{r}'_1 : p'_1), \ldots, (\mathsf{r}'_{l_1} : p'_{l_1}) \in [\![h_1]\!]_{\mathfrak{I}}$ and $(\mathsf{s}'_1 : q'_1), \ldots, (\mathsf{s}'_{l_2} : q'_{l_2}) \in [\![h_2]\!]_{\mathfrak{I}}$ be all tuple-count pairs such that, for every j_1, $1 \leq j_1 \leq l_1$, $\mathsf{r}'_{j_1} \equiv_{\mathrm{A}_1} \mathsf{t}$, and, for every j_2, $1 \leq j_2 \leq l_2$, $\mathsf{s}'_{j_2} \equiv_{\mathrm{A}_2} \mathsf{t}$. By construction, $p' = p'_1 + \cdots + p'_{l_1}$ and $q' = q'_1 + \cdots + q'_{l_2}$ and, for every j_1 and j_2, $1 \leq j_1 \leq l_1$ and $1 \leq j_2 \leq l_2$, $\mathsf{r}'_{j_1} \cdot \mathsf{s}'_{j_2}$ satisfies E and $(\mathsf{r}'_{j_1} \cdot \mathsf{s}'_{j_2} : p'_{j_1} \cdot q'_{j_2}) \in [\![h_1 \ltimes_E h_2]\!]_{\mathfrak{I}}$. We conclude $(\mathsf{t} : p' \cdot q') \in [\![\dot{\pi}_{\mathrm{A}_1 \cup \mathrm{A}_2}(h_1 \ltimes_E h_2)]\!]_{\mathfrak{I}}$ and $n' = p' \cdot q'$.

Next, we prove *(ii)* by bootstrapping the proof of *(i)*. Observe that, by $g_1 \mathrel{\dot{=}}_{\mathrm{A}_1} h_1$ and $g_2 \mathrel{\dot{=}}_{\mathrm{A}_2} h_2$, we have that $p = p'$ and $q = q'$. Hence, $n = n'$ and $(\mathsf{t} : n) \in [\![\dot{\pi}_{\mathrm{A}_1 \cup \mathrm{A}_2}(g_1 \ltimes_E g_2)]\!]_{\mathfrak{I}}$ if and only if $(\mathsf{t} : n) \in [\![\dot{\pi}_{\mathrm{A}_1 \cup \mathrm{A}_2}(h_1 \ltimes_E h_2)]\!]_{\mathfrak{I}}$.

The other statements can be proven in analogous ways. □

The rules of Theorem 2 can be specialized to semi-joins only:

Corollary 1. *Let* g_1*,* g_2*,* h_1*, and* h_2 *be expressions which satisfy the conditions of Theorem 2 and let* $\hat{=}$ *be either* $\dot{=}$ *or* $\tilde{=}$*. If* $g_1 \mathrel{\hat{=}}_{\mathrm{A}_1} h_1$ *and* $g_2 \mathrel{\tilde{=}}_{\mathrm{A}_2} h_2$*, then* $g_1 \ltimes_E g_2 \mathrel{\hat{=}}_{\mathrm{A}_1} h_1 \ltimes_E h_2$.

4.2 Deduplication

The second optimization goal we have set ourselves is to eliminate the need for removing duplicates. This is possible if we can push down deduplication operators to a level where subexpressions are guaranteed to evaluate to set relations, in which case deduplication becomes redundant.

The rewrite rules relevant for pushing down deduplication are the following:

Proposition 5. *Let* e*,* e_1*,* e_2 *be expressions over* $\mathfrak{D}$*. We have*

(i) $\dot{\delta}(\dot{\sigma}_E(e)) \doteq \dot{\sigma}_E(\dot{\delta}(e))$;
(ii) $\dot{\delta}(\dot{\pi}_{\mathrm{B}}(e) \doteq \dot{\pi}_{\mathrm{B}}(\dot{\delta}(e))$ *if* B *is a key of* e;
(iii) $\dot{\delta}(\dot{\rho}_f(e)) \doteq \dot{\rho}_f(\dot{\delta}(e))$;
(iv) $\dot{\delta}(\dot{\delta}(e)) \doteq \dot{\delta}(e)$;
(v) $\dot{\delta}(e_1 \cup e_2) \doteq \dot{\delta}(e_1) \dot{\sqcup} \dot{\delta}(e_2)$;
(vi) $\dot{\delta}(e_1 \mathbin{\dot{\cap}} e_2) \doteq \dot{\delta}(e_1) \mathbin{\dot{\cap}} \dot{\delta}(e_2)$;
(vii) $\dot{\delta}(e_1 \ltimes_E e_2) \doteq \dot{\delta}(e_1) \ltimes_E \dot{\delta}(e_2)$;
(viii) $\dot{\delta}(e_1 \mathbin{\dot{\ltimes}}_E e_2) \doteq \dot{\delta}(e_1) \mathbin{\dot{\ltimes}}_E e_2$;
(ix) $\dot{\delta}(e_1 \mathbin{\dot{\sqcup}} e_2) \doteq \dot{\delta}(e_1) \mathbin{\dot{\sqcup}} \dot{\delta}(e_2)$; *and*
(x) $\dot{\delta}(i_f(e)) \doteq i_f(\dot{\delta}(e))$.

