Reasoning about Duplicate Elimination with Description Logic
(preliminary report)

Vitaliy L. Khizder, David Toman, and Grant Weddell

Department of Computer Science
University of Waterloo, Canada

Abstract. Queries commonly perform much better if they manage to avoid duplicate elimination operations in their execution plans. In this paper, we report on a technique that provides a necessary and sufficient condition for removing such operators from object relational conjunctive queries under the standard duplicate semantics. The condition is fully captured as a membership problem in a dialect of description logic called CFD, which is capable of expressing a number of common constraints implicit in object relational database schemas. We also present a PTIME algorithm for arbitrary membership problems in CFD.

1 Introduction

The paper presents a combination of two techniques to provide a very powerful tool for reasoning about duplicate semantics of conjunctive object relational queries, in particular about elimination operations and situations in which these operations can be completely removed. The query language studied in the paper contains many other practical query languages, in particular the conjunctive fragments of SQL92 [13, 17] and OQL [7, 8].

The first technique provides a characterization of situations in which queries can be mowed out of the scope of a duplicate elimination operator: it defines a sufficient and necessary condition in terms of deducing a *uniqueness constraint* in an abstraction of the query. Unlike many other approaches, we show that our technique *completely* characterizes these situations: in the cases the appropriate condition does not hold, we know that the duplicate elimination operator may not be removed.

The second technique is complementary to the first: it defines a dialect of *description logic* called CFD that

- captures usual schema declarations, including the "decidable" part of the SQL IEF (integrity enhancement feature) declarations, and
- has an efficient inference mechanism that runs in PTIME both in the size of the schema and the query.

In addition to the usual integrity constraints, CFD can capture many other *schema declarations*, e.g., inheritance constraints, path functions, etc.

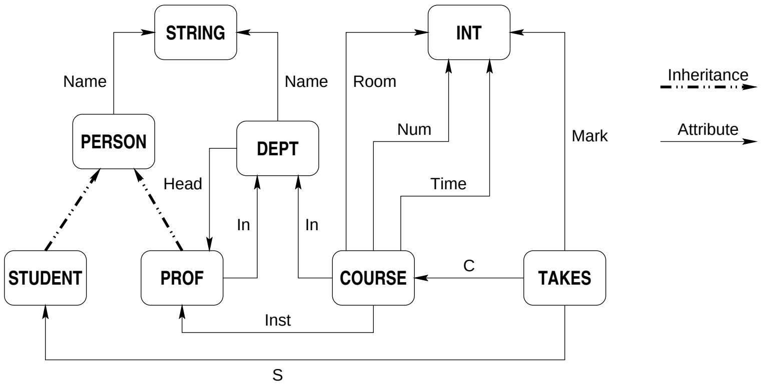

Fig. 1. University Database Schema.

Example 1 We illustrate the approach with an object relational schema for a hypothetical university application illustrated in Figure 1. In particular, consider a request to "*find the unique names of students taking some course taught at the same time as some other course numbered :P2 with an instructor in the department named :P1*". In SQL-like syntax, the query can be formulated as:

```
select distinct S.Name as Name
from  STUDENT as S, TAKES as T, COURSE as C
where S = T.S and C.Time = T.C.Time and
      C.Inst.In.Name = :P1 and C.Num = :P2
```

The results of this paper enable an efficient rewriting of the query to an equivalent formulation that does not involve any use of a duplicate elimination operation (the removal of the `distinct` keyword in this case):

```
select S.Name as Name
from  STUDENT as S, TAKES as T, COURSE as C
where S = T.S and C.Time = T.C.Time and
      C.Inst.In.Name = :P1 and C.Num = :P2
```

This goal is achieved by describing the integrity constraints that hold in the database schema (Figure 2) and then applying a rewriting rule (Example 10).

1.1 Related Work

Description logics have long been recognized as a valuable tool for conceptual modeling and for the specification of database schemas [2, 6]. More recently, they have found applications in query optimization, in particular for issues relating to deciding view relevance [5]. However, so far as we are aware, such logics have not yet been used to reason about duplicate elimination in query plans.

Conversely, many proposals have utilized *functional dependencies* and *foreign key constraints* to optimize queries, e.g., for predicate movement [16], as a supplement to Magic Set optimization for bag semantics [18], for optimization of joins and semi-joins [19], or for decorelation of complex queries [22], and other more general integrity constraints to optimize queries [12].

This paper links the efforts seeking efficient decision procedures for dialects of description logics [4] with actual techniques used for optimizing queries. The link between these two techniques is non-trivial: conjunctive query subsumption is NP-hard for set semantics [9] and Π_2^p-hard for bag semantics [10], in general. However, using our approach we reduce questions about equivalence of *particular* queries related by common forms of rewrites to subsumption constraints in a dialect of description logic that has an efficient (PTIME) decision procedure.

Both the decision procedure and logic are refinements of earlier versions [4]. In this paper the variety of uniqueness constraints is generalized to include path functional dependencies [24, 23] and the presented decision procedure itself incorporates an optimization for membership tests that involve uniqueness constraints that satisfy a symmetry property.

The rest of the paper is organized as follows: Section 2 defines CFD along with an object relational query language used in the rest of the paper. Section 3 presents the first result: a complete rewriting rule for duplication elimination removal from conjunctive object relational queries with respect to their duplicate semantics. Section 4 presents a complete inference procedure for deducing arbitrary subsumption constraints in CFD. All development is illustrated step-by-step using a running example. The paper concludes with some summary comments and suggestion for several directions of further research.

2 Definitions

This section provides all necessary definitions needed to present the results of the paper. The definitions are followed by examples that illustrate how the individual concepts are used.

