From Relation Algebra to Semi-Join Algebra: An Approach for Graph Query Optimization[*]

Jelle Hellings
Hasselt University
Hasselt, Belgium

Catherine L. Pilachowski
Indiana University
Bloomington, IN, USA

Dirk Van Gucht
Indiana University
Bloomington, IN, USA

Marc Gyssens
Hasselt University
Hasselt, Belgium

Yuqing Wu
Pomona College
Claremont, CA, USA

ABSTRACT

Many graph query languages rely on the composition operator to navigate graphs and select nodes of interests, even though evaluating compositions of relations can be costly. Often, this need for composition can be reduced by rewriting towards queries that use semi-joins instead. In this way, the cost of evaluating queries can be significantly reduced.

We study techniques to recognize and apply such rewritings. Concretely, we study the relationship between the expressive power of the relation algebras, that heavily rely on composition, and the semi-join algebras, that replace the composition operator in favor of the semi-join operators.

As our main result, we show that each fragment of the relation algebras where intersection and/or difference is only used on edges (and not on complex compositions) is expressively equivalent to a fragment of the semi-join algebras. This expressive equivalence holds for node queries that evaluate to sets of nodes. For practical relevance, we exhibit constructive steps for rewriting relation algebra queries to semi-join algebra queries, and prove that these steps lead to only a well-bounded increase in the number of steps needed to evaluate the rewritten queries.

In addition, on node-labeled graphs that are sibling-ordered trees, we establish new relationships among the expressive power of Regular XPath, Conditional XPath, FO-logic, and the semi-join algebra augmented with restricted fixpoint operators.

CCS CONCEPTS

• **Theory of computation → Database query processing and optimization (theory)**; *Logic and databases*; *Database query languages (principles)*; Finite Model Theory;

[*]This material is based upon work supported by the National Science Foundation under Grant No. NSF 1438990.

DBPL 2017, September 1, 2017, Munich, Germany
© 2017 Association for Computing Machinery.
ACM ISBN 978-1-4503-5354-0/17/09…$15.00
https://doi.org/10.1145/3122831.3122833

ACM Reference format:
Jelle Hellings, Catherine L. Pilachowski, Dirk Van Gucht, Marc Gyssens, and Yuqing Wu. 2017. From Relation Algebra to Semi-Join Algebra: An Approach for Graph Query Optimization. In *Proceedings of DBPL 2017, Munich, Germany, September 1, 2017,* 10 pages.
https://doi.org/10.1145/3122831.3122833

1 INTRODUCTION

The graph data model (representing labeled binary relations) is a versatile and natural data model for representing RDF data, social networks, gene and protein networks, and other sources of data.

Example 1.1. In Figure 1 we show a simple social network represented by graph data. Its nodes represent objects corresponding to persons and its labeled edges represent various semantic relationships between these persons. In this example, we have the *ParentOf* and *FriendOf* relationships. In Figure 2 we show the same data represented as labeled binary relations.

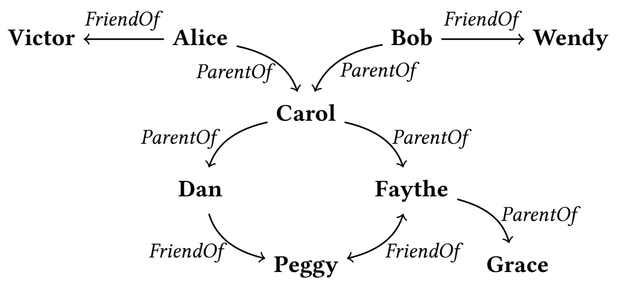

Figure 1: An example of social network graph data.

ParentOf

Alice	Carol
Bob	Carol
Carol	Dan
Carol	Faythe
Faythe	Grace

FriendOf

Alice	Victor
Bob	Wendy
Dan	Peggy
Faythe	Peggy
Peggy	Faythe

Figure 2: Labeled binary relations representing the graph data in Figure 1.

To query such graph data, many navigational query languages have been developed which, at their core, use a fragment of the relation algebra of Tarski [25], augmented with the Kleene-star operator (transitive closure). Examples of such relation-algebra-inspired query languages include XPath and its many formalizations [2, 3, 5, 20, 21, 26, 27], GXPath [19], the (nested) regular path queries [1], and the navigational expressions [7, 8, 10, 12, 24]. In these languages, graph navigation is primarily supported by *composition* ($\circ$). To see this, consider the query defined by the following relation algebra expression:

$$\mathit{ParentOf} \circ \mathit{ParentOf} \circ \mathit{FriendOf}.$$

This query searches for all pairs of people (m, n) such that n is a friend of m's grandchild. When applied to the data shown in Figure 1, it returns the binary relation

Alice	Peggy
Bob	Peggy

As another example, consider the expression

$$\pi_1[\mathit{ParentOf} \circ \mathit{ParentOf} \circ \mathit{ParentOf}] \circ \mathit{FriendOf},$$

which defines the query that yields the set of all pairs of great-grandparents and their friends. Indeed, in this expression, the projection

$$\pi_1[\mathit{ParentOf} \circ \mathit{ParentOf} \circ \mathit{ParentOf}]$$

returns the set of pairs of the form (m, m) where m is a great-grandparent, i.e., the pairs $\{(\text{Alice}, \text{Alice}), (\text{Bob}, \text{Bob})\}$. When this relation is composed with the *FriendOf* relation, we get the desired result $\{(\text{Alice}, \text{Victor}), (\text{Bob}, \text{Wendy})\}$.

These examples illustrate that the composition operator captures the intent of graph navigation in a simple and intuitive way, which explains why many graph query languages rely on it. In the setting of big data, this use of composition for graph navigation has a major drawback, however. Computing query results by evaluating each of the compositions involved is costly, both in terms of runtime and in terms of memory requirements.

Instead of relying on composition for graph navigation, one can consider using the semi-join operators $\ltimes$ and $\rtimes$. Rather than computing the composition of relations, semi-joins instead only determine the pairs that are involved in such compositions. In particular, when R and S are binary relations, the left semi-join $\mathrm{R} \ltimes \mathrm{S}$ determines the pairs in R that can be composed with pairs in S, i.e., $\{(m, n) \in \mathrm{R} \mid \exists z\, (n, z) \in \mathrm{S}\}$ and the right semi-join $\mathrm{R} \rtimes \mathrm{S}$ determines the pairs in S that can be composed with pairs in R, i.e., $\{(m, n) \in \mathrm{S} \mid \exists z\ (z, m) \in \mathrm{R}\}$. Using the semi-join operators we can rewrite the above expression $\pi_1[\mathit{ParentOf} \circ \mathit{ParentOf} \circ \mathit{ParentOf}] \circ \mathit{FriendOf}$ into the following equivalent expression:

$$\pi_1[\mathit{ParentOf} \ltimes (\mathit{ParentOf} \ltimes \mathit{ParentOf})] \rtimes \mathit{FriendOf}.$$

The main advantage of replacing compositions by semi-joins in rewriting is that the evaluation of the resulting expression (by evaluating each operation involved) is possible in linear time with respect to the size of relations, whereas evaluation of the original expression takes quadratic time. This also holds in general, as it is well-known that evaluating semi-joins is more efficient than evaluating compositions, even in the worst-case [18].

To achieve these improvements in practice, we can add the semi-join operators to appropriate query languages. This does, however, put the burden of efficient query evaluation on the users: in the above rewriting, we needed both the left and right semi-join operators. With respect to the former, we additionally had to insert parenthesis to control the order of evaluation of this non-associative operator. So, even in this simple example, the resulting expression becomes less intuitive and harder to write. Therefore, we believe that in modern graph database systems, which use declarative high-level graph query languages, such rewritings should be performed *for the users, rather than by the users.*

As an alternative to manual rewriting, we study ways to apply semi-join optimizations automatically. Towards this goal, we study the relation algebras—augmented with the Kleene star—and how they relate to the semi-join algebras obtained by replacing composition by semi-joins and the Kleene-star by appropriate less-costly forms of fixpoint iteration.

To the best of our knowledge, we are the first to study the relationships between the expressive power of the relation algebra and the semi-join algebra in their *full* generality. We should point out that the study of semi-joins has already received attention in the setting of Codd's relational algebra [13, 15–17]. In this setting, the semi-join version of the relational algebra is studied as a query language that has limited expressive power, cheap query evaluation, and for which many decision problems are decidable.

In the design and implementation of relational database systems, *basic* semi-join rewrite rules are well-known and the automatic usage of semi-join steps plays an important role in the evaluation of distributed joins [4] and the evaluation of acyclic joins (Yannakakis algorithm) [28, 31]. In both cases, these semi-join steps are used as *reducers* that provide a preprocessing step aimed at reducing the size of intermediate relations before joining them. A similar reducer-based role for the semi-join has also been studied in the context of the multiset relational algebra [23]. This focus on using the semi-join as a reducer sharply contrasts with our usage, as we aim at eliminating compositions altogether in favor of semi-joins.

It is well-known that the relation algebra has the same expressive power as FO[3], the first-order logic in which formulae are restricted to having three variables [10, 25]. We show that the semi-join algebra has the same expressive power as FO[2]. Hence, our work also studies relationships between FO[3] and FO[2].

Our main results are as follows:

(1) We establish sufficient conditions on a relation algebra expression that, when satisfied, imply that we can rewrite the expression into a semi-join algebra expression that, when interpreted as a node query on graphs, is equivalent to the original expression.

(2) We extend the above results towards relation algebra augmented with the Kleene-star operator. For this augmented algebra, we show a collapse on the level of node queries towards the semi-join algebra augmented with fixpoint iteration.

(3) To substantiate the practical usefulness of the above collapses, we present a constructive rewrite procedure to automatically rewrite expressions and subexpressions written in the relation algebra into semi-join algebra expressions. If the input to this procedure is an expression of length s and using at most u

union-operations, then application of the rewrite procedure we propose yields a rewritten expression that can be evaluated in at most $s + u \leq 2 \cdot s$ evaluation steps, demonstrating the practical feasibility of our rewrite techniques.

(4) We only place conditions on the usage of intersection and difference. To show that these restrictions are not too severe, we show that every query expressible in the semi-join algebra can be rewritten into an equivalent expression in the relation algebra that satisfies all restrictions we put on intersection and difference. We provide a similar rewriting for the semi-join algebra augmented with fixpoint iteration, which we rewrite towards the relation algebra augmented with the Kleene-star.