One cannot push down deduplication through difference, however, as $\dot{\delta}(e_1 \dot{-} e_2) \doteq \dot{\delta}(e_1) \dot{-} \dot{\delta}(e_2)$ does *not* hold in general.

If, in the end, deduplication operates on expressions that evaluate to set relations, it can be eliminated altogether:

Proposition 6. *Let e be an expression over $\mathfrak{D}$. If e is a set relation, then $\dot{\delta}(e) \doteq e$.*

4.3 Other rewrite rules

To use the rewrite rules of Proposition 1, Theorem 1, Theorem 2, Proposition 2, and Proposition 5 in the most general way possible, we also need to know how the other operators interact with strong-B-equivalence and weak-B-equivalence. For the unary operators, we have the following:

Proposition 7. *Let g and h be expressions over $\mathfrak{D}$ with $\mathrm{A} \subseteq \mathcal{S}(g;\mathfrak{D}) \cap \mathcal{S}(h;\mathfrak{D})$. Let $\hat{=}$ be either $\doteq$ or $\tilde{=}$. If $g \hat{=}_{\mathrm{A}} h$, then we have*

(i) $\dot{\sigma}_E(g) \hat{=}_{\mathrm{A}} \dot{\sigma}_E(h)$ *if E is a set of equalities with* $\mathrm{attrs}(E) \subseteq \mathrm{A}$*;*
(ii) $\dot{\pi}_{\mathrm{B}_g}(g) \hat{=}_{\mathrm{A}\cap\mathrm{B}} \dot{\pi}_{\mathrm{B}_h}(h)$ *if $\mathrm{B}_g \subseteq \mathcal{S}(g;\mathfrak{D})$, $\mathrm{B}_h \subseteq \mathcal{S}(h;\mathfrak{D})$, and $\mathrm{B} = \mathrm{B}_g \cap \mathrm{B}_h$;*
(iii) $\dot{\rho}_{f_g}(g) \hat{=}_{\{f_g(a)\mid a\in\mathrm{A}\}} \dot{\rho}_{f_h}(h)$ *if $f_g : \mathcal{S}(g;\mathfrak{D}) \to \mathrm{B}_g$ and $f_h : \mathcal{S}(h;\mathfrak{D}) \to \mathrm{B}_h$ are bijections with, for all $a \in \mathrm{A}$, $f_g(a) = f_h(a)$;*
(iv) $\dot{\delta}(g) \hat{=}_{\mathrm{A}} \dot{\delta}(h)$*; and*
(v) $\dot{\iota}_f(g) \hat{=}_{\mathrm{A}\cup\mathrm{B}} \dot{\iota}_f(h)$ *if $\mathrm{B} \subseteq (\mathfrak{N} - (\mathcal{S}(g;\mathfrak{D}) \cup \mathcal{S}(h;\mathfrak{D})))$ and f is a set of assignment-pairs such that, for each $b \in \mathrm{B}$, there exists $(b := x) \in f$ with $x \in (\mathrm{A} \cup \mathfrak{U})$.*

For the binary operators, we observe that the θ-join and θ-semi-join operators are completely covered by Theorem 2 and Corollary 1.

For the union and max-union operators, we have the following:

Proposition 8. *Let g_1, g_2, h_1, and h_2 be expressions over $\mathfrak{D}$ and let A be a set of attributes with $\mathrm{A}_g = \mathcal{S}(g_1;\mathfrak{D}) = \mathcal{S}(g_2;\mathfrak{D})$, $\mathrm{A}_h = \mathcal{S}(h_1;\mathfrak{D}) = \mathcal{S}(h_2;\mathfrak{D})$, and $\mathrm{A} \subseteq (\mathrm{A}_g \cap \mathrm{A}_h)$. Let $\hat{=}$ be either $\doteq$ or $\tilde{=}$. If $g_1 \hat{=}_{\mathrm{A}} h_1$ and $g_2 \hat{=}_{\mathrm{A}} h_2$, then we have $g_1 \dot{\cup} g_2 \hat{=}_{\mathrm{A}} h_1 \dot{\cup} h_2$ and $g_1 \dot{\sqcup} g_2 \tilde{=}_{\mathrm{A}} h_1 \dot{\sqcup} h_2$. If, in addition, $\mathrm{A} = \mathrm{A}_g = \mathrm{A}_h$, then we also have $g_1 \dot{\sqcup} g_2 \hat{=}_{\mathrm{A}} h_1 \dot{\sqcup} h_2$.*