2.1 Description Logics and Database Schemas

We begin by defining CFD, a dialect of a description logic that incorporates a constructor for capturing uniqueness constraints. CFD is short for "Classic/FD", the name of an earlier version first defined in [4] in which a simpler form of uniqueness construct is added to an earlier dialect called CLASSIC [3]. CFD has enough expressiveness to capture object relational database schemas including a variety of integrity constraints (cf. Example 3).

Definition 2 (Description Logic CFD) *Let C denote primitive concepts and A primitive attributes. The* concept descriptions *are defined by the following grammar:*

$$
\begin{array}{lll}
D :: & ANY & (\textit{all objects}) \\
 & \mid C & (\textit{superclass construct}) \\
 & \mid (\mathsf{Pf} : D) & (\textit{typing construct}) \\
 & \mid C(\mathsf{Pf}_1, \ldots, \mathsf{Pf}_n \to \mathsf{Pf}) & (\textit{uniqueness construct}; n \geq 1) \\
 & \mid \mathsf{Pf}_1 = \mathsf{Pf}_2 & (\textit{equational construct}) \\
 & \mid D_1, D_2 & (\textit{conjunction construct})
\end{array}
$$

```
PERSON < (Name: STRING),                DEPT < (Name: STRING),
         PERSON(Name -> Id)                    (Head: PROF),
                                               DEPT(Name -> Id),
STUDENT < PERSON                               DEPT(Head -> Id)

PROF < PERSON, (In: DEPT)               COURSE < (Num: INT),
                                                 (Room: INT),
TAKES < (S: STUDENT),                            (Time: INT),
        (C: COURSE),                             (Inst: PROF),
        (Mark: INT),                             (In: DEPT),
        TAKES(S, C -> Id),                       COURSE(Num, In -> Id),
        TAKES(S, C.Time -> C)                    COURSE(Room, Time -> Id),
                                                 COURSE(Inst.In -> In)
```

Fig. 2. University Database Schema in CFD.

where the attribute descriptions Pf *are of the form* **Id** *(identity) or* $A.\mathsf{Pf}$ *(path composition). The concept descriptions are used to formulate* subsumption constraints *of the form* $C < D$.

Intuitively, the *descriptions* describe sets of objects (object id's), the *attributes* (or paths) describe the relationships between objects, and *subsumption constraints* restrict valid instances of the database.

Example 3 The University database schema from Figure 1 is captured using CFD in Figure 2. The constraints introduce primitive classes `PERSON`, `STUDENT`, `PROF`, `COURSE`, `DEPT`, `TAKES`, `INT` (for integers), and `STRING` (for strings), and assert that `PERSON`s have `Name`s (strings), that `STUDENT`s and `PROF`s are `PERSON`s (inheritance constraints); that `Name` is a key of `PERSON` (because it determines **Id**—the object id of the person object), and so on. For readability we omit the trailing **Id** in attribute descriptions in our examples.

The semantics of CFD is defined with respect to a possibly infinite collection of entities Dom and a function $(.)^I$. We call the pair $I = (\mathsf{Dom}, (.)^I)$ an *interpretation* or a *database* (these are synonymous in our approach). The second component, $(.)^I$, maps subsumption constraints to truth values, concept descriptions to subsets of Dom, and attribute descriptions to total functions over Dom. There are no further conditions on the interpretation of primitive concepts and attributes. The interpretation of the remaining constructs is constrained as follows:

Definition 4 (Semantics of CFD) *Let* $\mathcal{C}_C \subseteq \mathsf{Dom}$ *be a set for each primitive concept* C*, and* $\mathcal{F}_A : \mathsf{Dom} \to \mathsf{Dom}$ *a function on* Dom *for each attribute* A*. Then*

$$
\begin{array}{ll}
(ANY)^I & = \mathsf{Dom} \\
(C)^I & = \mathcal{C}_C \\
(\mathsf{Pf} : D)^I & = \{v : (\mathsf{Pf})^I v \in (D)^I\} \\
(C(\mathsf{Pf}_1, \ldots, \mathsf{Pf}_n \to \mathsf{Pf}))^I & = \{v : \forall w \in (C)^I . \\
 & \qquad (\mathsf{Pf}_i)^I v = (\mathsf{Pf}_i)^I w \Rightarrow (\mathsf{Pf})^I v = (\mathsf{Pf})^I w\} \\
(\mathsf{Pf}_1 = \mathsf{Pf}_2)^I & = \{v : (\mathsf{Pf}_1)^I v = (\mathsf{Pf}_2)^I v\} \\
(D_1, D_2)^I & = (D_1)^I \cap (D_2)^I
\end{array}
$$

The meanings of the attribute descriptions **Id** *and* $A.\mathsf{Pf}$ *are defined as* $\lambda x.x$ *and* $\lambda x.(\mathsf{Pf})^I(\mathcal{F}_A\ x)$*, respectively. The meaning of a subsumption constraint* $C < D$ *is defined as* $(C)^I \subseteq (D)^I$*.*

A database schema S is a set of subsumption constraints in CDF. We say that a database I *satisfies* schema S, written $I \models S$, if all subsumption constraints in S are true with respect to $(.)^I$.

Similarly, given subsumption constraint $C < D$, we write $S \models C < D$ to express the condition that $C < D$ is a logical consequence of S, that is $I \models S$ implies $I \models C < D$ for all databases I.

To run in PTIME, the decision procedure presented in Section 4 requires two restrictions on the set S. First, only *regular* path functional dependencies are allowed in S. Description $C(\mathsf{Pf}_1, \ldots, \mathsf{Pf}_n \rightarrow \mathsf{Pf})$ is *regular* if Pf (with the possible exception of the last primitive attribute) is a prefix of Pf_i for some $1 \leq i \leq n$ [14]. Second, S may not contain equational constraints. We say that S is *regular* and *equation-free* when it satisfies these conditions. It is easy to verify that the schema in Figure 2 is regular and equation-free.

