

An Equational Chase for Path-Conjunctive Queries, Constraints, and Views

Lucian Popa and Val Tannen

University of Pennsylvania,
Department of Computer and Information Science,
200 South 33rd Street, Philadelphia, PA 19104, USA
{lpopa, val}@saul.cis.upenn.edu

Abstract. We consider the class of *path-conjunctive* queries and constraints (dependencies) defined over complex values with dictionaries. This class includes the relational conjunctive queries and embedded dependencies, as well as many interesting examples of complex value and oodb queries and integrity constraints. We show that some important classical results on containment, dependency implication, and chasing extend and generalize to this class.

1 Motivation

We are interested in distributed, mediator-based systems [Wie92] with multiple layers of nodes implementing mediated views (unmaterialized or only partially materialized) that integrate heterogenous data sources. Most of the queries that flow between the nodes of such a system are not formulated by a user but are instead generated automatically by composition with views and decomposition among multiple sources. Unoptimized, this process causes queries to quickly snowball into forms with superfluous computations. Even more importantly, without additional optimizations this process ignores intra- and inter-data source integrity constraints¹.

It has been recognized for some time that exploiting integrity constraints (so-called semantic optimization) plays a crucial role in oodbs [CD92] and in integrating heterogenous data sources [QR95,LSK95]. Relational database theory has studied extensively such issues [Mai83,Ull89,AHV95] but in the recent literature the use of constraints in optimization has been limited to special cases (see the related work in section 7). In contrast, we propose here a foundation for a systematic and quite general approach encompassing the old relational theory and a significant class of non-relational queries, constraints and views. A novel property of our approach, one that we hope to exploit in relation to rule-based optimization as in [CZ96], is that optimizing under constraints or deriving other constraints can be done *within* the equational theory of our internal framework by rewriting with the constraints themselves.

¹ In this paper, we use the terms *constraints* and *dependencies* interchangeably.

In this opening section we plan to show two motivating examples: one illustrating the important effect of constraints on query optimization, the other—coming from the announced paradigm of distributed mediator-base systems—concerned with deriving constraints that hold in views. The novel ideas to look for in these examples are (1) the use of the constraints themselves as “rewrite” rules (what we call the *equational chase*) (2) the ability to “compose” views with constraints yielding other constraints that can be checked directly (3) the ease with which we handle the heterogenous nature of the examples’ schema (figure 1) which features a relation interacting with a class.

Both examples are resolved with the general techniques that we develop in this paper. For simplicity these examples are shown in the syntax of the ODMG [Cat96] proposal although the actual method is developed and justified primarily for our more general internal framework (see section 3). Consider the schema in figure 1. It is written following mostly the syntax of ODL, the data definition language of ODMG, extended with referential integrity (foreign key) constraints in the style of data definition in SQL. It consists of a class **Dept** whose objects represent departments, with name, manager name, and **DProjs**, the set of names of all the projects done in the department. It also consists of a relation **Proj** whose tuples represent projects, with name, customer name, and **PDept**, the name of the department in which the project is done.

```

Proj : set<struct{
    string PName;
    string CustName;
    string PDept;>
primary key PName;
foreign key PDept
    references Dept::DName;
relationship PDept
    inverse Dept::DProjs;

class Dept
    (extent depts key DName){
    attribute string DName;
    relationship set<string> DProjs
        inverse Proj(PDept);
    attribute string MgrName;}
foreign key DProjs
    references Proj(PName);

```

Fig.1. The Proj-Dept schema, expressed in extended ODMG

The meaning of the referential integrity (RICs), inverse relationship, and key constraints specified in the schema is expressed by the following logical statements.

$$\begin{aligned}
 \text{(RIC1)} \quad & \forall(d \in \text{depts}) \forall(s \in d.\text{DProjs}) \exists(p \in \text{Proj}) \ s = p.\text{PName} \\
 \text{(RIC2)} \quad & \forall(p \in \text{Proj}) \exists(d \in \text{depts}) \ p.\text{PDept} = d.\text{DName} \\
 \text{(INV1)} \quad & \forall(d \in \text{depts}) \forall(s \in d.\text{DProjs}) \forall(p \in \text{Proj}) \\
 & \qquad \qquad \qquad s = p.\text{PName} \Rightarrow p.\text{PDept} = d.\text{DName} \\
 \text{(INV2)} \quad & \forall(p \in \text{Proj}) \forall(d \in \text{depts}) \\
 & \qquad \qquad \qquad p.\text{PDept} = d.\text{DName} \Rightarrow \exists(s \in d.\text{DProjs}) \ p.\text{PName} = s
 \end{aligned}$$

(KEY1) $\forall(d \in \text{depts}) \forall(d' \in \text{depts}) d.\text{DName} = d'.\text{DName} \Rightarrow d = d'$

(KEY2) $\forall(p \in \text{Proj}) \forall(p' \in \text{Proj}) p.\text{PName} = p'.\text{PName} \Rightarrow p = p'$

A query equivalence. In the presence of the constraints above, the following OQL² query:

```
define ABC_DP as select distinct struct(PN: s, DN: d.DName)
  from depts d, d.DProjs s, Proj p
  where s = p.PName and p.CustName = "ABC"
```

is equivalent to the (likely better) query:

```
define ABC_P as select distinct struct(PN: p.PName, DN: p.PDept)
  from Proj p
  where p.CustName = "ABC"
```

Because it involves both a class and a relation, and because it doesn't hold if the constraints are absent, the equivalence between `ABC_DP` and `ABC_P` does not appear to fit within previously proposed optimization frameworks. We will see that the equational chase method that we develop in this paper can be used to obtain this equivalence. Each of the constraints determines a semantics-preserving transformation rule. Specifically, one step of chasing with a constraint of the form

$$\forall(r_1 \in R_1) \cdots \forall(r_m \in R_m) [B_1 \Rightarrow \exists(s_1 \in S_1) \cdots \exists(s_n \in S_n) B_2]$$

produces the transformation

```
select distinct E
  from R_1 r_1, ..., R_m r_m
  where B_1 and C
   $\stackrel{\equiv}{\mapsto}$ 
  select distinct E
  from R_1 r_1, ..., R_m r_m, S_1 s_1, ..., S_n s_n
  where B_1 and B_2 and C
```

This transformation may seem somewhat mysterious, but we shall see that it can be justified by *equational* rewriting with the constraint itself put in an equational form and with the (idemloop) law (see section 3). We shall also see that when restricted to the relational model, the method amounts exactly to the *chase* [BV84]. Anyway, chasing `ABC_DP` with (INV1) gives

```
select distinct struct(PN: s, DN: d.DName)
  from depts d, d.DProjs s, Proj p
  where s = p.PName and p.PDept = d.DName and p.CustName = "ABC"
```

while chasing `ABC_P` with (RIC2) and then with (INV2) gives

```
select distinct struct(PN: p.PName, DN: p.PDept)
  from Proj p, depts d, d.DProjs s
  where p.PDept = d.DName and p.PName = s and p.CustName = "ABC"
```

² OQL is ODMG's query language.

