A PC Chase

MS-CIS-98-24

Lucian Popa and Val Tannen

University of Pennsylvania
School of Engineering and Applied Science
Computer and Information Science Department
Philadelphia, PA 19104-6389

1998

A PC Chase

Lucian Popa Val Tannen *

University of Pennsylvania

Abstract

PC stands for *path-conjunctive*, the name of a class of queries and dependencies that we define 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 Introduction

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 generated automatically by composition with views and decomposition among multiple sources. Unoptimized, this process quickly snowballs queries into forms with superfluous joins and would not exploit intra- and inter-data source integrity constraints. Exploiting integrity constraints (so-called semantic optimization) plays a crucial role in oodbs [CD92] and in integrating heterogenous data sources [CGMH$^+$94, QR95, LSK95]. Fortunately, relational database theory has studied intensively issues such as minimizing # of joins and using constraints for certain classes of queries and dependencies [Mai83, Ull89, AHV95].

In this paper we extend and generalize some basic results about conjunctive queries and embedded dependencies from the relational model to complex values and dictionaries (finite functions), the latter allowing the representation of oodb schemas and queries. This is done by representing queries and constraints (dependencies) in CoDi [1] 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*. While here we focus on set-related queries, we show elsewhere [2] that the (full) CoDi collection, aggregation and dictionary primitives suffices for implementing the quasi-totality of ODMG/ODL/OQL [Cat96]. Using boolean aggregates, CoDi can represent dependencies as equalities between boolean-valued queries. An important property of our approach, one that we hope to exploit in relation to rule-based optimization as in [CZ96], is that optimizing under dependencies or deriving other dependencies can be done *within* the equational theory by rewriting with the dependencies themselves.

Overview The types, expressions, and basic equational laws of CoDi are introduced in section 2. (Appendix A contains the rest of the axiomatization of the equational theory.) In section 3 we show how to represent in CoDi relational conjunctive queries (with equality) [CM77, ASU79] embedded dependencies, which are the multi-relation and un-typed versions of Fagin's embedded implicational dependencies [Fag82], and the chase [ABU79, MMS79, BV84b]. This suggests the definition in section 4 of a general notion of chase by rewriting, which connects dependencies to query equivalence (and therefore containment via intersection). We show also that implication of certain boolean-valued aggregate queries can be reduced to equivalences and comparison of certain number-valued

*Contact author. Address: University of Pennsylvania, Department of Computer and Information Science, 200 South 33rd Street, Philadelphia, PA 19104, Tel: (215)898-2665, Fax: (215)898-0587, Email: `val@cis.upenn.edu`

[1] From "collections and dictionaries"

[2] "An OQL interface to the K2 system", by J. Crabtree, S. Harker, and V. Tannen, forthcoming.

aggregate queries can be derived from equivalences, and that both can sometimes be proved by chasing. In the same section we discuss composing dependencies with views. In section 5 we offer examples of dependencies and equivalences beyond the relational model on which we use the generalized rewriting chase defined earlier. One example has an interesting notion of inverse relationship between a class and a relation that validates a significant optimization. Another captures the relationship between a relation and a dictionary representing a secondary index on it. Our main results are in section 6. 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 earlier are complete, and in certain cases decidable. Theorem 6.7 and corollary 6.8 extend and generalize the containment decidability/NP result of [CM77]. Theorem 6.11 and corollary 6.14 extend and generalize the corresponding results on the chase in [BV84b]. (We also extend and generalize Beeri and Vardi's result on the completeness of the infinite chase, see section 6.3.) Theorem 6.7 and theorem 6.11 also extend and generalize the corresponding completeness results on algebraic dependencies from [YP82, Abi83], although in a different equational theory. Proposition 6.3 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). The decidability and chase completeness results for containment of set-valued PC queries also hold for implication of boolean-valued disjunction aggregate PC queries.

2 CoDi

Types and expressions Only some of the language elements of CoDi are used in this paper and therefore described here. (Other elements, relating to bags, lists, conversions, etc., will be described elsewhere.)

Types $\sigma ::= \mathsf{bool} \mid \langle \mathtt{A}_1 : \sigma_1, \ldots, \mathtt{A}_n : \sigma_n \rangle \mid \{\sigma\} \mid \sigma_1 \times\!\!> \sigma_2 \mid \mathsf{num} \mid \ldots$other base types

Monad Algebras $\alpha ::= \mathsf{free} \mid \mathsf{or} \mid \mathsf{and} \mid \mathsf{max} \mid \mathsf{min}$

Expressions $E ::= x \mid E_1 \,\underline{\mathsf{and}}\, E_2 \mid \underline{\mathsf{if}}\, E_1 \,\underline{\mathsf{then}}\, E_2 \,\underline{\mathsf{else}}\, E_3 \mid \underline{\mathsf{eq}}(E_1, E_2) \mid \langle \mathtt{A}_1 : E_1, \ldots, \mathtt{A}_n : E_n \rangle \mid E.\mathtt{A} \mid$

$\underline{\mathsf{sng}}\, E \mid \underline{\mathsf{Loop}}\,[\alpha]\,(x \in E_1)\, E_2(x) \mid \underline{\mathsf{null}}\,[\alpha] \mid \underline{\mathsf{dom}}\, E \mid E_1 \,!\, E_2 \mid \underline{\mathsf{key}}\; x \;\underline{\mathsf{in}}\; E_1 \;\Rightarrow\; E_2(x)$

x is a *bound* variable (as in $\lambda x.e$) in $\underline{\mathsf{Loop}}\,[\alpha]\,(x \in E_1)\, E_2(x)$ and $\underline{\mathsf{key}}\; x \;\underline{\mathsf{in}}\; E_1 \;\Rightarrow\; E_2(x)$ and we write $E(x)$ to describe the *scope* of x, that is, x may occur in E_2, but not in E_1. (In fact, $\underline{\mathsf{key}}\; x \;\underline{\mathsf{in}}\; E_1 \;\Rightarrow\; E_2(x)$ is just the restriction to E_1 of $\lambda x.E_2(x)$.) For substitution, we will write $E(a)$ for the result of replacing x with a in $E(x)$.

Set restructuring and aggregation The CoDi fragment used here focuses on sets. We denote by $\{\sigma\}$ the type of finite (except in section 6) homogenous sets of elements of type σ. $\underline{\mathsf{sng}}\, E$ denotes singleton set. $\underline{\mathsf{Loop}}\,[\alpha]\,(x \in S)\, E(x)$ and $\underline{\mathsf{null}}\,[\alpha]$ are generic notations that depend on the *monad algebra* α. The monad algebras used here are structures associated with the set monad and are "enriched" with a nullary operation [LT97]. We do need the category-theoretic definitions in this paper because they are subsumed by the generic equivalence laws satisfied by $\underline{\mathsf{Loop}}\,[\alpha]\,(x \in S)\, E(x)$ and $\underline{\mathsf{null}}\,[\alpha]$ (below). However, we need to point out that these constructs have different semantics for each monad algebra α. These semantics are (for $S \stackrel{\text{def}}{=} \{a_1, \ldots, a_n\}$):

$$\begin{array}{rclrcl} \underline{\mathsf{Loop}}\,[\mathsf{free}]\,(x \in S)\, E(x) & = & \bigcup_{i=1}^{n} E(a_i) & \underline{\mathsf{Loop}}\,[\mathsf{max}]\,(x \in S)\, E(x) & = & max_{i=1}^{n} E(a_i) \\ \underline{\mathsf{Loop}}\,[\mathsf{or}]\,(x \in S)\, E(x) & = & \bigvee_{i=1}^{n} E(a_i) & \underline{\mathsf{Loop}}\,[\mathsf{and}]\,(x \in S)\, E(x) & = & \bigwedge_{i=1}^{n} E(a_i) \end{array}$$

We will use the following *abbreviations* to improve readability:

$$\begin{array}{rclrcl} \underline{\mathsf{BigU}}\,(x \in S)\, E(x) & \stackrel{\text{def}}{=} & \underline{\mathsf{Loop}}\,[\mathsf{free}]\,(x \in S)\, E(x) & \underline{\mathsf{Max}}\,(x \in S)\, E(x) & \stackrel{\text{def}}{=} & \underline{\mathsf{Loop}}\,[\mathsf{max}]\,(x \in S)\, E(x) \\ \underline{\mathsf{Some}}\,(x \in S)\, E(x) & \stackrel{\text{def}}{=} & \underline{\mathsf{Loop}}\,[\mathsf{or}]\,(x \in S)\, E(x) & \underline{\mathsf{All}}\,(x \in S)\, E(x) & \stackrel{\text{def}}{=} & \underline{\mathsf{Loop}}\,[\mathsf{and}]\,(x \in S)\, E(x) \end{array}$$

We can now explain the typing of $\underline{\mathsf{Loop}}$ in CoDi. Each monad algebra α has a *support type*, T_α, namely $T_{\mathsf{free}} = \{\sigma\}$ [3], $T_{\mathsf{or}} = T_{\mathsf{and}} = \mathsf{bool}$, $T_{\mathsf{max}} = T_{\mathsf{min}} = \mathsf{num}$. $\underline{\mathsf{Loop}}$ obeys the following uniform typing rule: if $S : \{\sigma\}$ and,

[3] We have simplified the notation of free which should be $\mathsf{free}(\sigma)$ because there is a whole family of free monad algebras, one for

```
Proj : set<struct{string PName;          class Dept (extent depts  key DName)
                  double Budg;             { attribute string DName;
                  string PDept;}>            attribute Set< string > DProj;};
           primary key (PName);
```

$\mathtt{Proj}\ :\ \{\langle \mathtt{PName} : \mathsf{string}, \mathtt{Budg} : \mathsf{num}, \mathtt{PDept} : \mathsf{string}\rangle\}$ $\qquad$ $\mathtt{Dept}\ :\ \mathit{Doid} \mathbin{\times\!\!>} \langle \mathtt{DName} : \mathsf{string}, \mathtt{DProj} : \{\mathsf{string}\}\rangle$

```
select distinct struct(PN: s, DN: d.DName)
from  depts d, d.DProj s, Proj p
where  s = p.PName  and  p.Budg > 100000
```

$$\mapsto\quad \underline{\mathsf{BigU}}\,(d \in \underline{\mathsf{dom}}\,\mathtt{Dept})\ \underline{\mathsf{BigU}}\,(s \in d\,!\,\mathtt{Dept.DProj})\ \underline{\mathsf{BigU}}\,(p \in \mathtt{Proj})\ \underline{\mathsf{if}}\ \underline{\mathsf{eq}}(s, p.\mathtt{PName})\ \underline{\mathsf{and}}\ p.\mathtt{Budg} > 100000\ \underline{\mathsf{then}}\ \underline{\mathsf{sng}}\langle \mathtt{PN} : s, \mathtt{DN} : d\,!\,\mathtt{Dept.DName}\rangle$$

Figure 1: An ODMG schema and query and their CoDi translations

assuming $x : \sigma$ we have $E(x) : T_\alpha$, then $\underline{\mathsf{Loop}}\,[\alpha]\,(x \in S)\,E(x) : T_\alpha$. The semantics given above does not cover the case $\underline{\mathsf{Loop}}\,[\alpha]\,(x \in \emptyset)\,E(x)$. It turns out [LT97] that the uniform way of dealing with a nullary constructor such as the empty set is to enrich each monad algebra α with a corresponding nullary operation, $\underline{\mathsf{null}}\,[\alpha] : T_\alpha$. Semantically, $\underline{\mathsf{null}}\,[\mathsf{free}]$ is indeed the empty set (CoDi abbreviation: $\underline{\mathsf{empty}}$), $\underline{\mathsf{null}}\,[\mathsf{or}]$ is the boolean false (CoDi abbreviation: $\underline{\mathsf{false}}$), $\underline{\mathsf{null}}\,[\mathsf{and}]$ is the boolean true (CoDi abbreviation: $\underline{\mathsf{true}}$), while $\underline{\mathsf{null}}\,[\mathsf{max}]$ and $\underline{\mathsf{null}}\,[\mathsf{min}]$ are the smallest, respectivelly largest element of type num (assume a symbolic completion of numbers with $\pm\infty$). We will also use the abbreviation

$$\underline{\mathsf{if}}\,[\alpha]\,B\ \underline{\mathsf{then}}\ E \ \stackrel{\text{def}}{=}\ \underline{\mathsf{if}}\ B\ \underline{\mathsf{then}}\ E\ \underline{\mathsf{else}}\ \underline{\mathsf{null}}\,[\alpha]$$

and to improve readability we will omit the α's in generic contexts. As for expressive power (so far), note that $\underline{\mathsf{BigU}}$ 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. In CoDi membership is expressible with disjunction aggregation:

$$\underline{\mathsf{member}}(E, S) \ \stackrel{\text{def}}{=}\ \underline{\mathsf{Some}}\,(x \in S)\,\underline{\mathsf{eq}}(x,\, E)$$

