Chase and Axioms for PC Queries and Dependencies

Lucian Popa and Val Tannen

University of Pennsylvania,

Department of Computer and Information Science,

200 South 33rd Street, Philadelphia, PA 19104, USA

E-mail: {lpopa, val}@saul.cis.upenn.edu

January 8, 1999

1 Overview

This report is the extended version of [PT99] and while they are some overlapping parts (mainly the results) we don't repeat here some of the examples given in [PT99], even though we frequently refer to them. The overview of this report is as follows. In section 2 we present some aspects of our internal framework, called CoDi [1]. This is a language and equational theory that combines a treatment of dictionaries with our previous work [BBW92, BNTW95, LT97] on collections and aggregates using the theory of *monads*. In addition we use dictionaries (finite functions), which allow us to represent oodb schemas and queries. While here we focus on set-related queries, we show elsewhere[2] that the (full) CoDi collection, aggregation and dictionary primitives suffice for implementing the quasi-totality of ODMG/ODL/OQL [Cat96]. Using boolean aggregates, CoDi can represent constraints as equalities between boolean-valued queries. We also give the basic equational laws of CoDi while the rest of the axiomatization can be found in section 11.

In section 3 we give a simple form for the equational chase which connects dependencies to query equivalence (and therefore containment via intersection). We also show there how we can 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 that the relational chase of [ABU79, MMS79, BV84b] is a particular case of our equational chase. In section 4 we offer examples of dependencies and equivalences beyond the relational model on which we use the equational chase defined earlier. One example captures a many-many inverse relationship in an OO schema, while in [PT99] we showed an example of an inverse between a class and a relation. In the same section we point out that view definitions can be equivalently characterized by dependencies and, in the special case of *path-conjunctive* (PC) views, the dependencies are path-conjunctive as well (EPCDs).

In section 5 we exhibit a class of queries and dependencies on complex values with dictionaries called path-conjunctive (PC queries and embedded PC dependencies (EPCDs)) for which the methods illustrated in earlier sections are complete, and in certain cases decidable. Section 6 gives the technical background that will be used throughout the rest of the paper. It describes well-defined path expressions, well-defined tableaux, and gives the construction of the canonical instance associated to each tableau. Theorem 7.9 in section 7 extends and generalizes the containment decidability/NP result of [CM77]. Theorem 7.11 in the same section shows that triviality of EPCDs is decidable and NP-complete. We also show in section 7 that PC query containment is

[1] From "**Co**llections and **Di**ctionaries"

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

reducible to EPCD triviality and they can both be characterized by homomorphisms of tableaux. Theorem 8.6 in section 8 extends and generalizes the corresponding results on the terminating chase in [BV84b]. A class of EPCDs that generalizes the total tgds of [BV84b] and that we call *full EPCDs* is defined in section 9. We show there that chasing with full dependencies always terminates(theorem 9.7), thus generalizing the result of [BV84b], moreover within the same complexity upper bound. In the same section we show that some nice properties that the relational chase has in the presence of total tgds still hold for the PC chase with full EPCDs: confluence and semantic invariance. We prove in section 10 that even when the chase does not terminate, it still a proof procedure for dependency implication and query containment/equivalence. This generalizes the similar first-order result of [BV84b]. We also argue that the all the results that we proved for set-valued PC queries apply for disjunction aggregates as well. Section 11 gives the rest of the axiomatization of the equational theory of CoDi and we show that it is complete for proving PC containment/EPCD triviality and EPCD implication/PC containment under dependencies (the latter by showing how to equationally simulate the PC chase).

Finally, we point out that the definition for full EPCDs that we give here is a correction of the erroneous one given in [PT99] (see footnote 19). Also, the complexity formula for dependency implication in the presence of full EPCDs, given in [PT99], is corrected in this report (see footnote 22).

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 8) 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 don't 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

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 $\sigma \gg \tau$ 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 $\mathtt{A}$ (need not be a key). The following dictionary is a logical level representation of a secondary index for $\mathtt{R}$ built on $\mathtt{A}$:

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

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.

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 $\bar{E}\,!\,M.\mathtt{A}$.