Relational Databases and Schemas. We have defined databases as pairs of a domain Dom and an interpretation function $(.)^I$. However, it is easy to see that standard SQL table/attribute declarations map to simple subsumption constraints and the databases that satisfy these constraints are equivalent to SQL databases. Consider the declaration

```
CREATE TABLE PERSON(Name char(100))
```

saying that we have a table `PERSON` with a single attribute `NAME`. This is equivalently captured by the subsumption constraint `PERSON < (Name: STRING)`; the function associated with the attribute `Name`, given a person object, returns a string. The SQL IEF (Integrity Enhancement Feature) also allows declaration of *integrity constraints* that must hold on the schema. In this paper we consider the *entity integrity* and the *referential integrity* constraints (the *domain* constraints are captured as above[1]).

The *entity integrity* (key) constraints are captured in CFD using the uniqueness construct. To declare that `Name` as a key of the `PERSON` table we say `PERSON < PERSON(Name -> Id)` where `Id` is essentially a reference to the object identifier of person (presumed to be unique for each object in the class).

The *referential integrity* constraints in our approach are more limited than in SQL. However, such a restriction is necessary as SQL's referential integrity constraints allow general embedded inclusion dependencies and thus reasoning becomes undecidable [1]. We restrict the referential integrity to unary foreign key constraints (i.e., the target of the dependency must be a key—in our case the object identifier—which reduces to attribute declarations in our model). This restriction leads to efficiently decidable theory (cf. section 4) while capturing a large class of practical schema declarations in SQL.

[1] With the exception of general `CHECK` constraints.

2.2 Conjunctive Object Relational Queries

We define the core syntax of the conjunctive fragment of object relational queries by the following grammar:

Definition 5 (Object Relational Conjunctive Queries)

$$
\begin{array}{ll}
Q :: C \;\texttt{as}\; A & (\textit{Primitive class}) \\
\quad \mid A.\mathsf{Pf}_1 = B.\mathsf{Pf}_2 & (\textit{Equational constraint}) \\
\quad \mid \texttt{select}\; A_1, \ldots, A_n\; Q & (\textit{Projection}) \\
\quad \mid \texttt{elim}\; Q & (\textit{Duplicate elimination}) \\
\quad \mid \texttt{from}\; Q_1, \ldots, Q_k & (\textit{Natural join})
\end{array}
$$

In addition we assume standard syntactic safety *conditions to be satisfied by the queries*[2]*.*

Note that the only *apriori interpreted* operation in the language is equality. All other classes of objects, including built-in classes (relations) are modeled using primitive concepts. Therefore, we have primitive concepts that model all the usual built-in relations, e.g., LESS for linear order, PLUS for addition, etc.

It is easy to see that the language captures all *conjunctive queries* under duplicate semantics. All other common constructs can be translated to queries formulated in the above fragment and are considered to be mere syntactic sugar. In particular:

$$
\begin{array}{l}
Q\;\texttt{where}\; E \;\texttt{and}\; E' \ldots \; = \texttt{from}\; Q, E, E', \ldots \\
Q\;\texttt{where exists}\; Q' \quad = \texttt{from}\; Q, (\texttt{elim select}\; V Q'),\ \text{for } V \text{ parameters of } Q' \\
\texttt{select distinct}\; V Q = \texttt{elim select}\; V Q
\end{array}
$$

The formal semantics of queries is defined with respect to a database I as follows:

Definition 6 (Semantics of Queries) *We define the semantic function* $[\![.]\!]$ *that maps queries to functions from databases* I *to bags of tuples as follows:*

$$
\begin{array}{rl}
[\![C \;\texttt{as}\; A]\!]I = & \{\!|\langle A : v\rangle : v \in (C)^I|\!\} \\
[\![A.\mathsf{Pf}_1 = B.\mathsf{Pf}_2]\!]I = & \{\!|\langle A : v, B : w\rangle : v, w \in \mathsf{Dom}, (\mathsf{Pf}_1)^I(v) = (\mathsf{Pf}_2)^I(w)|\!\} \\
[\![\texttt{select}\; A_1, \ldots, A_n\; Q]\!]I = & \{\!|\langle A_1 : v@A_1, \ldots, A_n : v@A_n\rangle : v \in [\![Q]\!]I|\!\} \\
[\![\texttt{elim}\; Q]\!]I = & \{v : v \in [\![Q]\!]I\} \\
[\![\texttt{from}\; Q_1, \ldots, Q_k]\!]I = & [\![Q_1]\!]I \bowtie \ldots \bowtie [\![Q_k]\!]I
\end{array}
$$

where $\{\!|\langle A_i : v_i\rangle|\!\}$ *denotes a bag of tuples and* $v@A_i$ *denotes the value associated with* A_i *in the tuple* v*; we require the attribute labels* A_i *to be the same for each tuple in the bag and for each individual tuple to form a set.*

---

[2] For our results to hold we can even allow infinite bags as long as every tuple is duplicated only finitely many times.

SQL Queries. As in the case of SQL tables mapping directly to primitive concepts, conjunctive queries in SQL (the `SELECT-FROM-WHERE` blocks) map directly to our object relational query language. The semantics of SQL maps immediately to the semantics presented in Definition 6. The only difference is that, for base tables, SQL retrieves values of all declared attributes in addition to the "tuple id", and thus in equality constraints the already retrieved values are simply compared. (Recall that all path functions in SQL are of the form $A.B$ where A is a tuple variable and B an attribute.)

From this point of view, it is interesting to observe that for SQL queries *tuple identifiers* are never used during query evaluation and therefore do not have to be stored in the database or manipulated by the queries. This approach is similar to introducing *virtual* attributes (record ids) [11, 20].

3 Reasoning about Duplicate Elimination Operators

This section presents the first result: a technique that allows us to remove subqueries (base classes) from the scope of a duplicate elimination operator. The condition of the rewrite rule presents a complete characterization of the situations in which the rewrite is valid.