Dictionaries We denote by $\sigma \mathbin{\times\!\!>} \tau$ the type of dictionaries (finite functions) with keys of type σ and entries of type τ. $\underline{\mathsf{dom}}\,M$ denotes the set of keys (the *domain*) of the dictionary M. $K\,!\,M$ denotes the entry of M corresponding to the key K. This operation *fails* unless K is in $\underline{\mathsf{dom}}\,M$ and we will take care to use it in contexts in which it is guaranteed not to fail. If k is a variable of type σ, $D : \{\sigma\}$, and $E(k) : \tau$ is an expression in which k may occur, then $\underline{\mathsf{key}}\ k\ \underline{\mathsf{in}}\ D \ \Rightarrow\ E(k)$ denotes the dictionary with domain D that associates to an arbitrary key k the entry $E(k)$. The set of all entries is called the *range* of the dictionary and is definable $\underline{\mathsf{range}}\,M \stackrel{\text{def}}{=} \underline{\mathsf{BigU}}\,(k \in \underline{\mathsf{dom}}\,M)\,\underline{\mathsf{sng}}(k\,!\,M)$. As another example, consider a relation $\mathtt{R} : \{\langle \mathtt{A} : \sigma, \ldots\rangle\}$ and any one of its attributes A (need not be a key). The following dictionary is a logical level representation of a secondary index for R built on A:

$$\underline{\mathsf{ix2}}(\mathtt{R}, \mathtt{A}) \ \stackrel{\text{def}}{=}\ \underline{\mathsf{key}}\ a\ \underline{\mathsf{in}}\ \Pi_{\mathtt{A}}\mathtt{R} \Rightarrow \underline{\mathsf{BigU}}\,(r \in \mathtt{R})\,\underline{\mathsf{if}}\ \underline{\mathsf{eq}}(r.\mathtt{A},\, a)\ \underline{\mathsf{then}}\ \underline{\mathsf{sng}}(r) \qquad \text{where } \Pi_{\mathtt{A}}\mathtt{R} \ \stackrel{\text{def}}{=}\ \underline{\mathsf{BigU}}\,(r \in \mathtt{R})\,\underline{\mathsf{sng}}(r.\mathtt{A})$$

Representing classes with extents Dictionaries can be used to model object-oriented database classes with extents. In many semantic formalizations of oodb (eg., [AK89, Kos95]), instances of classes with extents are finite functions on object identities. This suggests the following natural internal representation. We say that the CoDi dictionary M *represents* the class C when:

- The keys for M correspond to the oids of C, hence the domain of M corresponds to the extent of C. The type of these keys is a *fresh base type*, distinct from other base types like num or bool and also distinct from fresh types used for other classes. [4]
- The type of entries in M is the record type of the components (attributes/relationships) of objects in C.

---

each type σ, acting on the values of type $\{\sigma\}$.

[4]This is in accordance with the principle of "oid abstraction" and allows us to achieve a faithful representation of oo query languages, a topic pursued elsewhere.

$$
\begin{array}{llrcl}
\text{(sng)} & & \underline{\mathsf{BigU}}\,(x \in S)\,\underline{\mathsf{sng}}(x) & = & S \\
\text{(monad-}\beta\text{)} & & \underline{\mathsf{Loop}}\,(x \in \underline{\mathsf{sng}}(E))\,E'(x) & = & E'(E) \\
\text{(assoc)} & & \underline{\mathsf{Loop}}\,(x \in (\underline{\mathsf{BigU}}\,(y \in R)\,S(y)))\,E(x) & = & \underline{\mathsf{Loop}}\,(y \in R)\,(\underline{\mathsf{Loop}}\,(x \in S(y))\,E(x)) \\
\text{(null)} & & \underline{\mathsf{Loop}}\,(x \in \underline{\mathsf{empty}})\,E(x) & = & \underline{\mathsf{null}} \\
\text{(commute)} & & \underline{\mathsf{Loop}}\,(x \in R)\,\underline{\mathsf{Loop}}\,(y \in S)\,E(x,y) & = & \underline{\mathsf{Loop}}\,(y \in S)\,\underline{\mathsf{Loop}}\,(x \in R)\,E(x,y) \\
\\
\text{(idemloop)} & & \underline{\mathsf{Loop}}\,(x \in S)\,\underline{\mathsf{if}}\ B(x)\ \underline{\mathsf{then}}\ E & = & \underline{\mathsf{if}}\ \underline{\mathsf{Some}}\,(x \in S)\,B(x)\ \underline{\mathsf{then}}\ E \\
\\
\text{(dict-}\beta\text{)} & x \in D \quad \vdash & x\,!\,(\underline{\mathsf{key}}\ k\ \underline{\mathsf{in}}\ D \Rightarrow E(k)) & = & E(x) \\
\text{(dom)} & & \underline{\mathsf{dom}}\,(\underline{\mathsf{key}}\ k\ \underline{\mathsf{in}}\ D \Rightarrow E(k)) & = & D \\
\text{(dict-}\eta\text{)} & & \underline{\mathsf{key}}\ k\ \underline{\mathsf{in}}\ (\underline{\mathsf{dom}}\,M) \Rightarrow k\,!\,M & = & M \quad (k \text{ not free in } M)
\end{array}
$$

Figure 2: Equivalence laws

To translate object-oriented queries into CoDi we need just two additions to the way we translate relational and complex-value queries:

- We translate the extent of C by $\underline{\mathsf{dom}}\,M$.
- If E is an expression of type C that gets translated as an expression $\bar{E}$ of the same type as the keys of M, then the *implicit oid dereferencing* $E.\mathtt{A}$ gets translated as $\bar{E}\,!\,M.\mathtt{A}$.

We illustrate the process in figure 1 with an ODMG [Cat96] schema and query (in OQL) and their translations into CoDi. This example features a class and the representation of a relation in (naturally extended) ODL. Note that CoDi can represent directly *dependent joins* (see [SZ90, CM93]).

Equivalence laws In figure 2 we show the basic laws used in CoDi. Some of these laws are derived from the theory of monads and monad algebras and the extensions we worked out in [BNTW95, LT97]. We note that (sng,β, assoc) hold for all monads, (null) is true for monads with a nullary constructor (certain tree types don't have it) and (commute) holds for "commutative" monads such as sets and bags, but not for lists or trees. (idemloop) plays a special role here and is discusssed below. All the laws hold for sets and their aggregates, the only collection type considered here. The equivalence laws for dictionaries are also in figure 2. Note the form of (dict-β). In general, the assertions of CoDi's equational theory have the form $\Gamma \vdash E_1 = E_2$ where

$$\Gamma \quad \stackrel{\text{def}}{=} \quad x_1 \in S_1, x_2 \in S_2(x_1), \ldots, x_n \in S_n(x_1, \ldots, x_{n-1}) \qquad (n \geq 0)$$

is a *context* that defines the "range" of each variable $S_i : \{\sigma_i\}$ and therefore also its type $x_i : \sigma_i$. We shall use the notation $\vec{x} \in \vec{S}$ for Γ, omitting for readability the part of the notation that shows which variables may occur in the S_i's.

The rest of the equational axiomatization is in appendix A. Some laws, such as (commute) and (from the appendix) the laws governing $\underline{\mathsf{eq}}$, the conditional (especially (eqcond)), $\underline{\mathsf{and}}$, 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 ubiquituous rearrangements.

Idemloop and its consequences This law depends on the idempotence property of set union and the corresponding operations of the set monad algebras (disjunction, conjunction, max, min) therefore it will not hold for bags, lists or trees. It also depends on $\underline{\mathsf{null}}$ being an identity for these operations. Note that x does not occur in E. The disjunction aggregate tests whether E contributes at least once in $\underline{\mathsf{Loop}}$. Because of (idemloop) the equational theory of CoDi has an interesting property. The proof rule (subst) $\Gamma, x \in S \vdash E_1(x) = E_2(x)$ and $\Gamma \vdash \underline{\text{member}}(E, S) = \underline{\text{true}}$ implies $\Gamma \vdash E_1(E) = E_2(E)$

does not have to be axiomatically stipulated (as in first-order algebraic theories) because it is already derivable. (The lambda calculus has a similar property but it is justified differently.) Moreover, (subst) and (dict-β) prove the expected operational justification

$$\underline{\text{member}}(K, D) = \underline{\text{true}} \quad \text{implies} \quad K\,!\,(\underline{\text{key}}\ k\ \underline{\text{in}}\ D \Rightarrow E(k)) \;=\; E(K).$$

We will see that the (idemloop) law plays a central role in relating dependencies and query containments and in the representation of the chase.

3 Relational conjunctive queries and embedded dependencies

Consider the **tableau minimization** [AHV95] on the right. Note that Q' is a subtableau of Q and that there is also a *homomorphism* (*containment mapping*) from Q to Q' (by $u \mapsto x$, $v \mapsto z$). Below we express Q, Q' in CoDi (note the correspondence between the rows of the tableaux and the CoDi bound variables). Then we write the equation (FOLD) below which is a *valid* (holds in all instances, aka. *trivial*) equation, is provable in CoDi and is equivalent to the existence of the homomorphism above. Now, by rewriting Q' first with (FOLD) and then twice with (idemloop), we obtain Q. (Remember that we convened to omit mention of certain minor manipulations.)

Q/d	A B
R	$x\ z$
R	$z\ y$
R	$u\ v$
R	$v\ y$
	$x\ \ y$

Q'/d'	A B
R	$x\ z$
R	$z\ y$
	$x\ \ y$

$Q \;=\; \underline{\text{BigU}}\,(p \in \text{R})\,\underline{\text{BigU}}\,(q \in \text{R})\,\underline{\text{BigU}}\,(s \in \text{R})\,\underline{\text{BigU}}\,(t \in \text{R})$
$\qquad\qquad \underline{\text{if}}\ \underline{\text{eq}}(p.\text{B},\ q.\text{A})\ \underline{\text{and}}\ \underline{\text{eq}}(s.\text{B},\ t.\text{A})\ \underline{\text{and}}\ \underline{\text{eq}}(q.\text{B},\ t.\text{B})\ \underline{\text{then}}\ \underline{\text{sng}}\langle p.\text{A}, q.\text{B}\rangle$

$Q' \;=\; \underline{\text{BigU}}\,(p \in \text{R})\,\underline{\text{BigU}}\,(q \in \text{R})\,\underline{\text{if}}\ \underline{\text{eq}}(p.\text{B},\ q.\text{A})\ \underline{\text{then}}\ \underline{\text{sng}}\langle p.\text{A}, q.\text{B}\rangle$

(FOLD) $p \in \text{R}, q \in \text{R} \vdash \underline{\text{eq}}(p.\text{B},\ q.\text{A}) \;=\; \underline{\text{eq}}(p.\text{B},\ q.\text{A})\ \underline{\text{and}}\ \underline{\text{Some}}\,(s \in \text{R})\,\underline{\text{Some}}\,(t \in \text{R})\,\underline{\text{eq}}(s.\text{B},\ t.\text{A})\ \underline{\text{and}}\ \underline{\text{eq}}(q.\text{B},\ t.\text{B})$

General conjunctive query containment can be represented similarly (see section 4) in CoDi's equational theory. The correspondence between homomorphisms and CoDi equations is made clear in theorem 6.7. (FOLD) is actually a simplified form of the equations we use for containment because it corresponds to a *folding* [CM77].

Embedded dependencies [AHV95] can be expressed in CoDi as equations between boolean-valued expressions by using conjunction and disjunction aggregation for quantification and conditionals for implication. For example, Q' above, seen as a tuple-generating dependency (tgd) d' is represented by

(d') $\underline{\text{All}}\,(p \in \text{R})\,\underline{\text{All}}\,(q \in \text{R})\,\underline{\text{if}}\ \underline{\text{eq}}(p.\text{B},\ q.\text{A})\ \underline{\text{then}}\ \underline{\text{Some}}\,(r \in \text{R})\,\underline{\text{eq}}(r.\text{A},\ p.\text{A})\ \underline{\text{and}}\ \underline{\text{eq}}(r.\text{B},\ q.\text{B}) \;=\; \underline{\text{true}}$

Chasing with embedded dependencies [AHV95] can also be represented by a certain kind of rewriting in CoDi's equational theory. For this, we need another form for (d'). First we introduce a notation that is convenient when we code up implication as an equality using conjunction:

$$A \wedge\!\!= B \quad \stackrel{\text{def}}{=} \quad A \;=\; (A\ \underline{\text{and}}\ B)$$

Lemma 3.1 (Two forms for dependencies) *For any $B_1(\vec{x}), B_2(\vec{x})$ the following two equations are derivable from each other in CoDi's equational theory:*

$$\begin{array}{ll} (1) & \underline{\text{All}}\,(\vec{x} \in \vec{S})\,\underline{\text{if}}\ B_1(\vec{x})\ \underline{\text{then}}\ B_2(\vec{x}) \;=\; \underline{\text{true}} \\ (2) & \qquad\qquad \vec{x} \in \vec{S} \vdash\ B_1(\vec{x}) \;\wedge\!\!=\; B_2(\vec{x}) \end{array}$$

Indeed, (2) follows from (1) using (all) and an implication rule (see appendix A). To derive (1) from (2) it suffices by ($\underline{\text{Loop}}$-cong) to show that $\underline{\text{All}}\,(\vec{x} \in \vec{S})\,\underline{\text{true}} \;=\; \underline{\text{true}}$. This follows by (idemloop) from $\underline{\text{if}}\ C\ \underline{\text{then}}\ \underline{\text{true}}\ \underline{\text{else}}\ \underline{\text{true}} \;=$