Example. In the following we will refer to the same running example that is used in [PT99]. The ODL definitions and all the other names can be found there. The type specs of the Proj-Dept schema translate in CoDi as

$$\mathtt{Proj}\ :\ \{\langle \mathtt{PName} : \mathsf{string}, \mathtt{CustName} : \mathsf{string}, \mathtt{PDept} : \mathsf{string}\rangle\}$$

$$\mathtt{Dept}\ :\ \mathit{Doid} \gg \langle \mathtt{DName} : \mathsf{string}, \mathtt{DProjs} : \{\mathsf{string}\}, \mathtt{MgrName} : \mathsf{string}\rangle$$

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.

the query ABC_DP translates in CoDi as the expression

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

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

$$\begin{array}{l}\underline{\mathsf{All}}\,(p \in \mathtt{Proj})\,\underline{\mathsf{All}}\,(d \in \underline{\mathsf{dom}}\,\mathtt{Dept})\\ \quad \underline{\mathsf{if}}\ \underline{\mathsf{eq}}(p.\mathtt{PDept},\, d\,!\,\mathtt{Dept.DName})\ \underline{\mathsf{then}}\ \underline{\mathsf{Some}}\,(s \in d\,!\,\mathtt{Dept.DProjs})\,\underline{\mathsf{eq}}(p.\mathtt{PName},\, s)\\ \qquad\qquad\qquad = \ \underline{\mathsf{true}}\end{array}$$

Expressing constraints this way will allow us to achieve both our objectives: rewriting queries (and constraints!) with constraints and composing views with constraints.

Equivalence laws In figure 1 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 1. 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 section 11. Some laws, such as (commute) and (from section 11) 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 in our examples 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

$$\text{(subst)} \qquad \Gamma, x \in S \ \vdash \ E_1(x) = E_2(x) \qquad \text{and} \qquad \Gamma \ \vdash \ \underline{\mathsf{member}}(E, S) = \underline{\mathsf{true}} \qquad \text{implies} \qquad \Gamma \ \vdash \ E_1(E) = E_2(E)$$

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

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

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

Example. We show how we obtained (FD-RETRO) from (FD) (again see [PT99]). First, we express UJP_View in CoDi:

$$\begin{array}{l}\underline{\mathsf{BigU}}\,(p \in \mathtt{Proj})\,\underline{\mathsf{BigU}}\,(d \in \underline{\mathsf{dom}}\,\mathtt{Dept})\,\underline{\mathsf{BigU}}\,(s \in d\,!\,\mathtt{Dept.DProjs})\\ \quad \underline{\mathsf{if}}\ \underline{\mathsf{eq}}(s,\, p.\mathtt{PName})\ \underline{\mathsf{then}}\ \underline{\mathsf{sng}}\langle \mathtt{PN} : p.\mathtt{PName}, \mathtt{CN} : p.\mathtt{CustName}, \mathtt{MN} : d\,!\,\mathtt{Dept.MgrName}\rangle\end{array}$$

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

Figure 1: Equivalence laws

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

$$
\begin{array}{l}
\underline{\mathsf{All}}\,(p \in \mathtt{Proj})\,\underline{\mathsf{All}}\,(d \in \underline{\mathsf{dom}}\,\mathtt{Dept})\,\underline{\mathsf{All}}\,(s \in d\,!\,\mathtt{Dept.DProjs}) \\
\underline{\mathsf{All}}\,(v \in \underline{\mathsf{if}}\ \underline{\mathsf{eq}}(s,\, p.\mathtt{PName})\ \underline{\mathsf{then}}\ \underline{\mathsf{sng}}\langle \mathtt{PN} : p.\mathtt{PName}, \mathtt{CN} : p.\mathtt{CustName}, \mathtt{MN} : d\,!\,\mathtt{Dept.MgrName}\rangle) \\
\underline{\mathsf{All}}\,(v' \in \mathtt{UJP\_View})\,\underline{\mathsf{if}}\ \underline{\mathsf{eq}}(v.\mathtt{PN},\, v'.\mathtt{PN})\ \underline{\mathsf{then}}\ \underline{\mathsf{eq}}(v.\mathtt{MN},\, v'.\mathtt{MN}) \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad = \ \underline{\mathsf{true}}
\end{array}
$$