(5) In the setting of finite sibling ordered trees [20], we show that our results imply that first-order logic on trees ($\mathrm{FO}^{\mathrm{tree}}$) collapses to FO[2] (i.e., the semi-join algebra) with respect to node queries.

2 GRAPH DATA MODEL AND QUERIES

In this work, we use an edge-labeled graph data model. A *graph* is a triple $\mathcal{G} = (\mathbf{N}, \Sigma, \mathbf{E})$, with $\mathbf{N}$ a finite set of nodes, Σ a finite set of edge labels, and $\mathbf{E} : \Sigma \to 2^{\mathbf{N}\times\mathbf{N}}$ a function mapping edge labels to edge relations.

We define a *query* as a mapping from graphs to binary relations over $\mathbf{N}$. It is useful to adopt the following notations: if R is a binary relation, then $\mathrm{R}|_1 = \{m \mid \exists n\, (m, n) \in \mathrm{R}\}$ and $\mathrm{R}|_2 = \{n \mid \exists m\, (m, n) \in \mathrm{R}\}$ denote the first and second column of R. If a query maps every graph into a set of identical pairs of nodes, i.e. into a subset of the pairs $\{(n, n) \mid n \in \mathbf{N}\}$, then we call it a *node query*. The intuition is that we interpret a pair (n, n) as a representation for the node n and, as such, a node query "maps" each graph into a subset of the nodes of that graph.

In subsequent sections, we consider *expressions* of various algebraic languages and restricted first-order logic languages which define queries. If q is such an expression, then we denote the *evaluation* of q on graph $\mathcal{G}$ by $[\![q]\!]_{\mathcal{G}}$. If an expression defines a node query, we call it a *node expression*.

2.1 Equivalence notions

To reason about the soundness of rewrite rules for expressions, we need notions of expression equivalence. We consider three such notions: path-equivalence, left-projection-equivalence, and right-projection-equivalence. Let q_1 and q_2 be expressions. We say that q_1 and q_2 are

path-equivalent, denoted by $q_1 \equiv_{\mathrm{path}} q_2$, if, for every graph $\mathcal{G}$, $[\![q_1]\!]_{\mathcal{G}} = [\![q_2]\!]_{\mathcal{G}}$;

left-projection-equivalent, denoted by $q_1 \equiv_{\pi_1} q_2$, if, for every graph $\mathcal{G}$, $[\![q_1]\!]_{\mathcal{G}}|_1 = [\![q_2]\!]_{\mathcal{G}}|_1$; and

right-projection-equivalent, denoted by $q_1 \equiv_{\pi_2} q_2$, if, for every graph $\mathcal{G}$, $[\![q_1]\!]_{\mathcal{G}}|_2 = [\![q_2]\!]_{\mathcal{G}}|_2$.

Clearly, expressions that are path-equivalent are also left- and right-projection-equivalent. The reverse is not true in general. Indeed, the expressions $\mathrm{R} \circ \mathrm{S}$ and $\mathrm{R} \ltimes \mathrm{S}$ are left-projection-equivalent, but not path-equivalent. Likewise, the expressions $\mathrm{R} \circ \mathrm{S}$ and $\mathrm{R} \rtimes \mathrm{S}$ are right-projection-equivalent, but not path-equivalent. Finally, we observe that the three equivalence notions coincide on the class of node expressions.

Example 2.1. Looking back at the example queries used in the Introduction, we formally have

$$\mathit{ParentOf} \ltimes (\mathit{ParentOf} \ltimes \mathit{ParentOf}) \equiv_{\pi_1} \mathit{ParentOf} \circ \mathit{ParentOf} \circ \mathit{ParentOf}$$

and also

$$\pi_1[\mathit{ParentOf} \circ \mathit{ParentOf} \circ \mathit{ParentOf}] \circ \mathit{FriendOf} \equiv_{\mathrm{path}} \pi_1[\mathit{ParentOf} \ltimes (\mathit{ParentOf} \ltimes \mathit{ParentOf})] \rtimes \mathit{FriendOf}.$$

2.2 Expressive power

The equivalence notions introduced in the previous sections extend naturally to subsumption and equivalence notions between classes of expressions.

Let $z \in \{\mathrm{path}, \pi_1, \pi_2\}$. We say that the class of expressions $\mathcal{L}_1$ is *z-subsumed* by the class of expressions $\mathcal{L}_2$, denoted by $\mathcal{L}_1 \preceq_z \mathcal{L}_2$, if every expression in $\mathcal{L}_1$ is z-equivalent to an expression in $\mathcal{L}_2$. We say that the classes of expressions $\mathcal{L}_1$ and $\mathcal{L}_2$ are *z-equivalent*, denoted by $\mathcal{L}_1 \equiv_z \mathcal{L}_2$, if $\mathcal{L}_1 \preceq_z \mathcal{L}_2$ and $\mathcal{L}_2 \preceq_z \mathcal{L}_1$. We say that $\mathcal{L}_1$ and $\mathcal{L}_2$ are *projection-equivalent*, denoted by $\mathcal{L}_1 \equiv_\pi \mathcal{L}_2$, if $\mathcal{L}_1 \equiv_{\pi_1} \mathcal{L}_2$ and $\mathcal{L}_1 \equiv_{\pi_2} \mathcal{L}_2$.

3 NAVIGATIONAL GRAPH QUERIES

In this work, we mainly study the relationship between the expressive power of the relation algebra and the semi-join algebra, which are algebraic representations of FO[3] and FO[2], respectively. In our study, we also include iteration—in the form of transitive closure—as iteration is essential in graph querying.

3.1 Relation algebra and the semi-join algebra

The *graph expressions* are defined by the grammar

$$e := \emptyset \mid \mathrm{id} \mid \mathrm{di} \mid \ell \mid \ell^{\frown} \mid \pi_j[e] \mid \overline{\pi}_j[e] \mid e \circ e \mid e \ltimes e \mid e \rtimes e \mid e \cup e \mid e \cap e \mid e - e,$$

in which $\ell \in \Sigma$ and $j \in \{1, 2\}$. Let $\mathcal{G} = (\mathbf{N}, \Sigma, \mathbf{E})$ be a graph and let e be an expression. The semantics of evaluation is defined as follows:

$$\begin{aligned}
[\![\emptyset]\!]_{\mathcal{G}} &= \emptyset;\\
[\![\mathrm{id}]\!]_{\mathcal{G}} &= \{(m, m) \mid m \in \mathbf{N}\};\\
[\![\mathrm{di}]\!]_{\mathcal{G}} &= \{(m, n) \mid m, n \in \mathbf{N} \wedge m \neq n\};\\
[\![\ell]\!]_{\mathcal{G}} &= \{(m, n) \mid (m, n) \in \mathbf{E}(\ell)\} \quad (\text{with } \ell \in \Sigma);\\
[\![\ell^{\frown}]\!]_{\mathcal{G}} &= \{(n, m) \mid (m, n) \in \mathbf{E}(\ell)\} \quad (\text{with } \ell \in \Sigma);\\
[\![\pi_j[e]]\!]_{\mathcal{G}} &= \{(m, m) \mid m \in [\![e]\!]_{\mathcal{G}}|_j\} \quad (\text{with } j \in \{1, 2\});\\
[\![\overline{\pi}_j[e]]\!]_{\mathcal{G}} &= [\![\mathrm{id}]\!]_{\mathcal{G}} - [\![\pi_j[e]]\!]_{\mathcal{G}} \quad (\text{with } j \in \{1, 2\});\\
[\![e_1 \circ e_2]\!]_{\mathcal{G}} &= \{(m, n) \mid \exists z\, (m, z) \in [\![e_1]\!]_{\mathcal{G}} \wedge (z, n) \in [\![e_2]\!]_{\mathcal{G}}\};\\
[\![e_1 \ltimes e_2]\!]_{\mathcal{G}} &= \{(m, n) \mid (m, n) \in [\![e_1]\!]_{\mathcal{G}} \wedge \exists z\, (n, z) \in [\![e_2]\!]_{\mathcal{G}}\};\\
[\![e_1 \rtimes e_2]\!]_{\mathcal{G}} &= \{(m, n) \mid (m, n) \in [\![e_2]\!]_{\mathcal{G}} \wedge \exists z\, (z, m) \in [\![e_1]\!]_{\mathcal{G}}\};\\
[\![e_1 \oplus e_2]\!]_{\mathcal{G}} &= [\![e_1]\!]_{\mathcal{G}} \oplus [\![e_2]\!]_{\mathcal{G}} \quad (\text{with } \oplus \in \{\cup, \cap, -\}).
\end{aligned}$$

The *relation algebra*, which we denote by $\mathcal{N}_3$, allows every operator above except for the semi-joins ($\ltimes$ and $\rtimes$). The *semi-join algebra*, which we denote by $\mathcal{N}_2$, allows every operator above except for composition ($\circ$).

Example 3.1. The queries

$$\mathsf{Q}_1 = \pi_1[\mathit{ParentOf} \circ \overline{\pi}_1[\mathit{OwnsPet}] \circ \mathit{ResearcherAt}];$$
$$\mathsf{Q}_2 = \pi_1[\mathit{ParentOf} \ltimes (\overline{\pi}_1[\mathit{OwnsPet}] \ltimes \mathit{ResearcherAt})],$$

both return people that are parents of researchers that do not own any pets. The query Q_1 is a relation algebra expression and the query Q_2 is a semi-join algebra expression. Both expressions are node expressions and we have $\mathsf{Q}_1 \equiv_{\text{path}} \mathsf{Q}_2$. (Observe that, more generally, for any expression e, $\pi_1[e]$ and $\pi_2[e]$ are node expressions.)

3.2 Adding iteration

The relation algebra, as a graph query language, is usually augmented with a general Kleene-star operator (transitive closure): if e is an expression, then so is $[e]^*$. The semantics of evaluating $[e]^*$ on graph $\mathcal{G}$ is $[\![[e]^*]\!]_{\mathcal{G}} = \bigcup_{0 \leq i} [\![e^i]\!]_{\mathcal{G}}$ with $e^0 = \text{id}$ and $e^k = e \circ e^{k-1}$. We denote the relation algebra, augmented with the Kleene-star, by $\mathcal{N}_3^*$.