To simplify the exposition, we first show that every object relational conjunctive query can be written in a *normal form*:

Lemma 7 (Normal Form) *Let Q be an arbitrary object relational conjunctive query. Then there is an equivalent query with the form*

$$\begin{array}{l} \texttt{select}\ V \\ \texttt{from}\ C_1\ \texttt{as}\ A_1, \ldots, C_m\ \texttt{as}\ A_m, (\texttt{elim} \\ \qquad\qquad\qquad\qquad\qquad\qquad \texttt{select}\ W \\ \qquad\qquad\qquad\qquad\qquad\qquad \texttt{from}\ C_{m+1}\ \texttt{as}\ A_{m+1}, \ldots, C_n\ \texttt{as}\ A_n, R) \end{array}$$

where R is a list of equalities of the form $A_i.\mathsf{Pf}_1 = A_j.\mathsf{Pf}_2$, and V and W sets of attribute names that appear in the signatures of the respective subqueries. Also, we assume that R references $A_{m+1}, \ldots, A_n$ and all attributes in W.

Proof. *By induction on the structure of Q using standard equivalence rules.*

The normal form yields a set of subsumption constraints in CFD as follows:

Definition 8 *Let Q be an object relational conjunctive query in normal form and C_Q a distinguished concept name. We define a set of subsumption constraints*

$$S_Q = \{C_Q < (A_1 : C_1), \ldots, C_Q < (A_n : C_n), C_Q < R\}$$

The constraints S_Q mimic the semantics of Q within the description logic. This way, we reduce reasoning about the query Q to reasoning about subsumption constraints in CFD. The ability to reason in CFD allows us to formulate the main result of this section:

Theorem 9 (Duplicate Elimination Reduction) *Let S be a database schema. Then for all databases I such that $I \models S$ the query*

$$
\begin{array}{lll}
Q : \texttt{select}\ V & & \\
\quad \texttt{from} & C_1\ \texttt{as}\ A_1, \ldots, C_m\ \texttt{as}\ A_m, (\texttt{elim} & \\
& \texttt{select}\ W & \\
& \texttt{from}\ C_{m+1}\ \texttt{as}\ A_{m+1}, \ldots, C_n\ \texttt{as}\ A_n, R) &
\end{array}
$$

is semantically equivalent to

$$
\begin{array}{lll}
Q' : \texttt{select}\ V & & \\
\quad \texttt{from} & C_1\ \texttt{as}\ A_1, \ldots, C_{m+1}\ \texttt{as}\ A_{m+1}, (\texttt{elim} & \\
& \texttt{select}\ W \cup \{A_{m+1}\} & \\
& \texttt{from}\ C_{m+2}\ \texttt{as}\ A_{m+2}, \ldots, C_n\ \texttt{as}\ A_n, R) &
\end{array}
$$

if and only if $S \cup S_Q \models C_Q < C_Q(A \cup W \to A_{m+1})$ where $A = \{A_1, \ldots, A_m\}$.

Proof. (sketch)
"$\Rightarrow$": Assume that $S \cup S_Q \not\models C_Q < C_Q(A \cup W \to A_{m+1})$. Then there must be an interpretation I such that $v, w \in (C_Q)^I$ and $\mathcal{F}_{A_i}(v) = \mathcal{F}_{A_i}(w)$ for $A_i \in A \cup W$, but where $\mathcal{F}_{A_{m+1}}(v) \neq \mathcal{F}_{A_{m+1}}(w)$. In addition, v and w satisfy all equational constraints in R. Therefore, using the definition of semantics for the outer `from` clauses of Q and Q' we have

- a single tuple $\langle A_i : \mathcal{F}_{A_i}(v)\rangle$ for Q, but
- at least the two tuples $\langle A_i : \mathcal{F}_{A_i}(v), A_{m+1} : \mathcal{F}_{A_{m+1}}(v)\rangle$
  and $\langle A_i : \mathcal{F}_{A_i}(w), A_{m+1} : \mathcal{F}_{A_{m+1}}(w)\rangle$ for Q';

This fact is preserved by the final duplicate-preserving projection and thus we have $[\![Q]\!]I \neq [\![Q']\!]I$.
"$\Leftarrow$": Conversely, $[\![Q]\!]I \neq [\![Q']\!]I$ is only possible if, similarly to the previous case, the interpretation of Q's outer `from` contains a single tuple while the interpretation of Q' contains two tuples that differ only in the value of A_{m+1}. However, these two tuples can be used to define an interpretation I with exactly two objects in $(C_Q)^I$ that agree on all attributes in $A \cup W$ but disagree on A_{m+1}. As these two objects also satisfy all the other subsumption constraints in S_Q we have $S \cup S_Q \not\models C_Q < C_Q(A \cup W \to A_{m+1})$. □

The theorem states a sufficient and necessary condition needed to move a conjunct (C_{m+1} `as` A_{m+1}) from the scope of the duplicate elimination operator (`elim`) into the outer `from` clause (and vice versa). Let us illustrate this on the query from our introductory Example 1:

Example 10 First, the original query from Example 1 is transformed to the normal form (after replacing syntactic sugar):

```
select :P1, :P2, Name
from elim (
     select :P1, :P2, Name
     from STUDENT as S, TAKES as T, COURSE as C,
          S = T.S, C.Time = T.C.Time, C.Inst.In.Name = :P1,
          C.Num = :P2, Name = S.Name )
```

Note that parameters `:P1` and `:P2` are converted to "result" attributes on the normalized query (using the standard observation that there is no difference between these). The *query schema* S_Q is defined as follows:

```
CQ < (S: STUDENT), (T: TAKES), (C: COURSE),
     (P1: STRING), (P2: INT), (Name: STRING),
     S = T.S, C.Time = T.C.Time, C.Inst.In.Name = :P1,
     C.Num = :P2, Name = S.Name
```