These two queries are clearly equivalent.

A constraint derivation for views. Consider again the **Proj-Dept** schema in figure 1 and the following view, which combines *unnesting*, *joining*, and *projecting*:

```
define UJP_View as
  select distinct
    struct(PN: p.PName, CN: p.CustName, MN: d.MgrName)
  from Proj p, depts d, d.DProjs s,
  where s = p.PName
```

We will establish that the functional dependency $\text{PN} \rightarrow \text{MN}$ holds in this view, namely

$$\text{(FD)} \quad \forall(v \in \text{UJP_View}) \forall(v' \in \text{UJP_View}) v.\text{PN} = v'.\text{PN} \Rightarrow v.\text{MN} = v'.\text{MN}$$

Note that standard reasoning about functional dependencies is not applicable because of the nature of the view. Indeed we shall see that in addition to the key dependencies we will also use (INV1) to establish (FD).

We perform the derivation in two phases. First we *compose* (FD) with the view, obtaining a constraint on the original **Proj-Dept** schema. This is done by substituting the definition of **UJP_View** for the name **UJP_View** in (FD) and then rewriting the resulting constraint to the following form:

$$\begin{aligned} \text{(FD-RETRO)} \quad & \forall(p \in \text{Proj}) \forall(d \in \text{depts}) \forall(s \in d.\text{DProjs}) \\ & \forall(p' \in \text{Proj}) \forall(d' \in \text{depts}) \forall(s' \in d'.\text{DProjs}) \\ & s = p.\text{PName} \wedge s' = p'.\text{PName} \wedge p.\text{PName} = p'.\text{PName} \Rightarrow \\ & \quad d.\text{MgrName} = d'.\text{MgrName} \end{aligned}$$

Thus, (FD) holds in the view iff (FD-RETRO) holds in **Proj-Dept**. In the second phase we show that (FD-RETRO) follows from the constraints in the **Proj-Dept** schema. The equational chase method applies to constraints as well as to queries. Specifically, one step of chasing with a constraint of the form

$$\forall(r_1 \in R_1) \cdots \forall(r_m \in R_m) [B_1 \Rightarrow \exists(s_1 \in S_1) \cdots \exists(s_n \in S_n) B_2]$$

produces the transformation

$$\begin{aligned} & \forall(r_1 \in R_1) \cdots \forall(r_m \in R_m) [B_1 \Rightarrow \exists(t_1 \in T_1) \cdots \exists(t_p \in T_p) B_3] \\ & \quad \xrightarrow{\Leftrightarrow} \\ & \forall(r_1 \in R_1) \cdots \forall(r_m \in R_m) \forall(s_1 \in S_1) \cdots \forall(s_n \in S_n) \\ & \quad [B_1 \wedge B_2 \Rightarrow \exists(t_1 \in T_1) \cdots \exists(t_p \in T_p) B_3] \end{aligned}$$

Again, the rewriting that produces (FD-RETRO) and the transformation of constraints we just saw may seem somewhat mysterious, but they too have a

simple and clear justification in terms of equational rewriting (see section 3). Now, chasing (twice) with (INV1), then with (KEY2) and finally with (KEY1), transforms (FD-RETRO) into

$$\begin{aligned}
 \text{(TRIV)} \quad & \forall(p \in \text{Proj}) \quad \forall(d \in \text{depts}) \quad \forall(s \in d.\text{DProjs}) \\
 & \forall(p' \in \text{Proj}) \quad \forall(d' \in \text{depts}) \quad \forall(s' \in d.\text{DProjs}) \\
 & s = p.\text{PName} \wedge p.\text{PDept} = d.\text{DName} \wedge \\
 & s' = p'.\text{PName} \wedge p'.\text{PDept} = d'.\text{DName} \wedge \\
 & p.\text{PName} = p'.\text{PName} \wedge p = p' \wedge \\
 & \qquad \qquad \qquad d = d' \Rightarrow d.\text{MgrName} = d'.\text{MgrName}
 \end{aligned}$$

(Along the way, some simple facts about conjunction and equality came into play.) But (TRIV) is a constraint that holds in *all* instances, regardless of other constraints (such a constraint or dependency is traditionally called *trivial*). Therefore, (FD-RETRO) follows from (INV1, KEY2, KEY1).

How general is this method? The rest of the paper is concerned with proposing an answer to this question. We give first an overview, then we use our internal framework to justify the soundness of the method illustrated in the previous examples, and finally using *path-conjunctions* we identify a class of queries, constraints and views for which this method leads to decision procedures and complete equational reasoning.

2 Overview (of the rest of the paper)

In section 3 we present some aspects of our internal framework, called CoDi³. This is a language and equational theory that combines a treatment of dictionaries with our previous work [BBW92,BNTW95,LT97] on collections and aggregates using the theory of *monads*. In addition we use dictionaries (finite functions), which allow us to represent oodb schemas and queries. While here we focus on set-related queries, we show elsewhere⁴ that the (full) CoDi collection, aggregation and dictionary primitives suffice for implementing the quasi-totality of ODMG/ODL/OQL [Cat96]. Using boolean aggregates, CoDi can represent constraints like the ones we used in the examples in section 1 as equalities between boolean-valued queries. We also give the basic equational laws of CoDi while the rest of the axiomatization can be found in [PT98]. In section 3 we also show how that the transformations shown in section 1 correspond to precise rewrites in CoDi. This suggests the definition of a general notion of chase by rewriting, which connects dependencies to query equivalence (and therefore containment via intersection). This generalizes the relational chase of [ABU79,MMS79,BV84] for conjunctive queries (with equality) [CM77,ASU79] and embedded dependencies [Fag82]. In the same section we discuss composing dependencies with views.

³ From “Collections and Dictionaries”

⁴ “An OQL interface to the K2 system”, by J. Crabtree, S. Harker, and V. Tannen, forthcoming.

Our main results are in sections 4 and 5. In section 4 we exhibit a class of queries and dependencies on complex values with dictionaries called *path-conjunctive* (PC queries and embedded PC dependencies (EPCDs)) for which the methods illustrated in earlier sections are complete, and in certain cases decidable. Theorem 1 in section 4 extends and generalizes the containment decidability/NP result of [CM77]. Theorem 3 and corollary 1 in section 5 extend and generalize the corresponding results on the chase in [BV84]. Theorem 1 and theorem 3 also state that CoDi’s equational axiomatization is complete for deriving dependencies and containments, which extends and generalizes the corresponding completeness results on algebraic dependencies from [YP82,Abi83], although in a different equational theory. Proposition 1 which in our framework is almost “for free” immediately implies an extension and generalization of the corresponding SPC algebra [AHV95] result of [KP82] (but not the SPCU algebra result).

Due to the space restrictions, the proofs of the theorems stated here and some technical definitions are in the companion technical report [PT98]. This report also contains more results (that we summarize in section 6) and more examples. Related work and further investigations are in section 7.

