Fine Grained Information Integration with Description Logics

Huizhu Liu, David Toman, and Grant Weddell

Department of Computer Science, University of Waterloo
Email: {h4liu,david,gweddell}@uwaterloo.ca

Abstract

We outline an approach to query optimization in which a *description logic* (DL) reasoner serves a crucial strategic role, and present an example application of our approach in fine grained information integration. In particular, the application demonstrates how the internal structure of an unfolded B-tree can be captured as a terminology, and how an access plan that navigates this internal structure can be found with the aid of a DL reasoner.

1 Introduction

An *embedded control program* (ECP) is a new application area for database technology [14]. An ECP that is a *legacy* system presents an additional challenge. In particular, the ECP will already have code (sometimes a great deal of code) that specifies the internal encoding for a database of *control data*. To enable high-level access to this data using SQL-like languages, it becomes necessary to optimize queries over conceptual views. A query optimizer must therefore be capable of *fine grained information integration*; that is, it must be possible to supply descriptions of the internal encoding of control data as part of the input to the optimizer.

We have developed a novel resource bounded query optimizer in which integrity constraints that abstract the internal encoding of control data are used to extend the search space of possible query plans for a given source query [14]. This paper links the optimizer with a powerful DL reasoner that enables complex query rewrites. In particular, we show how the data structures that constitute an internal control data encoding can be captured by a terminology in $\mathcal{DLFDE}$, a refinement of an earlier DL dialect called CFD [10]. We then show how this optimizer uses a DL reasoner for $\mathcal{DLFDE}$ to help find access plans that navigate the data structures for queries over conceptual views. The process relies on dynamic construction of descriptions that characterize properties of subqueries generated during query optimization. A complicating factor, due to the nature of the application itself, is the presumption of a *bag semantics* for an underlying query algebra.

An unfolded B-tree data structure will be used as a running example. A schema for the B-tree is given in Figure 1; squares and circles represent primitive classes in this schema, while unlabeled wide arcs denote sub-classing and labeled narrow arcs denote attributes. The Pi classes on the right correspond to data pages of the B-tree at level i, while the Ej classes in the middle correspond to employee records within

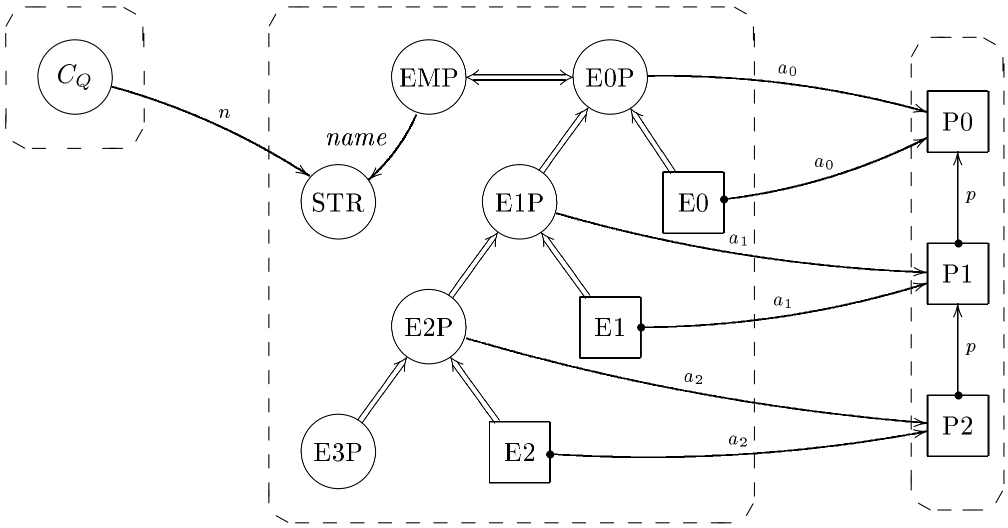

Figure 1: SCHEMA FOR THREE-LEVEL B-TREE ON CLASS EMP.

the data pages. Classes EjP denote *all* employee records in a data page at level i for $i \geq j$. The records are presumed to have a *name* field. A conceptual view of the data is represented by the EMP and STR classes. Also illustrated is a primitive concept on the left that abstracts a sample query over EMP and STR.

Squares have the added significance of denoting low-level indexes for which dotted outgoing arcs represent search parameters. For example, since no parameters for the root page P0 are required, a global variable provides access to P0; and because of their respective parameters a_0 and p, access to employee record addresses in E0 and the level one pages P1 can be obtained by scanning an array within P0.

Based on a terminological abstraction $\mathcal{T}$ of the B-tree that is expressed in terms of $\mathcal{DLFDE}$, we show how our optimizer uses a DL reasoner to translate a request for all distinct employee names,

select distinct $e.name$ as n from EMP as e,

to the following equivalent formulation:

```
select n
from (from [P0 as v0], (
        (select e, v0 [from E0 as e, e.a0 = v0] ) union all
        (select e, v0 from [from P1 as v1, v1.p = v0] , (
           (select e, v1 [from E1 as e, e.a1 = v1] ) union all
           (select e, v1 from [from P2 as v2, v2.p = v1] ,
                              [from E2 as e, e.a2 = v2] ))))),
     [n = e.name] .
```

To interpret the latter as an access plan, the subexpressions enclosed in all but the last box should be understood as index scans (note that some of the indices are supplied with parameters), and the last box as an assignment. The `union all`, `select`, and remaining `from` operations should be understood as concatenations, as duplicate-preserving projections, and as binary nested loop joins, respectively. With these assumptions, the second formulation obtains all employee names by performing a pre-order traversal of the index pages that comprise the B-tree.