Finally, for the operators intersection and difference, we propose the following straightforward rewrite rules:

Proposition 9. *Let g_1, g_2, h_1 and h_2 be expressions over $\mathfrak{D}$ with $\mathcal{S}(g_1;\mathfrak{D}) = \mathcal{S}(g_2;\mathfrak{D})$ and $\mathcal{S}(h_1;\mathfrak{D}) = \mathcal{S}(h_2;\mathfrak{D})$. Let $\hat{=}$ be either $\doteq$ or $\tilde{=}$. If $g_1 \hat{=} h_1$ and $g_2 \hat{=} h_2$, then we have $g_1 \dot{\cap} g_2 \hat{=} h_1 \dot{\cap} h_2$; and if $g_1 \doteq h_1$ and $g_2 \doteq h_2$, then $g_1 \dot{-} g_2 \doteq h_1 \dot{-} h_2$.*

The above rewrite rules for max-union, intersection, and difference are very restrictive. Next, we illustrate that we cannot simply relax these restrictive rewrite rules to more general strong-B-equivalence or weak-B-equivalence rewrite rules:

Example 8. Let $\text{A} = \{m, n\}$, let U, V, V', and W be four relation names, and let $\mathfrak{I}$ be the database instance mapping these four relation names to the following multiset relations over A:

$$\begin{aligned}
[\![U]\!]_{\mathfrak{I}} &= \{(\{m \mapsto \mathtt{u}, n \mapsto \mathtt{v}\} : 1), (\{m \mapsto \mathtt{u}, n \mapsto \mathtt{w}\} : 1)\};\\
[\![V]\!]_{\mathfrak{I}} &= \{(\{m \mapsto \mathtt{u}, n \mapsto \mathtt{v}\} : 2)\}; \qquad [\![V']\!]_{\mathfrak{I}} = \{(\{m \mapsto \mathtt{u}, n \mapsto \mathtt{v}\} : 3)\};\\
[\![W]\!]_{\mathfrak{I}} &= \{(\{m \mapsto \mathtt{u}, n \mapsto \mathtt{w}\} : 2)\}.
\end{aligned}$$

We have $V \doteq V'$ and, due to $[\![\dot{\pi}_m(U)]\!]_{\mathfrak{I}} = [\![\dot{\pi}_m(V)]\!]_{\mathfrak{I}} = [\![\dot{\pi}_m(W)]\!]_{\mathfrak{I}} = \{(\{m \mapsto \mathtt{u}\} : 2)\}$, we have $U \doteq_m V \doteq_m W$. We also have

$$\begin{aligned}
[\![V \dot{-} V']\!]_{\mathfrak{I}} &= \emptyset; & [\![V' \dot{-} V]\!]_{\mathfrak{I}} &= \{(\{m \mapsto \mathtt{u}, n \mapsto \mathtt{v}\} : 1)\};\\
[\![V \dot{\cup} V]\!]_{\mathfrak{I}} &= [\![V]\!]_{\mathfrak{I}} & [\![V \dot{\cup} W]\!]_{\mathfrak{I}} &= [\![V]\!]_{\mathfrak{I}} \cup [\![W]\!]_{\mathfrak{I}};\\
[\![V \dot{\cap} U]\!]_{\mathfrak{I}} &= \{(\{m \mapsto \mathtt{u}, n \mapsto \mathtt{v}\} : 1)\}; & [\![V \dot{\cap} W]\!]_{\mathfrak{I}} &= \emptyset;\\
[\![V \dot{-} U]\!]_{\mathfrak{I}} &= \{(\{m \mapsto \mathtt{u}, n \mapsto \mathtt{v}\} : 1)\}; & [\![V \dot{-} W]\!]_{\mathfrak{I}} &= \{(\{m \mapsto \mathtt{u}, n \mapsto \mathtt{v}\} : 2)\}.
\end{aligned}$$

Hence, $V \dot{-} V' \not\equiv V' \dot{-} V$, $V \dot{\cup} V \not\equiv_m V \dot{\cup} W$, $V \dot{\cap} U \not\equiv_m V \dot{\cap} W$, $V \dot{\cap} U \not\equiv_m V \dot{\cap} W$, and $V \dot{-} U \not\equiv_m V \dot{-} W$. Let $\mathfrak{I}'$ be the database instance obtained from $\mathfrak{I}$ by changing all the counts in $[\![U]\!]_{\mathfrak{I}}$ to 2. Then,

$$\begin{aligned}
[\![V \dot{-} U]\!]_{\mathfrak{I}'} &= \emptyset; & [\![V \dot{-} W]\!]_{\mathfrak{I}'} &= \{(\{m \mapsto \mathtt{u}, n \mapsto \mathtt{v}\} : 2)\},
\end{aligned}$$

and, hence, we also have $V \dot{-} U \not\equiv_m V \dot{-} W$. We must conclude that we cannot hope for meaningful rewrite rules for max-union, intersection, and difference if the conditions of Propositions 8 and 9 are not satisfied.