3 Chase and view composition in CoDi

We want to “rewrite” queries using constraints and we want to “compose” views with constraints. But constraints are logical assertions and queries and views are functional expressions (at least when we use OQL, SQL, or various algebras). The reconciliation could be attempted within first-order logic using appropriate translations of queries and views. Since we deal with complex values (nested sets and records) and with oo classes, some kind of “flattening” encoding (eg. [LS97]) would be necessary. Here we take the opposite approach, by translating constraints into a functional algebra, CoDi where queries and views are comfortably represented. We shall see that the fragment of logic that we need for constraints also has a natural representation.

The three major constructs in our functional algebra are the following (given here with their semantics, for $S \stackrel{\text{def}}{=} \{a_1, \dots, a_n\}$):

$$\begin{aligned} \underline{\text{BigU}}(x \in S) R(x) &\stackrel{\text{def}}{=} \bigcup_{i=1}^n R(a_i) & \underline{\text{All}}(x \in S) B(x) &\stackrel{\text{def}}{=} \bigwedge_{i=1}^n B(a_i) \\ \underline{\text{Some}}(x \in S) B(x) &\stackrel{\text{def}}{=} \bigvee_{i=1}^n B(a_i) \end{aligned}$$

$R(x)$ and S are set expressions, i.e. with types of the form $\{\sigma\}$ and $B(x)$ is a boolean expression. The variable x is *bound* in these constructs (as in lambda abstraction) and we represent by $R(x)$ and $B(x)$ the fact that it may occur in R and B . $B(a)$, $R(a)$ are the results of substituting a for x . We shall use the generic notation Loop to stand for BigU, All, or Some and we shall consider mostly expressions of the form

$$\underline{\text{Loop}}(x \in S) \text{ if } B(x) \text{ then } E(x)$$

which is an abbreviation for

$$\underline{\text{Loop}}(x_1 \in S_1) \cdots \underline{\text{Loop}}(x_n \in S_n(x_1, \dots, x_{n-1})) \\ \underline{\text{if}} \ B(x_1, \dots, x_n) \ \underline{\text{then}} \ E(x_1, \dots, x_n) \ \underline{\text{else}} \ \underline{\text{null}}$$

where null is another generic notation⁵ standing for empty in a BigU, for true in an All and for false in a Some. We shall also use sng \overline{E} , denoting a singleton set. The syntax for tuples (records) is standard. We also use an equality test: eq(E_1, E_2) and boolean conjunction. As for expressive power (so far), note that BigU is the operation ext/ Φ of [BNTW95], shown there to have (with singleton and primitives for tuples and booleans) the expressive power of the relational algebra over flat relations and the expressive power of the nested relational algebra over complex objects.

Dictionaries We denote by $\sigma \times \tau$ the type of dictionaries (finite functions) with keys of type σ and entries of type τ . dom M denotes the set of keys (the *domain*) of the dictionary M . $K ! M$ denotes the entry of M corresponding to the key K (lookup!). This operation *fails* unless K is in dom M and we will take care to use it in contexts in which it is guaranteed not to fail. We model oo classes with extents using dictionaries whose keys are the oids. The extents become the domains of the dictionaries and the implicit oid dereferencing in OQL is translated by a lookup in the dictionary. For example, **depts** d and $d.\text{DProjs}$ s in **ABC_DP**, section 1, become $d \in \underline{\text{dom}} \text{Dept}$ and $s \in d ! \text{Dept.DProjs}$, see below.

Example. Referring to the schema in figure 1, the type specs translate in CoDi as

```
Proj : {(PName : string, CustName : string, PDept : string)}
Dept : Doid × > {DName : string, DProjs : {string}, MgrName : string}
```

the query **ABC_DP** translates in CoDi as the expression

$$\underline{\text{BigU}}(d \in \underline{\text{dom}} \text{Dept}) \underline{\text{BigU}}(s \in d ! \text{Dept.DProjs}) \underline{\text{BigU}}(p \in \text{Proj}) \\ \underline{\text{if}} \ \underline{\text{eq}}(s, p.\text{PName}) \ \underline{\text{and}} \ \underline{\text{eq}}(p.\text{CustName}, \text{"ABC"}) \ \underline{\text{then}} \ \underline{\text{sng}}(\text{PN} : s, \text{DN} : d ! \text{Dept.DName})$$

and the constraint (INV2) translates in CoDi as the *equation*

$$\underline{\text{All}}(p \in \text{Proj}) \underline{\text{All}}(d \in \underline{\text{dom}} \text{Dept}) \\ \underline{\text{if}} \ \underline{\text{eq}}(p.\text{PDept}, d ! \text{Dept.DName}) \ \underline{\text{then}} \ \underline{\text{Some}}(s \in d ! \text{Dept.DProjs}) \ \underline{\text{eq}}(p.\text{PName}, s) \\ = \underline{\text{true}}$$

⁵ These generic notations are not *ad-hoc*. We show in [LT97] that they correspond to *monad algebras* which here are structures associated with the set monad and are “enriched” with a nullary operation.

Expressing constraints this way will allow us to achieve both our objectives: rewriting queries (and constraints!) with constraints and composing views with constraints. All our manipulations are justified by CoDi's *equivalence laws* and we show the basic ones in figure 2. Some of these laws are derived from the theory of monads and monad algebras and the extensions we worked out in [BNTW95,LT97]. The important new axiom is (idemloop). Its validity is based on the fact that x does not occur in E and that union, conjunction, and disjunction are all idempotent operations. The rest of the equational axiomatization is in [PT98]. Some laws, such as (commute) and (from [PT98]) the laws governing eq, the conditional, and, and the congruence laws are used rather routinely to rearrange expressions in preparation for a more substantial rewrite step. When describing rewritings we will often omit mentioning these ubiquitous rearrangements.

(sng)	$\text{BigU}(x \in S) \text{sng}(x) = S$
(monad- β)	$\text{Loop}(x \in \text{sng}(E)) E'(x) = E'(E)$
(assoc)	$\text{Loop}(x \in (\text{BigU}(y \in R) S(y))) E(x) = \text{Loop}(y \in R) (\text{Loop}(x \in S(y)) E(x))$
(null)	$\text{Loop}(x \in \text{empty}) E(x) = \text{null}$
(cond-loop)	$\text{Loop}(x \in S) \text{if } B \text{ then } E(x) = \text{if } B \text{ then } \text{Loop}(x \in S) E(x)$
(commute)	$\text{Loop}(x \in R) \text{Loop}(y \in S) E(x, y) = \text{Loop}(y \in S) \text{Loop}(x \in R) E(x, y)$
(idemloop)	$\text{Loop}(x \in S) \text{if } B(x) \text{ then } E = \text{if } \text{Some}(x \in S) B(x) \text{ then } E$

Fig. 2. Equivalence laws

Example. We show how we obtained (FD-RETRO) from (FD) (see section 1). First, we express UJP_View in CoDi:

$$\frac{\text{BigU}(p \in \text{Proj}) \text{BigU}(d \in \text{domDept}) \text{BigU}(s \in d! \text{Dept.DProjs})}{\text{if } \text{eq}(s, p.\text{PName}) \text{ then } \text{sng}(\text{PN} : p.\text{PName}, \text{CN} : p.\text{CustName}, \text{MN} : d! \text{Dept.MgrName})}$$