Part of what is needed for this to work is an ability to include certain kinds of equational constraints in a terminology. For example, there is a crucial logical relationship between employee records not occurring in the root page, E1P, and the data pages on the first level of the B-tree, P1. This dependency can be captured by asserting that *the* a_0 *value of each employee record in class* E1P *is equal to its* $a_1.p$ *value*, i.e., by adding the inclusion dependency E1P $\sqsubseteq (a_0 = a_1.p)$ to $\mathcal{T}$. Thus, another of our contributions is to incorporate an equational concept constructor in $\mathcal{DLFDE}$ in a way that retains decidability of the logical implication problem for $\mathcal{DLFDE}$, and that remains sufficiently expressive for fine grained data descriptions such as our unfolded B-tree case.

1.1 Related Work

A good survey of how DL reasoning can be useful in information systems, circa 1995, can be found in [1]. Of particular relevance to our own work is the use of DLs as a lingua franca for various database schema languages [2, 8], and then as a means for information integration [4, 6, 7]. Recently, there has been some work on using DLs to reason about set query containment [3, 9] and about bag query equivalence [10] in the presence of database schema expressed in terms of DLs. Much of this more recent work also depends on an ability to express functional constraints [5, 11].

The remainder of the paper is organized as follows. A DL dialect called $\mathcal{DLFDE}$ and a query language called $\mathcal{QL}$ is first introduced in Section 2. Section 3 then defines a number of bidirectional rules for rewriting queries. Our focus is on rules that involve reasoning with $\mathcal{DLFDE}$ terminologies that in turn abstract database schema and the structural properties of the queries themselves. Section 3 also outlines a general approach to query optimization based on these rules, and efficient ways to integrate reasoning in $\mathcal{DLFDE}$ with query rewriting. Section 4 gives a brief summary and outline of further extensions of the approach.

2 Definitions

We begin with a description (or feature) logic that allows the use of a concept constructor for equational constraints in possibly cyclic terminologies. This is only permitted in a way that ensures decidability for the associated implication problem. The new DL generalizes an earlier version first defined in [10]. To the best of our knowledge, this earlier version was the first to allow both functional and equational concept constructors to be used in terminologies.

Definition 1 (Syntax and Semantics of $\mathcal{DLFDE}$)
Let F be a set of attribute names. A path expression *is defined by the grammar "*$\mathsf{Pf} ::= f^{i,j}.\,\mathsf{Pf} \,|\, \mathit{Id}$*" for $f^{i,j} \in F$. Also, the right and left superscripts of consecutive attributes must match in every well formed path description.*

	Syntax	Semantics: Defn of "$(\cdot)^{\mathcal{I}}$"
D^i ::=	C^i	$(C^i)^{\mathcal{I}} \subseteq \Delta^i$
$\mid$	$D_1^i \sqcap D_2^i$	$(D_1^i)^{\mathcal{I}} \cap (D_2^i)^{\mathcal{I}}$
$\mid$	$D_1^i \sqcup D_2^i$	$(D_1^i)^{\mathcal{I}} \cup (D_2^i)^{\mathcal{I}}$
$\mid$	$\neg D^i$	$\Delta^i \setminus (D^i)^{\mathcal{I}}$
$\mid$	$\forall f^{i,j}.D^j$	$\{x : (f^{i,j})^{\mathcal{I}}(x) \in (D^j)^{\mathcal{I}}\}$
E^i ::=	D^i	
$\mid$	$E_1^i \sqcap E_2^i$	$(E_1^i)^{\mathcal{I}} \cap (E_2^i)^{\mathcal{I}}$
$\mid$	$\forall f^{i,j}.E^j$	$\{x : (f^{i,j})^{\mathcal{I}}(x) \in (E^j)^{\mathcal{I}}\}$
$\mid$	$(\mathsf{Pf}_1 = \mathsf{Pf}_2)$	$\{x : (\mathsf{Pf}_1)^{\mathcal{I}}(x) = (\mathsf{Pf}_2)^{\mathcal{I}}(x)\}$
$\mid$	$D^i\{\, \mathsf{Pf}_1, ..., \mathsf{Pf}_k \,\} \rightarrow \mathsf{Pf}$	$\{x : \forall\, y \in (D^i)^{\mathcal{I}}.\ \bigwedge_{i=1}^{k} (\mathsf{Pf}_i)^{\mathcal{I}}(x) = (\mathsf{Pf}_i)^{\mathcal{I}}(y) \Rightarrow (\mathsf{Pf})^{\mathcal{I}}(x) = (\mathsf{Pf})^{\mathcal{I}}(y)\}$

Figure 2: Syntax and Semantics of $\mathcal{DLFDE}$.

Let C^i for $i \in N$ be primitive concept description(s). We define derived concept descriptions *by the grammar in Figure 2.*

An inclusion dependency *is an expression of the form $D^i \sqsubseteq E^i$.*

The semantics *of expressions is defined with respect to a many-sorted structure $(\Delta^i, \cdot^{\mathcal{I}})$, where Δ^i are disjoint domains of "objects" and $(.)^{\mathcal{I}}$ an interpretation function that fixes the interpretations of primitive concepts C^i to be subsets of Δ^i and primitive attributes $f^{i,j}$ to be total functions $(f^{i,j})^{\mathcal{I}} : \Delta^i \rightarrow \Delta^j$. The interpretation is extended to path expressions, $(Id)^{\mathcal{I}} = \lambda x.x$, $(f^{i,j}.\mathsf{Pf})^{\mathcal{I}} = (\mathsf{Pf})^{\mathcal{I}} \circ (f^{i,j})^{\mathcal{I}}$ and derived concept descriptions D^i and E^i as defined in Figure 2.*

An interpretation satisfies an inclusion dependency *$D^i \sqsubseteq E^i$ if $(D^i)^{\mathcal{I}} \subseteq (E^i)^{\mathcal{I}}$.*