Now we use our rewriting rule from Theorem 9 to "move" the reference to `COURSE` out of the scope of the `elim` operator. The necessary condition is $S \cup S_Q \models$ `CQ < CQ(Name,:P1,:P2 -> C)`; (which is true as we shall see in Section 4). Therefore the query is equivalent to

```
select :P1, :P2, Name
from COURSE as C, elim (
     select C, :P1, :P2, Name
     from STUDENT as S, TAKES as T
          S = T.S, C.Time = T.C.Time, C.Inst.In.Name = :P1,
          C.Num = :P2, Name = S.Name )
```

A sequence of two additional applications of the duplicate elimination rewrite (that check for $S \cup S_Q \models$ `CQ < CQ(Name,:P1,:P2,C -> T)` to move the `TAKES as T` subquery and $S \cup S_Q \models$ `CQ < CQ(Name,:P1,:P2,C,T -> S)` for `STUDENT as S`) results in the following query:

```
select :P1, :P2, Name
from COURSE as C, TAKES as T, STUDENT as S, elim (
     select C, T, S, :P1, :P2, Name
     from S = T.S, C.Time = T.C.Time, C.Inst.In.Name = :P1,
          C.Num = :P2, Name = S.Name )
```

Note that the set of subsumption constraints S_Q is the same for each of the three steps. In Section 4 we see that the decision procedure can take advantage of this fact and reuse internal data structures to answer the second and third subsumption questions. Using standard simplification that commutes equalities with the duplicate elimination operation we obtain the query we desired in Example 1.

The rewriting rule (Theorem 9) can be integrated with existing query optimizers in two ways:

1. By "moving" an atomic query out of the scope of the `elim` operator (`DISTINCT` and `EXISTS` in SQL), we expand the search space of a join-ordering optimizer. If we manage to move all class references out of the scope of the `elim` operator (as in Example 10), we can completely remove the duplicate elimination operation from the query.
2. Conversely, we can move atomic subqueries into the scope of the `elim` operator. This is especially important in cases where the moved class reference does not contribute values to tuples in the result of the query. In this case the duplicate elimination operation can be replaced with a single index lookup (similarly to processing an `EXISTS` *clause* in SQL).

4 A Decision Procedure for CFD

Our overall algorithm works with *description graphs*, first introduced in [3] and modified in [4] to enable reasoning about simpler forms of uniqueness constraints that resembled (possibly asymmetric) functional dependencies. The algorithm presented here also incorporates an optimization that avoids copying entire description graphs when deducing subsumption by symmetric uniqueness constraints, a property always satisfied when reasoning about duplicate elimination on queries.

Definition 11 *A description graph G is a triple (N, E, n) consisting of a set N of nodes, a bag E of directed edges labeled with either primitive attribute names or* **Id***, and a distinguished node $n \in N$. Each node $n' \in N$ has three labels: a finite set $\mathsf{DS}(n')$ of concept descriptions, a finite set $\mathsf{PF}(n')$ of attribute descriptions, and a finite set $\mathsf{Fired}(n')$ of subsumption constraints that have already "fired" on n'. (Unless specified otherwise, we will assume that each of the three labels is initially empty for any newly created node.) Elements in E are written as triples (n_1, A, n_2) where n_1 and n_2 are nodes and A is either* **Id** *or a primitive attribute name.*

Intuitively, a description graph may be viewed as a partial database in which nodes correspond to hypothetical or prototypical objects that belong to descriptions in their DS labels, while edges correspond to (some of) their attribute values. Given a database schema S and subsumption constraint $C < D$, the decision procedure constructs this partial database by first creating an initial graph with a single node containing C in its DS label and then invoking a chase-like procedure EXP to create a normalized form for the graph that makes explicit additional constraints that are implicit in S. This normalized form represents a pattern that must always match the structure of *any* C object. The final step of the decision procedure calls a boolean function $\mathsf{SUBSUMES}$ to verify that the graph has the properties required by D.

A formal specification of these two procedures is given in the Appendix A. Full proofs of the following theorems are given in [14].

Theorem 12 (Soundness and Completeness) *Let S be a regular equation-free database schema and $C_1 < D_1$ and $C_2 < D_2$ arbitrary subsumption constraints for which C_1 does not occur in S. Then*

$$S \cup \{C_1 < D_1\} \models C_2 < D_2$$

if and only if

$$\mathsf{SUBSUMES}(D_2, \mathsf{EXP}(G, S \cup \{C_1 < D_1\}), S \cup \{C_1 < D_1\}, C_2, \mathit{true})$$

where description graph G has the form $(\{n\}, \{\}, n)$ in which $\mathsf{DS}(n) = \{C_2\}$ and $\mathsf{PF}(n) = \mathsf{Fired}(n) = \{\}$.

One of the desirable properties of our decision procedure is the fact that it is incremental; description graphs constructed by earlier invocations of EXP can

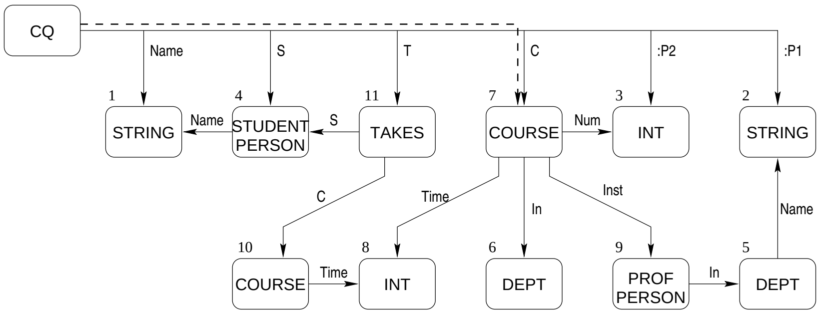

Fig. 3. A description graph for query schema S_Q.

be reused when the procedure is invoked on a sequence of similar membership problems. For example, a sequence of problems of the form $S \models C < D_i$ can always reuse the graph constructed by the first call to EXP, a circumstance that happens with our example of duplicate elimination (Example 10). Further ways of reusing the work are also possible and are explored in more detail in [14].