$\underline{\mathsf{true}}$ which is a consequence of the implication, conditional and $\underline{\mathsf{and}}$ rules. Therefore we can use for (d') the equivalent form:

$$(d') \qquad p \in \mathtt{R}, q \in \mathtt{R} \;\vdash\; \underline{\mathsf{eq}}(p.\mathtt{B},\, q.\mathtt{A}) \;\wedge\!\!= \;\underline{\mathsf{Some}}\,(r \in \mathtt{R})\,\underline{\mathsf{eq}}(r.\mathtt{A},\, p.\mathtt{A})\;\underline{\mathsf{and}}\;\underline{\mathsf{eq}}(r.\mathtt{B},\, q.\mathtt{B})$$

As an example, we will chase the query Q with this (d'). One possible chase step is represented by a rewrite with (d') and (idemloop). Another possible chase step is represented by renaming $p \mapsto s, q \mapsto t$ in d', rewriting with (d') and (idemloop) and obtaining:

$$Q \;\xrightarrow{d'}\; \begin{array}{l}\underline{\mathsf{BigU}}\,(p \in \mathtt{R})\,\underline{\mathsf{BigU}}\,(q \in \mathtt{R})\,\underline{\mathsf{BigU}}\,(s \in \mathtt{R})\,\underline{\mathsf{BigU}}\,(t \in \mathtt{R})\,\underline{\mathsf{BigU}}\,(r \in \mathtt{R})\\ \quad \underline{\mathsf{if}}\;\underline{\mathsf{eq}}(r.\mathtt{A},\, s.\mathtt{A})\;\underline{\mathsf{and}}\;\underline{\mathsf{eq}}(r.\mathtt{B},\, t.\mathtt{B})\;\underline{\mathsf{and}}\;\underline{\mathsf{eq}}(p.\mathtt{B},\, q.\mathtt{A})\;\underline{\mathsf{and}}\;\underline{\mathsf{eq}}(s.\mathtt{B},\, t.\mathtt{A})\;\underline{\mathsf{and}}\;\underline{\mathsf{eq}}(q.\mathtt{B},\, t.\mathtt{B})\;\underline{\mathsf{then}}\;\underline{\mathsf{sng}}\langle p.\mathtt{A}, q.\mathtt{B}\rangle\end{array}$$

which represents the query obtained by chasing.

Chasing a dependency instead of a query amounts to the same kind of rewriting. Consider Q seen a tgd d and represented as boolean-valued-$\underline{\mathsf{All}}$ query equals $\underline{\mathsf{true}}$ (like the first form of d'). Rewriting the left-hand side with (d'), without renaming, and then with (idemloop) gives a dependency which, in fact, is trivial (valid):

$$d \;\xrightarrow{d'}\; \begin{array}{l}\underline{\mathsf{All}}\,(p \in \mathtt{R})\,\underline{\mathsf{All}}\,(q \in \mathtt{R})\,\underline{\mathsf{All}}\,(s \in \mathtt{R})\,\underline{\mathsf{All}}\,(t \in \mathtt{R})\,\underline{\mathsf{All}}\,(r \in \mathtt{R})\\ \quad \underline{\mathsf{if}}\;\underline{\mathsf{eq}}(r.\mathtt{A},\, p.\mathtt{A})\;\underline{\mathsf{and}}\;\underline{\mathsf{eq}}(r.\mathtt{B},\, q.\mathtt{B})\;\underline{\mathsf{and}}\;\underline{\mathsf{eq}}(p.\mathtt{B},\, q.\mathtt{A})\;\underline{\mathsf{and}}\;\underline{\mathsf{eq}}(s.\mathtt{B},\, t.\mathtt{A})\;\underline{\mathsf{and}}\;\underline{\mathsf{eq}}(q.\mathtt{B},\, t.\mathtt{B})\\ \quad \underline{\mathsf{then}}\;\underline{\mathsf{Some}}\,(z \in \mathtt{R})\,\underline{\mathsf{eq}}(z.\mathtt{A},\, p.\mathtt{A})\;\underline{\mathsf{and}}\;\underline{\mathsf{eq}}(z.\mathtt{B},\, q.\mathtt{B}) \qquad = \quad \underline{\mathsf{true}}\end{array}$$

This represents the proof by chasing that (d) is true whenever (d') is true, i.e. (d) is implied by (d').

4 Chasing containments, dependencies, and views in CoDi

Any equation separates the instances in which it holds from those in which it doesn't so it is fair to call it a *constraint* or a *dependency*. We will be interested in dependencies of the form (d) below which generalizes the relational embedded dependencies as well as the trivial equations like (FOLD) used in section 3. We will also be interested in queries of the form Q below that generalizes the relational conjunctive queries as well as the (set-related) OQL query translations [5] ($\underline{\mathsf{Loop}}(\vec{x} \in \vec{S})$ means $\underline{\mathsf{Loop}}(x_1 \in S_1) \cdots \underline{\mathsf{Loop}}(x_n \in S_n(x_1, \ldots, x_{n-1}))$):

$$(d) \quad \vec{x} \in \vec{R_1} \;\vdash\; B_1(\vec{x}) \;\wedge\!\!=\; \underline{\mathsf{Some}}\,(\vec{y} \in \vec{R_2}(\vec{x}))\,B_2(\vec{x},\vec{y}) \qquad\qquad Q \;\stackrel{\text{def}}{=}\; \underline{\mathsf{Loop}}\,(\vec{x} \in \vec{R_1})\,\underline{\mathsf{if}}\;B_1(\vec{x})\;\underline{\mathsf{then}}\;E(\vec{x})$$

Generalizing from section 3, we call *chasing* Q with (d) in CoDi the rewriting of Q with (d) followed by rewriting with (idemloop)s and resulting in

$$Q' \;\stackrel{\text{def}}{=}\; \underline{\mathsf{Loop}}\,(\vec{x} \in \vec{R_1})\,\underline{\mathsf{Loop}}\,(\vec{y} \in \vec{R_2}(\vec{x}))\,\underline{\mathsf{if}}\;B_1(\vec{x})\;\underline{\mathsf{and}}\;B_2(\vec{x},\vec{y})\;\underline{\mathsf{then}}\;E(\vec{x})$$

A particular case, built just from (d) captures the essence (skeleton!) of the transformation. (d) proves by chasing that $\mathit{front}(d) = \mathit{back}(d)$ where

$$\mathit{front}(d) \;\stackrel{\text{def}}{=}\; \underline{\mathsf{BigU}}\,(\vec{x} \in \vec{R_1})\,\underline{\mathsf{if}}\;B_1(\vec{x})\;\underline{\mathsf{then}}\;\underline{\mathsf{sng}}\langle \vec{\mathtt{A}} : \vec{x}\rangle \qquad\qquad (\vec{\mathtt{A}} \text{ are fresh labels})$$

$$\mathit{back}(d) \;\stackrel{\text{def}}{=}\; \underline{\mathsf{BigU}}\,(\vec{x} \in \vec{R_1})\,\underline{\mathsf{BigU}}\,(\vec{y} \in \vec{R_2}(\vec{x}))\,\underline{\mathsf{if}}\;B_1(\vec{x})\;\underline{\mathsf{and}}\;B_2(\vec{x},\vec{y})\;\underline{\mathsf{then}}\;\underline{\mathsf{sng}}\langle \vec{\mathtt{A}} : \vec{x}\rangle$$

The CoDi chase yields directly equivalences, but of course containment can be reduced to equivalence using intersection. Doing this in some generality allows us a partial treatment of aggregate queries. Consider two queries of the studied form and define *meld*(-,-) and *cont*(-,-): [6]

$$Q_1 \;\stackrel{\text{def}}{=}\; \underline{\mathsf{Loop}}\,(\vec{x} \in \vec{R_1})\,\underline{\mathsf{if}}\;B_1(\vec{x})\;\underline{\mathsf{then}}\;E_1(\vec{x}) \qquad\qquad Q_2 \;\stackrel{\text{def}}{=}\; \underline{\mathsf{Loop}}\,(\vec{y} \in \vec{R_2})\,\underline{\mathsf{if}}\;B_2(\vec{y})\;\underline{\mathsf{then}}\;E_2(\vec{y})$$

[5] Actually, all dependencies and queries are equivalent to dependencies and queries in such form. Here we are just pointing out the soundness of certain syntactic methods. By putting restrictions on the expressions R, B, E we will show later (section 6) that these methods are in fact complete for an interesting class of queries and dependencies.

[6] These, like *front*(-) and *back*(-) are just syntactic abbreviations: in particular they are not semantically invariant.

$$meld(Q_1, Q_2) \stackrel{\text{def}}{=} \underline{\mathsf{Loop}}\,(\vec{x} \in \vec{R_1})\,\underline{\mathsf{Loop}}\,(\vec{y} \in \vec{R_2})\ \underline{\mathsf{if}}\ B_1(\vec{x})\ \underline{\mathsf{and}}\ B_2(\vec{y})\ \underline{\mathsf{and}}\ \underline{\mathsf{eq}}(E_1(\vec{x}),\, E_2(\vec{y}))\ \underline{\mathsf{then}}\ E_1(\vec{x}) \qquad \text{(or, by (eqcond), } E_2(\vec{y})\text{)}$$

$$cont(Q_1, Q_2) \stackrel{\text{def}}{=} \vec{x} \in \vec{R_1} \ \vdash\ B_1(\vec{x}) \ \wedge\!\!= \underline{\mathsf{Some}}\,(\vec{y} \in \vec{R_2})\, B_2(\vec{y})\ \underline{\mathsf{and}}\ \underline{\mathsf{eq}}(E_1(\vec{x}),\, E_2(\vec{y}))$$

Clearly, $cont(Q_1, Q_2)$ proves by chasing that $Q_1 = meld(Q_1, Q_2)$. If Q_1 and Q_2 are set-valued $\underline{\mathsf{BigU}}$ queries we will assume (by (sng)—without loss of generality) that $E_i = \underline{\mathsf{sng}}(E_i')$ Then, the meaning of $meld(Q_1, Q_2)$ is $Q_1 \cap Q_2$ and hence $Q_1 = meld(Q_1, Q_2)$ means $Q_1 \subseteq Q_2$. Thus, $cont(Q_1, Q_2)$ can be used to prove containment.

Moreover, it is easy to see that if Q_1, Q_2 are boolean-valued-$\underline{\mathsf{Some}}$ queries then $meld(Q_1, Q_2)$ means $Q_1 \wedge Q_2$ and therefore *implication* of such queries reduces to equivalence. In fact, set-valued and boolean-valued-$\underline{\mathsf{Some}}$ *path-conjunctive* queries (defined in section 6) are the form of queries for which we can prove the decidability and completeness results of section 6.

(Reverse) implication of boolean-valued-$\underline{\mathsf{All}}$ queries similarly reduces to equivalence. For number-valued $\underline{\mathsf{Max}}$ or $\underline{\mathsf{Min}}$ queries, $Q_1 = meld(Q_1, Q_2)$ is a sufficient (but not necessary) criterion for deriving $Q_1 < Q_2$ or $Q_1 > Q_2$.

The CoDi framework is nicely compositional and its equational theory often helps in reducing dependencies that hold in *views* to dependencies that hold in the original instance. To illustrate, consider a schema $\vec{R}$, a simple view V and a dependency (d) on instances of this view as follows (all occurrences of V in d are shown):

$$V \stackrel{\text{def}}{=} \underline{\mathsf{BigU}}\,(\vec{x} \in \vec{R})\ \underline{\mathsf{if}}\ B(\vec{x})\ \underline{\mathsf{then}}\ \underline{\mathsf{sng}}(E(\vec{x}))$$

$$(d) \qquad y_0 \in V, \vec{y} \in \vec{S_1}(y_0) \ \vdash\ B_1(y_0, \vec{y}) \ \wedge\!\!= \underline{\mathsf{Some}}\,(z_0 \in V)\,\underline{\mathsf{Some}}\,(\vec{z} \in \vec{S_2}(y_0, \vec{y}, z_0))\, B_2(y_0, \vec{y}, z_0, \vec{z})$$

Now, substituting in (d) the expression for V and using (assoc,monad-β) yields a similar form (d') which holds in some instance I of $\vec{R}$ iff (d) holds in the view instance $V(I)$:

$$(d') \quad \vec{x} \in \vec{R}, \vec{y} \in \vec{S_1}(E(\vec{x})) \ \vdash\ B(\vec{x})\ \underline{\mathsf{and}}\ B_1(E(\vec{x}), \vec{y}) \ \wedge\!\!= \underline{\mathsf{Some}}\,(\vec{x'} \in \vec{R})\,\underline{\mathsf{Some}}\,(\vec{z} \in \vec{S_2}(E(\vec{x}), \vec{y}, E(\vec{x'})))\, B(\vec{x'})\ \underline{\mathsf{and}}\ B_2(E(\vec{x}), \vec{y}, E(\vec{x'}), \vec{z})$$

5 Examples of dependencies on complex values and dictionaries

In this section we show that the queries and dependencies of the form given in section 4 and the chasing by rewriting that we defined there capture examples beyond the relational model.