A terminology *$\mathcal{T}$ consists of a finite set of inclusion dependencies, and is* stratified *if:*

1. *For each description $\forall f^{i,j}.D$, we have $i \leq j$;*
2. *For each description $D\{\, \mathsf{Pf}_1, \ldots, \mathsf{Pf}_k \,\} \rightarrow \mathsf{Pf}$, each Pf_i consists only of primitive attributes $f^{i,j}$ for which $i = j$; and*
3. *For each description $(\mathsf{Pf}_1 = \mathsf{Pf}_2)$, each Pf_j consists only of primitive attributes $f^{i,j}$ for which $i \leq j$ and has at least one primitive attribute for which $i < j$.*

The logical implication problem *asks if $\mathcal{T} \models D^i \sqsubseteq E^i$ holds; that is, if all interpretations that satisfy all constraints in $\mathcal{T}$ also satisfy $(D^i)^{\mathcal{I}} \subseteq (E^i)^{\mathcal{I}}$ (the posed question).*

Herein, we simplify the notation for path expressions by omitting the superscripts for descriptions and primitive attributes whenever clear from the context, and allow a syntactic composition $\mathsf{Pf}_1 \,.\, \mathsf{Pf}_2$ of path expressions that stands for their concatenation. The following theorem is an extension of an earlier result [13].

Proposition 2 *Let $\mathcal{T}$ be a stratified $\mathcal{DLFDE}$ terminology. Then the $\mathcal{DLFDE}$ implication problem $\mathcal{T} \models D^i \sqsubseteq E^i$ is decidable and complete for DEXPTIME.*

Relaxing the restrictions on terminologies leads to undecidable implication problems. In particular, an absence of either condition (1) or (3) will allow equations of the form $(f_1.f_2 = f_3)$ on a cyclic schema (which is well known to be undecidable [12]). Perhaps surprisingly, allowing path functional dependencies (and in turn keys) to constrain the behavior of $f^{i,j}$-attributes for which $i < j$ also leads to undecidability, as illustrated by the following.

1: $\mathrm{EMP} \sqsubseteq \forall name.\mathrm{STR}$
2: $\mathrm{EMP} \sqsubseteq \mathrm{EMP}\{name\} \to Id$
3:
4: $\mathrm{P0} \sqsubseteq \mathrm{P0}\{\} \to Id$
5: $\mathrm{P1} \sqsubseteq \forall p.P0$
6: $\mathrm{P2} \sqsubseteq \forall p.P1$
7:
8: $\mathrm{EMP} \sqsubseteq \mathrm{E0P}$
9: $\mathrm{E0P} \sqsubseteq \mathrm{EMP} \sqcap (\forall a_0.\mathrm{P0}) \sqcap (\mathrm{E0} \sqcup \mathrm{E1P})$

10: $\mathrm{E0} \sqsubseteq \mathrm{E0P}$
11: $\mathrm{E1P} \sqsubseteq \mathrm{E0P} \sqcap (\forall a_1.\mathrm{P1}) \sqcap (\mathrm{E1} \sqcup \mathrm{E2P}) \sqcap (a_1.p = a_0)$
12: $\mathrm{E0} \sqcap \mathrm{E1P} \sqsubseteq \bot$
13:
14: $\mathrm{E1} \sqsubseteq \mathrm{E1P}$
15: $\mathrm{E2P} \sqsubseteq \mathrm{E1P} \sqcap (\forall a_2.\mathrm{P2}) \sqcap (\mathrm{E2} \sqcup \mathrm{E3P}) \sqcap (a_2.p = a_1)$
16: $\mathrm{E1} \sqcap \mathrm{E2P} \sqsubseteq \bot$
17:
18: $\mathrm{E2} \sqsubseteq \mathrm{E2P}$
19: $\mathrm{E3P} \sqsubseteq \bot$

Figure 3: AN UNFOLDED B-TREE SCHEMA IN $\mathcal{DLFDE}$.

Example 3 Consider a general (undecidable) decision problem of the form

$$\{C^0 \sqsubseteq \forall f^{0,0}_{i,j}.C^0\} \cup \boxed{\{C^0 \sqsubseteq (f^{0,0}_{i,1}.f^{0,0}_{i,2} = f^{0,0}_{i,3})\}} \models C^0 \sqsubseteq (f^{0,0}_1.f^{0,0}_2 = f^{0,0}_3)$$

where $0 < i \leq n$ and $0 < j \leq 3$. The inclusion dependencies enclosed in the box have the following equivalent formulation (note the use of $f^{0,1}$-attributes).

$$\{C^0 \sqsubseteq (f^{0,0}_{i,1}.f^{0,1}_{i,1} = f^{0,1}_{i,2}) \sqcap (f^{0,0}_{i,2}.f^{0,1}_{i,3} = f^{0,1}_{i,1}) \sqcap (f^{0,0}_{i,3}.f^{0,1}_{i,3} = f^{0,1}_{i,2}) \sqcap C^0\{f^{0,1}_{i,3}\} \to Id\}$$

The reformulated terminology now satisfies conditions (1) and (3), but fails to satisfy condition (2).