Besides the rewrite rules we introduced, the usual strong-equivalent (multiset) relational algebra rewrite rules can, of course, also be used in the setting of non-strong-equivalent rewriting. This includes rules for pushing down selections and projections [5, 12] and the usual associativity and distributivity rules for union, intersection, difference, and joins [2].

5 Deriving structural query information

Some of the rewrite rules of Proposition 1, Theorem 2, and Proposition 6 can only be applied if it is known that some subexpression is a set relation or has certain keys. Therefore, we introduce rules to derive this information.

We use the well-known functional dependencies as a framework to reason about keys. A *functional dependency over* A is of the form $\text{B} \to \text{C}$, with $\text{B}, \text{C} \subseteq \text{A}$. Let $\text{B} \to \text{C}$ be a functional dependency over A. A relation $\mathscr{R}$ over A satisfies $\text{B} \to \text{C}$ if, for every pair of tuples $\mathsf{r}, \mathsf{s} \in \mathscr{R}$ with $\mathsf{r} \equiv_{\text{B}} \mathsf{s}$, we have $\mathsf{r} \equiv_{\text{C}} \mathsf{s}$. A multiset relation $\langle \mathscr{R}; \tau \rangle$ over A satisfies $\text{B} \to \text{C}$ if $\mathscr{R}$ satisfies $\text{B} \to \text{C}$. Let D be a set of functional dependencies over A. The *closure of* D, which we denote by $\mathsf{+}(D)$, is the set of all functional dependencies over A that are logically implied by D [1].

By $\mathrm{fds}(e;\mathfrak{D})$, we denote the functional dependencies we derive from expression e over $\mathfrak{D}$. We define $\mathrm{fds}(e;\mathfrak{D})$ inductively. The base cases are relation names $R \in \mathbf{N}$, for which we have $\mathrm{fds}(R;\mathfrak{D}) = {+}(\{\textsc{k} \to \mathbf{A}(R) \mid \textsc{k} \in \mathbf{K}(R)\})$. For the other operations, we have

$$
\begin{aligned}
\mathrm{fds}(\dot\sigma_E(e);\mathfrak{D}) &= {+}(\{\textsc{a} \to \textsc{b} \mid \exists \textsc{c}\ \textsc{b} \subseteq \det(\textsc{c}; E) \wedge (\textsc{a} \to \textsc{c}) \in \mathrm{fds}(e;\mathfrak{D})\});\\
\mathrm{fds}(\dot\pi_{\textsc{c}}(e);\mathfrak{D}) &= \{\textsc{a} \to \textsc{b} \cap \textsc{c} \mid \textsc{a} \subseteq \textsc{c} \wedge (\textsc{a} \to \textsc{b}) \in \mathrm{fds}(e;\mathfrak{D})\};\\
\mathrm{fds}(\dot\rho_f(e);\mathfrak{D}) &= \{f(\textsc{a}) \to f(\textsc{b}) \mid (\textsc{a} \to \textsc{b}) \in \mathrm{fds}(e;\mathfrak{D})\};\\
\mathrm{fds}(\dot\delta(e);\mathfrak{D}) &= \mathrm{fds}(e;\mathfrak{D});\\
\mathrm{fds}(e_1 \dot\cup e_2;\mathfrak{D}) &= {+}(\emptyset);\\
\mathrm{fds}(e_1 \dot\cap e_2;\mathfrak{D}) &= {+}((\mathrm{fds}(e_1;\mathfrak{D}) \cup \mathrm{fds}(e_2;\mathfrak{D})));\\
\mathrm{fds}(e_1 \dot- e_2;\mathfrak{D}) &= \mathrm{fds}(e_1;\mathfrak{D});\\
\mathrm{fds}(e_1 \dot\bowtie e_2;\mathfrak{D}) &= {+}((\mathrm{fds}(e_1;\mathfrak{D}) \cup \mathrm{fds}(e_2;\mathfrak{D})));\\
\mathrm{fds}(e_1 \dot\bowtie_E e_2;\mathfrak{D}) &= \mathrm{fds}(\dot\sigma_E(e_1 \dot\bowtie e_2);\mathfrak{D});\\
\mathrm{fds}(e_1 \dot\ltimes_E e_2;\mathfrak{D}) &= \mathrm{fds}(\dot\pi_{\mathcal{S}(e_1;\mathfrak{D})}(e_1 \dot\bowtie_E e_2);\mathfrak{D})\\
\mathrm{fds}(e_1 \dot\sqcup e_2;\mathfrak{D}) &= {+}(\emptyset);\\
\mathrm{fds}(\dot\iota_f(e);\mathfrak{D}) &= {+}(\{(\{x\} \cap \mathcal{S}(e;\mathfrak{D}) \to \{b\}) \mid (b := x) \in f\} \cup \mathrm{fds}(e;\mathfrak{D})).
\end{aligned}
$$