Inverse relationships in oo schemas. Consider two oodb classes, represented by the dictionaries

$$\mathtt{M}_1 : \sigma_1 \mathrel{\times\!\!>} \langle \mathtt{A}_1 : \{\sigma_2\}, \ldots \rangle \qquad\qquad \mathtt{M}_2 : \sigma_2 \mathrel{\times\!\!>} \langle \mathtt{A}_2 : \{\sigma_1\}, \ldots \rangle$$

A *many-many inverse relationship* between attributes $\mathtt{A}_1$ and $\mathtt{A}_2$ can be represented by the following dependencies:

$$\begin{array}{llclcl}
\text{(RIC1)} & o_1 \in \underline{\mathsf{dom}}\,\mathtt{M}_1, x_2 \in o_1!\mathtt{M}_1.\mathtt{A}_1 & \vdash & \underline{\mathsf{true}} & = & \underline{\mathsf{Some}}\,(o_2 \in \underline{\mathsf{dom}}\,\mathtt{M}_2)\,\underline{\mathsf{eq}}(x_2,\, o_2) \\
\text{(RIC2)} & o_2 \in \underline{\mathsf{dom}}\,\mathtt{M}_2, x_1 \in o_2!\mathtt{M}_2.\mathtt{A}_2 & \vdash & \underline{\mathsf{true}} & = & \underline{\mathsf{Some}}\,(o_1 \in \underline{\mathsf{dom}}\,\mathtt{M}_1)\,\underline{\mathsf{eq}}(x_1,\, o_1) \\
\text{(INV1)} & o_1 \in \underline{\mathsf{dom}}\,\mathtt{M}_1, x_2 \in o_1!\mathtt{M}_1.\mathtt{A}_1, o_2 \in \underline{\mathsf{dom}}\,\mathtt{M}_2 & \vdash & \underline{\mathsf{eq}}(x_2,\, o_2) & \wedge\!\!= & \underline{\mathsf{Some}}\,(x_1 \in o_2!\mathtt{M}_2.\mathtt{A}_2)\,\underline{\mathsf{eq}}(o_1,\, x_1) \\
\text{(INV2)} & o_2 \in \underline{\mathsf{dom}}\,\mathtt{M}_2, x_1 \in o_2!\mathtt{M}_2.\mathtt{A}_2, o_1 \in \underline{\mathsf{dom}}\,\mathtt{M}_1 & \vdash & \underline{\mathsf{eq}}(x_1,\, o_1) & \wedge\!\!= & \underline{\mathsf{Some}}\,(x_2 \in o_1!\mathtt{M}_1.\mathtt{A}_1)\,\underline{\mathsf{eq}}(o_2,\, x_2)
\end{array}$$

The first two are examples of *inclusion dependencies* or *referential integrity contraints* (RICs), while the last two constraints complete the representation of the inverse relationship. We will see that all four dependencies are *full EPCDs* (see section 6) and therefore any chase with them is guaranteed to terminate. For any monad algebra and any expression $E(y, z)$ of the right type consider the equation

$$\underline{\mathsf{Loop}}\,(o_1 \in \underline{\mathsf{dom}}\,\mathtt{M}_1)\,\underline{\mathsf{Loop}}\,(x_2 \in o_1!\mathtt{M}_1.\mathtt{A}_1)\, E(o_1, x_2) \quad = \quad \underline{\mathsf{Loop}}\,(o_2 \in \underline{\mathsf{dom}}\,\mathtt{M}_2)\,\underline{\mathsf{Loop}}\,(x_1 \in o_2!\mathtt{M}_2.\mathtt{A}_2)\, E(x_1, o_2)$$

This equation is derivable from the dependencies above, by chasing the left-hand side with (RIC1, INV1) and the right-hand side with (RIC2, INV2). Therefore, in this schema, we can move back and forth between certain queries "centered" on $\mathtt{M}_1$ and queries "centered" on $\mathtt{M}_2$. This is the kind of semantic optimization discussed in [BK90] and [CD92]. Many-one and one-one inverse relationships can be similarly characterized by full EPCDs. The next example uses the same ideas in a heterogeneous context.

An inverse relationship between a class and a relation. Consider the oo schema in figure 1 and the following constraints on its instances. The values of `DProj` are sets of strings that should appear as values of `PName` (thus, RIC1). There is another obvious RIC2 for `PDept` and more subtly, we expect a certain inverse relationship constraint between `DProj` and `PDept`. Finally, there are two *key constraints* (*dependencies*):

$$\text{(RIC1)} \quad d \in \underline{\mathsf{dom}}\,\mathtt{Dept}, s \in d!\mathtt{Dept.DProj} \;\vdash\; \underline{\mathsf{true}} \;=\; \underline{\mathsf{Some}}\,(p \in \mathtt{Proj})\,\underline{\mathsf{eq}}(s,\, p.\mathtt{PName})$$

$$\text{(RIC2)} \quad p \in \mathtt{Proj} \;\vdash\; \underline{\mathsf{true}} \;=\; \underline{\mathsf{Some}}\,(d \in \underline{\mathsf{dom}}\,\mathtt{Dept})\,\underline{\mathsf{eq}}(p.\mathtt{PDept},\, d!\mathtt{Dept.DName})$$

$$\text{(INV1)} \quad d \in \underline{\mathsf{dom}}\,\mathtt{Dept}, s \in d!\mathtt{Dept.DProj}, p \in \mathtt{Proj} \;\vdash\; \underline{\mathsf{eq}}(s,\, p.\mathtt{PName}) \;\wedge\!=\; \underline{\mathsf{eq}}(p.\mathtt{DProj},\, d!\mathtt{Dept.DName})$$

$$\text{(INV2)} \quad p \in \mathtt{Proj}, d \in \underline{\mathsf{dom}}\,\mathtt{Dept} \;\vdash\; \underline{\mathsf{eq}}(p.\mathtt{DProj},\, d!\mathtt{Dept.DName}) \;\wedge\!=\; \underline{\mathsf{Some}}\,(s \in d!\mathtt{Dept.DProj})\,\underline{\mathsf{eq}}(p.\mathtt{PName},\, s)$$

$$\text{(KEY1)} \quad d \in \underline{\mathsf{dom}}\,\mathtt{Dept}, d' \in \underline{\mathsf{dom}}\,\mathtt{Dept} \;\vdash\; \underline{\mathsf{eq}}(d!\mathtt{Dept.DName},\, d'!\mathtt{Dept.DName}) \;\wedge\!=\; \underline{\mathsf{eq}}(d,\, d')$$

$$\text{(KEY2)} \quad p \in \mathtt{Proj}, p' \in \mathtt{Proj} \;\vdash\; \underline{\mathsf{eq}}(p.\mathtt{PName},\, p'.\mathtt{PName}) \;\wedge\!=\; \underline{\mathsf{eq}}(p,\, p')$$

It can be shown that for any monad algebra and any expression $E(x, y)$ of the right type:

$$\underline{\mathsf{Loop}}\,(d \in \underline{\mathsf{dom}}\,\mathtt{Dept})\,\underline{\mathsf{Loop}}\,(s \in d!\mathtt{Dept.DProj})\,E(d!\mathtt{Dept}, \mathtt{DName}, s) \;=\; \underline{\mathsf{Loop}}\,(p \in \mathtt{Proj})\,E(p.\mathtt{PDept}, p.\mathtt{PName})$$

is provable by chasing the left side with (RIC1, INV1) and the right side with (RIC2, INV2). We can use this to show that the OQL query in figure 1 is equivalent to a much better query. Indeed, in the CoDi translation of the query in figure 1, the innermost loop depends only on $d!\mathtt{Dept.DName}$ and s. With the equivalence we have just derived (or the chases used in its proof) with a (KEY2) chase step, and with a tableau minimization (which is chasing with trivial dependencies), we obtain the query:

```
select distinct struct(PN: p.PName, DN: p.PDept)
from   Proj p
where  p.Budg > 100000
```

Nest/unnest. Let $\mathtt{R} : \{\langle \mathtt{A} : \sigma, \mathtt{B} : \tau \rangle\} \quad \mathtt{W} : \{\langle \mathtt{A} : \sigma, \mathtt{Bs} : \{\tau\} \rangle\}$ and define the well-known operations

$$\underline{\mathsf{unnest}}(\mathtt{W}) \stackrel{\text{def}}{=} \underline{\mathsf{BigU}}\,(w \in \mathtt{W})\,\underline{\mathsf{BigU}}\,(b \in w.\mathtt{Bs})\,\underline{\mathsf{sng}}\langle \mathtt{A} : w.\mathtt{A}, \mathtt{B} : b \rangle$$

$$\underline{\mathsf{nest}}(\mathtt{R}) \stackrel{\text{def}}{=} \underline{\mathsf{BigU}}\,(r \in \mathtt{R})\,\underline{\mathsf{sng}}\langle \mathtt{A} : r.\mathtt{A}, \mathtt{Bs} : \underline{\mathsf{BigU}}\,(s \in \mathtt{R})\,\underline{\mathsf{if}}\;\underline{\mathsf{eq}}(s.\mathtt{A},\, r.\mathtt{A})\;\underline{\mathsf{then}}\;\underline{\mathsf{sng}}(s.\mathtt{B})\rangle$$

With (assoc,monad-β) rewriting it can be shown that $\underline{\mathsf{unnest}}(\underline{\mathsf{nest}}(\mathtt{R}))$ is equivalent not directly to `R` but to a spurious join of `R` with itself. In turn, this becomes `R` by a tableau minimization rewriting as in section 3. It can be shown that the equality $\mathtt{R} \;=\; \underline{\mathsf{unnest}}(\mathtt{W})$ is equivalent to the dependencies

$$\text{(UNNEST1)} \quad r \in \mathtt{R} \;\vdash\; \underline{\mathsf{true}} \;=\; \underline{\mathsf{Some}}\,(w \in \mathtt{W})\,\underline{\mathsf{Some}}\,(b \in w.\mathtt{Bs})\,\underline{\mathsf{eq}}(r.\mathtt{A},\, w.\mathtt{A})\;\underline{\mathsf{and}}\;\underline{\mathsf{eq}}(r.\mathtt{B},\, b)$$

$$\text{(UNNEST2)} \quad w \in \mathtt{W}, b \in w.\mathtt{Bs} \;\vdash\; \underline{\mathsf{true}} \;=\; \underline{\mathsf{Some}}\,(r \in \mathtt{R})\,\underline{\mathsf{eq}}(w.\mathtt{A},\, r.\mathtt{A})\;\underline{\mathsf{and}}\;\underline{\mathsf{eq}}(b,\, r.\mathtt{B})$$

and further equivalent to the family of equalities

$$\underline{\mathsf{Loop}}\,(w \in \mathtt{W})\,\underline{\mathsf{Loop}}\,(b \in w.\mathtt{Bs})\,E(w.\mathtt{A}, b) \;=\; \underline{\mathsf{Loop}}\,(r \in \mathtt{R})\,E(r.\mathtt{A}, r.\mathtt{B})$$

This is similar to (and simpler than) proposition 5.1 below for which a proof sketch in given. Moreover, it turns out that the equality $\mathtt{W} = \underline{\mathsf{nest}}(\underline{\mathsf{unnest}}(\mathtt{W}))$ is equivalent to the dependencies

$$\text{(KEY)} \quad w \in \mathtt{W}, b \in w.\mathtt{Bs}, w' \in \mathtt{W} \;\vdash\; \underline{\mathsf{eq}}(w.\mathtt{A},\, w'.\mathtt{A}) \;=\; \underline{\mathsf{eq}}(w,\, w')$$

$$\text{(NON-EMPTY)} \quad w \in \mathtt{W} \;\vdash\; \underline{\mathsf{true}} \;=\; \underline{\mathsf{Some}}\,(b \in w.\mathtt{Bs})\,\underline{\mathsf{true}}$$

Indeed, (KEY) and (NON-EMPTY) hold provably in any view of the form $\mathtt{W} = \underline{\mathsf{nest}}(\mathtt{R})$. Conversely, we can chase $\underline{\mathsf{nest}}(\underline{\mathsf{unnest}}(\mathtt{W}))$ with (NON-EMPTY) and (KEY) (at different levels of nesting[7]). Putting it all together, we also conclude that $\mathtt{W} = \underline{\mathsf{nest}}(\mathtt{R})$ is equivalent to all four dependencies: (UNNEST1, UNNEST2, KEY, NON-EMPTY). Note that all but (UNNEST1) are full. The additional $b \in w.\mathtt{Bs}$ in (KEY) was put in to make it an *inhabited* dependency (see section 6). Because (KEY) is used with (NON-EMPTY) it does not impede its applicability.