Translating (FD) into CoDi (as we did for (INV2)), then substituting the first occurrence of UJP_View with its definition and using (assoc) we obtain:

$$\begin{aligned} & \text{All}(p \in \text{Proj}) \text{All}(d \in \text{domDept}) \text{All}(s \in d! \text{Dept.DProjs}) \\ & \text{All}(v \in \text{if } \text{eq}(s, p.\text{PName}) \text{ then } \text{sng}(\text{PN} : p.\text{PName}, \text{CN} : p.\text{CustName}, \text{MN} : d! \text{Dept.MgrName})) \\ & \text{All}(v' \in \text{UJP_View}) \text{if } \text{eq}(v.\text{PN}, v'.\text{PN}) \text{ then } \text{eq}(v.\text{MN}, v'.\text{MN}) \\ & \hspace{15em} = \text{true} \end{aligned}$$

After some manipulations of conditionals we apply, in sequence, (monad- β), (null) and (cond-loop), and obtain:

$$\begin{aligned} & \underline{\text{All}}(p \in \text{Proj}) \underline{\text{All}}(d \in \text{domDept}) \underline{\text{All}}(s \in d! \text{Dept.DProjs}) \underline{\text{All}}(v' \in \text{UJP_View}) \\ & \quad \underline{\text{if}} \underline{\text{eq}}(s, p.\text{PName}) \underline{\text{and}} \underline{\text{eq}}(p.\text{PName}, v'.\text{PN}) \underline{\text{then}} \underline{\text{eq}}(d! \text{Dept.MgrName}, v'.\text{MN}) \\ & = \underline{\text{true}} \end{aligned}$$

A similar rewriting in which the other occurrence of `UJP_View` is substituted with its definition yields the direct translation in CoDi of (FD-RETRO). Note that these transformations explain tableau view “replication” [KP82].

The form in which we have written the constraints turns out to be convenient if we want them rewritten but not if we want to rewrite *with* them. In order to replace equals by equals within the scope of `Loop(x ∈ S)` bindings, we will use a form of equations “guarded” by set membership conditions for the variables:

$$x_1 \in S_1, \dots, x_n \in S_n(x_1, \dots, x_{n-1}) \vdash E_1(x_1, \dots, x_n) = E_2(x_1, \dots, x_n)$$

The congruence rules for the `Loop` construct together with some additional rules that can be found in [PT98] allow us to rewrite the set membership conditions within the scope of `All` and vice-versa. Thus we have the following:

Lemma 1 (Two Forms for Constraints). *The following two equations are derivable from each other in CoDi’s equational theory:*

$$\begin{aligned} & \underline{\text{All}}(x \in S) \underline{\text{if}} B(x) \underline{\text{then}} C(x) = \underline{\text{true}} \\ & x \in S \vdash B(x) = B(x) \underline{\text{and}} C(x) \end{aligned}$$

The Equational Chase Step. Consider the constraint (d) and the expression G :

$$\begin{aligned} (d) \quad & x \in \mathbf{R}_1 \vdash B_1(x) = B_1(x) \underline{\text{and}} \underline{\text{Some}}(y \in \mathbf{R}_2(x)) B_2(x, y) \\ G \quad & \stackrel{\text{def}}{=} \underline{\text{Loop}}(x \in \mathbf{R}_1) \underline{\text{if}} B_1(x) \underline{\text{then}} E(x) \end{aligned}$$

Rewriting G with (d) we obtain

$$\underline{\text{Loop}}(x \in \mathbf{R}_1) \underline{\text{if}} B_1(x) \underline{\text{and}} \underline{\text{Some}}(y \in \mathbf{R}_2(x)) B_2(x, y) \underline{\text{then}} E(x)$$

Commutativity of conjunction and nesting of conditionals yields

$$\underline{\text{Loop}}(x \in \mathbf{R}_1) \underline{\text{if}} \underline{\text{Some}}(y \in \mathbf{R}_2(x)) B_2(x, y) \underline{\text{then}} \underline{\text{if}} B_1(x) \underline{\text{then}} E(x)$$

Finally, applying (idemloop) from right to left gives

$$G' \quad \stackrel{\text{def}}{=} \underline{\text{Loop}}(x \in \mathbf{R}_1) \underline{\text{Loop}}(y \in \mathbf{R}_2(x)) \underline{\text{if}} B_1(x) \underline{\text{and}} B_2(x, y) \underline{\text{then}} E(x)$$

Therefore, the equation $G = G'$ is derivable in CoDi’s equational theory (and hence follows semantically) from (d). We say that G' is the result of (one-step)

chasing G with (d) . G could be a query but it could also be the boolean-valued expression A when we translate constraints as equations $A = \underline{\text{true}}$. Chasing a constraint means chasing the expression A .

Query Containment. The chase step illuminates the connection between query containment and constraints (in the relational case a similar connection was put in evidence in [YP82,Abi83]). Consider

$$Q_i \stackrel{\text{def}}{=} \underline{\text{BigU}}(\mathbf{x} \in \mathbf{R}_i) \text{ if } B_i(\mathbf{x}) \text{ then } \underline{\text{sng}}(E_i(\mathbf{x})) \quad (i = 1, 2)$$

and observe that we can express intersection by

$$Q_1 \cap Q_2 = \underline{\text{BigU}}(\mathbf{x} \in \mathbf{R}_1) \underline{\text{BigU}}(\mathbf{y} \in \mathbf{R}_2) \text{ if } B_1(\mathbf{x}) \text{ and } B_2(\mathbf{y}) \text{ and } \underline{\text{eq}}(E_1(\mathbf{x}), E_2(\mathbf{y})) \text{ then } \underline{\text{sng}}(E_1(\mathbf{x}))$$

Now consider the constraint $\text{cont}(Q_1, Q_2)$ defined as

$$\mathbf{x} \in \mathbf{R}_1 \vdash B_1(\mathbf{x}) = B_1(\mathbf{x}) \text{ and } \underline{\text{Some}}(\mathbf{y} \in \mathbf{R}_2) B_2(\mathbf{y}) \text{ and } \underline{\text{eq}}(E_1(\mathbf{x}), E_2(\mathbf{y}))$$

It is easy to see that one step of chasing Q_1 with $\text{cont}(Q_1, Q_2)$ yields $Q_1 \cap Q_2$. Since $Q_1 \subseteq Q_2$ is equivalent to $Q_1 = Q_1 \cap Q_2$, it follows that containments can be established by deriving certain constraints. In fact, we show in sections 4 and 5 that for path-conjunctive queries containment is *reducible* to path-conjunctive dependencies.