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

$$
\begin{array}{l}
\underline{\mathsf{All}}\,(p \in \mathtt{Proj})\,\underline{\mathsf{All}}\,(d \in \underline{\mathsf{dom}}\,\mathtt{Dept})\,\underline{\mathsf{All}}\,(s \in d\,!\,\mathtt{Dept.DProjs})\,\underline{\mathsf{All}}\,(v' \in \mathtt{UJP\_View}) \\
\qquad \underline{\mathsf{if}}\ \underline{\mathsf{eq}}(s,\, p.\mathtt{PName})\ \underline{\mathsf{and}}\ \underline{\mathsf{eq}}(p.\mathtt{PName},\, v'.\mathtt{PN})\ \underline{\mathsf{then}}\ \underline{\mathsf{eq}}(d\,!\,\mathtt{Dept.MgrName},\, v'.\mathtt{MN}) \\
= \ \underline{\mathsf{true}}
\end{array}
$$

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

3 Equational Chase

The form in which we have written the constraints turns out to be convenient if we want them rewritten but not if we want to rewrite *with* them. In order to replace equals by equals within the scope of $\underline{\mathsf{Loop}}(\vec{x} \in \vec{S})$ bindings, we will use equations of the form (already mentioned in section 2) $\Gamma \vdash E_1 = E_2$, where $\Gamma = \vec{s} \in \vec{S}$. The congruence rules for the $\underline{\mathsf{Loop}}$ construct together with additional rules that can be found in section 11 allow us to rewrite the set membership conditions within the scope of $\underline{\mathsf{All}}$ and vice-versa. Thus we have the following:

Lemma 3.1 (Two Forms for Constraints) *The following two equations are derivable from each other in CoDi's equational theory (for the proof see section 11):*

$$\underline{\mathsf{All}}\,(\vec{x} \in \vec{S})\,\underline{\mathsf{if}}\ B(\vec{x})\ \underline{\mathsf{then}}\ C(\vec{x}) \;=\; \underline{\mathsf{true}}$$

$$\vec{x} \in \vec{S} \;\vdash\; B(\vec{x}) \;=\; B(\vec{x})\ \underline{\mathsf{and}}\ C(\vec{x})$$

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

$$(d) \qquad \vec{x} \in \vec{R_1} \;\vdash\; B_1(\vec{x}) \;=\; B_1(\vec{x})\ \underline{\mathsf{and}}\ \underline{\mathsf{Some}}\,(\vec{y} \in \vec{R_2}(\vec{x}))\,B_2(\vec{x},\vec{y})$$

$$G \;\stackrel{\text{def}}{=}\; \underline{\mathsf{Loop}}\,(\vec{x} \in \vec{R_1})\,\underline{\mathsf{if}}\ B_1(\vec{x})\ \underline{\mathsf{then}}\ E(\vec{x})$$

Rewriting G with (d) we obtain

$$\underline{\mathsf{Loop}}\,(\vec{x} \in \vec{R_1})\,\underline{\mathsf{if}}\ B_1(\vec{x})\ \underline{\mathsf{and}}\ \underline{\mathsf{Some}}\,(\vec{y} \in \vec{R_2}(\vec{x}))\,B_2(\vec{x},\vec{y})\ \underline{\mathsf{then}}\ E(\vec{x})$$

Commutativity of conjunction and nesting of conditionals yields

$$\underline{\mathsf{Loop}}\,(\vec{x} \in \vec{R_1})\,\underline{\mathsf{if}}\ \underline{\mathsf{Some}}\,(\vec{y} \in \vec{R_2}(\vec{x}))\,B_2(\vec{x},\vec{y})\ \underline{\mathsf{then}}\ \underline{\mathsf{if}}\ B_1(\vec{x})\ \underline{\mathsf{then}}\ E(\vec{x})$$

Finally, applying (idemloop) from right to left gives

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

Therefore, the equation $G = G'$ is derivable in CoDi's equational theory (and hence follows semantically) from (d). We say that G' is the result of (one-step) chasing G with (d). G could be a query but it could also be the boolean-valued expression A when we translate constraints as equations $A = \underline{\mathsf{true}}$. Chasing a constraint means chasing the expression A. We note however that this is only a particular case of chase. In general we need to be able to (homomorphically) map the bound variables of d to the bound variables of G. The full definition is given in section 8.