For our unfolded B-tree, however, the restriction to stratified terminologies does not cause any problems. In particular, a terminology that corresponds to the graphical representation of this data structure given earlier in Figure 1 is listed in Figure 3. An interpretation $(.)^{\mathcal{I}}$ induces a (class of) *database instance*s over which we can formulate queries. The query language we use for this purpose is a positive fragment of an object-relational first-order language with duplicate semantics:

$Q ::= D$ `as` a	$[\![Q]\!]\mathcal{I} := \{\!\!\{\langle a : v\rangle : v \in (D)^{\mathcal{I}}\}\!\!\}$
$\mid\ a.\mathsf{Pf}_1 = b.\mathsf{Pf}_2$	$\{\!\!\{\langle a : v, b : w\rangle : (\mathsf{Pf}_1)^{\mathcal{I}}(v) = (\mathsf{Pf}_2)^{\mathcal{I}}(w)\}\!\!\}$
$\mid$ `select` $a_1, \ldots, a_n\ Q$	$\{\!\!\{\langle a_1 : v@a_1, \ldots, a_n : v@a_n\rangle : v \in [\![Q]\!]\mathcal{I}\}\!\!\}$
$\mid$ `elim` $a_1, \ldots, a_n\ Q$	$\{\langle a_1 : v@a_1, \ldots, a_n : v@a_n\rangle : v \in [\![Q]\!]\mathcal{I}\}$
$\mid$ `true`	$\{\langle\,\rangle\}$
$\mid$ `from` Q_1, Q_2	$[\![Q_1]\!]\mathcal{I} \bowtie [\![Q_2]\!]\mathcal{I}$
$\mid$ `empty` $a_1, \ldots, a_n$	$\emptyset$
$\mid\ Q_1$ `union all` Q_2	$[\![Q_1]\!]\mathcal{I} \uplus [\![Q_2]\!]\mathcal{I}$

Figure 4: SYNTAX AND SEMANTICS OF $\mathcal{QL}$.

Definition 4 (Object Relational Queries) *Figure 4 defines the syntax (left) and semantics (right) of the query language $\mathcal{QL}$. In the syntax, the symbols a, a_i, and b stand for* query variables *(identifiers). The semantics is given with respect to an interpretation $(.)^{\mathcal{I}}$. In addition we assume standard* syntactic safety *conditions to be satisfied by the queries.*

For the remainder of the paper, we use parentheses and common abbreviations to make queries more readable; e.g., `from` $Q_1, \ldots, Q_k$ stands for iterated binary joins.

3 Query Optimization

We now present a collection of bidirectional rules for rewriting queries expressed in $\mathcal{QL}$, with a particular focus on the rules that depend critically on an ability to reason about $\mathcal{DLFDE}$ terminologies. We then outline an optimization process that uses these rules, primarily by illustrating their behavior on our B-tree case.

3.1 Transformation Rules

Typically, rule-based query optimizers use rules that are designed to apply universally to all subqueries, possibly with respect to integrity constraints that hold in a database schema. Often, however, a subquery nested within another fixed query, what we shall call a *query context*, can be rewritten to another subquery that preserves equivalence *only with respect to the enclosing query context*. Our formulation of query rules apply in this more general circumstance, and therefore depend on the following definition.

Definition 5 (Context) *An expression $Q[\,]$ in the language $\mathcal{QL}$ enriched by an additional terminal symbol $[\,]$ is called a* query context. *For a query $Q \in \mathcal{QL}$, the expression $Q[Q']$, denotes the syntactical substitution of Q' for $[\,]$. We also say that Q' is* compatible *with $Q[\,]$ if $Q[Q'] \in \mathcal{QL}$.*

To benefit from terminological reasoning, our rules refer to descriptions that abstract both subqueries Q and enclosing query contexts $Q[\,]$, denoted E_Q and $E_{Q[\,]}$, respectively. These descriptions capture important structural information about subquery results and considerably enhance the power of the rules. In particular, by adding inclusion dependencies of the form $C_Q \sqsubseteq E_Q$ and $C_Q \sqsubseteq E_{Q[\,]}$ to a given terminology (C_Q denotes a primitive concept anchor for the descriptions), a $\mathcal{DLFDE}$ reasoner can deduce additional information needed by the rules. Definitions of E_Q and $E_{Q[\,]}$ are as follows.

$$E_Q = \begin{cases} \forall a.D, & \text{if } Q = \text{“}D \texttt{ as } a\text{”;} \\ (a_1.\mathsf{Pf}_1 = a_2.\mathsf{Pf}_2), & \text{if } Q = \text{“}(a_1.\mathsf{Pf}_1 = a_2.\mathsf{Pf}_2)\text{”;} \\ E_{Q'}, & \text{if } Q = \text{“}\texttt{select } V\ Q'\text{” or “}\texttt{elim } V\ Q'\text{”;} \\ \top, & \text{if } Q = \text{“}\texttt{true}\text{”;} \\ E_{Q_1} \sqcap E_{Q_2}, & \text{if } Q = \text{“}\texttt{from } Q_1, Q_2\text{”;} \\ \bot, & \text{if } Q = \text{“}\texttt{empty } V\text{”; or} \\ E_{Q_1} \sqcup E_{Q_2}, & \text{if } Q = \text{“}Q_1 \texttt{ union all } Q_2\text{”.} \end{cases}$$

$$E_{Q[\,]} = \begin{cases} \top, & \text{if } Q[\,] = \text{“}[\,]\text{”;} \\ E_{Q[\,]} \sqcap E_{Q'}, & \text{if } Q[\,] = \text{“}Q[\texttt{from } Q', [\,]]\text{” or “}Q[\texttt{from } [\,], Q']\text{”; or} \\ E_{Q[\,]}, & \text{if } Q[\,] = \text{“}Q[\texttt{elim } V[\,]]\text{”, “}Q[\texttt{select } V[\,]]\text{”,} \\ & \qquad \text{“}Q[Q' \texttt{ union all } [\,]]\text{” or “}Q[[\,] \texttt{ union all } Q']\text{”.} \end{cases}$$

For every query Q and query context $Q[\,]$, we also define the set α_Q of *free variables* of Q and the set $\alpha_{Q[\,]}$ of *variables captured* by the context $Q[\,]$. These two sets are defined inductively on the structure of Q and $Q[\,]$, respectively.

The rules are listed in Figure 5, and operate as follows.