The Relational Case. Conjunctive (tableau) queries [AHV95] are easily expressible in CoDi (using variables that range over tuples rather than over individuals, much as in SQL and OQL). Embedded dependencies (tuple- and equality-generating) are expressible just as we have expressed the constraints of the schema in figure 1⁶. It turns out that mirroring in CoDi one step of the relational chase of a tableau query Q (or another embedded dependency) with an embedded dependency (d) corresponds exactly to the equational chase step that we have described above! Moreover, the special case of chasing with trivial dependencies explains equationally tableaux query containment and tableaux minimization. When $\text{cont}(Q_1, Q_2)$ is trivial, it corresponds precisely with a containment mapping ([CM77]). We show in section 4 (and in full details in [PT98]) that this is the case for the more general class of path-conjunctive queries and dependencies. Moreover, we show that CoDi is complete for proving triviality and containment/equivalence for this class.

⁶ Since we are using tuple variables, full relational dependencies still require existential quantification in CoDi. With extra care, a generalization of full dependencies is still possible (section 5).

4 Path-Conjunctive Queries, Dependencies, and Views

A *schema* consists simply of some names (roots) \mathbf{R} and their types. An *instance* consists of complex values (with dictionaries) of the right type for each root name. We will distinguish between *finite* and *unrestricted* instances, where $\{\sigma\}$ means *all* sets. We now define:

Paths: $P ::= x \mid \mathbf{R} \mid P.\mathbf{A} \mid \underline{\text{dom}} P \mid x!P \mid \underline{\text{true}} \mid \underline{\text{false}}$

Path-Conjunctions: $C ::= \underline{\text{eq}}(P_1, P'_1) \underline{\text{and}} \cdots \underline{\text{and}} \underline{\text{eq}}(P_n, P'_n)$

Path-Conjunctive (PC) Queries: $Q ::= \underline{\text{BigU}}(x \in P) \underline{\text{if}} C(x) \underline{\text{then}} \underline{\text{sng}}(E(x))$
 where ⁷: $E ::= P \mid \langle A_1 : P_1, \dots, A_n : P_n \rangle$

Path-Conjunctive View: a view expressed using a PC query.

Embedded Path-Conjunctive Dependency (EPCD):

$d ::= \underline{\text{All}}(x \in P_1) \underline{\text{if}} C_1(x) \underline{\text{then}} \underline{\text{Some}}(y \in P_2(x)) C_2(x, y) = \underline{\text{true}}$

or equivalently (see lemma 1):

$d ::= x \in P_1 \vdash C_1(x) = C_1(x) \underline{\text{and}} \underline{\text{Some}}(y \in P_2(x)) C_2(x, y)$

Equality-Generating Dependency (EGD): an EPCD of the form

$\underline{\text{All}}(x \in P_1) \underline{\text{if}} C_1(x) \underline{\text{then}} C_2(x) = \underline{\text{true}}$

Restrictions. All PC queries and EPCDs are subject to the following restrictions.

1. A *simple* type is defined (inductively) as either a base type or a record type in which the types of the components are simple types (in other words, it doesn't involve set or dictionary types). Dictionary types $\sigma \times \tau$ are restricted such that σ is a simple type. The paths P_i, P'_i appearing in path conjunctions must be of simple type. The expression E appearing in PC queries must be of simple type.
2. A *finite set type* is a type of the form $\{\tau\}$ where the only base type occurring in τ is `bool` or $\langle \rangle$ (the empty record type). We do not allow in PC queries or EPCDs bindings of the form $x \in P$ such that P is of finite set type ⁸.
3. $x!P$ can occur only in the scope of a binding of the form $x \in \underline{\text{dom}} P$ ⁹.

⁷ Although we don't allow record constructors in paths, their equality is still representable, componentwise.

⁸ Finite set types cause some difficulties in our current proof method. However there are more serious reasons to worry about them: it is shown in [BNTW95] that they can be used to encode set difference, although the given encoding uses a language slightly richer than that of PC queries.

⁹ This restriction could be removed at the price of tedious reasoning about partiality, but we have seen no need to do it for the results and examples discussed here.

Views. The following shows that composing EPCDs with PC views yields EPCDs. Thus all subsequent results regarding implication/triviality of EPCDs can be used to infer implication/triviality of EPCDs over views as well.

Proposition 1. *Composing an EPCD over a schema \mathbf{S} with a PC view from \mathbf{R} to \mathbf{S} yields an expression that is provably equivalent to an EPCD over \mathbf{R} . Moreover composing EGDs with PC views gives expressions provably equivalent to EGDs.*

The **main result** of this section is about containment of queries and triviality (validity, holds in all instances) of dependencies.

Theorem 1 (Containment and Triviality). *Containment of PC queries and triviality of EPCDs are reducible to each other. Both hold in all (unrestricted) instances iff they hold in all finite instances. Both are decidable and in NP. CoDi's equational axiomatization is complete for deriving them.*

As for relational tableau queries, the proof of the theorem relies on a reduction to the existence of certain kinds of *homomorphisms* between *tableaux*. Except that here we must invent a more complex notion of tableau that takes set and dictionary nesting into consideration. In CoDi, the (PC) tableau corresponding to the PC query $\text{BigU}(\mathbf{x} \in \mathbf{P}) \text{ if } C(\mathbf{x}) \text{ then sng}(E(\mathbf{x}))$ is just a piece of syntax consisting of several expressions $T ::= \{\mathbf{x} \in \mathbf{P} ; C(\mathbf{x})\}$. Note that relational tableaux can also be represented this way, with the variables in \mathbf{x} corresponding to the rows. We define a homomorphism between two tableaux to be a mapping between variables satisfying certain PTIME-checkable conditions. The precise definition is given in [PT98]. A basic insight about relational tableaux is that they can be considered themselves as instances. In the presence of nested sets and dictionaries, we must work some to construct from any tableau T a canonical instance, $\text{Inst}(T)$ such that a valuation from T_2 into $\text{Inst}(T_1)$ induces a homomorphism from T_2 to T_1 .¹⁰ Deciding containment then comes down to testing for the existence of a homomorphism, which is in NP. Since the problem is already NP-hard in the relational case [CM77], containment of PC queries is in fact NP-complete. The full details are given in [PT98]. As for the reductions between containment and triviality, we show that for PC queries $Q_1 \subseteq Q_2$ iff $\text{cont}(Q_1, Q_2)$ is trivial, hence triviality of EPCDs is also NP-hard. Conversely, we associate to an EPCD (d) two queries¹¹, where \mathbf{A} are fresh labels:

$$\begin{aligned} \text{front}(d) &\stackrel{\text{def}}{=} \text{BigU}(\mathbf{x} \in \mathbf{P}_1) \text{ if } C_1(\mathbf{x}) \text{ then sng}\langle \mathbf{A} : \mathbf{x} \rangle \\ \text{back}(d) &\stackrel{\text{def}}{=} \text{BigU}(\mathbf{x} \in \mathbf{P}_1) \text{BigU}(\mathbf{y} \in \mathbf{P}_2(\mathbf{x})) \text{ if } C_1(\mathbf{x}) \text{ and } C_2(\mathbf{x}, \mathbf{y}) \\ &\quad \text{then sng}\langle \mathbf{A} : \mathbf{x} \rangle \end{aligned}$$

and we show that (d) is trivial iff $\text{front}(d) \subseteq \text{back}(d)$, hence triviality of EPCDs is NP-complete.