Example 3.2. The query

$$\mathsf{Q}_3 = \pi_1[[\mathit{ParentOf} \circ \overline{\pi}_1[\mathit{OwnsPet}]]^* \circ \mathit{ResearcherAt}]$$

returns people that are ancestors of a chain of descendants that do not own pets, where the youngest descendant is also a researcher.

As an FO[2]-like counterpart of the Kleene-star, which is inherently based on the composition, we introduce a form of fixpoint iteration. We add the operator $\mathsf{fp}_{i,\mathfrak{R}}[e \text{ union } b]$ with $i \in \{1, 2\}$, b an expression, e an expression, and $\mathfrak{R}$ the single free variable of e. We do not allow $\mathfrak{R}$ to occur elsewhere.

The semantics of evaluating $\mathsf{fp}_{i,\mathfrak{R}}[e \text{ union } b]$ on graph $\mathcal{G}$ is defined next. Let $s_0 := [\![b]\!]_{\mathcal{G}}|_i$ and define $s_j := s_{j-1} \cup [\![e]\!]_{\mathcal{G}+s_{j-1}}|_i$ in which $\mathcal{G}+s_{j-1}$ is the graph obtained from $\mathcal{G}$ by interpreting the set of nodes s_{j-1} as the edge relation $\{(n, n) \mid n \in s_{j-1}\}$ labeled with $\mathfrak{R}$. Due to monotonicity of $\cup$, there is bound to exist a k, $k \leq |\mathbf{N}|$, such that $s_k = s_{k+1}$. We define $[\![\mathsf{fp}_{i,\mathfrak{R}}[e \text{ union } b]]\!]_{\mathcal{G}} = \{(n, n) \mid n \in s_k\}$.

Example 3.3. Let $e = \mathit{ParentOf} \ltimes (\overline{\pi}_1[\mathit{OwnsPet}] \ltimes \mathfrak{R})$. This expression has a single free variable $\mathfrak{R}$. Now consider the queries Q_3 of Example 3.2 and

$$\mathsf{Q}_4 = \mathsf{fp}_{1,\mathfrak{R}}[e \text{ union } \mathit{ResearcherAt}].$$

We have $\mathsf{Q}_3 \equiv_{\text{path}} \mathsf{Q}_4$ and we observe that Q_4 does not have free variables.

We only introduce fixpoint iteration here as a less-costly alternative to the Kleene-star. For this purpose, general fixpoints are too strong, however. Therefore, we put restrictions on the expression e used in $\mathsf{fp}_{i,\mathfrak{R}}[e \text{ union } b]$: if $i = 1$, then e must be *right-recursive* in $\mathfrak{R}$ and, if $i = 2$, then e must be *left-recursive* in $\mathfrak{R}$.

Let $x \in \{\text{left}, \text{right}\}$. If $\mathfrak{R}$ is a variable, then the expression $\mathfrak{R}$ is x-recursive in $\mathfrak{R}$. Expressions of the form $e = e_1 \cup e_2$ are x-recursive in $\mathfrak{R}$ if e_1 and e_2 are x-recursive in $\mathfrak{R}$. Expressions of the form $e = \mathsf{fp}_{j,\mathfrak{R}'}[e' \text{ union } b']$ are x-recursive in $\mathfrak{R}$ if b' is x-recursive in $\mathfrak{R}$ ($j = 1$ if $x = \text{right}$ and $j = 2$ if $x = \text{left}$). Expressions of the form $e = e_1 \ltimes e_2$ are right-recursive in $\mathfrak{R}$ if e_1 does not have free variables and e_2 is right-recursive in $\mathfrak{R}$. Finally, expressions of the form $e = e_1 \rtimes e_2$ are left-recursive in $\mathfrak{R}$ if e_2 does not have free variables and e_1 is left-recursive in $\mathfrak{R}$.

Example 3.4. The expression $e = \mathit{ParentOf} \ltimes (\overline{\pi}_1[\mathit{OwnsPet}] \ltimes \mathfrak{R})$, as used in query Q_4 of Example 3.3, is right-recursive. The expression $e' = \mathfrak{R} \rtimes \mathit{FamilyOf} \cup \mathfrak{R} \rtimes \mathit{FriendOf}$ is left-recursive. The query

$$\mathsf{Q}_5 = \mathsf{fp}_{2,\mathfrak{R}}[e' \text{ union } \mathit{OwnsPet}^\frown]$$

yields pet owners and people that are related to pet owners via friend and family relations.

We denote the semi-join algebra, augmented with this restricted form of fixpoint iteration, by $\mathcal{N}_2^{\mathsf{fp}}$.

3.3 Relationships with other languages

Observe that the relation algebra and the semi-join algebra are single-sorted languages, i.e., every expression operates on binary relations and yields binary relations. This is in contrast with well-known languages such as GXPath and KAT [14, 19], in which expressions can yield unary relations (sets of nodes) or binary relations. The results in this paper indicate that it is not necessary to have these two-sorted expressions from the point of view of automatic query optimization. Hence, we prefer using simpler single-sorted languages instead.

We finish this section with some observations about the first-order counterparts of the relation algebra and the semi-join algebra. Let $\mathcal{G} = (\mathbf{N}, \Sigma, \mathbf{E})$ be a graph with $\Sigma = \{\ell_1, \dots, \ell_{|\Sigma|}\}$. We write FO[$k$] to denote the first-order logic where variables are restricted to a set of at most k variables over the structure $(\mathbf{N}; \ell_1, \dots, \ell_{|\Sigma|})$, in which ℓ_j, $1 \leq j \leq |\Sigma|$, represents the edge relation $\mathbf{E}(\ell_j)$.

It is well-known that the relation algebra has the same expressive power as FO[3] formulae with two free variables [10, 25]. Likewise, based on the equivalence of FO[2] and the multi-dimensional modal logic MLR_2 [22], we can establish the following proposition. (Due to space limitation we omit the proof.)

Proposition 3.5. *We have* FO[2] $\equiv_{\text{path}} \mathcal{N}_2$. *The language* $\mathcal{N}_2^{\mathsf{fp}}$ *is path-subsumed by* FO[2] *to which either inflationary fixpoints or infinitary connectives have been added.*

4 REWRITING THE RELATION ALGEBRAS

In this section, we explore ways to automatically rewrite expressions with composition and Kleene-star operators into expression with semi-join and fixpoint operators. The aim of this is to obtain queries that can be evaluated more efficiently.

The rewrite rules we present try to eliminate costly composition and Kleene-star operations. Towards this goal, the following observations are useful. Expressions of the form $f_j[e]$, with $f \in \{\pi, \overline{\pi}\}$ and $j \in \{1, 2\}$, are node expressions. We identify two situations in which the presence of such node expressions allow for optimizations by eliminating composition in favor of semi-joins:

(1) The expression itself is a node expression due to the usage of projections or coprojections at the outer level. An example is the expression $\pi_1[e_1 \circ e_2]$, where a straightforward rewriting yields $\pi_1[e_1 \circ e_2] \equiv_{\text{path}} \pi_1[e_1 \ltimes e_2]$.

(2) In a composition $e_1 \circ e_2$, one of the two expressions involved is a node expression. An example is the expression $e_1 \circ \pi_1[e_2]$, where a straightforward rewriting yields $e_1 \circ \pi_1[e_2] \equiv_{\text{path}} e_1 \ltimes \pi_1[e_2]$.

Since the semantics of the Kleene-star is defined in terms of composition, similar observations can be made with respect to the Kleene-star.

To support the rewritings of compositions towards semi-joins and Kleene-stars to fixpoints in cases similar to the situations discussed above, we propose the rewrite rules of Figure 3. We will argue later that $\tau(e) \equiv_{\text{path}} e$, $\tau_{\pi_1}(e) \equiv_{\pi_1} e$, and $\tau_{\pi_2}(e) \equiv_{\pi_2} e$.

We observe that in these rewrite rules expressions restricted to contain only edge labels ℓ, $\ell \in \Sigma$, conversed edge labels $\ell^\frown$, the relations $\emptyset$, id, and di, and the binary operators $\cup$, $\cap$, and $-$ are expressible in both $\mathcal{N}_3$ and in $\mathcal{N}_2$. We refer to these expressions as *basic expressions.* The rewrite rules we consider do not change these basic expressions. If e is an expression in $\mathcal{N}_3^*$, then $\tau(e)$ results in a path-equivalent expression. Likewise, $\tau_{\pi_1}(e)$ and $\tau_{\pi_2}(e)$ result in left-, respectively, right-projection-equivalent expression.

Example 4.1. Consider the query

$$\mathsf{Q}_6 = \pi_1[((\mathit{WorksOn} \circ \mathit{WorksOn}^\frown) \cap \mathit{FriendOf}) \circ \mathit{EditorOf}] \circ \mathit{StudentOf}.$$

This query returns pairs of professors and their students, such that the professor is friends with an editor with whom the professor collaborates on a project. For clarity, we abbreviate each edge label in Q_6, resulting in $\pi_1[((W \circ W^\frown) \cap F) \circ E] \circ S$. We have the following:

$$\begin{aligned}
\tau(\mathsf{Q}_6) &= \tau_{\pi_2}(\pi_1[((W \circ W^\frown) \cap F) \circ E]) \rtimes \tau(S)\\
&= \pi_1[\tau_{\pi_1}(((W \circ W^\frown) \cap F) \circ E)] \rtimes S\\
&= \pi_1[\tau_{\circ_1}((W \circ W^\frown) \cap F; \tau_{\pi_1}(E))] \rtimes S\\
&= \pi_1[(\tau(W \circ W^\frown) \cap \tau(F)) \ltimes E] \rtimes S\\
&= \pi_1[((\tau(W) \circ \tau(W^\frown)) \cap F) \ltimes E] \rtimes S\\
&= \pi_1[((W \circ W^\frown) \cap F) \ltimes E] \rtimes S.
\end{aligned}$$

We shall prove (Theorem 4.4) that Q_6 and $\tau(\mathsf{Q}_6)$ are path-equivalent. This rewriting results in a query in which two out of three applications of composition are eliminated in favor of semi-joins. In Example 6.4 (Section 6), we shall show that the last remaining composition step is unavoidable.