A logical level representation of secondary indexes Recall the definition of $\underline{\mathsf{ix2}}(\text{-},\text{-})$ in section 2. We have

Proposition 5.1 *For any* $\mathtt{R} : \{\tau\}$ *where* $\tau \equiv \langle \mathtt{A} : \sigma, \ldots \rangle$ *and any* $M : \sigma \times\!> \{\tau\}$ *the following are equivalent in the CoDi equational theory:*

(i) $\mathtt{M} = \underline{\mathsf{ix2}}(\mathtt{R}, \mathtt{A})$

(ii)

$$\begin{array}{llcl}
\text{(FLAT-RANGE1)} & r \in \mathtt{R} \;\vdash\; \underline{\mathsf{true}} & = & \underline{\mathsf{Some}}\,(a \in \underline{\mathsf{dom}}\,\mathtt{M})\,\underline{\mathsf{Some}}\,(t \in a\,!\,\mathtt{M})\,\underline{\mathsf{eq}}(r,\, t) \\
\text{(FLAT-RANGE2)} & a \in \underline{\mathsf{dom}}\,\mathtt{M}, t \in a\,!\,\mathtt{M} \;\vdash\; \underline{\mathsf{true}} & = & \underline{\mathsf{Some}}\,(r \in \mathtt{R})\,\underline{\mathsf{eq}}(t,\, r) \\
\text{(DUPL-KEY)} & a \in \underline{\mathsf{dom}}\,\mathtt{M}, t \in a\,!\,\mathtt{M} \;\vdash\; \underline{\mathsf{true}} & = & \underline{\mathsf{eq}}(t.\mathtt{A},\, a) \\
\text{(NON-EMPTY)} & a \in \underline{\mathsf{dom}}\,\mathtt{M} \;\vdash\; \underline{\mathsf{true}} & = & \underline{\mathsf{Some}}\,(t \in a\,!\,\mathtt{M})\,\underline{\mathsf{true}}
\end{array}$$

(iii)

$$\begin{array}{rcl}
\underline{\mathsf{Loop}}\,(a \in \underline{\mathsf{dom}}\,\mathtt{M})\,E_1(a) & = & \underline{\mathsf{Loop}}\,(r \in \mathtt{R})\,E_1(r.\mathtt{A}) \\
a \in \underline{\mathsf{dom}}\,\mathtt{M} \;\vdash\; \underline{\mathsf{Loop}}\,(r \in \mathtt{R})\,\underline{\mathsf{if}}\;\underline{\mathsf{eq}}(r.\mathtt{A},\, a)\;\underline{\mathsf{then}}\;E_2(r.\mathtt{A}, r) & = & \underline{\mathsf{Loop}}\,(t \in a\,!\,\mathtt{M})\,E_2(a, t)
\end{array}$$

Proof sketch. (i)$\Rightarrow$(ii) Think of M as a view of R. Substituting the definition of $\underline{\mathsf{ix2}}(\mathtt{R}, \mathtt{A})$ (section 2) for M in each dependency in (ii) produces a trivial dependency on R. (ii)$\Rightarrow$(iii) By chasing both sides of the equalities in (iii) with dependencies in (ii). (iii)$\Rightarrow$(i) Take $\underline{\mathsf{BigU}}$ for $\underline{\mathsf{Loop}}$, $\underline{\mathsf{sng}}(a)$ for $E_1(a)$ and $\underline{\mathsf{sng}}(t)$ for $E_2(a, t)$. □

Note that the only constraint here that is not a full EPCD is (FLAT-RANGE1). There are several ways to replace (FLAT-RANGE1) with full EPCDs. One of the less obvious ways is to consider

$$\begin{array}{llcl}
\text{(RIC)} & r \in \mathtt{R} \;\vdash\; \underline{\mathsf{true}} & = & \underline{\mathsf{Some}}\,(a \in \underline{\mathsf{dom}}\,\mathtt{M})\,\underline{\mathsf{eq}}(r.\mathtt{A},\, a) \\
\text{(INV)} & r \in \mathtt{R}, a \in \underline{\mathsf{dom}}\,\mathtt{M} \;\vdash\; \underline{\mathsf{eq}}(r.\mathtt{A},\, a) & \wedge\!\!= & \underline{\mathsf{Some}}\,(t \in a\,!\,\mathtt{M})\,\underline{\mathsf{eq}}(r,\, t)
\end{array}$$

It can be shown by chasing that (FLAT-RANGE1) follows from (RIC) and (INV), and that (INV) and (RIC) follow from (FLAT-RANGE1) and (DUPL-KEY).

6 Decidability and completeness results

A *schema* consists simply of some names (roots) and their types: $\vec{R} : \vec{\sigma}$. An *instance* consists of complex values (with dictionaries) of the right type for each root name. In this section we distinguish between *finite* and *unrestricted* instances, in the latter $\{\sigma\}$ meaning *all* sets. We now define *paths* P and *path-conjunctions* C:

$$P ::= x \mid \mathtt{R} \mid P.\mathtt{A} \mid \underline{\mathsf{dom}}\,P \mid x\,!\,P \mid \langle \mathtt{A}_1 : P_1, \ldots, \mathtt{A}_n : P_n \rangle \mid \underline{\mathsf{null}}[\alpha] \mid \underline{\mathsf{sng}}P \qquad C ::= \underline{\mathsf{eq}}(P_1,\, P_1')\;\underline{\mathsf{and}}\;\cdots\;\underline{\mathsf{and}}\;\underline{\mathsf{eq}}(P_n,\, P_n')$$

A *path-conjunctive* (PC) query has the form $\underline{\mathsf{Loop}}\,(\vec{x} \in \vec{P_1})\,\underline{\mathsf{if}}\;C(\vec{x})\;\underline{\mathsf{then}}\;P_2(\vec{x})$

An *embedded path-conjunctive dependency* (EPCD) has one of the equivalent (see section 3) forms

$$\underline{\mathsf{All}}\,(\vec{x} \in \vec{P_1})\,\underline{\mathsf{if}}\;C_1(\vec{x})\;\underline{\mathsf{then}}\;\underline{\mathsf{Some}}\,(\vec{y} \in \vec{P_2}(\vec{x}))\,C_2(\vec{x}, \vec{y}) = \underline{\mathsf{true}} \qquad \vec{x} \in \vec{P_1} \;\vdash\; C_1(\vec{x}) \wedge\!\!= \underline{\mathsf{Some}}\,(\vec{y} \in \vec{P_2}(\vec{x}))\,C_2(\vec{x}, \vec{y})$$

An *equality-generating dependency* (EGD) is an EPCD of the form $\vec{x} \in \vec{P_1} \;\vdash\; C_1(\vec{x}) \wedge\!\!= \underline{\mathsf{eq}}(\vec{P_2}(\vec{x}), \vec{P_3}(\vec{x}))$

[7]Note that $\underline{\mathsf{nest}}(\underline{\mathsf{unnest}}(\mathtt{W}))$ is not a PC query (see section 6)

A PC *tableau* consists of a context and a path-conjunction of the form $T ::= \{\vec{x} \in \vec{P_1} \; ; \; C_1(\vec{x})\}$

For an EPCD as above we will also use the notation $dep(T, T')$, where T is as above and $T' = \{\vec{x} \in \vec{P_1}, \vec{y} \in \vec{P_2}(\vec{x}); C_1(\vec{x}) \; \underline{\mathsf{and}} \; C_2(\vec{x}, \vec{y})\}$. This is in the spirit with the notation for tuple generating dependencies using tableaux in [BV84a] and [BV84a]. Note however that our formalism doesn't necessarily distinguish between EPCDs and EGDs: any EGD can be written as $dep(T, T')$, where $T' = \{\vec{x} \in \vec{P_1}; C_1(\vec{x}) \; \underline{\mathsf{and}} \; \underline{\mathsf{eq}}(\vec{P_2}(\vec{x}), \vec{P_3}(\vec{x}))\}$. For a PC query Q as above we will use the abbreviation $\underline{\mathsf{Loop}}(T)P_2$.

Restrictions All PC queries, EPCDs, and tableaux are subject to the following restrictions. (1) 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 tableaux bindings of the form $x \in P$ such that P is of finite set type. (2) $x \,!\, P$ can occur only in the scope of a binding of the form $x \in \underline{\mathsf{dom}}\, P$ [8]. Note that if Q_1, Q_2 are PC queries then $cont(Q_1, Q_2)$ is an EPCD and that if d is an EPCD then $front(d)$ and $back(d)$ are PC queries (definitions in section 4). There is an additional restriction on EPCDs, *inhabitation*, that will be outlined shortly.

Definition 6.1 *A valuation* [9] *of a tableau* $T = \{\vec{x} \in \vec{P} \; ; \; C(\vec{x})\}$ *into an instance* I *is a type-preserving mapping* $v : \vec{x} \to I$ *that can be extended to path expressions and path conjunctions over* $\vec{x}$ *(i.e.* $v(R) = R^I$*, for any name* R*,* $v(P.A) = v(P).A$*, etc.) such that the following two conditions hold:*

(1) if $x \in P$ *occurs in* T *then* $v(x)$ *is an element of* $v(P)$ *in* I *(context-preserving property)*

(2) $v(C(\vec{x})) = true$

The key to proving the results in this section is the construction of a canonical instance $Inst(T)$ associated to each tableau T [10] (see appendix B for details). Briefly, we associate to each tableau T a graph $Inst_\emptyset(T)$ (not quite an instance!) having as nodes congruence classes (w.r.t. equalities mentioned in T) of path expressions over T, and a canonical "valuation" $\underline{\mathsf{cval}}_\emptyset : T \to Inst_\emptyset(T)$. Then, $Inst(T)$ and a canonical valuation $\underline{\mathsf{cval}} : T \to Inst(T)$ are constructed from $Inst_\emptyset(T)$ and $\underline{\mathsf{cval}}_\emptyset$ by identifying empty sets of the same type.

Inhabitated dependencies. All the EPCDs we consider are required to be *inhabited.* This problem is specific to complex values and is due to expressions being equated because they denote empty sets. In the subsequent definitions and theorems of this section, all given EPCDs/EGDs are inhabited, all given PC queries are such that the *cont*(-,-)s are inhabited, and all EPCDs/EGDs that are required to hold are also required to be inhabited.

Definition 6.2 $\underline{\mathsf{eq}}(Q(\vec{x}), Q'(\vec{x}))$ *is an* inhabited *formula over* $T = \{\vec{x} \in \vec{P}; C(\vec{x})\}$ *if* Q *and* Q' *are path expressions over* T *such that* $\underline{\mathsf{cval}}Q = \underline{\mathsf{cval}}Q'$ *implies* $\underline{\mathsf{cval}}_\emptyset Q = \underline{\mathsf{cval}}_\emptyset Q'$ *(the converse always holds).*

Since the construction of $Inst(T)$ can be carried out in PTIME, we can decide in PTIME whether a formula is inhabited. A conjunction of inhabited formulas is an inhabited formula. We call an EGD $\vec{x} \in \vec{P} \; \vdash \; C(\vec{x}) \wedge\!= D(\vec{x})$ an *inhabited EGD* if the formula $D(\vec{x})$ is inhabited over $T = \{\vec{x} \in \vec{P}; C(\vec{x})\}$. Intuitively, EGDs that are not inhabited are those that may be satisfied by $Inst(T)$, even though they are not valid. The collapsing of the "empty" sets in $Inst_\emptyset(T)$ causes this problem.

Examples. Let $S = (R, W, U)$ a schema with three relation names. Suppose $R : \{\langle A : \{\sigma\}, B : \tau\rangle\}$. Then $x \in R, y \in R \; \vdash \underline{\mathsf{eq}}(x, y) \wedge\!= \underline{\mathsf{eq}}(x.A, y.A)$ is inhabited, while $x \in R, y \in R \; \vdash \underline{\mathsf{true}} \wedge\!= \underline{\mathsf{eq}}(x.A, y.A)$ and $x \in R \; \vdash \; \underline{\mathsf{true}} \wedge\!= \underline{\mathsf{eq}}(W, U)$ are not inhabited.

An EPCD $\vec{x} \in \vec{P} \; \vdash \; C(\vec{x}) \wedge\!= \underline{\mathsf{Some}}\,(\vec{y} \in \vec{Q}(\vec{x}))\, D(\vec{x}, \vec{y})$ is an *inhabited EPCD* if the formula $D(\vec{x}, \vec{y})$ is inhabited over the tableau $T' = \{\vec{x} \in \vec{P}, \vec{y} \in \vec{Q}(\vec{x}); C(\vec{x})\}$.

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

[8] 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 in this paper

[9] The notion of valuation is useful in giving meaning of expressions with free variables. In particular, we are able to express in terms of valuations the notion of satisfiability of an EPCD by an instance.

[10] In the relational case this instance is isomorphic to the tableau itself.

Proposition 6.3 *Let (d) be an EPCD over a schema* $\vec{\mathsf{S}}$ *whose roots have set type and suppose that* $\vec{\mathsf{S}}$ *is a* PC view, *that is, each* S *is expressed as a set-valued PC query over another schema* $\vec{\mathsf{R}}$. *Composing this view with (d) is provably equivalent to another EPCD, this one over* $\vec{\mathsf{R}}$.

6.1 Triviality and containment

We state here without proof that an inhabited EGD $\vec{x} \in \vec{P} \ \vdash \ C(\vec{x}) \wedge\!\!= \underline{\mathsf{eq}}(Q(\vec{x}),\, Q'(\vec{x}))$ is trivial (fin/unr) if and only if $\underline{\mathsf{cval}}Q = \underline{\mathsf{cval}}Q'$ (and therefore if and only if $\underline{\mathsf{cval}}_{\emptyset}Q = \underline{\mathsf{cval}}_{\emptyset}Q'$). Thus, deciding the triviality of an inhabited EGD reduces to checking its satisfiability in *Inst*(T) under the canonical valuation. Moreover this can be done in PTIME (since the construction of *Inst*(T) can be carried out in PTIME). It is easy to see that any inhabited trivial EGD $\vec{x} \in \vec{P} \vdash C(\vec{x}) \wedge\!\!= \underline{\mathsf{eq}}(Q,\, Q')$ is provable in CoDi 's equational theory. This is because $\underline{\mathsf{cval}}_{\emptyset}Q = \underline{\mathsf{cval}}_{\emptyset}Q'$ implies that Q and Q' are in the same congruence class in $\mathit{Inst}_{\emptyset}(T)$ (see appendix B), thus $\underline{\mathsf{eq}}(Q,\, Q')$ follows from $C(\vec{x})$ using congruence rules. To summarize:

Theorem 6.4 *An EGD holds in all unrestricted instances iff it holds in all finite instances. Trivial EGDs are provable in CoDi and triviality is decidable in PTIME.*

Definition 6.5 (Homomorphism) *Let* $T = \{\vec{x} \in \vec{P}; C(\vec{x})\}$ *and* $T' = \{\vec{y} \in \vec{R}; D(\vec{y})\}$ *be two tableaux. A* homomorphism $h : T' \to T$ *is a type-preserving mapping from variables* $\vec{y}$ *into variables* $\vec{x}$ *such that* h *is* context-preserving *i.e., for any* $y_i \in R_i$ *in* T' *and* $x_j \in P_j$ *in* T, *if* $h(y_i) = x_j$ *then* $\vec{x} \in \vec{P} \ \vdash \ C(\vec{x}) \wedge\!\!= \underline{\mathsf{eq}}(P_j,\, h(R_i))$ *and such that* $\vec{x} \in \vec{P} \ \vdash \ C(\vec{x}) \wedge\!\!= D(h(\vec{y}))$.