¹⁰ Intuitively, $\text{Inst}(T)$ is the minimal instance that contains the "structure" of T , and we can express syntactical conditions on T as equivalent conditions on $\text{Inst}(T)$.

¹¹ Since \mathbf{x} may have non-simple type variables, these queries do not obey the restriction we imposed for PC queries. Nonetheless, their containment is decidable. This encourages the hope that the simple type restriction can be removed for this theorem.

In the case of **EGDs**, we can improve on complexity and we can even remove the restriction to simple types.

Theorem 2 (Trivial EGDs). *A non-simple EGD (with equality at non-simple types) holds in all (unrestricted) instances iff it holds in all finite instances. Triviality of non-simple EGDs is decidable in PTIME. CoDi's equational axiomatization is complete for deriving all trivial non-simple EGDs.*

5 Path-Conjunctive Chase

The chase as defined in section 3 is only a particular case. In general we need to be able to map the bound variables to other variables. Moreover, we need a precise definition of when the chase step should *not* be applicable because it would yield a query or dependency that is trivially equivalent to the one we already had. As with relational tableau queries, it suffices to define the chase on tableaux: chasing a PC query means chasing the corresponding PC tableau, while chasing an EPCD (d) means chasing the PC tableau that corresponds to $front(d)$. For example, just for the particular case justified in section 3 chasing the tableau $\{\mathbf{x} \in \mathbf{P}_1 ; C_1(\mathbf{x})\}$ with the dependency

$$(d) \quad \underline{\text{All}}(\mathbf{x} \in \mathbf{P}_1) \underline{\text{if}} C_1(\mathbf{x}) \underline{\text{then}} \underline{\text{Some}}(\mathbf{y} \in \mathbf{P}_2(\mathbf{x})) C_2(\mathbf{x}, \mathbf{y}) = \underline{\text{true}}$$

yields the tableau $\{\mathbf{x} \in \mathbf{P}_1, \mathbf{y} \in \mathbf{P}_2(\mathbf{x}) ; C_1(\mathbf{x}) \underline{\text{and}} C_2(\mathbf{x}, \mathbf{y})\}$. The complete definition of the chase step involves homomorphisms and due to lack of space we give it in [PT98]. Like the particular case we've shown, the complete definition is such that a chase step transforms a PC query (an EPCD) into one provably equal (equivalent) in the CoDi equational axiomatization. Most importantly, the complete definition is such that if $Inst(T) \not\models d$ then the chase step with d is applicable to T .

A *chase sequence* with a set of EPCDs D is a sequence of tableaux obtained by successive chase steps each with some dependency $d \in D$ (same d can be used repeatedly). We say that a sequence starting with T *terminates* if it reaches a tableau T' that cannot be chased with any $d \in D$ (and therefore $Inst(T') \models D$). Although in general T' depends on the choice of terminating chase sequence, we shall denote it by $chase_D(T)$ and extend the same notation to queries and dependencies.

Theorem 3 (Containment/Implication by Chasing). *Let D be a set of EPCDs.*

1. *Let Q_1, Q_2 be PC queries such that some chasing sequence of Q_1 with D terminates (with $chase_D(Q_1)$). The following are equivalent and the unrestricted and finite version of each of (a)–(d) are equivalent as well:*

- | | |
|--|----------------------------------|
| (a) $Q_1 \subseteq_D Q_2$ | (b) $chase_D(Q_1) \subseteq Q_2$ |
| (c) $chase_D(cont(Q_1, Q_2))$ is trivial | (d) $D \models cont(Q_1, Q_2)$ |
| (e) $cont(Q_1, Q_2)$ (hence $Q_1 \subseteq Q_2$) is provable from D in CoDi | |

2. Let d be an EPCD such that some chasing sequence of d with D terminates. The following are equivalent and the unrestricted and finite version of each of (a)–(d) are equivalent as well:

- (a) $D \models d$ (b) $\text{chase}_D(d)$ is trivial
(c) $\text{chase}_D(\text{front}(d)) \subseteq \text{back}(d)$ (d2) $\text{front}(d) \subseteq_D \text{back}(d)$
(e) d is provable from D in CoDi

Full EPCDs. This is a class of dependencies that generalizes the relational full dependencies [AHV95] (originally called total tgd 's and egd 's in [BV84]). Since we work with “tuple” variables, the definition needs a lot more care than in the first-order case.

Definition 1 (Full Dependencies). An EPCD

$$\underline{\text{All}}(\mathbf{r} \in \mathbf{R}) \underline{\text{if}} B_1(\mathbf{r}) \underline{\text{then}} \underline{\text{Some}}(\mathbf{s} \in \mathbf{S}(\mathbf{r})) B_2(\mathbf{r}, \mathbf{s}) = \underline{\text{true}}$$

is full if for any variable s_i in \mathbf{s} there exists a path $P_i(\mathbf{r})$ such that the following EGD is trivial

$$\underline{\text{All}}(\mathbf{r} \in \mathbf{R}) \underline{\text{All}}(\mathbf{s} \in \mathbf{S}(\mathbf{r})) \underline{\text{if}} B_1(\mathbf{r}) \underline{\text{and}} B_2(\mathbf{r}, \mathbf{s}) \underline{\text{then}} \underline{\text{eq}}(s_i, P_i(\mathbf{r})) = \underline{\text{true}}$$

Theorem 4 (Full Termination). If D is a set of full EPCDs and T is a PC tableau then any chase sequence of T with D terminates.

Corollary 1. PC query containment under full EPCDs and logical implication of EPCDs from full EPCDs are reducible to each other, their unrestricted and finite versions coincide, and both are decidable.

Proposition 2. Let D be a set of full EPCDs and (d) be another EPCD. Let T be the tableau part of (d) that gets chased. Then deciding whether $D \models d$ can be done in time

$$c_1 |D| (nh)^{c_2 w s}$$

where c_1 and c_2 are two constants, $|D|$ is the number of EPCDs in D , n is the number of variables in T , s is the maximum number of variables of an EPCD in D , while w and h are two schema parameters: w is the maximum number of attributes that a record type has in the schema (including nested attributes at any depth) and h is the maximum height of a type in the schema.

Therefore, the complexity of the chase decision procedure for full EPCDs is not worse than in the relational subcase [BV84] (recall also the lower bound of [CLM81]). For EGDs, which are always full, it is easy to see that the problem is actually in PTIME, as in the relational case.

The chase with full EPCDs also enjoys the following nice properties:

Theorem 5 (Confluence). Consider two terminal chase sequences of a PC tableau T with a set D of full EPCDs, ending in T_1 and T_2 respectively. Then $\text{Inst}(T_1)$ and $\text{Inst}(T_2)$ must be isomorphic.

Note that we cannot hope that T_1 and T_2 are “equal” (even modulo variable renaming) because the path-conjunctions may be different, although logically equivalent.