The rules of Figure 3 depend on the ability to determine if an expression e is a node expression. This is, in general, hard to determine without evaluation of e. We can, however, use the semantics of $\mathcal{N}_3^*$ to define a predicate ns(e) that evaluates to true *only if* the expression e is a node expression:

$$\begin{aligned}
\mathsf{ns}(\emptyset) = \mathsf{ns}(\text{id}) &= \mathtt{True};\\
\mathsf{ns}(\text{di}) &= \mathtt{False};\\
\mathsf{ns}(\ell) = \mathsf{ns}(\ell^\frown) &= \text{“}\ell \in \Sigma \text{ is a node label”};\\
\mathsf{ns}(f_j e) &= \mathtt{True} \quad (\text{with } f \in \{\pi, \overline{\pi}\} \text{ and } j \in \{1,2\});\\
\mathsf{ns}(e_1 \circ e_2) &= \mathsf{ns}(e_1) \land \mathsf{ns}(e_2);\\
\mathsf{ns}(e_1 \ltimes e_2) = \mathsf{ns}(e_2 \rtimes e_1) &= \mathsf{ns}(e_1);\\
\mathsf{ns}(e_1 \cup e_2) &= \mathsf{ns}(e_1) \land \mathsf{ns}(e_2);\\
\mathsf{ns}(e_1 \cap e_2) &= \mathsf{ns}(e_1) \lor \mathsf{ns}(e_2);\\
\mathsf{ns}(e_1 - e_2) &= \mathsf{ns}(e_1) \lor (e_2 = \text{di});\\
\mathsf{ns}([e]^*) &= \mathsf{ns}(e);\\
\mathsf{ns}(\mathsf{fp}_{j,\mathfrak{R}}[e \text{ union } b]) &= \mathtt{True} \quad (\text{with } j \in \{1,2\});.
\end{aligned}$$

We not only claim soundness of the rewrite rules of Figure 3. We also argue that the rewrite rules are practically useful for graph query optimization. Thereto, we also analyze the complexity of the expression resulting from the rewritten expression in terms of the expression size, the number of steps needed for evaluation, and the complexity of the operations involved. In this analysis, we use the following terminology:

Definition 4.2. The *size* of an expression e, denoted by $\|e\|$, is the number of operations in e. We have

$$\begin{aligned}
\|\emptyset\| = \|\text{id}\| = \|\text{di}\| = \|\ell\| &= \|\ell^\frown\| = 0;\\
\|f_j[e]\| &= 1 + \|e\|;\\
\|e_1 \otimes e_2\| &= 1 + \|e_1\| + \|e_2\|;\\
\|e_1 \oplus e_2\| &= 1 + \|e_1\| + \|e_2\|;\\
\|[e]^*\| &= 1 + \|e\|;\\
\|\mathfrak{R}\| &= 0;\\
\|\mathsf{fp}_{j,\mathfrak{R}}[e \text{ union } b]\| &= 1 + \|e\| + \|b\|,
\end{aligned}$$

with $f \in \{\pi, \overline{\pi}\}$, $j \in \{1,2\}$, $\otimes \in \{\circ, \ltimes, \rtimes\}$, and $\oplus \in \{\cup, \cap, -\}$. The *subexpression set* of e, denoted by $\mathcal{S}(e)$, is the set of all unique non-atomic subexpressions that must be evaluated:

$$\begin{aligned}
\mathcal{S}(\emptyset) = \mathcal{S}(\text{id}) = \mathcal{S}(\text{di}) = \mathcal{S}(\ell) &= \mathcal{S}(\ell^\frown) = \emptyset;\\
\mathcal{S}(f_j[e]) &= \{f_j[e]\} \cup \mathcal{S}(e);\\
\mathcal{S}(e_1 \otimes e_2) &= \{e_1 \otimes e_2\} \cup \mathcal{S}(e_1) \cup \mathcal{S}(e_2);\\
\mathcal{S}(e_1 \oplus e_2) &= \{e_1 \oplus e_2\} \cup \mathcal{S}(e_1) \cup \mathcal{S}(e_2);\\
\mathcal{S}([e]^*) &= \{[e]^*\} \cup \mathcal{S}(e);\\
\mathcal{S}(\mathfrak{R}) &= \emptyset;\\
\mathcal{S}(\mathsf{fp}_{j,\mathfrak{R}}[e \text{ union } b]) &= \{\mathsf{fp}_{j,\mathfrak{R}}[e \text{ union } b]\} \cup \mathcal{S}(e) \cup \mathcal{S}(b).
\end{aligned}$$

The *evaluation size* of e, denoted by eval-steps(e), is defined by eval-steps$(e) = |\mathcal{S}(e)|$.

Example 4.3. Consider the query

$$\mathsf{Q}_7 = ((\ell \circ \ell) \circ (\ell \circ \ell)) \circ ((\ell \circ \ell) \circ (\ell \circ \ell)).$$

We have $\|\mathsf{Q}_7\| = 7$, whereas we have eval-steps$(\mathsf{Q}_7) = 3$. Indeed, this expression can be evaluated in three steps, namely by first evaluating $e_1 = \ell \circ \ell$, next $e_2 = e_1 \circ e_1$, and, finally, $e = e_2 \circ e_2$. Now consider the query

$$\mathsf{Q}_7' = \ell \circ (\ell \circ (\ell \circ (\ell \circ (\ell \circ (\ell \circ (\ell \circ \ell)))))),$$

for which we have $\mathsf{Q}_7 \equiv_{\text{path}} \mathsf{Q}_7'$ and $\|\mathsf{Q}_7'\| = $ eval-steps$(\mathsf{Q}_7') = 7$.

We observe that the only rewrite rule of Figure 3 that increases the expression size significantly is the rewrite rule $\tau_{\circ_i}(e_1 \cup e_2; \varepsilon) = \tau_{\circ_i}(e_1; \varepsilon) \cup \tau_{\circ_1}(e_2; \varepsilon)$, as this rewrite rule duplicates the expression ε. By $\mathsf{u}(\tau(e))$, $\mathsf{u}(\tau_{\pi_j}(e))$, and $\mathsf{u}(\tau_{\circ_j}(e;\varepsilon))$ ($j \in \{1,2\}$), we denote the number of times the rewrite rule $\tau_{\circ_i}(e_1 \cup e_2; \varepsilon) = \tau_{\circ_i}(e_1; \varepsilon) \cup \tau_{\circ_1}(e_2; \varepsilon)$ has been applied in the rewriting of e using the rewrite rules $\tau(e)$, $\tau_{\pi_j}(e)$, or $\tau_{\circ_j}(e; \varepsilon')$, respectively.

Using structural induction on relation algebra expressions, we can establish the following main result about the soundness and evaluation complexity of the rewrite rules in Figure 3 (proof omitted):

Theorem 4.4. *Let e be an expression in $\mathcal{N}_3^*$.*

$$\begin{array}{llll}
\tau(b) = b & \tau_{\pi_i}(b) = b & \tau_{\circ_1}(b;\varepsilon) = b \ltimes \varepsilon & \tau_{\circ_2}(b;\varepsilon) = \varepsilon \rtimes b \\
\tau(f_j[e]) = f_j[\tau_{\pi_j}(e)] & \tau_{\pi_i}(f_j[e]) = f_j[\tau_{\pi_j}(e)] & \tau_{\circ_1}(f_j[e];\varepsilon) = f_j[\tau_{\pi_j}(e)] \ltimes \varepsilon & \tau_{\circ_2}(f_j[e];\varepsilon) = \varepsilon \rtimes f_j[\tau_{\pi_j}(e)] \\
\tau(e_1 \circ e_2) = \circ_{\text{path}}(e_1;e_2) & \tau_{\pi_i}(e_1 \circ e_2) = \circ_{\pi_i}(e_1;e_2) & \tau_{\circ_i}(e_1 \circ e_2;\varepsilon) = \tau_{\circ_i}(e_1;\tau_{\circ_i}(e_2;\varepsilon)) & \\
\tau(e_1 \cup e_2) = \tau(e_1) \cup \tau(e_2) & \tau_{\pi_i}(e_1 \cup e_2) = \tau_{\pi_i}(e_1) \cup \tau_{\pi_i}(e_2) & \tau_{\circ_i}(e_1 \cup e_2;\varepsilon) = \tau_{\circ_i}(e_1;\varepsilon) \cup \tau_{\circ_i}(e_2;\varepsilon) & \\
\tau(e_1 \oplus e_2) = \tau(e_1) \oplus \tau(e_2) & \tau_{\pi_i}(e_1 \oplus e_2) = \tau(e_1) \oplus \tau(e_2) & \tau_{\circ_1}(e_1 \oplus e_2;\varepsilon) = (\tau(e_1) \oplus \tau(e_2)) \ltimes \varepsilon & \tau_{\circ_2}(e_1 \oplus e_2;\varepsilon) = \varepsilon \rtimes (\tau(e_1) \oplus \tau(e_2)) \\
\tau([e]^*) = [\tau(e)]^* & \tau_{\pi_i}([e]^*) = \text{id} & \tau_{\circ_i}([e]^*;\varepsilon) = \text{fp}_{\mathfrak{N},i}[\tau_{\circ_i}(e;\mathfrak{N}) \text{ union } \varepsilon] &
\end{array}$$

$$\circ_{\text{path}}(e_1;e_2) = \begin{cases} \tau(e_1) \circ \tau(e_2) & \text{if } e_1 \text{ and } e_2 \text{ are not node expressions;} \\ \tau(e_1) \ltimes \tau_{\pi_1}(e_2) & \text{if } e_2 \text{ is a node expression;} \\ \tau_{\pi_2}(e_1) \rtimes \tau(e_2) & \text{if } e_1 \text{ is a node expression.} \end{cases} \qquad \circ_{\pi_i}(e_1;e_2) = \begin{cases} \tau_{\circ_1}(e_1;\tau_{\pi_1}(e_2)) & \text{if } i = 1; \\ \tau_{\circ_2}(e_2;\tau_{\pi_2}(e_1)) & \text{if } i = 2. \end{cases}$$

Figure 3: Rewrite rules aimed at rewriting compositions to semi-joins and Kleene-star operators to fixpoint operators. In these rules, b is a basic subexpression, ε is an already rewritten expression, $f \in \{\pi, \overline{\pi}\}$, $i \in \{1,2\}$, $j \in \{1,2\}$, $\oplus \in \{\cap, -\}$, and $\mathfrak{N}$ is a fresh variable.