The following lemma relates valuations and homomorphisms and is essential for the proof of Theorem 6.7.

Lemma 6.6 *Let* $T_1 = \{\vec{x} \in \vec{P_1}; C_1(\vec{x})\}$ *and* $T_2 = \{\vec{y} \in \vec{P_2}; C_2(\vec{y})\}$ *be two tableaux. Assume that* $C_2(\vec{y})$ *is an inhabited formula over* $\{\vec{y} \in \vec{P_2}; \underline{\mathsf{true}}\}$. *Then, for any valuation* $v : T_2 \to \mathit{Inst}(T_1)$, *there exists a homomorphism* $h : T_2 \to T_1$. *In addition* v *and* $\underline{\mathsf{cval}} \circ h$ *satisfy the same set of inhabited formulas over* $\{\vec{y} \in \vec{P_2}; \underline{\mathsf{true}}\}$

Theorem 6.7 (Containment/Trivial dependencies)

1. *Let* Q_1, Q_2 *be set-valued PC queries. The following are equivalent:*

   (a1) $Q_1 \subseteq^{unr} Q_2$ (a2) $Q_1 \subseteq^{fin} Q_2$

   (b1) *cont*(Q_1, Q_2) *is trivial (unrestricted)*   (b2) *cont*(Q_1, Q_2) *is trivial (in the finite)*

   (c) *there exists* $\{\vec{x} \in \vec{P_1} \ ; \ C_1(\vec{x})\} \xleftarrow{h} \{\vec{y} \in \vec{P_2} \ ; \ C_2(\vec{x})\}$ *such that* $\vec{x} \in \vec{P_1} \ \vdash \ C_1(\vec{x}) \wedge\!\!= \underline{\mathsf{eq}}(P_1'(\vec{x}),\, P_2'(h(\vec{y})))$

   (d) *cont*(Q_1, Q_2) *(and therefore the containment) is provable in CoDi's equational theory*

   where $Q_1 = \underline{\mathsf{BigU}}\,(\vec{x} \in \vec{P_1})\,\underline{\mathsf{if}}\ C_1(\vec{x})\ \underline{\mathsf{then}}\ \underline{\mathsf{sng}}(P_1'(\vec{x}))$ *and* $Q_2 = \underline{\mathsf{BigU}}\,(\vec{y} \in \vec{P_2})\,\underline{\mathsf{if}}\ C_2(\vec{y})\ \underline{\mathsf{then}}\ \underline{\mathsf{sng}}(P_2'(\vec{y}))$.

2. *Let* d *be an EPCD. The following are equivalent:*

   (a1) d *is trivial (unrestricted)*   (a2) d *is trivial (in the finite)*

   (b1) *front*$(d) =^{unr}$ *back*(d) (b2) *front*$(d) =^{fin}$ *back*(d)

   (c) *there exists* $\{\vec{x} \in \vec{P_1} \ ; \ C_1(\vec{x})\} \xleftarrow{h} \{\vec{x} \in \vec{P_1}, \vec{y} \in \vec{P_2}(\vec{x}) \ ; \ C_1(\vec{x})\ \underline{\mathsf{and}}\ C_2(\vec{x}, \vec{y})\}$ *such that* $\vec{x} \in \vec{P_1} \ \vdash \ C_1(\vec{x}) \wedge\!\!= \underline{\mathsf{eq}}(\vec{x},\, h(\vec{x}))$

   (d) d *is provable in CoDi's equational theory*[11]

   where d *is* $\vec{x} \in \vec{P_1} \ \vdash \ C_1(\vec{x}) \wedge\!\!= \underline{\mathsf{Some}}\,(\vec{y} \in \vec{P_2}(\vec{x}))\, C_2(\vec{x}, \vec{y})$

Corollary 6.8 *Existence of a homomorphism of tableaux, and therefore containment/equivalence of set-valued PC query and EPCD triviality are decidable and in NP (and hence NP-complete by* [CM77]*).*

[11] We show this in appendix A

6.2 Terminating chase

The definition of the chase in section 4 was somewhat simplified by the coincidence of variable names. The general definition is given next. Note that chasing (d) mimics chasing *front*(d), that chasing *cont*(Q_1, Q_2) mimics chasing Q_1 and that in chasing $\underline{\mathsf{Loop}}\,(\vec{x} \in \vec{R})\ \underline{\mathsf{if}}\ B(\vec{x})\ \underline{\mathsf{then}}\ E(\vec{x})$ the chase only affects the *underlying* tableau $\{\vec{x} \in \vec{R}\ ;\ B(\vec{x})\}$. Therefore it suffices to define the chase on tableaux.

Definition 6.9 (Chase step) *Let (d) be the EPCD $\vec{r} \in \vec{R}\ \vdash\ B_1(\vec{r})\ \wedge\!\!=\ \underline{\mathsf{Some}}\,(\vec{s} \in \vec{S}(\vec{r}))\, B_2(\vec{r}, \vec{s})$ and T be the tableau $\{\vec{x} \in \vec{P}\ ;\ C(\vec{x})\}$. Suppose that there is $T \xleftarrow{h} \{\vec{r} \in \vec{R}; B_1(\vec{r})\}$ but there is no $T \xleftarrow{h'} T'$ such that $\vec{x} \in \vec{P}\ \vdash\ C(\vec{x})\ \wedge\!\!=\ \underline{\mathsf{eq}}(\vec{x},\ h'(\vec{x}))$ where $T' = \{\vec{x} \in \vec{P}, \vec{s} \in \vec{S}(h(\vec{r}))\ ;\ C(\vec{x})\ \underline{\mathsf{and}}\ B_2(h(\vec{r}), \vec{s})\}$. Then we say that (d) is* applicable *to T and chases it to T', written $T \xrightarrow{d} T'$. We also write $Q \xrightarrow{d} Q'$ and $d' \xrightarrow{d} d''$.*

Lemma 6.10 (Chase properties) (1) *If $Q \xrightarrow{d} Q'$ then $Q = Q'$ is provable from d in CoDi*[12]. (2) *If Inst$(T) \not\models d$ then d is applicable to T.*

Part (2) of the previous lemma allows us to observe that, for any *terminating* chase sequence $T = T_0 \longrightarrow \ldots \longrightarrow T_n$ of T by a set of EPCDs D (terminating means no d in D is applicable to T_n), $T_n \models D$. For any PC query $Q = \underline{\mathsf{Loop}}(T)P'$ we use the notation $chase_D(Q)$ for $\underline{\mathsf{Loop}}(T_n)P'$ (and similarly, we have $chase_D(d)$).

Theorem 6.11 (Containment with dependencies/Dependency implication) *Let D be a set of EPCDs.*

1. *Let Q_1, Q_2 be set-valued PC queries such that some chasing sequence of Q_1 with D terminates (with $chase_D(Q_1)$). The following are equivalent:*

(a1)	$Q_1 \subseteq^{unr}_D Q_2$	(a2)	$Q_1 \subseteq^{fin}_D Q_2$
(b1)	$chase_D(Q_1) \subseteq^{unr} Q_2$	(b2)	$chase_D(Q_1) \subseteq^{fin} Q_2$
(c1)	$chase_D(cont(Q_1, Q_2))$ *is trivial (unr)*	(c2)	$chase_D(cont(Q_1, Q_2))$ *is trivial (fin)*
(d1)	$D \models^{unr} cont(Q_1, Q_2)$	(d2)	$D \models^{fin} cont(Q_1, Q_2)$

   (e) *cont(Q_1, Q_2) (and therefore the containment) is provable from D in CoDi's equational theory*

2. *Let d be an EPCD such that some chasing sequence of d with D terminates. The following are equivalent:*

(a1)	$D \models^{unr} d$	(a2)	$D \models^{fin} d$
(b1)	$chase_D(d)$ *is trivial (unr)*	(b2)	$chase_D(d)$ *is trivial (fin)*
(c1)	$chase_D(front(d)) \subseteq^{unr} back(d)$	(c2)	$chase_D(front(d)) \subseteq^{fin} back(d)$
(d1)	$front(d) \subseteq^{unr} back(d)$	(d2)	$front(d) \subseteq^{fin} back(d)$

   (e) *d is provable from D in CoDi's equational theory*

Definition 6.12 (Full dependencies) *An EPCD $\vec{r} \in \vec{R}\ \vdash\ B_1(\vec{r})\ \wedge\!\!=\ \underline{\mathsf{Some}}\,(\vec{s} \in \vec{S}(\vec{r}))\, B_2(\vec{r}, \vec{s})$ is* full *if for any variable s_i in $\vec{s}$ there exists a path $P_i(\vec{r})$ such that $\vec{r} \in \vec{R}, \vec{s} \in \vec{S}(\vec{r})\ \vdash\ B_1(\vec{r})\ \underline{\mathsf{and}}\ B_2(\vec{r}, \vec{s})\ \wedge\!\!=\ \underline{\mathsf{eq}}(s_i,\ P_i(\vec{r}))$*

Theorem 6.13 *If D is a set of full EPCDs and T is a tableau then any chase of T by D terminates.*

Corollary 6.14 *Set-valued PC query containment/equivalence 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.*

Relational full/total tgds are full EPCDs. We conjecture that the complexity of the PC problem is exponential, hence not worse than in the relational case [BV84b, CLM81]. Note that EGDs are always full. It is easy to see for EGDs the problem is actually in PTIME, as in the relational case.

[12] See appendix A for proof

6.3 Non-terminating chase

We also generalize the results of [BV84b] for non-terminating chase, that is, we show that in the PC case the chase is still a proof procedure. As opposed to the relational case where one can also invoke Gödel's completeness theorem, the recursive enumerability of the PC problem was not obvious.

Let $Q = \underline{\mathsf{BigU}}(T)P'$ be a set-valued PC query and $dep(T,T')$ an EPCD where $T = \{\vec{x} \in \vec{P}\ ;\ C(\vec{x})\}$ and $T' = \{\vec{x} \in \vec{P}, \vec{y} \in \vec{R}(\vec{x}); C(\vec{x})\ \underline{\mathsf{and}}\ D(\vec{x},\vec{y})\}$. Suppose $T_m = \{\vec{x_m} \in \vec{P_m}; C_m(\vec{x_m})\}$ is the mth tableau in a chase sequence (not necessarily terminating) $T = T_0 \longrightarrow \ldots \longrightarrow T_n \longrightarrow \ldots$ of T by a set of EPCDs D. We use the notations:

$$chase^m_D(d) \stackrel{def}{=} dep(T_m, T'_m) \qquad chase^m_D(Q) \stackrel{def}{=} \underline{\mathsf{BigU}}(T_m)P'$$

where $T'_m = \{\vec{x_m} \in \vec{P_m}, \vec{y} \in \vec{R}(\vec{x});\ C_m(\vec{x_m})\ \underline{\mathsf{and}}\ D(\vec{x},\vec{y})\}$.

We show here that if $D \models dep(T,T')$ then for any infinite chase of T by D there is a tableau T_m (with m finite) in the chase such that $dep(T_m, T'_m)$ is trivial. A similar result holds for query containment/equivalence. The techniques and the results generalize the ones of [BV84b] regarding the relational case. We make the following assumptions:

1. every EPCD that is applicable infinitely many times should be applied infinitely many times (non-starvation of dependencies)
2. all path expressions are over a fixed, infinite, totally ordered and well-founded set of variables. Moreover, path expressions are not only totally ordered, but well-founded as well (this can be done by lifting the well-founded order on variables to path expressions).