Proposition 3 (Semantic Invariance). *Let D_1 and D_2 be two semantically equivalent (hold in the same instances) sets of full EPCDs and let T_1 and T_2 be the resulting tableaux of two arbitrary terminal chase sequences of a given tableau T with D_1 and, respectively, D_2 . Then $\text{Inst}(T_1)$ and $\text{Inst}(T_2)$ must be isomorphic.*

6 Other Results

The companion technical report [PT98] contains several other results that space restrictions prevent us from outlining here. In particular, we show in [PT98] how to extend and generalize the complete proof procedure result of Beeri and Vardi [BV84] for the case when the chase may not terminate. The result also applies to query containment. Note that since we are not in the first-order case, even the r.e.-ness is non-obvious. The equational axiomatization of CoDi is complete in this case too. Moreover, we show in [PT98] that the containment parts of theorems 1 and 3 (as well as the result for the non-terminating chase) also hold, with similar proofs, for boolean-valued-Some queries in PC form, where containment means boolean implication.

We have shown how to use dictionaries to model oo classes but in fact they are much more versatile. In [PT98] we give examples of queries, views, and constraints that use dictionaries to model indexes. We also use the chase to characterize nesting of relations into nested sets or dictionaries, as well as unnesting of the corresponding structures.

7 Related Work and Further Investigations

Related work. The monad algebra approach to aggregates [LT95] is related to the monoid comprehensions of [FM95b] but it is somewhat more general since there exist monads (trees for example) whose monad algebras are not monoids. The maps of [ALPR91], the treatment of object types in [BK93] and in [DHP97], that of views in [dSDA94], and that of arrays in [LMW96] are related to our use of dictionaries. An important difference is made by the operations on dictionaries used here.

The idea of representing constraints as equivalences between boolean-valued (OQL actually) queries already appears in [FRV96]. The equational theory of CoDi proves almost the entire variety of proposed algebraic query equivalences beginning with the standard relational algebraic ones, and including [SZ89a,SZ89b], [CD92,Clu91,FM95b,FM95a] and the very comprehensive work by Beeri and Kornatzky [BK93]. Moreover, using especially (commute), CoDi validates and generalizes standard join reordering techniques, thus the problem of join associativity in object algebras raised in [CD92] does not arise. Our PC queries are less

general than COQL queries [LS97], by not allowing alternations of conditionals and BigU. However we are more general in other ways, by incorporating dictionaries and considering constraints. Containment of PC queries is in NP while a double exponential upper bound is provided for containment of COQL queries. In [Bid87] it is shown that containment of conjunctive queries for the Verso complex value model and algebra is reducible to the relational case. Other studies include semantic query optimization for unions of conjunctive queries [CGM88], containment under Datalog-expressible constraints and views [DS96], and containment of non-recursive Datalog queries with regular expression atoms under a rich class of constraints [CGL98]. We are not aware of any extension of the chase to complex values and oodb models. Hara and Davidson [HD98] provide a complete intrinsic axiomatization of generalized functional dependencies for complex value schemas without empty sets. Fan and Weinstein [FW98] examine the un/decidability of logical implication for path constraints in various classes of oo-typed semistructured models.

Further investigations. We conjecture that the simple type restriction can be removed without affecting the containment/triviality result (Theorem 1). When equality at non-simple types is allowed, the chase is incomplete. However, the chase seems to be able to prove the two containments that make a set or dictionary equality. This suggests that a complete proof procedure might exist that combines the chase with an extensionality rule. Another important direction of work is allowing alternations of conditionals and BigU and trying to extend the result of [LS97] from weak equivalence to equivalence. The axiomatization of inclusions in [Abi83] can be soundly translated into CoDi's equational theory. We conjecture that CoDi is a conservative extension of this axiomatization. An interesting observation is that the equational chase does not require the PC restrictions, but just a certain form of query and dependency. It is natural to ask how far we can extend the completeness of this method. Most EPCDs in our examples are full. Some of those who are not may be amenable to the ideas developed for special cases with inclusion dependencies [JK84,CKV90]. Another question regards the decidable properties of classes of first-order queries and sentences that might correspond (by encoding, eg. [LS97]) to PC queries and EPCDs. Other encodings might allow us to draw comparisons with the interesting results of [CGL98].

It is an intriguing question why the powerful relational optimization techniques using tableaux and dependencies have not made their way into commercial optimizers. There seem to be two reasons for this. One is that queries crafted by users tend not to introduce spurious joins and tend to take advantage of certain constraints (even implicit ones!). The other reason is that the techniques have generally exponential algorithms. But do these reasons carry through to the paradigm of distributed mediator-based systems that interests us? We have already argued that in such systems there are a lot of (secondary) queries that are generated by not-so-smart software components. The complexity issue remains a serious one and only an experimental approach might put it to rest. Recall that the algorithms are exponential in the size of queries and dependencies, not of

data. Note also that standard relational optimization using dynamic programming is also exponential in theory, yet practical. Finally, some work on PTIME subcases exists [CR97,Sar91] and might be extended. On a more practical note, rewriting with individual CoDi axioms generates too large a search space to be directly useful in practical optimization. An important future direction is the modular development of coarser derived CoDi transformations corresponding to various optimization techniques in a rule-based approach.

Anecdote We were happily proving equalities in CoDi by rewriting with dependencies and (idemloop) for quite some time before we realized the connection with the chase!

Many thanks to Serge Abiteboul, Peter Buneman, Sophie Cluet, Susan Davidson, Alin Deutsch, Wenfei Fan, Carmem Hara, Rona Machlin, Dan Suciu, Scott Weinstein, and the paper's reviewers.

References

- [Abi83] S. Abiteboul. Algebraic analogues to fundamental notions of query and dependency theory. Technical report, INRIA, 1983.
- [ABU79] A. V. Aho, C. Beeri, and J. D. Ullman. The theory of joins in relational databases. *ACM Transactions on Database Systems*, 4(3):297–314, 1979.
- [AHV95] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
- [ALPR91] M. Atkinson, C. Lecluse, P. Philbrow, and P. Richard. Design issues in a map language. In *Proc. of the 3rd Int'l Workshop on Database Programming Languages (DBPL91)*, Nafplion, Greece, August 1991.
- [ASU79] A. V. Aho, Y. Sagiv, and J. D. Ullman. Equivalences among relational expressions. *SIAM Journal of Computing*, 8(2):218–246, 1979.
- [BBW92] Val Breazu-Tannen, Peter Buneman, and Limsoon Wong. Naturally embedded query languages. In J. Biskup and R. Hull, editors, *LNCS 646: Proceedings of 4th International Conference on Database Theory, Berlin, Germany, October, 1992*, pages 140–154. Springer-Verlag, October 1992. Available as UPenn Technical Report MS-CIS-92-47.
- [Bid87] N. Bidoit. The verso algebra or how to answer queries with fewer joins. *Journal of Computer and System Sciences*, 35:321–364, 1987.
- [BK93] Catriel Beeri and Yoram Kornatzky. Algebraic optimisation of object oriented query languages. *Theoretical Computer Science*, 116(1):59–94, August 1993.
- [BNTW95] Peter Buneman, Shamim Naqvi, Val Tannen, and Limsoon Wong. Principles of programming with collection types. *Theoretical Computer Science*, 149:3–48, 1995.
- [BV84] Catriel Beeri and Moshe Y. Vardi. A proof procedure for data dependencies. *Journal of the ACM*, 31(4):718–741, 1984.
- [Cat96] R. G. G. Cattell, editor. *The Object Database Standard: ODMG-93*. Morgan Kaufmann, San Mateo, California, 1996.
- [CD92] Sophie Cluet and Claude Delobel. A general framework for the optimization of object oriented queries. In M. Stonebraker, editor, *Proceedings ACM-SIGMOD International Conference on Management of Data*, pages 383–392, San Diego, California, June 1992.