Example 13 Figure 3 illustrates the description graph produced by an invocation of SUBSUMES of the form

$$\mathsf{SUBSUMES}(\texttt{CQ(Name,:P1,:P2 -> C)}, \mathsf{EXP}((\{n\},\{\},n),S),S,\texttt{CQ},\mathit{true})$$

where $\mathsf{DS}(n) = \{\texttt{CQ}\}$ and S consists of the constraints in Figure 1 together with query schema S_Q. Note that only DS labels for nodes are indicated. The nodes are numbered according to the order in which updates E12, E13, E14 of EXP add **Id** to their PF labels. Since in particular the graph "ACCEPTS" attribute description `C` (note the dashed arc outgoing from the distinguished node), SUBSUMES returns true, thus enabling the first rewrite in Example 10.

Theorem 14 (Complexity) *Let S be a regular equation-free database schema with size k, and let $C_1 < D_1$ and $C_2 < D_2$ denote arbitrary subsumption constraints of total size m for which C_1 does not occur in S. Then there is an implementation of* EXP *and* SUBSUMES *such that an invocation of the form*

$$\mathsf{SUBSUMES}(D_2, \mathsf{EXP}(G, S \cup \{C_1 < D_1\}), S \cup \{C_1 < D_1\}, C_2, \mathit{true})$$

terminates in $O(mk)$ time if $C_2 < D_2$ has the form $C < C(\mathsf{Pf}_1, \ldots, \mathsf{Pf}_n \to \mathsf{Pf})$, in $O(fm^2k)$ time otherwise (f is the number of uniqueness constructs in D_2).

Note that both Theorem 12 and 14 only hold for cases in which database schemas are both regular and equation-free. It is well-known that the membership problem becomes undecidable if equations are allowed in a database schema. If the regularity property is relaxed, the membership problem for CFD becomes DEXPTIME complete [15]. Moreover, regular uniqueness constraints encompass the majority of practical situations (in particular, all of classical functional dependencies and primary keys).

5 Conclusion

The following list summarizes the contributions of the paper.

1. We have defined CFD, a description logic dialect sufficiently expressive to describe object relational database schemas and a wide class of integrity constraints on such schemas.
2. We have devised a rewriting rule for object relational conjunctive queries that completely characterizes situations in which a duplicate elimination operator can be removed from the query. We have shown that a necessary and sufficient condition maps to a question in CFD.
3. We presented an efficient (linear both in size of the schema and the query) procedure that answers the above question.

In addition we have presented a decision procedure for subsumption checking in CFD and discussed the impact of various restrictions we imposed on database schema (or lack of thereof) on the complexity of CFD's decision procedure.

Future work includes several possible directions (presented as conjectures here). Theorem 9 can be straightforwardly modified to handle aggregates in object relational queries (the `elim` operator can be viewed as a degenerate case of the group-by operator). The same theorem can also be easily adapted to handle rewrites that perform various kinds of order optimization, and to discovering *interesting orders* of tables [21]. The addition of disjunction to CFD and/or the query language still leads to a decidable theory, and query rewrites similar to the one presented in this paper are then possible. Moreover, we conjecture that there is a subset of database schemas with amenable computational properties (that includes a limited use of disjunction to capture wider classes of schemas than CFD).

References

1. Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases.* Addison-Wesley, 1995.
2. Alexander Borgida. Description logics in data management. *IEEE Transactions on Knowledge and Data Engineering*, 7(5):671–682, 1995.
3. Alexander Borgida and Peter F. Patel-Schneider. A Semantics and Complete Algorithm for Subsumption in the CLASSIC Description Logic. *J. of AI Research*, 1:277–308, 1994.
4. Alexander Borgida and Grant Weddell. Adding uniqueness constraints to description logics (preliminary report). In *International Conference on Deductive and Object-Oriented Databases*, pages 85–102, 1997.
5. Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Answering Queries Using Views in Description Logics. In *6th International Workshop on Knowledge Representation meets Databases (KRDB'99)*, pages 6–10, 1999.
6. 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.
7. Rick G. G. Cattell. ODMG-93: A Standard for Object-Oriented DBMSs. In *ACM SIGMOD International Conference on Management of Data*, page 480, 1994.