(i) Let $u = \text{u}(\tau(e))$. We have $\tau(e) \equiv_{\text{path}} e$, eval-steps$(\tau(e)) \leq u + \|e\|$, *and* $\|\tau(e)\| = \Theta(\|e\| \cdot 2^u)$ *in the worst case.*

(ii) Let $i \in \{1,2\}$ and $u = \text{u}(\tau_{\pi_i}(e))$. We have $\tau_{\pi_i}(e) \equiv_{\pi_i} e$, eval-steps$(\tau_{\pi_i}(e)) \leq u + \|e\|$, *and* $\|\tau_{\pi_i}(e)\| = \Theta(\|e\| \cdot 2^u)$ *in the worst case.*

The rewrite rules of Figure 3 are sound, as stated in Theorem 4.4, but not complete, as illustrated by the following example:

Example 4.5. Consider the query

$$\mathsf{Q}_8 = (\mathit{FriendOf} \cap (\mathit{FriendOf} \circ \mathit{FriendOf})) - (\text{id} \cup \text{di}).$$

Due to the presence of intersection, The rewritings $\tau(\mathsf{Q}_8)$, $\tau_{\pi_1}(\mathsf{Q}_8)$, and $\tau_{\pi_2}(\mathsf{Q}_8)$ do not result in an expression in (a fragment of) $\mathcal{N}_2$ or $\mathcal{N}_2^{\text{fp}}$. Since Q_8 always evaluates to $\emptyset$, however, we have $\mathsf{Q}_8 \equiv_{\text{path}} \emptyset$, and $\emptyset$ is, by definition, in $\mathcal{N}_2$ and $\mathcal{N}_2^{\text{fp}}$.

5 REWRITING AND QUERY OPTIMIZATION

The rules of Figure 3 are purposely kept simple. These rewrite rules will not necessarily improve query evaluation performance at every evaluation step, as illustrated by the following example:

Example 5.1. Consider the query

$$\mathsf{Q}_9 = \pi_1[\ell_1 \circ \ell_2].$$

We have $\tau(\mathsf{Q}_9) = \pi_1[\ell_1 \ltimes \ell_2]$. Now consider the graph $\mathcal{G} = (\mathrm{N}, \Sigma, \mathrm{E})$ with $\mathrm{N} = \{m, n_1, \dots, n_{|\mathrm{N}|-1}\}$, $\Sigma = \{\ell_1, \ell_2\}$, $\mathrm{E}(\ell_1) = \{(m, n_i) \mid 1 \leq i \leq |\mathrm{N}| - 1\}$, and $\mathrm{E}(\ell_2) = \{(n_i, m) \mid 1 \leq i \leq |\mathrm{N}| - 1\}$. We shall argue that straightforward evaluation of Q_9—first evaluating the composition and then evaluating the projection—is less costly than straightforward evaluation of $\tau(\mathsf{Q}_9)$. Observe that we have

$$[\![\ell_1 \circ \ell_2]\!]_{\mathcal{G}} = \{(m, m)\};$$
$$[\![\ell_1 \ltimes \ell_2]\!]_{\mathcal{G}} = \mathrm{E}(\ell_1).$$

Due to the intermediate result of evaluating $\ell_1 \circ \ell_2$ being much smaller than the intermediate result of evaluating $\ell_1 \ltimes \ell_2$, straightforward projection algorithms will perform the projection step in Q_9 at a much lower cost than the projection step in $\tau(\mathsf{Q}_9)$. Moreover, in this specific example, algorithms for computing the composition can easily achieve comparable performance to algorithms for computing the semi-join.

In the following, we explore how the rewrite rules of Figure 3 can be used for graph query optimization. In Section 5.1, we look at how to deal with the situation outlined in Example 5.1. In Section 5.2, we look at the complexity of evaluating fixpoints. Finally, in Section 5.3, we look at the cost of evaluating expressions and their optimized counterparts.

5.1 Query rewriting and optimization

The complexity of evaluating the relation algebra and semi-join algebra operators can be derived from well-known complexities for evaluating relational algebra [9, 18, 28]. In practice, the cost of each operation o depends on the size of the relations obtained by evaluating the operands of o. From this observation, the issue shown in Example 5.1 can easily be explained: the rewrite rules of Figure 3 can rewrite expression e into an expression e' such that $|[\![e]\!]_{\mathcal{G}}| < |[\![e']\!]_{\mathcal{G}}|$.

We change the rewrite rules of Figure 3 such that the resulting rewrite rules provide strong-guarantees with respect to the size of the evaluation of a rewritten expression. We do so by modifying $\tau_{\circ_i}(e;\varepsilon)$ by replacing rewritings of the form $g \ltimes \varepsilon$ by $\pi_1[g \ltimes \varepsilon]$ and of the form $\varepsilon \rtimes g$ by $\pi_2[\varepsilon \rtimes g]$.

Proposition 5.2. *Let $\mathcal{G}$ be a graph, let e be an expression in $\mathcal{N}_3^*$, and let ε be an expression. If we use the rewrite rules of Figure 3 with the above modifications, then $\tau_{\circ_1}(e;\varepsilon)$ and $\tau_{\circ_2}(e;\varepsilon)$ are node expressions and their evaluation yields the smallest possible sets such that $[\![\tau_{\circ_1}(e;\varepsilon)]\!]_{\mathcal{G}}|_1 = [\![e \ltimes \varepsilon]\!]_{\mathcal{G}}|_1$ and $[\![\tau_{\circ_2}(e;\varepsilon)]\!]_{\mathcal{G}}|_2 = [\![\varepsilon \rtimes e]\!]_{\mathcal{G}}|_2$.*

With a minimal modification to the rewrite rules for $\tau_{\pi_i}(e)$, $i \in \{1,2\}$, we can also guarantee that $\tau_{\pi_i}(e)$ minimizes intermediate evaluation results in the way as the modified version of $\tau_{\circ_i}(e;\varepsilon)$ does. We do so by, additionally, applying the following two changes to $\tau_{\pi_i}(e)$:

$$\tau_{\pi_i}(b) = \pi_i[b];$$
$$\tau_{\pi_i}(e_1 \oplus e_2) = \pi_i[\tau(e_1) \oplus \tau(e_2)],$$

in which b is a basic expression and $\oplus \in \{\cap, -\}$. Using a straightforward induction argument, we obtain

Proposition 5.3. *Let $\mathcal{G}$ be a graph and let e be an expression in $\mathcal{N}_3^*$. If we use the rewrite rules of Figure 3 with the above modifications, then $\tau_{\pi_1}(e)$ and $\tau_{\pi_2}(e)$ are node expressions and their evaluation yields the smallest possible sets such that $[\![\tau_{\pi_1}(e)]\!]_{\mathcal{G}}|_1 = [\![e]\!]_{\mathcal{G}}|_1$ and $[\![\tau_{\pi_2}(e)]\!]_{\mathcal{G}}|_2 = [\![e]\!]_{\mathcal{G}}|_2$.*

From Proposition 5.3, we conclude:

Corollary 5.4. *Let $\mathcal{G}$ be a graph, let e be an expression in $\mathcal{N}_3^*$, and $i \in \{1, 2\}$. If we use the rewrite rules of Figure 3 with the above modifications, then we have $|[\![\tau_{\pi_i}(e)]\!]_{\mathcal{G}}| \leq |[\![e]\!]_{\mathcal{G}}|$.*

We observe that, in practice, the projection-steps introduced by modifying $\tau_{\circ_i}(e; \varepsilon)$ and $\tau_{\pi_i}(e)$ can easily be integrated into specialized versions of the algorithms used to evaluate $\ltimes$, $\rtimes$, $\cup$, $\cap$, and $-$, this without increasing the cost for the resulting specialized versions of the algorithms. Hence, the changes made do not increase the evaluation cost.

5.2 Efficient evaluation of fixpoints

Let $f = \mathsf{fp}_{i,\mathfrak{R}}[e \text{ union } b]$ be an expression without free variables. The complexity of evaluating f is determined by the so-called *recursion steps of f*, denoted by $\mathcal{R}(f)$, and the cost of evaluating the so-called *non-recursive terms of f*, which we denote by $\mathcal{T}(f)$. We define $\mathcal{R}(f) = \mathcal{R}(e)$ with

$$\begin{aligned}\mathcal{R}(\mathfrak{R}) &= 1 \quad \text{(with } \mathfrak{R} \text{ a variable)};\\ \mathcal{R}(e_1 \ltimes e_2) = \mathcal{R}(e_2 \rtimes e_1) &= 1 + \mathcal{R}(e_1);\\ \mathcal{R}(e_1 \cup e_2) &= \mathcal{R}(e_1) + \mathcal{R}(e_2) + 1;\\ \mathcal{R}(\mathsf{fp}_{j,\mathfrak{R}'}[e' \text{ union } b']) &= \mathcal{R}(b') + \mathcal{R}(e') + 1,\end{aligned}$$

and we define $\mathcal{T}(f)$ to be the multiset $\mathcal{T}(f) = [b] + \mathcal{T}(e)$ with

$$\begin{aligned}\mathcal{T}(\mathfrak{R}) &= [\;] \quad \text{(with } \mathfrak{R} \text{ a variable)}\\ \mathcal{T}(e_1 \ltimes e_2) = \mathcal{T}(e_2 \rtimes e_1) &= [e_2] + \mathcal{T}(e_1);\\ \mathcal{T}(e_1 \cup e_2) &= \mathcal{T}(e_1) + \mathcal{T}(e_2);\\ \mathcal{T}(\mathsf{fp}_{j,\mathfrak{R}'}[e' \text{ union } b']) &= \mathcal{T}(b') + \mathcal{T}(e').\end{aligned}$$

Proposition 5.5. *Let $\mathcal{G} = (\mathbf{N}, \Sigma, \mathbf{E})$ be a graph and let $f = \mathsf{fp}_{i,\mathfrak{R}}[e \text{ union } b]$ be an expression without free variables. The worst-case cost for evaluating $[\![f]\!]_{\mathcal{G}}$ is $O(|\mathcal{R}(f)| \cdot n + s + c)$, in which*

$$n = \max\{|[\![t]\!]_{\mathcal{G}}|_i| \mid t \in \mathcal{T}(f)\};$$

$$s = \sum\{|[\![t]\!]_{\mathcal{G}}| \mid t \in \mathcal{T}(f)\},$$

and c is the total cost of evaluating the expressions in $\mathcal{T}(f)$.