We define $\mathrm{keys}(e;\mathfrak{D}) = \{\textsc{a} \mid (\textsc{a} \to \mathcal{S}(e;\mathfrak{D})) \in \mathrm{fds}(e;\mathfrak{D})\}$ to be the keys we derive from expression e over $\mathfrak{D}$. Next, we define the predicate $\mathrm{set}(e;\mathfrak{D})$ which is `true` if we can derive that expression e over $\mathfrak{D}$ always evaluates to a set relation. We define $\mathrm{set}(e;\mathfrak{D})$ inductively. The base cases are relation names $R \in \mathbf{N}$, for which we have $\mathrm{set}(R;\mathfrak{D}) = \mathbf{S}(R)$. For the other operations, we have

$$
\begin{aligned}
\mathrm{set}(\dot\sigma_E(e);\mathfrak{D}) &= \mathrm{set}(e;\mathfrak{D}); & \mathrm{set}(\dot\pi_{\textsc{b}}(e);\mathfrak{D}) &= \mathrm{set}(e;\mathfrak{D}) \wedge \textsc{b} \in \mathrm{keys}(e;\mathfrak{D});\\
\mathrm{set}(\dot\rho_f(e);\mathfrak{D}) &= \mathrm{set}(e;\mathfrak{D}); & \mathrm{set}(\dot\delta(e);\mathfrak{D}) &= \texttt{true};\\
\mathrm{set}(e_1 \dot\cup e_2;\mathfrak{D}) &= \texttt{false}; & \mathrm{set}(e_1 \dot\cap e_2;\mathfrak{D}) &= \mathrm{set}(e_1;\mathfrak{D}) \vee \mathrm{set}(e_2;\mathfrak{D});\\
\mathrm{set}(e_1 \dot- e_2;\mathfrak{D}) &= \mathrm{set}(e_1;\mathfrak{D}); & \mathrm{set}(e_1 \dot\bowtie_E e_2;\mathfrak{D}) &= \mathrm{set}(e_1;\mathfrak{D}) \wedge \mathrm{set}(e_2;\mathfrak{D});\\
\mathrm{set}(e_1 \dot\ltimes_E e_2;\mathfrak{D}) &= \mathrm{set}(e_1;\mathfrak{D}); & \mathrm{set}(e_1 \dot\sqcup e_2;\mathfrak{D}) &= \mathrm{set}(e_1;\mathfrak{D}) \wedge \mathrm{set}(e_2;\mathfrak{D});\\
\mathrm{set}(\dot\iota_f(e);\mathfrak{D}) &= \mathrm{set}(e;\mathfrak{D}).
\end{aligned}
$$

The derivation rules for $\mathrm{fds}(e;\mathfrak{D})$ and $\mathrm{set}(e;\mathfrak{D})$ are not complete: it is not guaranteed that $\mathrm{fds}(e;\mathfrak{D})$ contains *all* functional dependencies that must hold in $[\![e]\!]_{\mathfrak{I}}$, for every database instance $\mathfrak{I}$ over $\mathfrak{D}$. Likewise, it is not guaranteed that $\mathrm{set}(e;\mathfrak{D}) = \texttt{false}$ implies that e is *not* a set relation.

Example 9. Let $\mathtt{u}_1, \mathtt{u}_2 \in \mathfrak{U}$ with $\mathtt{u}_1 \neq \mathtt{u}_2$ and let e be an expression over $\mathfrak{D}$. Consider the expressions $e_1 = \dot\sigma_{a=\mathtt{u}_1, a=\mathtt{u}_2}(e)$ and $e_2 = e - e$. For every database instance $\mathfrak{I}$ over $\mathfrak{D}$, we have $[\![e_1]\!]_{\mathfrak{I}} = [\![e_2]\!]_{\mathfrak{I}} = \emptyset$. Hence, every functional dependency $\textsc{a} \to \textsc{b}$ with $\textsc{a}, \textsc{b} \subseteq \mathcal{S}(e;\mathfrak{D})$ holds on $[\![e_1]\!]_{\mathfrak{I}}$ and on $[\![e_2]\!]_{\mathfrak{I}}$. In addition, both $[\![e_1]\!]_{\mathfrak{I}}$ and $[\![e_2]\!]_{\mathfrak{I}}$ are set relations. If $\mathrm{fds}(e;\mathfrak{D}) = \emptyset$, then we derive that $\mathrm{fds}(e_1;\mathfrak{D}) = {+}(\{\emptyset \to \{a\}\})$ and $\mathrm{fds}(e_2;\mathfrak{D}) = {+}(\emptyset)$. Observe that $\mathrm{fds}(e_1;\mathfrak{D})$ contains all functional dependencies over $\mathcal{S}(e_1;\mathfrak{D})$ if and only if $\mathcal{S}(e;\mathfrak{D}) = \{a\}$ and

that $\mathrm{fds}(e_2;\mathfrak{D})$ never contains all functional dependencies over $\mathcal{S}(e_1;\mathfrak{D})$. We also derive that $\mathrm{set}(e_1;\mathfrak{D}) = \mathrm{set}(e_2;\mathfrak{D}) = \mathtt{false}$, despite e_1 and e_2 evaluating to set relations.