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

$$Q_i \;\stackrel{\text{def}}{=}\; \underline{\mathsf{BigU}}\,(\vec{x} \in \vec{R_i})\,\underline{\mathsf{if}}\ B_i(\vec{x})\ \underline{\mathsf{then}}\ \underline{\mathsf{sng}}(E_i(\vec{x})) \qquad (i = 1, 2)$$

and observe that we can express intersection by

$$\begin{aligned} Q_1 \cap Q_2 \;=\; & \underline{\mathsf{BigU}}\,(\vec{x} \in \vec{R_1})\,\underline{\mathsf{BigU}}\,(\vec{y} \in \vec{R_2}) \\ & \qquad \underline{\mathsf{if}}\ B_1(\vec{x})\ \underline{\mathsf{and}}\ B_2(\vec{y})\ \underline{\mathsf{and}}\ \underline{\mathsf{eq}}(E_1(\vec{x}),\, E_2(\vec{y}))\ \underline{\mathsf{then}}\ \underline{\mathsf{sng}}(E_1(\vec{x})) \end{aligned}$$

Now consider the constraint $cont(Q_1, Q_2)$ defined as

$$\vec{x} \in \vec{R_1} \;\vdash\; B_1(\vec{x}) \;=\; B_1(\vec{x})\ \underline{\mathsf{and}}\ \underline{\mathsf{Some}}\,(\vec{y} \in \vec{R_2})\,B_2(\vec{y})\ \underline{\mathsf{and}}\ \underline{\mathsf{eq}}(E_1(\vec{x}),\, E_2(\vec{y}))$$

It is easy to see that one step of chasing Q_1 with $cont(Q_1, Q_2)$ yields $Q_1 \cap Q_2$. Since $Q_1 \subseteq Q_2$ is equivalent to $Q_1 = Q_1 \cap Q_2$, it follows that containments can be established by deriving certain constraints. In fact, we show in sections 7 and 8 that for path-conjunctive queries containment is *reducible* to path-conjunctive dependencies.

The Relational Case. Conjunctive (tableau) queries [AHV95] are easily expressible in CoDi (using variables that range over tuples rather than over individuals, much as in SQL and OQL).

The example on the right shows two tableau queries, Q' and Q. 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	$z\ y$
	$x\ y$

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

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

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

The correspondence between homomorphisms and CoDi equations is made clear in theorem 7.11. (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[5]. For example, Q' above, seen as a tuple-generating dependency (tgd) d' is represented by

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

Chasing with embedded dependencies [AHV95] can also be represented by a certain kind of rewriting in CoDi's equational theory. For this, we use the second form for (d') from Lemma 3.1. As a parenthesis, in the rest of the paper we will use a notation that is convenient when we code up implication as an equality using conjunction:

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

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:

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

which represents the query obtained by chasing.

[5] Since we are using tuple variables, full relational dependencies still require existential quantification in CoDi. With extra care, a generalization of full dependencies is still possible (section 9).

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):

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

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

Coming back to the minimization example, the rewriting of Q' into Q by using (FOLD) is nothing more but the special case of chasing with trivial dependencies. This explains equationally tableaux minimization (a similar observation holds for tableaux containment: when $cont(Q_1, Q_2)$ is trivial, it corresponds precisely to a containment mapping ([CM77]). We are able to generalize the classical relational theorems, regarding query containment, dependency triviality and dependency implication (by chase), in sections 7 and 8, for the more general class of path-conjunctive queries and dependencies. Moreover, we show that CoDi is complete for *proving* triviality, containment/equivalence, and dependency implication for this class.

4 Examples with complex values and object-oriented data

We have shown in[PT99] on the Proj-Dept example (involving a class and a relation with a inverse constraint) how the chase can prove that a query with a three-loop join (namely `ABC_DP`) is equivalent to a query with only one loop (namely `ABC_P`), thus giving rise to a significant optimization. In this section we show that the chasing by rewriting capture other examples beyond the relational model.