$\bowtie$-intro: Given a existing query variable a, the DL reasoner infers that the hypothetical query object C_Q satisfies the constraint $C_Q \sqsubseteq \forall a.D$. This means that all valuations of a in $Q[\,]$ must also belong to $(D)^{\mathcal{I}}$ and thus we can add or remove a conjunct “D `as` a” to or from the query. Also note that the terminal `true` can be introduced or removed anywhere in Q using a standard rule for join.

$$\bowtie\text{-intro: } \frac{Q[\mathtt{true}]}{Q[D\ \mathtt{as}\ a]}\quad \mathcal{T} \cup \{C_Q \sqsubseteq E_{Q[\,]}\} \models C_Q \sqsubseteq \forall a.D \text{ and } a \in \alpha_{Q[\,]}$$

$$=\text{-intro: } \frac{Q[\mathtt{true}]}{Q[a.\,\mathsf{Pf}_1 = b.\,\mathsf{Pf}_2]}\quad \mathcal{T} \cup \{C_Q \sqsubseteq E_{Q[\,]}\} \models C_Q \sqsubseteq (a.\,\mathsf{Pf}_1 = b.\,\mathsf{Pf}_2) \text{ and } a, b \in \alpha_{Q[\,]}$$

$$\text{var-intro: } \frac{Q[a_1.\,\mathsf{Pf}_1 = a_2.\,\mathsf{Pf}_2\,.\,\mathsf{Pf}_3]}{Q[\mathtt{from}\ a_1.\,\mathsf{Pf}_1 = b.\,\mathsf{Pf}_3, b = a_2.\,\mathsf{Pf}_2]}\quad b \notin \alpha_{Q[\,]} \cup \{a_1, a_2\}$$

$$\uplus\text{-intro: } \frac{Q[(E_1 \sqcup E_2)\ \mathtt{as}\ a]}{Q[\mathtt{elim}\ a\ ((E_1\ \mathtt{as}\ a)\ \mathtt{union\ all}\ (E_2\ \mathtt{as}\ a))]}$$

$$\varepsilon\text{-elim: } \frac{Q[\mathtt{elim}\ a\ D\ \mathtt{as}\ a]}{Q[D\ \mathtt{as}\ a]}\qquad \varepsilon\text{-elim: } \frac{Q[\mathtt{elim}\ a, b\ \ a.\,\mathsf{Pf}_1 = b.\,\mathsf{Pf}_2]}{Q[a.\,\mathsf{Pf}_1 = b.\,\mathsf{Pf}_2]}$$

$$\bowtie\text{-}\varepsilon\text{-dist: } \frac{Q[\mathtt{elim}\ V\ \mathtt{from}\ Q_1, Q_2]}{Q[\mathtt{select}\ V\ \mathtt{from}\ (\mathtt{elim}\ \alpha_{Q_1}\ Q_1), (\mathtt{elim}\ W\ Q_2)]}\quad \begin{array}{l} \mathcal{T} \cup \{C_Q \sqsubseteq E_{Q[\,]} \sqcap E_{Q_1} \sqcap E_{Q_2}\} \\ \models C_Q \sqsubseteq C_Q\{V \cup \alpha_{Q[\,]}\} \to \alpha_{Q_1} \\ \text{where } W = (V \cup \alpha_{Q_1}) \cap \alpha_{Q_2} \end{array}$$

$$\uplus\text{-}\varepsilon\text{-dist: } \frac{Q[\mathtt{elim}\ V\ (Q_1\ \mathtt{union\ all}\ Q_2)]}{Q[(\mathtt{elim}\ V\ Q_1)\ \mathtt{union\ all}\ (\mathtt{elim}\ V\ Q_2)]}\quad \mathcal{T} \cup \{C_Q \sqsubseteq E_{Q[\,]} \sqcap E_{Q_1} \sqcap E_{Q_2}\} \models C_Q \sqsubseteq \bot$$

Figure 5: QUERY TRANSFORMATION RULES.

=-intro: As above, an equational constraint on query variables that is implied in the context $Q[\,]$ by the terminology and the abstraction $C_Q \sqsubseteq E_{Q[\,]}$ can be freely added or removed.

var-intro: Existing equational constraints in queries can be split by introducing a new query variable with a unique name. Although this rule does not depend on DL reasoning, it enables the introduction of further information from the database schema into the query by subsequent use of rules $\bowtie$-intro and =-intro.

$\uplus$-intro: Query variables labeled with disjunctive descriptions can be translated to the `union all` operation. Note the use of the duplicate elimination operator `elim` to account for the discrepancy between the set semantics of the description and the bag semantics of the `union all` operation.

ε-elim: The (usually expensive) `elim` operation can be removed from queries when its input is duplicate free. This is guaranteed by the semantics of the atomic queries "D `as` a" and "$a.\,\mathsf{Pf}_1 = b.\,\mathsf{Pf}_2$".

$\bowtie$-ε-dist: An `elim` operator can be distributed over a join to subqueries. However, this is only possible if the DL reasoner can deduce that the query variables of one of the subqueries are functionally determined by the "remaining" variables. This rule generalizes an earlier version [10].

$\uplus$-ε-dist: An `elim` operator can also be distributed over a `union all` if the DL reasoner can deduce that the two subqueries must be disjoint by virtue of their abstractions.

The above observations, together with an induction on query contexts, yield the following result.

Proposition 6 *The rules in Figure 5 are sound.*

In our running example we also use several *administrative rules*, e.g., commutativity and associativity of joins, distributivity of unions over joins, absorption of nested duplicate elimination operators, etc.

3.2 Implementation

Figure 6 illustrates a sequence of applications of rules in Figure 5 that progressively transforms the "names of employees" query to the query plan given in the introduction. The order of application of these rewrites generally follows the approach outlined in [14] for a query language with set semantics. In particular, there are three phases that altogether find query plans for conjunctive fragments.