Although the above derivation rules are not complete, they are sound:

Theorem 3. *Let e be an expression over $\mathfrak{D}$. The derivation rules for* $\mathrm{fds}(e;\mathfrak{D})$ *and* $\mathrm{set}(e;\mathfrak{D})$ *are sound: if $\mathfrak{I}$ is a database instance over $\mathfrak{D}$, then*

(i) $[\![e]\!]_{\mathfrak{I}}$ *satisfies every functional dependency in* $\mathrm{fds}(e;\mathfrak{D})$*; and*
(ii) *if* $\mathrm{set}(e;\mathfrak{D}) = \mathtt{true}$*, then* $[\![e]\!]_{\mathfrak{I}}$ *is a set relation.*

We introduce the following notions. Let $\langle \mathscr{R}_1;\tau_1\rangle$ and $\langle \mathscr{R}_2;\tau_2\rangle$ be multiset relations over A. We say that $\langle \mathscr{R}_1;\tau_1\rangle$ *is a weak subset of* $\langle \mathscr{R}_2;\tau_2\rangle$, denoted by $\langle \mathscr{R}_1;\tau_1\rangle \tilde{\subseteq} \langle \mathscr{R}_2;\tau_2\rangle$, if $\mathscr{R}_1 \subseteq \mathscr{R}_2$. We say that $\langle \mathscr{R}_1;\tau_1\rangle$ *is a strong subset of* $\langle \mathscr{R}_2;\tau_2\rangle$, denoted by $\langle \mathscr{R}_1;\tau_1\rangle \dot{\subseteq} \langle \mathscr{R}_2;\tau_2\rangle$, if $(\mathtt{t}:n) \in \langle \mathscr{R}_1;\tau_1\rangle$ implies $(\mathtt{t}:m) \in \langle \mathscr{R}_2;\tau_2\rangle$ with $n \leq m$.[12] Lemma 2 lists the main properties of these notions needed to prove Theorem 3.

Lemma 2. *Let $\langle \mathscr{R}_1;\tau_1\rangle$ and $\langle \mathscr{R}_2;\tau_2\rangle$ be multiset relations over* A*. We have*

(i) $\langle \mathscr{R}_1;\tau_1\rangle \tilde{\subseteq} \langle \mathscr{R}_2;\tau_2\rangle$ *if* $\langle \mathscr{R}_1;\tau_1\rangle \dot{\subseteq} \langle \mathscr{R}_2;\tau_2\rangle$*;*
(ii) $\langle \mathscr{R}_1;\tau_1\rangle$ *satisfies* $\mathrm{B} \to \mathrm{C}$ *if* $\langle \mathscr{R}_1;\tau_1\rangle \tilde{\subseteq} \langle \mathscr{R}_2;\tau_2\rangle$ *and* $\langle \mathscr{R}_2;\tau_2\rangle$ *satisfies functional dependency* $\mathrm{B} \to \mathrm{C}$*; and*
(iii) $\langle \mathscr{R}_1;\tau_1\rangle$ *is a set relation if* $\langle \mathscr{R}_1;\tau_1\rangle \dot{\subseteq} \langle \mathscr{R}_2;\tau_2\rangle$ *and* $\langle \mathscr{R}_2;\tau_2\rangle$ *is a set relation.*

6 Rewriting the example queries

To illustrate the techniques introduced in this paper, we provide a detailed rewriting of the queries exhibited in Example 3. We start by considering the queries $\dot{\delta}(\dot{\pi}_{age}(e))$ and $\dot{\delta}(\dot{\pi}_{cname,type}(e))$ with $e = \dot{\sigma}_{type=\mathtt{food}}(\mathit{Customer} \bowtie \mathit{Bought} \bowtie \mathit{Product})$. As a first step, we use well-known push-down rules for selection [12], and we get $e \doteq \mathit{Customer} \bowtie \mathit{Bought} \bowtie_{type=\mathtt{food}} \mathit{Product}$.