Proof (sketch). We observe that, in expression e, there is no negation on the path towards the variable $\mathfrak{R}$: we only allow union, semi-joins, and fixpoints ($\cup$, $\ltimes$, $\rtimes$, and fp), and we do not allow difference and coprojections ($-$ and $\overline{\pi}$). Hence, if we interpret the expressions in $\mathcal{T}(f)$ as pre-computed edge labels, then the restricted language we consider is expressible in a subset of the alternation-free μ-calculus, for which very efficient evaluation algorithms exist [6].

We sketch how to evaluate the fixpoint expression f when $i = 1$. The case for $i = 2$ is analogous. To evaluate the fixpoint expression f, we first translate the expression into a graph representation. We do so by making edge-connections between expressions in the following way:

(1) Add an unlabeled connection from the expression e to the expression $\mathfrak{R}$.

(2) For any right-recursive subexpression $e_1 \ltimes e_2$, add a connection labeled e_1 from e_2 to $e_1 \ltimes e_2$.

(3) For any right-recursive subexpression $e_1 \cup e_2$, add unlabeled connections from e_1 and e_2 to $e_1 \cup e_2$.

(4) For any right-recursive subexpression $\mathsf{fp}_{1,\mathfrak{R}'}[e' \text{ union } b']$, add an unlabeled connection from $\mathfrak{R}'$ to $\mathsf{fp}_{1,\mathfrak{R}'}[e' \text{ union } b']$ and from b' to $\mathfrak{R}'$.

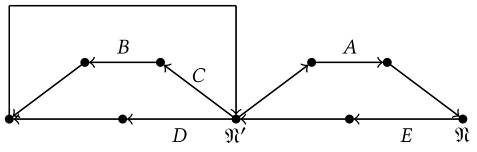

Figure 4: Graph representation of $\mathsf{fp}_{1,\mathfrak{R}}[A \ltimes e' \text{ union } F]$ **with** $e' = \mathsf{fp}_{1,\mathfrak{R}'}[B \ltimes (C \ltimes \mathfrak{R}') \cup D \ltimes \mathfrak{R}') \text{ union } E \ltimes \mathfrak{R}]$.

Figure 4 provides an example of the resulting graph representation of a fixpoint expression.

The graph representation is used for a message-passing evaluation algorithm: each expression-node maintains a set of graph-nodes. When an expression-node receives a graph-node v it has not yet received, then (a) it sends v to every expression-node to which it has an unlabeled connection and (b) it sends w to every expression-node to which it has a connection labeled e' with $(w, v) \in [\![e']\!]_{\mathcal{G}}$. We initialize this process by sending each graph-node in $[\![b]\!]_{\mathcal{G}}|_1$ to the expression-node $\mathfrak{R}$. Let S be the set of all graph-nodes received by $\mathfrak{R}$ after all messages have been processed. We have $[\![f]\!]_{\mathcal{G}} = \{(v, v) \mid v \in S\}$.

Over each unlabeled connection, at most n messages are sent and over each connection labeled with e, at most $|[\![e]\!]_{\mathcal{G}}|$ messages are sent. The number of expression-nodes and the number of unlabeled connections are both worst-case $O(\mathcal{R}(f))$ and for each non-recursive term in $\mathcal{T}(f)$ there is exactly one labeled connection. Hence, at most $O(\mathcal{R}(f) \cdot n + s)$ messages need to be sent. □

5.3 Complexity of rewritten expressions

We consider the complexity of query evaluation in the usual framework [29]. If e is an expression, then the worst-case complexity of evaluating e is $O(\text{eval-steps}(e) \cdot c)$, where c is the maximum cost for performing a single operation. The *query complexity*—the cost of evaluating an expression in terms of the size of the expression given a fixed graph—is $O(\text{eval-steps}(e))$. The *data complexity*—the cost of evaluating a query in terms of the size of the graph given a fixed query—is $O(c)$.

For most operators, the maximum cost for performing a single operation can easily be derived or is well-known from the literature [9, 18, 28], the only exception being the fixpoint operator fp, which we discussed in Section 5.2. Of these operators, only the constant relation di, the binary operator $\circ$, and the unary operator * can cause large query results. This is illustrated by the following, very easy, example:

Example 5.6. Consider the graph $\mathcal{G} = (\mathbf{N}, \Sigma, \mathbf{E})$ with $\mathbf{N} = \{m, n_1, \ldots, n_{|\mathbf{N}|-1}\}$, $\Sigma = \{\ell\}$, and $\mathbf{E}(\ell) = \{(m, n_i), (n_i, m) \mid 1 \leq i \leq$

$|\mathbf{N}| - 1\}$. We have

$$[\![\mathrm{di}]\!]_{\mathcal{G}} = \mathbf{N}^2 - \{(v, v) \mid v \in \mathbf{N}\};$$
$$[\![\ell \circ \ell]\!]_{\mathcal{G}} = \mathbf{N}^2 - \mathbf{E}(\ell);$$
$$[\![[\ell]^*]\!]_{\mathcal{G}} = \mathbf{N}^2.$$

Observe that $|[\![\mathrm{di}]\!]_{\mathcal{G}}| = |\mathbf{N}|^2 - |\mathbf{N}|$ and $|[\![\ell \circ \ell]\!]_{\mathcal{G}}| = |\mathbf{N}|^2 - 2\cdot(|\mathbf{N}|-1)$.

The maximum evaluation size and cost of evaluating the operators not mentioned in Example 5.6 is worst-case upper-bounded by either the size of their operands only or, additionally, by the number of nodes in the graph. Hence, Example 5.6 clearly illustrates why we consider composition and the Kleene-star to be expensive.

We can verify that the rewrite rules $\tau(e)$, $\tau_{\pi_1}(e)$, and $\tau_{\pi_2}(e)$ reduce the data-complexity in two distinct ways. First, the number of "expensive" operators (composition and Kleene-star) is reduced in favor of cheaper operators. Second, by Corollary 5.4, the size of evaluation results for subqueries is minimized whenever possible, reducing the cost of evaluating non-rewritten operators.

The cost of the reduction in the data-complexity of evaluating a query optimized by $\tau(e)$, $\tau_{\pi_1}(e)$, or $\tau_{\pi_2}(e)$ is an increase in the query-complexity of the optimized query. This increase in the query-complexity is caused by an increase in the evaluation size. This increase is, in the worst case, an exponential increase, as illustrated in the following example:

Example 5.7. Consider the queries Q_7 of Example 4.3 and

$$\mathsf{Q}_{10} = \tau_{\pi_1}(\mathsf{Q}_7) = \ell \ltimes (\ell \ltimes (\ell \ltimes (\ell \ltimes (\ell \ltimes (\ell \ltimes (\ell \ltimes \ell)))))).$$

By Theorem 4.4, we have $\mathsf{Q}_7 \equiv_{\pi_1} \mathsf{Q}_{10}$. During rewriting, the expression size did not increase, while the evaluation size did sharply increase: we have $\|\mathsf{Q}_7\| = \|\mathsf{Q}_{10}\| = 7$, eval-steps($\mathsf{Q}_7$) = 3, and eval-steps($\mathsf{Q}_{10}$) = 7. As a consequence, we can evaluate Q_{10} in worst-case $O(7 \cdot |\mathbf{E}(\ell)|)$ and Q_7 in worst-case $O(3 \cdot |\mathbf{E}(\ell)|^2)$. Hence, the increase in the query-complexity is accompanied by a sharp decrease in the data-complexity.

Even with a worst-case exponential increase in the query-complexity, Theorem 4.4 guarantees that the query complexity is linearly upper-bounded by the size of the original query. Hence, in the usual case where queries are small and graphs are very large, the increase of the query complexity is not an issue.

6 EXPRESSIVE POWER OF $\mathcal{N}_3^*$ AND $\mathcal{N}_2^{\mathrm{fp}}$

The rewrite rules of Figure 3 do not fully rewrite every expression in $\mathcal{N}_3^*$ to $\mathcal{N}_2^{\mathrm{fp}}$. To better understand the limits of these rewrite rules, we take a look at how they rewrite fragments of $\mathcal{N}_3$ and $\mathcal{N}_3^*$.

We write $\mathscr{F} \subseteq \{{}^\frown, \mathrm{di}, \overline{\pi}, \cap, -\}$ to denote a set of operators in which $\overline{\pi}$ represents both $\overline{\pi}_1$ and $\overline{\pi}_2$. By $\mathcal{N}_3(\mathscr{F})$ we denote the fragment of $\mathcal{N}_3$ that only allows $\emptyset$, $\ell \in \Sigma$, id, π_1, π_2, $\circ$, $\cup$, and all operators in $\mathscr{F}$ and by $\mathcal{N}_2(\mathscr{F})$ we denote the fragment of $\mathcal{N}_2$ that only allows $\emptyset$, $\ell \in \Sigma$, id, π_1, π_2, $\ltimes$, $\rtimes$, $\cup$, and all operators in $\mathscr{F}$. By $\mathcal{N}_3^*(\mathscr{F})$ and $\mathcal{N}_2^{\mathrm{fp}}(\mathscr{F})$ we denote the languages based on $\mathcal{N}_3(\mathscr{F})$ and $\mathcal{N}_2(\mathscr{F})$ to which the Kleene-star and the fixpoints, respectively, are added. We have the following:

Theorem 6.1. *If $\mathscr{F} \subseteq \{{}^\frown, \mathrm{di}, \overline{\pi}\}$, then*
(i) $\mathcal{N}_2(\mathscr{F}) \preceq_{\mathrm{path}} \mathcal{N}_3(\mathscr{F})$ and $\mathcal{N}_2(\mathscr{F}) \equiv_\pi \mathcal{N}_3(\mathscr{F})$; and
(ii) $\mathcal{N}_2^{\mathrm{fp}}(\mathscr{F}) \preceq_{\mathrm{path}} \mathcal{N}_3^*(\mathscr{F})$ and $\mathcal{N}_2^{\mathrm{fp}}(\mathscr{F}) \equiv_\pi \mathcal{N}_3^*(\mathscr{F})$.