Let (T) be an infinite chase sequence of T by a set of EPCDs D: $T_0 \longrightarrow \ldots \longrightarrow T_n \longrightarrow \ldots$. We define first an infinite tableau $T^\infty = \{\vec{x} \in \vec{P}; C(\vec{x})\}$ that satisfies the following:

1. for any prefix $\vec{x_n} \in \vec{P_n}$ of $\vec{x} \in \vec{P}$ there exists a tableau $T_m = \{\vec{x_m} \in \vec{P_m}; C_m(\vec{x_m})\}$ in (T) such that $\vec{x_n} \in \vec{P_n}$ is a prefix of $\vec{x_m} \in \vec{P_m}$
2. $C(\vec{x}) = \bigwedge_{T_m \in (T)} C_m(\vec{x_m})$ ($C(\vec{x})$ is an infinite conjunction)

Next, we define the canonical instance of T^∞, denote it by I^∞, as the limit of the sequence $(Inst(T_n))_{n\geq 0}$ (see appendix B for definition of $Inst(T_n)$). For any finite path expression Q over T^∞, it must be the case that Q is defined over T_m in (T), for some finite m. Consider the sequence $\underline{\mathsf{cval}}^{(n)}_\emptyset(Q)$, for all $n \geq m$. $\underline{\mathsf{cval}}^{(n)}_\emptyset(Q)$ is the smallest path expression in the congruence class of Q with respect to T_n. One can see that $\underline{\mathsf{cval}}^{(n+1)}_\emptyset(Q)$ is either identical to $\underline{\mathsf{cval}}^{(n)}_\emptyset(Q)$ (if the congruence class of Q w.r.t T_n remains the same in T_{n+1} or is unioned with other congruence classes but the smallest element doesn't change) or smaller than $\underline{\mathsf{cval}}^{(n)}_\emptyset(Q)$ (the congruence class of Q w.r.t T_n is unioned with other congruence classes and the smallest element does change). By our assumption of well-foundedness, there must exist a $p \geq m$ s.t. $\underline{\mathsf{cval}}^{(p)}_\emptyset(Q) = \underline{\mathsf{cval}}^{(p+1)}_\emptyset(Q) = \ldots$. Define $\underline{\mathsf{cval}}_\emptyset(Q) = \underline{\mathsf{cval}}^{(p)}_\emptyset(Q)$. We can verify that the graph induced by $\underline{\mathsf{cval}}_\emptyset$ preserves the context $\vec{x} \in \vec{P}$ of T and moreover satisfies $C(\vec{x})$. Finally, we collapse the empty sets, to obtain I^∞. As in the finite case one can show that $I^\infty \models D$.

Theorem 6.15 *(Containment with dependencies/Dependency implication) Let D be a set of EPCDs.*

1. *Let Q_1, Q_2 be set-valued PC queries and consider an arbitrary infinite chasing sequence of Q_1 with D. The following are equivalent:*

   (a) $Q_1 \subseteq^{unr}_D Q_2$

   (b) *there is a finite m such that:*

   (1) $chase^m_D(Q_1) \subseteq^{unr} Q_2$ and/or (2) $chase^m_D(cont(Q_1,Q_2))$ is trivial (unr)

   (c) $D \models^{unr} cont(Q_1, Q_2)$

   (d) *$cont(Q_1,Q_2)$ (and therefore the containment) is provable from D*

2. *Let d be an EPCD and consider an arbitrary infinite chasing sequence of d with D. The following are equivalent:*

   (a) $D \models^{unr} d$

   (b) *there is m finite such that:*

   (1) $chase_D^m(d)$ is trivial (unr) and/or (2) $chase_D^m(front(d)) \subseteq^{unr} back(d)$

   (c) $front(d) \subseteq^{unr} back(d)$

   (d) *d is provable from D*

6.4 Disjunction aggregates

Parts (1) of theorems 6.7, 6.11 and 6.15 also hold, with similar proofs, for boolean-valued-$\underline{\mathsf{Some}}$ PC queries, where containment means boolean implication. Alternatively, we can give a more elegant proof of this by observing the following reduction from $\underline{\mathsf{Some}}$ query containment/equivalence to $\underline{\mathsf{BigU}}$ query containment/equivalence. Each disjunction aggregate query $Q = \underline{\mathsf{Some}}\,(\vec{r} \in \vec{\mathrm{R}})\, B(\vec{r})$ has a corresponding set-valued PC query $Q' = \underline{\mathsf{BigU}}\,(\vec{r} \in \vec{\mathrm{R}})\, \underline{\mathsf{if}}\ B(\vec{r})\ \underline{\mathsf{then}}\ \underline{\mathsf{sng}}\langle\rangle$ such that Q evaluates to $\underline{\mathsf{true}}$ if and only if Q' evaluates to $\underline{\mathsf{sng}}\langle\rangle$. (This is an immediate consequence of idemloop).

7 Related work and further investigations

Related work The monad algebra approach to aggregates 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. A different approach based on parameterized algebraic specifications appears in [BTS93]. 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.

Arrays, as dealt with in [LMW96] can be formalized as dictionaries, given some arithmetic and operations that produce integer intervals. In [DHP97] the Kleisli/CPL system is extended to represent and query oodbs, specifically Shore. The ideas used there can be represented with dictionaries, but dictionaries are more flexible. The maps of [ALPR91], the treatment of object types in [BK93] and that of views in [dSDA94] are related to our use of dictionaries. An important difference is made by the operations on dictionaries used here.

Our PC queries are less general than COQL queries [LS97], by not allowing alternations of conditionals and $\underline{\mathsf{BigU}}$. However they are more general in other ways, by incorporating dictionaries and allowing equalities beyond base type. 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 class inheritance constraints [Cha92], containment under Datalog-expressible constraints and views [DS96], equivalence between queries with set and bag aggregates [NSS98], 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.

Davidson and Hara [HD98] consider generalized functional dependencies for complex value schemas. Their main objective is an intrinsic axiomatization of such dependencies. Our paper does not examine at all the problem of intrinsic axiomatizations [BV84a]. Fan and Weinstein [FW98] examine the un/decidability of logical implication for path constraints in various classes oo-typed semistructured models. Path constraints are first-order expressible and are both weaker than our EPCDs in some respects (cannot express (NON-EMPTY) for instance) and probably stronger in other respects (they allow more quantifier nesting).

Further investigations We conjecture that the restriction to inhabited dependencies could be totally or partially removed. This may be also related to the restriction to *weak* equivalence in [LS97]. 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. We conjecture that confluence and semantic invariance of the chase generalizes from the relational case to full EPCDs. 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]. 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. Finally, CoDi is an equational rendition of a ramified higher-order logic and the question arises if it is related to a weak form of *topos theory* [FS90].

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.

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.

[AK89] S. Abiteboul and P. Kanellakis. Object identity as a query language primitive. In *Proceedings of ACM SIGMOD Conference on Management of Data*, pages 159–173, Portland, Oregon, 1989.

[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.

[BK90] Catriel Beeri and Yoram Kornatzky. Algebraic optimisation of object oriented query languages. In S. Abiteboul and P. C. Kanellakis, editors, *LNCS 470: 3rd International Conference on Database Theory, Paris, France, December 1990*, pages 72–88, Berlin, December 1990. Springer-Verlag.

[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 proramming with collection types. *Theoretical Computer Science*, 149:3–48, 1995.

[BTS93] C. Beeri and P. Ta-Shma. Bulk data types, a theoretical approach. In Catriel Beeri, Atsushi Ohori, and Dennis E. Shasha, editors, *Proceedings of 4th International Workshop on Database Programming Languages, New York, August 1993*, pages 80–96, New York City, 1993. Springer-Verlag.

[BV84a] Catriel Beeri and Moshe Y. Vardi. Formal systems for tuple and equality generating dependencies. *SIAM Journal of Computing*, 13(1):76–98, 1984.

[BV84b] 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.

[CGMH+94] S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, and J. Widom. The tsimmis project: Integration of heterogeneous information sources. In *Proc. of IPSJ*, Tokyo, Japan, October 1994.

[Cha92] Edward P. F. Chan. Containment and minimization of positive conjunctive queries in oodb's. In *Proc. 11th ACM Symposium on Principles of Database Systems*, pages 202–211, 1992.

[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.

[CM93] S. Cluet and G. Moerkotte. Nested queries in object bases. In *Proc. DBPL*, pages 226–242, 1993.

[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 ??–??, Montreal, Quebec, Canada, 1996.

[DHP97] S. B. Davidson, C. Hara, and L. Popa. Querying an object-oriented databases using cpl. Technical Report, To appear in Proc???, Brazil MS-CIS-97-07, University of Pennsylvania, December 1997.

[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.

[FS90] Peter J. Freyd and Andre Scedrov. *Categories, Allegories.* North-Holland, Amsterdam, 1990.

[FW98] Wenfei Fan and Scott Wenstein. Interaction between path and type constraints, June 1998. Manuscript available from `wfan@saul.cis.upenn.edu`.

[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 conjuctive queries under functional and inclusion dependencies. *Journal of Computer and System Sciences*, 28:167–189, 1984.

[Kos95] Anthony Kosky. Observational properties of databases with object identity. Technical Report MS-CIS-95-20, Dept. of Computer and Information Science, University of Pennsylva nia, 1995.

[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.

[LT97] Kazem Lellahi and Val Tannen. A calculus for collections and aggregates. In E. Moggi and G. Rosolini, editors, *LNCS 1290: Category Thory 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.

[NSS98] Werner Nutt, Yehoshua Sagiv, and Sara Shurin. Deciding equivalences among aggregate queries. In *Proc. 17th ACM Symposium on Principles of Database Systems*, pages 214–223, Seattle, Washington, June 1998.

[QR95] X. Qian and L. Raschid. Query interoperation among object-oriented and relational databases. In *Proc. ICDE*, 1995.

[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.

[SZ90] G. Shaw and S. Zdonik. A query algebra for object-oriented databases. In *Proc. IEEE Conference on Data Engineering*, pages 154–162, 1990.

[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.

A Equational axiomatization

In addition to the laws in figure 2, the axiomatization of the CoDi equational theory consists of the following:

1. **(exists) and (all)**:

$$\text{(exists)}\quad \Gamma, x \in \mathtt{R} \ \vdash\ \underline{\textsf{true}} = \underline{\textsf{Some}}\,(y \in \mathtt{R})\, \underline{\textsf{eq}}(y,\, x) \qquad\qquad \text{(all)}\ \frac{\Gamma\ \vdash\ \underline{\textsf{All}}\,(\vec{x} \in \vec{\mathtt{R}})\, B = \underline{\textsf{true}}}{\Gamma, \vec{x} \in \vec{\mathtt{R}}\ \vdash\ B = \underline{\textsf{true}}}$$

2. **Record axioms**:

$$\text{(rcd-proj)}\ \ \Gamma \vdash \langle A_1 : E_1, \ldots, A_n : E_n \rangle.A_i = E_i \qquad \text{(rcd-surj)}\ \ \Gamma \vdash E = \langle E.A_1, \ldots, E.A_n \rangle$$

3. **Conditional axioms**:

$$\text{(cond-nest)}\ \ \Gamma \vdash \underline{\textsf{if}}\,[\alpha]\, B_1\ \underline{\textsf{then}}\ \underline{\textsf{if}}\,[\alpha]\, B_2\ \underline{\textsf{then}}\ E = \underline{\textsf{if}}\,[\alpha]\, B_1\ \underline{\textsf{and}}\ B_2\ \underline{\textsf{then}}\ E$$

$$\text{(cond-true)}\ \ \Gamma \vdash \underline{\textsf{if}}\,[\alpha]\, \underline{\textsf{true}}\ \underline{\textsf{then}}\ E = E$$

$$\text{(eqcond)}\ \ \Gamma \vdash \underline{\textsf{if}}\,[\alpha]\, \underline{\textsf{eq}}(E_1,\, E_2)\ \underline{\textsf{then}}\ E(E_1) = \underline{\textsf{if}}\,[\alpha]\, \underline{\textsf{eq}}(E_1,\, E_2)\ \underline{\textsf{then}}\ E(E_2)$$

$$\text{(cond-loop1)}\ \ \Gamma \vdash \underline{\textsf{Loop}}\,[\alpha]\,(x \in \underline{\textsf{if}}\,[\textsf{free}]\, B\ \underline{\textsf{then}}\ S)\, E(x) = \underline{\textsf{if}}\,[\alpha]\, B\ \underline{\textsf{then}}\ \underline{\textsf{Loop}}\,[\alpha]\,(x \in S)\, E(x)$$

$$\text{(cond-loop2)}\ \ \Gamma \vdash \underline{\textsf{Loop}}\,[\alpha]\,(x \in S)\, \underline{\textsf{if}}\,[\alpha]\, B\ \underline{\textsf{then}}\ E(x) = \underline{\textsf{if}}\,[\alpha]\, B\ \underline{\textsf{then}}\ \underline{\textsf{Loop}}\,[\alpha]\,(x \in S)\, E(x)$$

4. $\underline{\textsf{and}}$ **rules**:

$$(\ \underline{\textsf{and}}\ \text{-assoc})\ \ \Gamma \vdash (B_1\ \underline{\textsf{and}}\ B_2)\ \underline{\textsf{and}}\ B_3 = B_1\ \underline{\textsf{and}}\ (B_2\ \underline{\textsf{and}}\ B_3)$$

$$(\ \underline{\textsf{and}}\ \text{-comm})\ \ \Gamma \vdash B_1\ \underline{\textsf{and}}\ B_2 = B_2\ \underline{\textsf{and}}\ B_1 \qquad (\ \underline{\textsf{and}}\ \text{-idemp})\ \ \Gamma \vdash B\ \underline{\textsf{and}}\ B = B$$

$$(\ \underline{\textsf{and}}\ \text{-cond})\ \ \Gamma \vdash B_1\ \underline{\textsf{and}}\ B_2 = \underline{\textsf{if}}\ B_1\ \underline{\textsf{then}}\ B_2\ \underline{\textsf{else}}\ \underline{\textsf{false}}$$