- [CGL98] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. On the decidability of query containment under constraints. In *Proc. 17th ACM Symposium on Principles of Database Systems*, pages 149–158, 1998.
- [CGM88] U.S. Chakravarthi, J. Grant, and J. Minker. Foundations of semantic query optimization for deductive databases. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 243–273, San Mateo, California, 1988. Morgan-Kaufmann.
- [CKV90] Stavros S. Cosmadakis, Paris C. Kanellakis, and Moshe Y. Vardi. Polynomial-time implication problems for unary inclusion dependencies. *Journal of the ACM*, 37(1):15–46, 1990.
- [CLM81] A. K. Chandra, H. R. Lewis, and J. A. Makowsky. Embedded implicational dependencies and their inference problem. In *Proceedings of ACM SIGACT Symposium on the Theory of Computing*, pages 342–354, 1981.
- [Clu91] S. Cluet. *Langages et Optimisation de requetes pour Systemes de Gestion de Base de donnees oriente-objet*. PhD thesis, Universite de Paris-Sud, 1991.
- [CM77] Ashok Chandra and Philip Merlin. Optimal implementation of conjunctive queries in relational data bases. In *Proceedings of 9th ACM Symposium on Theory of Computing*, pages 77–90, Boulder, Colorado, May 1977.
- [CR97] C. Chekuri and A. Rajaraman. Conjunctive query containment revisited. In *LNCS 1186: Database Theory - ICDT'97, Proceedings of the 6th Int'l Conference*, pages 56–70, Delphi, 1997. Springer-Verlag.
- [CZ96] M. Cherniack and S. B. Zdonik. Rule languages and internal algebras for rule-based optimizers. In *Proceedings of the SIGMOD International Conference on Management of Data*, pages 401–412, Montreal, Quebec, Canada, 1996.
- [DHP97] S. B. Davidson, C. Hara, and L. Popa. Querying an object-oriented database using CPL. In *Proceedings of the 12th Brazilian Symposium on Databases*, pages 137–153, 1997. Also available as technical report MS-CIS-97-07, University of Pennsylvania.
- [DS96] Guozhu Dong and Jianwen Su. Conjunctive query containment with respect to views and constraints. *Information Processing Letters*, 57(2):95–102, 1996.
- [dSDA94] C. Souza dos Santos, C. Delobel, and S. Abiteboul. Virtual schemas and bases. In *Proceedings ICEDT*, March 1994.
- [Fag82] Ronald Fagin. Horn clauses and database dependencies. *Journal of the ACM*, 29(4):952–985, 1982.
- [FM95a] L. Fegaras and D. Maier. An algebraic framework for physical oodb design. In *Proc. of the 5th Int'l Workshop on Database Programming Languages (DBPL95)*, Umbria, Italy, August 1995.
- [FM95b] Leonidas Fegaras and David Maier. Towards an effective calculus for object query languages. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 47–58, San Jose, California, May 1995.
- [FRV96] D. Florescu, L. Rashid, and P. Valduriez. A methodology for query reformulation in cis using semantic knowledge. *International Journal of Cooperative Information Systems*, 5(4), 1996.
- [FW98] W. Fan and S. Weinstein. Interaction between path and type constraints. Technical Report MS-CIS-98-16, University of Pennsylvania, 1998.
- [HD98] Carmem Hara and Susan Davidson. Inference rules for nested functional dependencies. Technical Report MS-CIS-98-19, University of Pennsylvania, 1998.

- [JK84] D. S. Johnson and A. Klug. Testing containment of conjunctive queries under functional and inclusion dependencies. *Journal of Computer and System Sciences*, 28:167–189, 1984.
- [KP82] A. Klug and R. Price. In determining view dependencies using tableaux. *ACM Transactions on Database Systems*, 7:361–381, 1982.
- [LMW96] L. Libkin, R. Machlin, and L. Wong. A query language for multidimensional arrays: Design, implementation and optimization techniques. In *SIGMOD Proceedings, Int’l Conf. on Management of Data*, 1996.
- [LS97] Alon Levy and Dan Suciu. Deciding containment for queries with complex objects. In *Proc. of the 16th ACM SIGMOD Symposium on Principles of Database Systems*, Tucson, Arizona, May 1997.
- [LSK95] A. Levy, D. Srivastava, and T. Kirk. Data model and query evaluation in global information systems. *Journal of Intelligent Information Systems*, 1995.
- [LT95] S. K. Lellahi and V. Tannen. Enriched monads. Technical Report ??, LRI, Univ. Paris-Sud, February 1995.
- [LT97] Kazem Lellahi and Val Tannen. A calculus for collections and aggregates. In E. Moggi and G. Rosolini, editors, *LNCS 1290: Category Theory and Computer Science Proceedings of the 7th Int’l Conference, CTCS’97*, pages 261–280, Santa Margherita Ligure, September 1997. Springer-Verlag.
- [Mai83] David Maier. *The Theory of Relational Databases*. Computer Science Press, Rockville, Maryland, 1983.
- [MMS79] D. Maier, A. O. Mendelzon, and Y. Sagiv. Testing implications of data dependencies. *ACM Transactions on Database Systems*, 4(4):455–469, 1979.
- [PT98] Lucian Popa and Val Tannen. Chase and axioms for PC queries and dependencies. Technical Report MS-CIS-98-34, University of Pennsylvania, 1998. Available online at <http://www.cis.upenn.edu/~techreports/>.
- [QR95] X. Qian and L. Raschid. Query interoperation among object-oriented and relational databases. In *Proc. ICDE*, 1995.
- [Sar91] Y. Saraiya. *Subtree elimination algorithms in deductive databases*. PhD thesis, Stanford University, 1991.
- [SZ89a] G. Shaw and S. Zdonik. Object-oriented queries: equivalence and optimization. In *Proceedings of International Conference on Deductive and Object-Oriented Databases*, 1989.
- [SZ89b] G. Shaw and S. Zdonik. An object-oriented query algebra. In *Proc. DBPL*, Salishan Lodge, Oregon, June 1989.
- [Ull89] Jeffrey D. Ullman. *Principles of Database and Knowledge-Base Systems*, volume 2. Computer Science Press, 1989.
- [Wie92] Gio Wiederhold. Mediators in the architecture of future information systems. *IEEE Computer*, pages 38–49, March 1992.
- [YP82] Mihalis Yannakakis and Christos Papadimitriou. Algebraic dependencies. *Journal of Computer and System Sciences*, 25:2–41, 1982.