1. An initial *query expansion* phase is applied for conjunctive subexpressions (e.g., Steps 1-7 in Figure 6). The phase introduces additional existentially quantified variables and conditions whose existence is implied by the terminology. Observe the use of rules $\bowtie$-intro, $=$-intro and var-intro.
2. The second phase essentially relates to so-called *join order selection*. A "choice-point" during join order selection is to select from among the variables and conditions that relate to index use. For example, index P0 is selected (Step 8) and, if possible removed from the scope of an `elim` operator (Step 9). An important detraction from normal practice, however, is an option to explore disjunctive descriptions using rule $\uplus$-intro in lieu of "selecting next index" (Step 12). In this latter case, administrative rules (Step 13) and (if applicable) rule $\uplus$-ε-dist (Step 14) are then used to "prepare" each disjunct for a recursive application of the three phases (Steps 15-16 for the first disjunct; the remaining steps for the second disjunct).
3. The third and final *query contraction* phase then removes redundant variables and conditions not used in the actual query plan. Observe the use of rules $\bowtie$-intro, $=$-intro and var-intro in reverse (Steps 16 and 19).

Another feature of our implementation of this process is the use of a graph encoding of queries that is also used directly by the $\mathcal{DLFDE}$ model building procedure. Also, query contraction is instead accomplished by applying query expansion on a separate *plan graph* constructed during join order selection.

4 Summary

We have outlined an approach to query optimization for a positive fragment of a query language with duplicate semantics for which a DL reasoner is used to search for possible index use, including cases that require exploring horizontally partitioned data, and to reason about duplicate elimination and its interaction with the join and union operations. We have also presented $\mathcal{DLFDE}$, a boolean-complete DL dialect with a DEXPTIME-complete implication problem that includes concept constructors for both functional and equational constraints, and allows a stratified use of these constructors to occur in possibly cyclic terminologies.

There are many opportunities for future work. For example, we are currently exploring alternative restrictions on $\mathcal{DLFDE}$ terminologies that continue to guarantee decidability. Completeness results that relate to query equivalence defined in terms of

1. $\texttt{elim}\ n\ \texttt{from}\ \mathrm{EMP}\ \texttt{as}\ e, n = e.\mathit{name}$

2. $\texttt{elim}\ n\ \texttt{from}\ \mathrm{EMP}\ \texttt{as}\ e, n = e.\mathit{name}, \mathrm{E0P}\ \texttt{as}\ e$ $\qquad\bowtie$-intro and 8

3. $\texttt{elim}\ n\ \texttt{from}\ \mathrm{EMP}\ \texttt{as}\ e, n = e.\mathit{name}, \mathrm{E0P}\ \texttt{as}\ e, (\mathrm{E0} \sqcup \mathrm{E1P})\ \texttt{as}\ e$ $\qquad\bowtie$-intro and 9

4. $\texttt{elim}\ n\ \texttt{from}\ \mathrm{EMP}\ \texttt{as}\ e, n = e.\mathit{name}, \mathrm{E0P}\ \texttt{as}\ e, (\mathrm{E0} \sqcup \mathrm{E1P})\ \texttt{as}\ e, e.a_0 = e.a_0$ $\qquad=$-intro

5. $\texttt{elim}\ n\ \texttt{from}\ \mathrm{EMP}\ \texttt{as}\ e, n = e.\mathit{name}, \mathrm{E0P}\ \texttt{as}\ e, (\mathrm{E0} \sqcup \mathrm{E1P})\ \texttt{as}\ e, e.a_0 = v_0, v_0 = e.a_0$ $\qquad$var-intro

6. $\texttt{elim}\ n\ \texttt{from}\ \mathrm{EMP}\ \texttt{as}\ e, n = e.\mathit{name}, \mathrm{E0P}\ \texttt{as}\ e, (\mathrm{E0} \sqcup \mathrm{E1P})\ \texttt{as}\ e, e.a_0 = v_0$ $\qquad$rev-$=$-intro

7. $\texttt{elim}\ n\ \texttt{from}\ \mathrm{EMP}\ \texttt{as}\ e, n = e.\mathit{name}, \mathrm{E0P}\ \texttt{as}\ e, (\mathrm{E0} \sqcup \mathrm{E1P})\ \texttt{as}\ e, e.a_0 = v_0, \mathrm{P0}\ \texttt{as}\ v_0$ $\qquad\bowtie$-intro and 9

8. $\texttt{select}\ n\ \texttt{from}\ \texttt{elim}\ v_0\ \mathrm{P0}\ \texttt{as}\ v_0,$
   $\texttt{elim}\ n, v_0\ \texttt{from}\ \mathrm{EMP}\ \texttt{as}\ e, n = e.\mathit{name}, \mathrm{E0P}\ \texttt{as}\ e, (\mathrm{E0} \sqcup \mathrm{E1P})\ \texttt{as}\ e, e.a_0 = v_0$ $\qquad\bowtie$-ε-dist

9. $\texttt{select}\ n\ \texttt{from}\ \mathrm{P0}\ \texttt{as}\ v_0,$
   $\texttt{elim}\ n, v_0\ \texttt{from}\ \mathrm{EMP}\ \texttt{as}\ e, n = e.\mathit{name}, \mathrm{E0P}\ \texttt{as}\ e, (\mathrm{E0} \sqcup \mathrm{E1P})\ \texttt{as}\ e, e.a_0 = v_0$ $\qquad\varepsilon$-elim