Consider the query $e' = \mathit{Bought} \bowtie_{type=\mathtt{food}} \mathit{Product}$, subquery of the above query. We have $\{\mathit{pname}\} \in \mathrm{keys}(\mathit{Product};\mathfrak{D})$ and *Product* is a set relation. Hence, we can apply Theorem 2.*(iv)* and Theorem 2.*(vi)* with $f = \{\mathit{type} := \mathtt{food}\}$, and we obtain $e' \doteq_{\{cname,pname\}} \mathit{Bought} \ltimes_{type=\mathtt{food}} \mathit{Product}$ and $e' \doteq_{\{cname,pname,type\}} \dot{\imath}_f(\mathit{Bought} \ltimes_{type=\mathtt{food}} \mathit{Product})$. Let $e'' = \mathit{Bought} \ltimes_{type=\mathtt{food}} \mathit{Product}$. For the rewriting of $\dot{\delta}(\dot{\pi}_{age}(e))$, we use Proposition 1.*(iii)* and Proposition 1.*(v)* and infer $e' \tilde{=}_{\{cname\}} e''$. Using Theorem 2.*(iii)* and transitivity, we infer $e \tilde{=}_{\{cname,age\}} \mathit{Customer} \ltimes e''$, and, using Proposition 1.*(v)*, we infer $e \tilde{=}_{\{age\}} \mathit{Customer} \ltimes e''$. Finally, we can apply Proposition 1.*(iv)* and Proposition 1.*(vi)* to conclude $\dot{\delta}(\dot{\pi}_{age}(e)) \doteq \dot{\delta}(\dot{\pi}_{age}(\mathit{Customer} \ltimes e''))$. Hence, $\dot{\delta}(\dot{\pi}_{age}(e)) \doteq \dot{\delta}(\dot{\pi}_{age}(\mathit{Customer} \ltimes (\mathit{Bought} \ltimes_{type=\mathtt{food}} \mathit{Product})))$, which is the query resulting from optimizing $\dot{\delta}(\dot{\pi}_{age}(e))$ in Example 3.

[12] Notice that $\langle \mathscr{R}_1;\tau_1\rangle \tilde{\subseteq} \langle \mathscr{R}_2;\tau_2\rangle$ does not imply that $\langle \mathscr{R}_1;\tau_1\rangle$ is fully included in $\langle \mathscr{R}_2;\tau_2\rangle$: there can be tuples $\mathtt{t} \in \mathscr{R}_1$ for which $\tau_1(\mathtt{t}) > \tau_2(\mathtt{t})$.

For the rewriting of $\dot{\delta}(\dot{\pi}_{cname,type}(e))$, we directly use Proposition 1.*(iii)* to obtain $e' \doteq_{\{cname,type\}} i_f(e'')$. We use Theorem 2.*(ii)* and transitivity to obtain $e \doteq_{\{cname,type\}} \mathit{Customer} \bowtie i_f(e'')$. Using Proposition 2.*(vi)* and commutativity of θ-join, we conclude $\mathit{Customer} \bowtie i_f(e'') \doteq_{\{cname,type\}} i_f(\mathit{Customer} \bowtie e'')$. Next, consider the query $\mathit{Customer} \bowtie e''$, subquery of the query $i_f(\mathit{Customer} \bowtie e'')$. Using Theorem 2.*(iii)*, we infer $\mathit{Customer} \bowtie e'' \tilde{=}_{\{cname\}} \mathit{Customer} \ltimes e''$ and we apply Proposition 7.*(v)* with f on both sides to obtain $i_f(\mathit{Customer} \bowtie e'') \tilde{=}_{\{cname,type\}} i_f(\mathit{Customer} \ltimes e'')$. Using transitivity, we get $e \tilde{=}_{\{cname,type\}} i_f(\mathit{Customer} \ltimes e'')$ and we apply Proposition 7.*(ii)* with $\mathrm{B} = \{cname, type\}$ to obtain $\dot{\pi}_{\mathrm{B}}(e) \tilde{=}_{\{cname,type\}} \dot{\pi}_{\mathrm{B}}(i_f(\mathit{Customer} \ltimes e''))$. Next, we apply Proposition 2.*(ii)* on $\dot{\pi}_{\mathrm{B}}(i_f(\mathit{Customer} \ltimes e''))$ and transitivity to obtain $\dot{\pi}_{\mathrm{B}}(e) \tilde{=}_{\{cname,type\}} i_f(\dot{\pi}_{cname}(\mathit{Customer} \ltimes e''))$. Then we apply Proposition 7.*(iv)* on both sides and we use Propositions 1.*(iv)* and 1.*(vii)* to obtain $\dot{\delta}(\dot{\pi}_{\mathrm{B}}(e)) \doteq \dot{\delta}(i_f(\dot{\pi}_{cname}(\mathit{Customer} \ltimes e'')))$. We observe that *Customer* is a set relation. Hence, $i_f(\dot{\pi}_{cname}(\mathit{Customer} \ltimes e''))$ is also set relation. Finally, we can now apply Proposition 6 and transitivity to conclude $\dot{\delta}(\dot{\pi}_{cname,type}(e)) \doteq i_f(\dot{\pi}_{cname}(\mathit{Customer} \ltimes e''))$ and, hence, $\dot{\delta}(\dot{\pi}_{cname,type}(e)) \doteq i_{type:=\texttt{food}}(\dot{\pi}_{cname}(\mathit{Customer} \ltimes (\mathit{Bought} \ltimes_{type=\texttt{food}} \mathit{Product})))$, which is the query resulting from optimizing $\dot{\delta}(\dot{\pi}_{cname,type}(e))$ in Example 3.