8. Rick G. G. Cattell, Douglas Barry, Dirk Bartels, Mark Berler, Jeff Eastman, Sophie Gamerman, David Jordan, Adam Springer, Henry Strickland, and Drew Wade. *The Object Database Standard: ODMG 2.0*. Morgan Kaufman, 1997.
9. Ashok K. Chandra and Philip M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *ACM Symposium on Theory of Computing*, pages 77–90, 1977.
10. Surajit Chaudhuri and Moshe Y. Vardi. Optimization of *real* conjunctive queries. In *ACM Symposium on Principles of Database Systems*, pages 59–70, 1993.
11. Umeshwar Dayal. Of Nests and Trees: A Unified Approach to Processing Queries That Contain Nested Subqueries, Aggregates, and Quantifiers. In *13th International Conference on Very Large Data Bases*, pages 197–208, 1987.
12. Parke Godfrey, John Grant, Jarek Gryz, and Jack Minker. Integrity constraints: Semantics and applications. In Jan Chomicki and Gunter Saake, editors, *Logics for Databases and Information Systems*, chapter 9. Kluwer, 1998.
13. ISO. Database Language SQL. ISO/IEC 9075:1992, International Organization for Standardization, 1992.
14. Vitaliy L. Khizder. *Uniqueness Constraints in Object-Relational Databases and Description Logics*. PhD thesis, University of Waterloo, 1999.
15. Vitaliy L. Khizder, David Toman, and Grant Weddell. On Decidability and Complexity of Description Logics with Uniquwness Constraints. Technical report, Dept. of Computer Science, University of Waterloo, 2000. (submitted).
16. Alon Y. Levy, Inderpal Singh Mumick, and Yehoshua Sagiv. Query Optimization by Predicate Move-Around. In *20th International Conference on Very Large Data Bases*, pages 96–107, 1994.
17. J. Melton and A. R. Simon. *Understanding the new SQL: A Complete Guide*. Morgan Kaufmann Publishers, 1993.
18. Inderpal Singh Mumick, Hamid Pirahesh, and Raghu Ramakrishnan. The Magic of Duplicates and Aggregates. In *16th International Conference on Very Large Data Bases*, pages 264–277, 1990.
19. Glenn N. Paulley and Per-Åke Larson. Exploiting uniqueness in query optimization. In *10th International Conference on Data Engineering*, pages 68–79, 1994.
20. Hamid Pirahesh, Joseph M. Hellerstein, and Waqar Hasan. Extensible/Rule Based Query Rewrite Optimization in Starburst. In *ACM SIGMOD International Conference on Management of Data*, pages 39–48, 1992.
21. Patricia G. Selinger, Morton M. Astrahan, Donald D. Chamberlin, Raymond A. Lorie, and Thomas G. Price. Access Path Selection in a Relational Database Management System. In *ACM SIGMOD International Conference on Management of Data*, pages 23–34, 1979.
22. Praveen Seshadri, Hamid Pirahesh, and T. Y. Cliff Leung. Complex Query Decorrelation. In *12th International Conference on Data Engineering*, pages 450–458, 1996.
23. Martin F. van Bommel and Grant Weddell. Reasoning About Equations and Functional Dependencies on Complex Objects. *IEEE Transactions on Knowledge and Data Engineering*, 6(3):455–469, 1994.
24. Grant Weddell. Reasoning about Functional Dependencies Generalized for Semantic Data Models. *ACM Transactions on Database Systems*, 17(1):32–64, 1992.

A EXP and SUBSUMES

A specification of procedure EXP and function SUBSUMES are given below in a production system style presumed by the full proofs of Theorems 12 and 14 that can be found in [14]. The actual implementation of these procedures is event driven, and makes use of indices on a given database schema.

procedure $\mathsf{EXP}(G, S)$

Input: description graph $G = (N, E, n)$; set of subsumption constraints S.

Exhaustively apply the following updates to G in the order presented.

E1. If $e = (n', \mathbf{Id}, n') \in E$, then $E := E - \{e\}$.

E2. If $e = (n', \mathbf{Id}, n) \in E$, then $E := E - \{e\} \cup \{(n, \mathbf{Id}, n')\}$[3].

E3. If $e = (n_1, \mathbf{Id}, n_2) \in E$, then
(a) $\mathsf{DS}(n_1) := \mathsf{DS}(n_1) \cup \mathsf{DS}(n_2)$.
(b) $\mathsf{PF}(n_1) := \mathsf{PF}(n_1) \cup \mathsf{PF}(n_2)$.
(c) $\mathsf{Fired}(n_1) := \mathsf{Fired}(n_1) \cup \mathsf{Fired}(n_2)$.
(d) For each $e' = (n', A, n_2) \in E$ do $E := E - \{e'\} \cup \{(n', A, n_1)\}$.
(e) For each $e' = (n_2, A, n') \in E$ do $E := E - \{e'\} \cup \{(n_1, A, n')\}$.
(f) $E := E - \{e\}$; $N := N - \{n_2\}$.

E4. If $e_1 = (n_1, A, n_2), e_2 = (n_1, A, n_3) \in E$,
then $E := E - \{e_2\} \cup \{(n_2, \mathbf{Id}, n_3)\}$.

E5. If $n' \in N$ s.t. $D = (D_1, \ldots, D_n) \in \mathsf{DS}(n')$,
then $\mathsf{DS}(n') := \mathsf{DS}(n') - \{D\} \cup \{D_1, \ldots, D_n\}$.

E6. If $n_1, n_2 \in N$ s.t. $D = (A.\mathsf{Pf} : D') \in \mathsf{DS}(n_1)$
and $(n_1, A, n_2) \in E$, then
(a) $\mathsf{DS}(n_1) := \mathsf{DS}(n_1) - \{D\}$.
(b) $\mathsf{DS}(n_2) := \mathsf{DS}(n_2) \cup \{(\mathsf{Pf} : D')\}$.

E7. If $n' \in N$ s.t. $D = (A.\mathsf{Pf} : D') \in \mathsf{DS}(n')$
and D' contains an equational constraint,
then invoke $\mathsf{FIND}(n', A.\mathbf{Id}, G)$.

E8. If $n' \in N$ s.t. $D = (\mathbf{Id} : D') \in \mathsf{DS}(n')$,
then $\mathsf{DS}(n') := \mathsf{DS}(n') - \{D\} \cup \{D'\}$.

E9. If $n' \in N$ s.t. $D = (\mathsf{Pf}_1 = \mathsf{Pf}_2) \in \mathsf{DS}(n')$, then
(a) $\mathsf{DS}(n') := \mathsf{DS}(n') - \{D\}$.
(b) Add $(\mathsf{FIND}(n', \mathsf{Pf}_1, G), \mathbf{Id}, \mathsf{FIND}(n', \mathsf{Pf}_2, G))$ to E.