10. $\texttt{select}\ n\ \texttt{from}\ \mathrm{P0}\ \texttt{as}\ v_0,$
    $\texttt{select}\ n, v_0\ \texttt{from}\ \texttt{elim}\ e, v_0\ \texttt{from}\ \mathrm{EMP}\ \texttt{as}\ e, \mathrm{E0P}\ \texttt{as}\ e, (\mathrm{E0} \sqcup \mathrm{E1P})\ \texttt{as}\ e, e.a_0 = v_0,$
    $\texttt{elim}\ n, e\ \ n = e.\mathit{name}$ $\qquad\bowtie$-ε-dist

11. $\texttt{select}\ n\ \texttt{from}\ \mathrm{P0}\ \texttt{as}\ v_0,$
    $\texttt{elim}\ e, v_0\ \texttt{from}\ \mathrm{EMP}\ \texttt{as}\ e, \mathrm{E0P}\ \texttt{as}\ e, (\mathrm{E0} \sqcup \mathrm{E1P})\ \texttt{as}\ e, e.a_0 = v_0,$
    $n = e.\mathit{name}$ $\qquad\varepsilon$-elim+administrative rules

12. $\texttt{select}\ n\ \texttt{from}\ \mathrm{P0}\ \texttt{as}\ v_0,$
    $\texttt{elim}\ e, v_0\ \texttt{from}\ \mathrm{EMP}\ \texttt{as}\ e, \mathrm{E0P}\ \texttt{as}\ e, e.a_0 = v_0, \texttt{elim}\ e\ (\mathrm{E0}\ \texttt{as}\ e)\ \texttt{union all}\ (\mathrm{E1P}\ \texttt{as}\ e),$
    $n = e.\mathit{name}$ $\qquad\uplus$-intro

13. $\texttt{select}\ n\ \texttt{from}\ \mathrm{P0}\ \texttt{as}\ v_0,$
    $\texttt{elim}\ e, v_0 (\texttt{from}\ \mathrm{EMP}\ \texttt{as}\ e, \mathrm{E0P}\ \texttt{as}\ e, e.a_0 = v_0, \mathrm{E0}\ \texttt{as}\ e)\ \texttt{union all}$
    $(\texttt{from}\ \mathrm{EMP}\ \texttt{as}\ e, \mathrm{E0P}\ \texttt{as}\ e, e.a_0 = v_0, \mathrm{E1P}\ \texttt{as}\ e),$
    $n = e.\mathit{name}$ $\qquad$administrative rules

14. $\texttt{select}\ n\ \texttt{from}\ \mathrm{P0}\ \texttt{as}\ v_0,$
    $(\texttt{elim}\ e, v_0\ \texttt{from}\ \mathrm{EMP}\ \texttt{as}\ e, \mathrm{E0P}\ \texttt{as}\ e, e.a_0 = v_0, \mathrm{E0}\ \texttt{as}\ e)\ \texttt{union all}$
    $(\texttt{elim}\ e, v_0\ \texttt{from}\ \mathrm{EMP}\ \texttt{as}\ e, \mathrm{E0P}\ \texttt{as}\ e, e.a_0 = v_0, \mathrm{E1P}\ \texttt{as}\ e),$
    $n = e.\mathit{name}$ $\qquad\uplus$-ε-dist

15. $\texttt{select}\ n\ \texttt{from}\ \mathrm{P0}\ \texttt{as}\ v_0,$
    $(\texttt{select}\ e, v_0\ \texttt{from}\ \mathrm{EMP}\ \texttt{as}\ e, \mathrm{E0P}\ \texttt{as}\ e, e.a_0 = v_0, \mathrm{E0}\ \texttt{as}\ e)\ \texttt{union all}$
    $(\texttt{elim}\ e, v_0\ \texttt{from}\ \mathrm{EMP}\ \texttt{as}\ e, \mathrm{E0P}\ \texttt{as}\ e, e.a_0 = v_0, \mathrm{E1P}\ \texttt{as}\ e),$
    $n = e.\mathit{name}$ $\qquad 3\times\ \bowtie$-ε-dist+$4\times\ \varepsilon$-elim

16. $\texttt{select}\ n\ \texttt{from}\ \mathrm{P0}\ \texttt{as}\ v_0,$
    $(\texttt{select}\ e, v_0\ \texttt{from}\ \mathrm{E0}\ \texttt{as}\ e, e.a_0 = v_0)\ \texttt{union all}$
    $(\texttt{elim}\ e, v_0\ \texttt{from}\ \mathrm{EMP}\ \texttt{as}\ e, \mathrm{E0P}\ \texttt{as}\ e, e.a_0 = v_0, \mathrm{E1P}\ \texttt{as}\ e),$
    $n = e.\mathit{name}$ $\qquad 2\times$ rev-$\bowtie$-intro and 9, 10

17. $\texttt{select}\ n\ \texttt{from}\ \mathrm{P0}\ \texttt{as}\ v_0,$
    $(\texttt{select}\ e, v_0\ \texttt{from}\ \mathrm{E0}\ \texttt{as}\ e, e.a_0 = v_0)\ \texttt{union all}$
    $(\texttt{elim}\ e, v_0\ \texttt{from}\ \mathrm{EMP}\ \texttt{as}\ e, \mathrm{E0P}\ \texttt{as}\ e, e.a_0 = v_0,$
    $\mathrm{E1P}\ \texttt{as}\ e, (\mathrm{E1} \sqcup \mathrm{E2P})\ \texttt{as}\ e, e.a_1 = v_1, \mathrm{P1}\ \texttt{as}\ v_1),$
    $n = e.\mathit{name}$ $\qquad$analogous to steps 3-7