7 Conclusion

In this work, we provide a formal framework for optimizing SQL queries using query rewriting rules aimed at optimizing query evaluation for relational algebra queries using multiset semantics. The main goals of our rewrite rules are the automatic elimination of costly join steps, in favor of semi-join steps, and the automatic elimination of deduplication steps. We believe that our rewrite rules can be applied to many practical queries on which traditional techniques fall short. Hence, we believe that our results provide a promising strengthening of traditional query rewriting and optimization techniques. Based upon the ideas of our work, there are several clear directions for future work.

We have primarily studied the automatic rewriting of queries using joins into queries using semi-joins instead. In the setting of SQL, this optimization is usually obtained by rewriting joins into `WHERE ... IN ...` clauses. The anti-semi-join plays a similar role in performing `WHERE ... NOT IN ...` clauses. As such, it is only natural to ask whether our framework can be extended to also automatically rewrite towards anti-semi-join operators. A careful investigation is needed to fully incorporate anti-semi-joins in our framework, however.

The multiset relational algebra we studied does not cover all features provided by SQL. Among the missing features are aggregation and recursive queries (via `WITH RECURSIVE`), and both are candidates for further study. With respect to recursive queries, we observe that in the setting of graph query languages, usage of transitive closure to express reachability can automatically be rewritten to very fast fixpoint queries [6, 7]. Similar optimizations also apply to simple `WITH RECURSIVE` queries, but it remains open whether a general technique exists to optimize such queries.

References

1. Abiteboul, S., Hull, R., Vianu, V. (eds.): Foundations of Databases. Addison-Wesley Publishing Company, 1st edn. (1995)
2. Albert, J.: Algebraic properties of bag data types. In: Proceedings of the 17th International Conference on Very Large Data Bases. pp. 211–219. VLDB '91, Morgan Kaufmann Publishers Inc. (1991)
3. Bernstein, P.A., Chiu, D.M.W.: Using semi-joins to solve relational queries. J. ACM **28**(1), 25–40 (1981). https://doi.org/10.1145/322234.322238
4. Dayal, U., Goodman, N., Katz, R.H.: An extended relational algebra with control over duplicate elimination. In: Proceedings of the 1st ACM SIGACT-SIGMOD Symposium on Principles of Database Systems. pp. 117–123. PODS '82, ACM (1982). https://doi.org/10.1145/588111.588132
5. Grefen, P.W.P.J., de By, R.A.: A multi-set extended relational algebra: a formal approach to a practical issue. In: Proceedings of 1994 IEEE 10th International Conference on Data Engineering. pp. 80–88. IEEE (1994). https://doi.org/10.1109/ICDE.1994.283002
6. Hellings, J., Pilachowski, C.L., Van Gucht, D., Gyssens, M., Wu, Y.: From relation algebra to semi-join algebra: An approach for graph query optimization. In: Proceedings of The 16th International Symposium on Database Programming Languages. ACM (2017). https://doi.org/10.1145/3122831.3122833
7. Hellings, J., Pilachowski, C.L., Van Gucht, D., Gyssens, M., Wu, Y.: From relation algebra to semi-join algebra: An approach to graph query optimization. The Computer Journal **64**(5), 789–811 (2020). https://doi.org/10.1093/comjnl/bxaa031
8. International Organization for Standardization: ISO/IEC 9075-1: Information technology – database languages – SQL (2011)
9. Klausner, A., Goodman, N.: Multirelations: Semantice and languages. In: Proceedings of the 11th International Conference on Very Large Data Bases. pp. 251–258. VLDB '85, VLDB Endowment (1985)
10. Lamperti, G., Melchiori, M., Zanella, M.: On multisets in database systems. In: Proceedings of the Workshop on Multiset Processing: Multiset Processing, Mathematical, Computer Science, and Molecular Computing Points of View. pp. 147–215. Springer-Verlag (2001)
11. Paulley, G.N.: Exploiting Functional Dependence in Query Optimization. Ph.D. thesis, University of Waterloo (2000)
12. Ullman, J.D.: Principles of Database and Knowledge-Base Systems: Volume II: The New Technologies. W. H. Freeman & Co. (1990)
13. Yannakakis, M.: Algorithms for acyclic database schemes. In: Proceedings of the Seventh International Conference on Very Large Data Bases - Volume 7. pp. 82–94. VLDB '81, VLDB Endowment (1981)