5. $\underline{\textsf{eq}}$ **rules**:

$$\text{(refl)}\ \ \Gamma \vdash \underline{\textsf{eq}}(E,\, E) = \underline{\textsf{true}} \qquad \frac{\Gamma \vdash E_1 = E_2}{\Gamma \vdash \underline{\textsf{eq}}(E_1,\, E_2) = \underline{\textsf{true}}} \qquad \frac{\Gamma \vdash \underline{\textsf{eq}}(E_1,\, E_2) = \underline{\textsf{true}}}{\Gamma \vdash E_1 = E_2}$$

6. **Implication rules**:

$$\frac{\Gamma \vdash B_1 \wedge\!\!= B_2}{\Gamma \vdash \underline{\text{if}}\ B_1\ \underline{\text{then}}\ B_2\ \underline{\text{else}}\ \underline{\text{true}} = \underline{\text{true}}} \qquad \frac{\Gamma \vdash \underline{\text{if}}\ B_1\ \underline{\text{then}}\ B_2\ \underline{\text{else}}\ \underline{\text{true}} = \underline{\text{true}}}{\Gamma \vdash B_1 \wedge\!\!= B_2}$$

7. **Congruence rules:**

$$(\underline{\text{Loop}}\text{-cong})\ \frac{\Gamma, x \in \mathtt{R}\ \vdash\ E_1 = E_2}{\Gamma\ \vdash\ \underline{\text{Loop}}\,[\alpha]\,(x \in \mathtt{R})\,E_1 = \underline{\text{Loop}}\,[\alpha]\,(x \in \mathtt{R})\,E_2} \qquad (\underline{\text{sng}}\text{-cong})\ \frac{\Gamma\ \vdash\ E_1 = E_2}{\Gamma\ \vdash\ \underline{\text{sng}}E_1 = \underline{\text{sng}}E_2}$$

$$(\text{dict-cong})\ \frac{\Gamma\ \vdash\ \mathtt{R}_1 = \mathtt{R}_2 \qquad \Gamma, k \in \mathtt{R}_1\ \vdash\ E_1 = E_2}{\Gamma\ \vdash\ \underline{\text{key}}\ k\ \underline{\text{in}}\ \mathtt{R}_1\ \Rightarrow\ E_1 = \underline{\text{key}}\ k\ \underline{\text{in}}\ \mathtt{R}_2\ \Rightarrow\ E_2} \qquad (!\text{-cong})\ \frac{\Gamma\ \vdash\ M_1 = M_2}{\Gamma, k \in \underline{\text{dom}}\,M_1\ \vdash\ k!\,M_1 = k!\,M_2}$$

$$(\underline{\text{dom}}\text{-cong})\ \frac{\Gamma\ \vdash\ M_1 = M_2}{\Gamma\ \vdash\ \underline{\text{dom}}\,M_1 = \underline{\text{dom}}\,M_2} \qquad (\text{cond-cong})\ \frac{\Gamma\ \vdash\ B_1 = B_2 \qquad \Gamma\ \vdash\ E_1 = E_2}{\Gamma\ \vdash\ \underline{\text{if}}\ B_1\ \underline{\text{then}}\ E_1 = \underline{\text{if}}\ B_2\ \underline{\text{then}}\ E_2}$$

$$(\text{rcd-cong})\ \frac{\Gamma\ \vdash\ E_1 = E_1' \ \ldots\ \Gamma\ \vdash\ E_n = E_n'}{\Gamma\ \vdash\ \langle A_1 : E_1, \ldots, A_n : E_n\rangle = \langle A_1 : E_1', \ldots, A_n : E_n'\rangle} \qquad (\text{prj-cong})\ \frac{\Gamma\ \vdash\ E_1 = E_2}{\Gamma\ \vdash\ E_1.A_i = E_2.A_i}$$

Some obvious rules such as symmetry and transitivity of equality are missing because they are derivable largely due to (eqcond).

Trivial EPCDs are provable. The following inference rule is derivable (even without the PC restriction):

$$(\text{triviality})\ \frac{\vec{x} \in \vec{P}\ \vdash\ C(\vec{x})\ \wedge\!\!= \underline{\text{eq}}(P_j,\, h(R_i))\ \ \underline{\text{and}}\ \ \underline{\text{eq}}(P_l,\, h(P_k))\ \ \underline{\text{and}}\ \ \underline{\text{eq}}(\vec{x},\, h(\vec{x}))\ \ \underline{\text{and}}\ \ D(\vec{x}, h(\vec{y}))}{\vec{x} \in \vec{P}\ \vdash\ C(\vec{x})\ \wedge\!\!= \underline{\text{Some}}\,(\vec{y} \in \vec{R}(\vec{x}))\,D(\vec{x}, \vec{y})}$$

where h is a mapping from variables $\{\vec{x}, \vec{y}\}$ into variables $\{\vec{x}\}$ such that $h(y_i) = x_j$ and $h(x_k) = x_l$. To derive the rule we infer first, by (exists), $\vec{x} \in \vec{P} \vdash \underline{\text{Some}}\,(y_i \in P_j)\,\underline{\text{eq}}(y_i,\, h(y_i)) = \underline{\text{true}}$, for any y_i. Then we use the premises and, mainly, congruences, ($\underline{\text{and}}$ -cond) and (eqcond), to bring in $C(\vec{x})$, and then to replace each P_j with $h(R_i)$, and then $h(\vec{x})$ with $\vec{x}$ and $h(\vec{y})$ with $\vec{y}$. This proves (c)$\Rightarrow$(d) in Theorem 6.7.

Provability of the chase step. Let $Q = \underline{\text{Loop}}\,(\vec{x} \in \vec{P})\,\underline{\text{if}}\ C(\vec{x})\ \underline{\text{then}}\ E$ and d and h as in the definition of the chase step (see Section 6). Observe that h extended to be the identity on $\vec{x}$ is a homomorphism from $\{\vec{x} \in \vec{P}, \vec{r'} \in \vec{R}; \underline{\text{eq}}(\vec{r'},\, h(\vec{r}))\}$ into $\{\vec{x} \in \vec{P}; \underline{\text{true}}\}$ such that it satisfies the condition of Theorem 6.7. Therefore $\vec{x} \in \vec{P} \vdash \underline{\text{true}} = \underline{\text{Some}}\,(\vec{r'} \in \vec{R})\,\underline{\text{eq}}(\vec{r'},\, h(\vec{r}))$ is trivial, hence provable. Since $\vec{x} \in \vec{P} \vdash C(\vec{x}) \wedge\!\!= B_1(h(\vec{r}))$ is valid (h is a homomorphism) and, therefore provable, we can rewrite $\underline{\text{Loop}}\,(\vec{x} \in \vec{P})\,\underline{\text{if}}\ C(\vec{x})\ \underline{\text{then}}\ E$ to

$$\underline{\text{Loop}}\,(\vec{x} \in \vec{P})\,\underline{\text{if}}\ C(\vec{x})\ \underline{\text{and}}\ B_1(h(\vec{r}))\ \underline{\text{and}}\ \underline{\text{Some}}\,(\vec{r'} \in \vec{R})\,\underline{\text{eq}}(\vec{r'},\, h(\vec{r}))\ \underline{\text{then}}\ E$$

Rewrites with (idemloop), (eqcond), and then d and idemloop, yield:

$$\underline{\text{Loop}}\,(\vec{x} \in \vec{P})\,\underline{\text{Loop}}\,(\vec{r'} \in \vec{R})\,\underline{\text{Loop}}\,(\vec{s} \in \vec{S}(\vec{r'}))\,\underline{\text{if}}\ C(\vec{x})\ \underline{\text{and}}\ B_1(\vec{r'})\ \underline{\text{and}}\ B_2(\vec{r'}, \vec{s})\ \underline{\text{and}}\ \underline{\text{eq}}(\vec{r'},\, h(\vec{r}))\ \underline{\text{then}}\ E$$

Applying (cond-loop2) we move $\underline{\text{eq}}(\vec{r'},\, h(\vec{r}))$ outside of the loop over $\vec{s}$ and apply (eqcond) to replace occurrences of $\vec{r'}$ with $h(\vec{r})$. Then a step of tableau minimization with $\vec{x} \in \vec{P} \vdash \underline{\text{true}} = \underline{\text{Some}}\,(\vec{r'} \in \vec{R})\,\underline{\text{eq}}(\vec{r'},\, h(\vec{r}))$, followed by replacing $C(\vec{x})$ $\underline{\text{and}}$ $B_1(h(\vec{r}))$ with $C(\vec{x})$ yields $\underline{\text{Loop}}\,(\vec{x} \in \vec{P})\,\underline{\text{Loop}}\,(\vec{s} \in \vec{S}(h(\vec{r})))\,\underline{\text{if}}\ C(\vec{x})\ \underline{\text{and}}\ B_2(h(\vec{r}), \vec{s})\ \underline{\text{then}}\ E$, the query Q' that we wanted. This proves lemma 6.10 (1)

B A canonical instance construction

We associate to each tableau $T = \{\vec{x} \in \vec{P}\ ;\ C(\vec{x})\}$ a special instance, *Inst*(T), crucial for proving our decidability and completeness results. We also use *Inst*(T) to define the class of inhabited EPCDs. Intuitively, *Inst*(T) is the minimal instance that contains the "structure" of T, and it allows us to express syntactical conditions on T as necessary and sufficient conditions on *Inst*(T). The construction is sketched next:

1) we built a directed acyclic graph $G(T)$: start with a set of nodes, V, containing one node for each path expression P occuring in T. Close this set under the operations $P.A, x!\,P$, $\underline{\mathsf{dom}}\,P$, R and $\underline{\mathsf{null}}_\alpha$. More precisely: if $P : \langle A_1 : \tau_1, \ldots, A_n : \tau_n\rangle$ is in V then set $V = V \cup \{p.A_1, \ldots, p.A_n\}$. Similarly, for $x!\,P$ and for $\underline{\mathsf{dom}}\,P$. For each name R in the schema, set $V = V \cup \{R\}$. Finally, $V = V \cup \{\underline{\mathsf{null}}_\alpha\}$. Next, we add edges between nodes in V in the natural way: for any $P.A$ in V, add an (unlabeled) edge from P into $P.A$. Similarly for $\underline{\mathsf{dom}}\,P$. For $x!\,P$ in V add an edge from x into $x!\,P$ and an edge from P into $x!\,P$. Finally, we populate set values: we add for each $x \in P$ occuring in T an edge labeled with $\in$ from P into x.

2) construct the congruence closure of $G(T)$ with respect to $C(\vec{x})$ and the normal congruence rules for $P.A, x!\,P, \underline{\mathsf{dom}}\,P$, $\underline{\mathsf{sng}}P$ and $\langle A_1 : P_1, \ldots, A_n : P_n\rangle$. Start with a partition of the nodes of $G(T)$ into classes, by putting nodes P_1 and P_2 in the same class whenever $\underline{\mathsf{eq}}(P_1, P_2)$ occurs in $C(\vec{x})$. Then coarsen this partition by collapsing classes through application of congruence rules, reflexivity, symmetry and transitivity. The process ends in polynomial time with a partition of $G(T)$ into classes, corresponding to the minimal congruence relation on G that satisfies $C(\vec{x})$. Each congruence class becomes a node in a new graph, $G(T)_{/C(\vec{x})}$. Add an edge from a node $[P_1, \ldots, P_n]$ into a node $[Q_1, \ldots, Q_k]$, if there is at least one edge from some P_i into some Q_j in $G(T)$.

3) $G(T)_{/C(\vec{x})}$ has all the properties to be a valid instance with one exception: there may be distinct nodes of set type $S_1, \ldots, S_n$, such that the $\in$-edges for all S_i's go into the same set of nodes, $\{e_1, \ldots, e_m\}$). Thus, $G(T)_{/C(\vec{x})}$ does not satisfy the extensionality property of sets. Our construction considers the two possible cases: First, $m > 0$, i.e. $S_1, \ldots, S_n$ are not empty. We simply add a new, distinct, node (and an $\in$-edge) to each S_i. Second case: $m = 0$, i.e. $S_1, \ldots, S_n$ are empty (one of them always comes from a $\underline{\mathsf{null}}_\alpha$). Call the graph obtained until now $\mathit{Inst}_\emptyset(T)$. Then *Inst*($T$) is obtained from $\mathit{Inst}_\emptyset(T)$ by identifying $S_1, \ldots, S_n$ (we also make sure that we close the result under the congruence rule for record constructors). For technical reasons, we identify each congruence class $[P_1, \ldots, P_n]$ with its smallest element P_i (we can always impose a total order on path expressions).

Note that *Inst*(T) is completely determined only up to the new nodes introduced in the last stage. The reason for not allowing path expressions of finite set type to occur in a binding becomes apparent from the construction: in that case, *Inst*(T) could have, for example, a node S of type $\{\mathsf{bool}\}$ with more than two distinct members! It is easy to see that there are two canonical mappings, $\underline{\mathsf{cval}}_\emptyset : T \to \mathit{Inst}_\emptyset(T)$, and $\underline{\mathsf{cval}} : T \to \mathit{Inst}(T)$, associating to each path expression occuring in T nodes in $\mathit{Inst}_\emptyset(T)$ and, respectively, *Inst*(T). Both $\underline{\mathsf{cval}}_\emptyset$ and $\underline{\mathsf{cval}}$ are naturally extended on path conjunctions, as well. It is then the case that $\underline{\mathsf{cval}}_\emptyset(C(\vec{x})) = \underline{\mathsf{cval}}(C(\vec{x})) = \text{true}$, i.e. each has the properties of a *valuation*.