18. $\texttt{select}\ n\ \texttt{from}\ \mathrm{P0}\ \texttt{as}\ v_0,$
    $(\texttt{select}\ e, v_0\ \texttt{from}\ \mathrm{E0}\ \texttt{as}\ e, e.a_0 = v_0)\ \texttt{union all}$
    $(\texttt{elim}\ e, v_0\ \texttt{from}\ \mathrm{EMP}\ \texttt{as}\ e, \mathrm{E0P}\ \texttt{as}\ e, e.a_0 = v_0,$
    $\mathrm{E1P}\ \texttt{as}\ e, (\mathrm{E1} \sqcup \mathrm{E2P})\ \texttt{as}\ e, e.a_1 = v_1, \mathrm{P1}\ \texttt{as}\ v_1, v_1.p = v_0),$
    $n = e.\mathit{name}$ $\qquad=$-intro

19. $\texttt{select}\ n\ \texttt{from}\ \mathrm{P0}\ \texttt{as}\ v_0,$
    $(\texttt{select}\ e, v_0\ \texttt{from}\ \mathrm{E0}\ \texttt{as}\ e, e.a_0 = v_0)\ \texttt{union all}$
    $(\texttt{elim}\ e, v_0\ \texttt{from}\ \mathrm{EMP}\ \texttt{as}\ e, \mathrm{E0P}\ \texttt{as}\ e,$
    $\mathrm{E1P}\ \texttt{as}\ e, (\mathrm{E1} \sqcup \mathrm{E2P})\ \texttt{as}\ e, e.a_1 = v_1, \mathrm{P1}\ \texttt{as}\ v_1, v_1.p = v_0),$
    $n = e.\mathit{name}$ $\qquad$rev-$=$-intro

20. $\texttt{select}\ n\ \texttt{from}\ \mathrm{P0}\ \texttt{as}\ v_0,$
    $(\texttt{select}\ e, v_0\ \texttt{from}\ \mathrm{E0}\ \texttt{as}\ e, e.a_0 = v_0)\ \texttt{union all}$
    $(\texttt{select}\ e, v_0\ \texttt{from}\ \mathrm{P1}\ \texttt{as}\ v_1, v_1.p = v_0,$
    $(\texttt{select}\ e, v_1\ \texttt{from}\ \mathrm{E1}\ \texttt{as}\ e, e.a_1 = v_1)\ \texttt{union all}$
    $(\texttt{select}\ e, v_1\ \texttt{from}\ \mathrm{P2}\ \texttt{as}\ v_2, v_2.p = v_1, \mathrm{E}_2\ \texttt{as}\ e, e.a_2 = v_2)),$
    $n = e.\mathit{name}$ $\qquad$analogous to steps 8-19 applied two additional times

Figure 6: QUERY COMPILATION.

our rules are another ongoing avenue of research. Another direction is the incorporation of *order dependencies* [13] for reasoning about order and aggregate optimizations and their interaction with existing rules. Finally, we are continuing our efforts on evaluating an experimental implementation of our query optimizer on a number of real-world test cases such as the Linux kernel data structures [14].

References

[1] Alexander Borgida. Description logics in data management. *IEEE Transactions on Knowledge and Data Engineering*, 7(5):671–682, 1995.

[2] Andrea Call, Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Reasoning on UML Class Diagrams in Description Logics. In *Proc. of IJCAR Workshop on Precise Modelling and Deduction for Object-oriented Software Development (PMD 2001)*, 2001.

[3] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Answering Queries Using Views over Description Logics Knowledge Bases. In *Proc. of the 16th Nat. Conf. on Artificial Intelligence (AAAI 2000)*, pages 386–391, 2000.

[4] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Description Logics for Information Integration. In A. Kakas and F. Sadri, editors, *Computational Logic: From Logic Programming into the Future (In honour of Bob Kowalski)*, Lecture Notes in Computer Science. Springer-Verlag, 2001. To appear.

[5] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Identification Constraints and Functional Dependencies in Description Logics. In *Proc. of the 17th Int. Joint Conf. on Artificial Intelligence (IJCAI 2001)*, pages 155–160, 2001.

[6] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Ontology of Integration and Integration of Ontologies. In *Proc. of the 2001 Description Logic Workshop (DL 2001)*, pages 10–19. CEUR-WS Vol.49, 2001.

[7] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Daniele Nardi, and Riccardo Rosati. Description Logic Framework for Information Integration. In *Principles of Knowledge Representation and Reasoning (KR'98)*, pages 2–13, 1998.

[8] Diego Calvanese, Maurizio Lenzerini, and Daniele Nardi. Description Logics for Conceptual Data Modelling. In Jan Chomicki and Gunter Saake, editors, *Logics for Databases and Information Systems*, chapter 8. Kluwer, 1998.

[9] Ian Horrocks, Ulrike Sattler, Sergio Tessaris, and Stephan Tobies. How to decide Query Containment under Constraints using a Description Logic. In *Proceedings of the 7th International Workshop on Knowledge Representation meets Databases (KRDB 2000)*, CEUR-WS vol.29, pages 59–72, 2000.

[10] Vitaliy L. Khizder, David Toman, and Grant E. Weddell. Reasoning about Duplicate Elimination with Description Logic. In *Rules and Objects in Databases, DOOD 2000 (part of Computational Logic 2000)*, pages 1017–1032, 2000.

[11] Vitaliy L. Khizder, David Toman, and Grant E. Weddell. On Decidability and Complexity of Description Logics with Uniqueness Constraints. In *International Conference on Database Theory ICDT'01*, pages 54–67, 2001.

[12] Michael Machtey and Paul Young. *An Introduction to the General Theory of Algorithms.* North-Holland Amsterdam, 1978.

[13] David Toman and Grant E. Weddell. On Attributes, Roles, and Dependencies in Description Logics and the Ackermann Case of the Decision Problem. In *Description Logics 2001*, pages 76–85. CEUR-WS vol.49, 2001.

[14] David Toman and Grant E. Weddell. Query Processing in Embedded Control Programs. In *2nd Int. Workshop on Databases in Telecommunications*, pages 68–87. Springer LNCS 2209, 2001.