E10. If $n_1, n_2 \in N$ s.t.:
(1) $n_1 \neq n_2$,
(2) there $D = C(\mathsf{Pf}_1, \ldots, \mathsf{Pf}_m \to \mathsf{Pf}) \in \mathsf{DS}(n_1)$,
(3) $C \in \mathsf{DS}(n_2)$,
(4) for each $1 \leq i \leq m : \mathsf{AGREES}(n_1, \mathsf{Pf}_i, n_2, G)$, and
(5) $\neg\mathsf{AGREES}(n_1, \mathsf{Pf}, n_2, G)$,
then add $(\mathsf{FIND}(n_1, \mathsf{Pf}, G), \mathbf{Id}, \mathsf{FIND}(n_2, \mathsf{Pf}, G))$ to E.

E11. If $n' \in N$ s.t. $(C < D) \in S - \mathsf{Fired}(n')$ and $C \in \mathsf{DS}(n')$, then
(a) $\mathsf{Fired}(n') := \mathsf{Fired}(n') \cup \{C < D\}$.
(b) $\mathsf{DS}(n') := \mathsf{DS}(n') \cup \{D\}$.

E12. If $n' \in N$ s.t. $\mathsf{Pf}_1 = A.\mathsf{Pf}_2 \in \mathsf{PF}(n')$, then
(a) $\mathsf{PF}(n') := \mathsf{PF}(n') - \{\mathsf{Pf}_1\}$.
(b) $\mathsf{PF}(\mathsf{FIND}(n', A.\mathbf{Id}, G)) := \{\mathsf{Pf}_2\}$.

E13. If $n' \in N$ s.t.:
(1) $C \in \mathsf{DS}(n')$,
(2) $C(\mathsf{Pf}_1, \ldots, \mathsf{Pf}_m \to \mathsf{Pf}) \in \mathsf{DS}(n')$,
(3) for each $1 \leq i \leq m : \mathsf{ACCEPTS}(n', \mathsf{Pf}_i, G)$, and
(4) $\neg\mathsf{ACCEPTS}(n', \mathsf{Pf}, G)$,
then $\mathsf{PF}(n') := \mathsf{PF}(n') \cup \{\mathsf{Pf}\}$.

E14. If $(n_1, A, n_2) \in E$ s.t. $\mathbf{Id} \in \mathsf{PF}(n_1)$ and $\mathbf{Id} \notin \mathsf{PF}(n_2)$,
then $\mathsf{PF}(n_2) := \mathsf{PF}(n_2) \cup \{\mathbf{Id}\}$.

[3] Recall that n is the distinguished node in G

function $\mathsf{SUBSUMES}(D, G, S, C, T)$**: Boolean**

Input: concept description D; description graph $G = (N, E, n)$; set of subsumption constraints S, primitive concept C and Boolean T.

Branch to the relevant case depending on the form of D.

Case C: Return $C \in \mathsf{DS}(n)$.

Case $(D_1, ..., D_m)$:
Return true iff $\mathsf{SUBSUMES}(D_i, G, S, C, T)$ for all $1 \leq i \leq m$.

Case $(\mathbf{Id} : D')$: Return $\mathsf{SUBSUMES}(D', G, S, C, T)$.

Case $(A.\mathsf{Pf} : D')$.
Return $\mathsf{SUBSUMES}((\mathsf{Pf} : D'), \mathsf{EXP}((N, E, \mathsf{FIND}(n, A.\mathbf{Id}, G)), S), S, C, false)$.

Case $\mathsf{Pf}_1 = \mathsf{Pf}_2$:
(a) $\mathsf{FIND}(n, \mathsf{Pf}_1, G)$, $\mathsf{FIND}(n, \mathsf{Pf}_2, G)$.
(b) $\mathsf{EXP}(G, S)$.
(c) Return $\mathsf{FIND}(n, \mathsf{Pf}_1, G) = \mathsf{FIND}(n, \mathsf{Pf}_2, G)$.

Case $C(\mathsf{Pf}_1, ..., \mathsf{Pf}_m \to \mathsf{Pf})$ where T:
(a) For each $n' \in N : \mathsf{PF}(n') := \{\}$.
(b) $\mathsf{PF}(n) := \{\mathsf{Pf}_1, ..., \mathsf{Pf}_m\}$.
(c) Invoke $\mathsf{EXP}((N, E, n), S)$.
(d) Return $\mathsf{ACCEPTS}(n, \mathsf{Pf}, G)$.

Case $C'(\mathsf{Pf}_1, \ldots, \mathsf{Pf}_m \to \mathsf{Pf})$ where $C' \neq C$ or $\neg T$:
(a) Create a copy $G' = (N', E', n')$ of G.
(b) Add a new node n to N' and a new edge (n, A, n') to E'.
(c) Add $(B : C')$ to $\mathsf{DS}(n)$.
(d) For each $1 \leq i \leq m$: add $(A.\mathsf{Pf}_i = B.\mathsf{Pf}_i)$ to $\mathsf{DS}(n)$.
(e) Invoke $\mathsf{EXP}(G', S)$.
(f) Return $\mathsf{FIND}(n, A.\mathsf{Pf}, G') = \mathsf{FIND}(n, B.\mathsf{Pf}, G')$.

Case ANY: Return true.

function $\mathsf{FIND}(n, \mathsf{Pf}, G)$: **node**

Input: description graph $G = (N, E, n'')$; node $n \in N$; attribute description Pf.

Returns the node at the end of the path Pf from a node n in G, potentially creating any missing nodes and edges to ensure such a path exists.

function $\mathsf{AGREES}(n_1, \mathsf{Pf}, n_2, G)$**: Boolean**

Input: description graph $G = (N, E, n)$; nodes $n_1, n_2 \in N$; attribute description Pf.

Returns true if there is a node $n' \in G$ such that n' is reachable from both n_1 and n_2 by the same prefix of Pf.

function $\mathsf{ACCEPTS}(n_1, \mathsf{Pf}, G)$**: Boolean**

Input: description graph $G = (N, E, n)$; node $n_1 \in N$; attribute description Pf.

Returns true if **Id** occurs in the PF label of any node on the Pf path starting from node n_1 in G.