Proof (sketch). The rewrite rules of Figure 3 satisfy two basic properties. First, no rewrite rule introduces operators not yet in the original expressions, except for semi-joins (introduced when rewriting compositions) and fixpoints (introduced when rewriting Kleene-stars). Second, compositions are only kept in path-equivalent rewritings. During left-projection-equivalent rewriting using $\tau_{\pi_1}(\cdot)$ and right-projection-equivalent rewriting using $\tau_{\pi_2}(\cdot)$, path-equivalent rewritings are only enforced by the usage of intersection and difference outside of basic expressions. □

6.1 Intersection and difference

Observe that Theorem 6.1 does not include intersection and difference. Within basic expressions, the expressive power of the intersection and difference only plays a minor role. As a consequence, we can extend Theorem 6.1 slightly. Let $\mathscr{F} \subseteq \{{}^\frown, \mathrm{di}, \overline{\pi}, \cap, -\}$. We write $\mathcal{B}_3(\mathscr{F})$, $\mathcal{B}_3^*(\mathscr{F})$, $\mathcal{B}_2(\mathscr{F})$, and $\mathcal{B}_2^{\mathrm{fp}}(\mathscr{F})$ to denote the fragments of $\mathcal{N}_3(\mathscr{F})$, $\mathcal{N}_3^*(\mathscr{F})$, $\mathcal{N}_2(\mathscr{F})$, and $\mathcal{N}_2^{\mathrm{fp}}(\mathscr{F})$, respectively, in which intersection and difference ($\cap$ and $-$) occur in basic expressions only.

Theorem 6.2. *If $\mathscr{F} \subseteq \{{}^\frown, \mathrm{di}, \overline{\pi}, \cap, -\}$, then*
(i) $\mathcal{B}_2(\mathscr{F}) \preceq_{\mathrm{path}} \mathcal{B}_3(\mathscr{F})$ and $\mathcal{B}_2(\mathscr{F}) \equiv_\pi \mathcal{B}_3(\mathscr{F})$;
(ii) $\mathcal{B}_2^{\mathrm{fp}}(\mathscr{F}) \preceq_{\mathrm{path}} \mathcal{B}_3^*(\mathscr{F})$ and $\mathcal{B}_2^{\mathrm{fp}}(\mathscr{F}) \equiv_\pi \mathcal{B}_3^*(\mathscr{F})$.

With regard to the semi-join algebra, we may require that intersection and difference occur in basic expressions only, as shown next.

Proposition 6.3. *If $\mathscr{F} \subseteq \{{}^\frown, \mathrm{di}, \overline{\pi}, \cap, -\}$, then $\mathcal{B}_2(\mathscr{F}) \equiv_{\mathrm{path}} \mathcal{N}_2(\mathscr{F})$ and $\mathcal{B}_2^{\mathrm{fp}}(\mathscr{F}) \equiv_{\mathrm{path}} \mathcal{N}_2^{\mathrm{fp}}(\mathscr{F})$.*

Proof (sketch). Push down intersection ($\cap$) and difference ($-$) through projections, coprojections, semi-joins, and unions using straightforward rewrite rules. We observe that we can treat fixpoints as if they are projections. By repeatedly pushing down intersection and difference until this is no longer possible, all intersections and differences occur in basic expressions only. □

The above collapse of the semi-join algebras to the basic expressions does not extend to the relation algebras:

Example 6.4. Consider the following query that is based on the part of Q_6 in Example 4.1 that could not be rewritten without using composition:

$$\mathsf{Q}_{11} = (\mathit{FriendOf} \circ \mathit{FriendOf}) \cap \mathit{FriendOf}.$$

The query Q_{11} has an occurrence of intersection beyond the scope of basic expressions. We claim that no expression in $\mathcal{B}_3^*$ or $\mathcal{B}_3$, or, equivalently, in $\mathcal{B}_2$ or $\mathcal{B}_2^{\mathrm{fp}}$, is path-equivalent, left-projection-equivalent, or right-projection-equivalent to Q_{11}.

To show this, consider the graphs $\mathcal{G}_{3,3}$ and $\mathcal{G}_4$ of Figure 5 and observe that $[\![\mathsf{Q}_{11}]\!]_{\mathcal{G}_{3,3}} \neq \emptyset$ and $[\![\mathsf{Q}_{11}]\!]_{\mathcal{G}_4} = \emptyset$. To show that no expression in $\mathcal{B}_2$ or $\mathcal{B}_2^{\mathrm{fp}}$ can distinguish between $\mathcal{G}_{3,3}$ and $\mathcal{G}_4$, we can use standard two-pebble game results for the FO[2]-variant of the infinitary finite variable logics [11, Example 3.10].

6.2 Conditional XPath on sibling-ordered trees

The above expressivity results have interesting implications for Regular XPath, Conditional XPath, and first-order logic evaluated

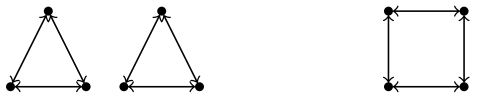

Figure 5: On the *left*, a two-3-cycle graph $\mathcal{G}_{3,3}$. On the *right*, a single-4-cycle graph $\mathcal{G}_4$.

on node-labeled sibling-ordered trees, which we detail next. Regular XPath is a query language for node-labeled sibling-ordered XML data [20]. Regular XPath distinguishes path formulae, which evaluate to binary relations, and node formulae, which evaluate to unary relations (sets of nodes). Path formulae are defined by the grammar[1]

$$\text{p\_wff} = \mathit{Edge} \mid \text{p\_wff} \circ \text{p\_wff} \mid \text{p\_wff} \cup \text{p\_wff} \mid [\text{p\_wff}]^* \mid ?\text{n\_wff},$$

in which $\mathit{Edge} \in \{\mathit{Child}, \mathit{Parent}, \mathit{Left}, \mathit{Right}\}$ denotes the edge relations (parent-child axis and the ordered sibling axis), n_wff is a node formula, and ?n_wff interprets the node formulae as a binary relation. Node formulae are defined by the grammar

$$\text{n\_wff} = \ell \mid \text{id} \mid \pi_1[\text{p\_wff}] \mid \overline{\text{n\_wff}} \mid \text{n\_wff} \cup \text{n\_wff} \mid \text{n\_wff} \cap \text{n\_wff},$$

in which ℓ denotes a node label.

To translate between Regular XPath and $\mathcal{N}_3(^\frown, \overline{\pi})$, we translate path formulae to general relation algebra expressions and node formulae to relation algebra expressions that are node expressions. We represent node labels by using restricted edge labels. These choices result in a straightforward rewriting $\tau_{\text{p\_wff}}(\text{p\_wff})$ for path formulae p_wff, and for node formulae we have:

$$\begin{aligned}
\tau_{\text{n\_wff}}(\ell) &= \ell;\\
\tau_{\text{n\_wff}}(\text{id}) &= \text{id};\\
\tau_{\text{n\_wff}}(\pi_1[\text{p\_wff}]) &= \pi_1[\tau_{\text{p\_wff}}(\text{p\_wff})];\\
\tau_{\text{n\_wff}}(\overline{\text{n\_wff}}) &= \overline{\pi}_1[\tau_{\text{n\_wff}}(\text{n\_wff})];\\
\tau_{\text{n\_wff}}(\text{n\_wff}_1 \cup \text{n\_wff}_2) &= \tau_{\text{n\_wff}}(\text{n\_wff}_1) \cup \tau_{\text{n\_wff}}(\text{n\_wff}_2);\\
\tau_{\text{n\_wff}}(\text{n\_wff}_1 \cap \text{n\_wff}_2) &= \tau_{\text{n\_wff}}(\text{n\_wff}_1) \circ \tau_{\text{n\_wff}}(\text{n\_wff}_2).
\end{aligned}$$

Conditional XPath is a restriction of Regular XPath in which the Kleene-star can only be applied to so-called steps instead of generic expressions. (A step is an edge relation to which, optionally, a test is applied of the form $\langle \text{p\_wff} \rangle$.) We have the following:

Proposition 6.5. *With respect to queries yielding binary relations evaluated on finite node-labeled sibling-ordered trees, we have Regular XPath* $\equiv_{\text{path}} \mathcal{N}_3^*(^\frown, \overline{\pi})$*, Conditional XPath* $\preceq_{\text{path}} \mathcal{N}_3^*(^\frown, \overline{\pi})$*, and* $\mathcal{N}_3^*(^\frown, \overline{\pi}) \not\preceq_\pi$ *Conditional XPath.*

Proof (sketch). To translate from Regular XPath to $\mathcal{N}_3(^\frown, \overline{\pi})$, we use the above translation $\tau_{\text{p\_wff}}(\text{p\_wff})$. For the other direction, we adapt the above translation. The only difficulty in this, are subexpressions of the form $\pi_2[e]$ and $\overline{\pi}_2[e]$. We deal with these subexpressions by rewriting them towards $\pi_1[e']$ and $\overline{\pi}_1[e']$, respectively, in which e' is obtained from e by pushing down a converse step towards the edge labels *Child* and *Right*. The other relations follow from the well-known relations between Regular XPath and Conditional XPath [20]. □

[1]We have adapted the Regular XPath syntax to match our $\mathcal{N}_3^*$ syntax.

Combining Theorem 6.1 and Proposition 6.5 with a result from Marx [20] yields

Corollary 6.6. *With respect to queries yielding binary relations evaluated on finite node-labeled sibling-ordered trees, we have Regular XPath* $\equiv_\pi \mathcal{N}_2^{\text{fp}}(^\frown, \overline{\pi})$*, Conditional XPath* $\preceq_\pi \mathcal{N}_2^{\text{fp}}(^\frown, \overline{\pi})$*, and* $\mathcal{N}_2^{\text{fp}}(^\frown, \overline{\pi}) \not\preceq_\pi$ *Conditional XPath.*

On finite node-labeled sibling-ordered trees, Conditional XPath is path-equivalent to FO^{tree}: first-order logic on tree structures represented by a descendant and a following-sibling relation, unary node-label predicates, and equality [20, Proposition 2.7]. Hence, we conclude the following:

Proposition 6.7. *With respect to queries yielding binary relations evaluated on finite node-labeled sibling-ordered trees we have* $\text{FO}^{\text{tree}} \preceq_\pi \mathcal{N}_2^{\text{fp}}(^\frown, \overline{\pi})$ *and* $\text{FO}^{\text{tree}} \preceq_\pi \mathcal{N}_2^{\text{fp}}(^\frown, \overline{\pi})$.

It is not possible to translate Conditional XPath to the two-variable fragment of FO^{tree} via $\mathcal{N}_2^{\text{fp}}(^\frown, \overline{\pi})$. In the two-variable fragment of FO^{tree}, there is no obvious way to express the relations *Child* and *Right* or express step-based iteration via the descendant and the following-sibling relations.

7 CONCLUSION AND FUTURE WORK

The main theme of this paper is the relationship between the relation algebra and the semi-join algebra. We studied these relationships with query optimization in mind: we aimed at rewriting relation algebra expressions containing costly composition and Kleene-star operators into semi-join algebra expressions containing less costly semi-join and fixpoint operators. To do so, we identified sufficient conditions on relation algebra expressions that allow us to perform such rewritings and we have shown that these conditions are not too restrictive.

To make the theory applicable, we presented rewrite rules which can be used to rewrite (parts of) relation algebra expressions that satisfy the conditions identified. In addition, we have provided a complexity analysis that shows that our rewrite rules lead to only a well-bounded increase in the number of steps needed to evaluate the rewritten queries (while, at the same time, strictly reducing the number of costly composition and Kleene-star operations).

Since the relation algebra and the semi-join algebra correspond to FO[3] and FO[2], respectively, our rewrite rules also provided new insights into the relationship between these first-order logics. In addition, by specializing our results to node-labeled sibling-ordered trees, we were able to obtain new insights in the relationship between the expressive power of full first-order logic and FO[2] on such trees.

We have identified several directions for future work, including the following:

(1) Even though our rewrite rules do not completely solve the efficient query evaluation problem for the relation algebra, we have been able to verify that our rewrite rules capture optimizations that have not yet been fully exploited by existing database systems. Hence, a practical empirical study of a query evaluation system that combines our rewrite rules with other query optimization and evaluation techniques is an obvious avenue of further research.

(2) In this paper we have introduced rewriting techniques for the semi-join algebra specialized to binary relations. The semi-join

algebra has also been studied extensively for traditional relational databases (see, e.g. [13, 15–17]) and several of its expressiveness properties and query evaluation benefits have been identified and used in practice. We plan to investigate how our techniques can be generalized and be of benefit for relational database query optimization and evaluation.

(3) The intersection and difference operators limit the applicability of our rewriting techniques. For several restricted graph structures, well-known collapse results exist to eliminate intersection and difference (see e.g. [2, 12, 30]). Unfortunately, these known elimination results blow up the size of the resulting query excessively. Still, it is worthwhile to investigate if approaches that are more practical exist to eliminate intersection and differences in the scope of semi-join based query optimization.

(4) As argued in the Introduction, the strength of the graph query language $\mathcal{N}_3^*$ is navigation. Beyond navigation, the expressive power of $\mathcal{N}_3^*$ is rather limited, even with respect to basic counting. The language is, for example, incapable to express that nodes have a minimum number of incoming or outgoing edges. For FO[2] and FO[3] these limitations are usually lifted by adding so-called counting quantifiers [11]. Such counting quantifiers can also be added to $\mathcal{N}_3^*$ and $\mathcal{N}_2^{\mathrm{fp}}$. It is open to determine to which extend our rewrite rules and query optimization results are generalizable towards these languages.

REFERENCES

[1] Pablo Barceló. 2013. Querying Graph Databases. In *Proceedings of the 32nd Symposium on Principles of Database Systems (PODS '13)*. ACM, 175–188.

[2] Michael Benedikt, Wenfei Fan, and Gabriel Kuper. 2005. Structural properties of XPath fragments. *Theoretical Computer Science* 336, 1 (2005), 3–31.

[3] Michael Benedikt and Christoph Koch. 2009. XPath Leashed. *ACM Computing Surveys (CSUR)* 41, 1 (2009), 3:1–3:54.

[4] Philip A. Bernstein and Dah-Ming W. Chiu. 1981. Using Semi-Joins to Solve Relational Queries. *J. ACM* 28, 1 (1981), 25–40.

[5] James Clark and Steve DeRose. 1999. *XML Path Language (XPath) Version 1.0*. W3C Recommendation. W3C. http://www.w3.org/TR/1999/REC-xpath-19991116.

[6] Rance Cleaveland and Bernhard Steffen. 1993. A linear-time model-checking algorithm for the alternation-free modal mu-calculus. *Formal Methods in System Design* 2, 2 (1993), 121–147.

[7] George H. L. Fletcher, Marc Gyssens, Dirk Leinders, Dimitri Surinx, Jan Van den Bussche, Dirk Van Gucht, Stijn Vansummeren, and Yuqing Wu. 2015. Relative expressive power of navigational querying on graphs. *Information Sciences* 298 (2015), 390–406.

[8] George H. L. Fletcher, Marc Gyssens, Dirk Leinders, Jan Van den Bussche, Dirk Van Gucht, Stijn Vansummeren, and Yuqing Wu. 2015. The impact of transitive closure on the expressiveness of navigational query languages on unlabeled graphs. *Annals of Mathematics and Artificial Intelligence* 73, 1-2 (2015), 167–203.

[9] Hector Garcia-Molina, Jeffrey D. Ullman, and Jennifer Widom. 2008. *Database Systems: The Complete Book* (2 ed.). Prentice Hall Press, Upper Saddle River, NJ, USA.

[10] Steven Givant. 2006. The Calculus of Relations as a Foundation for Mathematics. *Journal of Automated Reasoning* 37, 4 (2006), 277–322.

[11] Martin Grohe. 1998. Finite Variable Logics in Descriptive Complexity Theory. *The Bulletin of Symbolic Logic* 4 (1998), 345–398.

[12] Jelle Hellings, Marc Gyssens, Yuqing Wu, Dirk Van Gucht, Jan Van den Bussche, Stijn Vansummeren, and George H. L. Fletcher. 2015. Relative Expressive Power of Downward Fragments of Navigational Query Languages on Trees and Chains. In *Proceedings of the 15th Symposium on Database Programming Languages (DBPL 2015)*. 59–68.

[13] Aviel Klausner and Nathan Goodman. 1985. Multirelations: Semantice and Languages. In *Proceedings of the 11th International Conference on Very Large Data Bases (VLDB '85)*. VLDB Endowment, 251–258.

[14] Dexter Kozen. 1997. Kleene Algebra with Tests. *ACM Transactions on Programming Languages and Systems* 19, 3 (1997), 427–443.

[15] Dirk Leinders. 2008. *The semijoin algebra*. Ph.D. Dissertation. Hasselt University and transnational University of Limburg.

[16] Dirk Leinders, Maarten Marx, Jerzy Tyszkiewicz, and Jan Van den Bussche. 2005. The Semijoin Algebra and the Guarded Fragment. *Journal of Logic, Language and Information* 14, 3 (2005), 331–343.

[17] Dirk Leinders, Jerzy Tyszkiewicz, and Jan Van den Bussche. 2004. On the expressive power of semijoin queries. *Inform. Process. Lett.* 91, 2 (2004), 93–98.

[18] Dirk Leinders and Jan Van den Bussche. 2007. On the complexity of division and set joins in the relational algebra. *J. Comput. System Sci.* 73, 4 (2007), 538–549. Special Issue: Database Theory 2005.

[19] Leonid Libkin, Wim Martens, and Domagoj Vrgoč. 2013. Querying Graph Databases with XPath. In *Proceedings of the 16th International Conference on Database Theory (ICDT '13)*. ACM, New York, NY, USA, 129–140.

[20] Maarten Marx. 2005. Conditional XPath. *ACM Transactions on Database Systems* 30, 4 (2005), 929–959.

[21] Maarten Marx and Maarten de Rijke. 2005. Semantic Characterizations of Navigational XPath. *SIGMOD Record* 34, 2 (2005), 41–46.

[22] Maarten Marx and Yde Venema. 1997. *Multi-Dimensional Modal Logic*. Springer Netherlands, Dordrecht.

[23] Praveen Seshadri, Joseph M. Hellerstein, Hamid Pirahesh, T. Y. Cliff Leung, Raghu Ramakrishnan, Divesh Srivastava, Peter J. Stuckey, and S. Sudarshan. 1996. Cost-based Optimization for Magic: Algebra and Implementation. *SIGMOD Rec.* 25, 2 (1996), 435–446.

[24] Dimitri Surinx, George H. L. Fletcher, Marc Gyssens, Dirk Leinders, Jan Van den Bussche, Dirk Van Gucht, Stijn Vansummeren, and Yuqing Wu. 2015. Relative expressive power of navigational querying on graphs using transitive closure. *Logic Journal of the IGPL* 23, 5 (2015), 759–788.

[25] Alfred Tarski. 1941. On the Calculus of Relations. *The Journal of Symbolic Logic* 6, 3 (1941), 73–89.

[26] Balder ten Cate. 2006. The Expressivity of XPath with Transitive Closure. In *Proceedings of the Twenty-fifth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS '06)*. ACM, 328–337.

[27] Balder ten Cate and Maarten Marx. 2007. Navigational XPath: Calculus and Algebra. *SIGMOD Record* 36, 2 (2007), 19–26.

[28] Jeffrey D. Ullman. 1990. *Principles of Database and Knowledge-Base Systems: Volume II: The New Technologies*. W. H. Freeman & Co., New York, NY, USA.

[29] Moshe Y. Vardi. 1982. The Complexity of Relational Query Languages (Extended Abstract). In *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing (STOC '82)*. ACM, New York, NY, USA, 137–146.

[30] Yuqing Wu, Dirk Van Gucht, Marc Gyssens, and Jan Paredaens. 2011. A Study of a Positive Fragment of Path Queries: Expressiveness, Normal Form and Minimization. *Comput. J.* 54, 7 (2011), 1091–1118.

[31] Mihalis Yannakakis. 1981. Algorithms for Acyclic Database Schemes. In *Proceedings of the Seventh International Conference on Very Large Data Bases - Volume 7 (VLDB '81)*. VLDB Endowment, 82–94.