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

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

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

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

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 9) 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 [BK93] and [CD92]. Many-one and one-one inverse relationships can be similarly characterized by full EPCDs.

PC views and nesting views. The two example queries `ABC_DP` and `ABC_P` were restricted such that their output type was flat (no nested sets or dictionaries in the output), while their input types (the schema names occuring in the query) were allowed to be nested. We will see in the next section that the class of path-conjunctive (PC) queries, for which our decidability and completeness results are given, fall into this category of "unnesting" queries. We show in sections 7 and 8 that containment of PC queries is reducible to embedded path-conjunctive dependencies (EPCDs). This idea has several important consequences, one of them being the ability to equivalently characterize a PC view definition by two EPCDs (for the two containments, respectively). More precisely, assume the following view definition over a schema $\vec{R}$, $V \stackrel{\text{def}}{=} V(\vec{R})$, where $V(\vec{R})$ is the PC query:

$$\underline{\mathsf{BigU}}\,(\vec{x} \in \vec{P})\,\underline{\mathsf{if}}\ B(\vec{x})\ \underline{\mathsf{then}}\ \underline{\mathsf{sng}}P'(\vec{x})$$

Proposition 4.1 *In a schema obtained by adding the name V to $\vec{R}$, an instance I satisfies $V = V(\vec{R})$ if and only if I satisfies the following two EPCDs:*

$$d_V^1 \stackrel{\text{def}}{=} v \in V \ \vdash\ \underline{\mathsf{true}} \wedge\!= \underline{\mathsf{Some}}\,(\vec{x} \in \vec{P})\,B(\vec{x})\ \underline{\mathsf{and}}\ \underline{\mathsf{eq}}(P'(\vec{x}),\, v)$$

$$d_V^2 \stackrel{\text{def}}{=} \vec{x} \in \vec{P} \ \vdash\ B(\vec{x}) \wedge\!= \underline{\mathsf{Some}}\,(v \in V)\,\underline{\mathsf{eq}}(P'(\vec{x}),\, v)$$

Note that d_V^1 is $cont(V, V(\vec{R}))$ while d_V^2 is $cont(V(\vec{R}), V)$ if we use the equivalence $V = \underline{\mathsf{BigU}}\,(v \in V)\,\underline{\mathsf{sng}}\ v$ (by the (sng) law). Using EPCDs instead of the PC view definition allows us to use the chase for query rewriting. We are currently investigating the ability of using dependencies and chase for rewriting of queries with PC views. We are also looking at the ability of characterizing physical level structures, like indexes, (many of which are PC views) by dependencies.

A second remark is that the equivalent characterization with dependencies holds not only for PC queries but for nesting queries as well. However, the dependencies in that case are no longer EPCDs (they may involve set equality, or several alternations of quantifiers - EPCDs have only one alternation of quantifiers and no set/dictionary equality, see section 5). Therefore we cannot use the completeness results about the chase that we prove in this paper for the PC case. We are currently investigating classes of nesting views (*grouping* views, for example), which have equivalent characterization with EPCDs.

5 PC queries, EPCDs and tableaux

¿From now on, we will restrict ourselves to a fragment of CoDi for which we are able to generalize the decidability/completeness results from the relational theory regarding containment/triviality and chase. We call the queries and dependencies that we are dealing with *path-conjunctive* (PC in short) because they are the natural generalization of the conjunctive fragment of the relational theory. We note that most of the previous examples of queries and dependencies fall into the PC category (one notable exception is <u>nest</u>).

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. We will distinguish between *finite* and *unrestricted* instances, in the latter $\{\sigma\}$ meaning *all* sets. We now define *paths* P and *path-conjunctions* C:

$$P ::= x \mid \mathtt{R} \mid P.\mathtt{A} \mid \underline{\mathsf{dom}}\,P \mid x\,!\,P \mid \underline{\mathsf{true}} \mid \underline{\mathsf{false}} \qquad C ::= \underline{\mathsf{eq}}(P_1,\, P_1')\ \underline{\mathsf{and}}\ \cdots\ \underline{\mathsf{and}}\ \underline{\mathsf{eq}}(P_n,\, P_n')$$

A *path-conjunctive* (PC) query has the form $Q ::= \underline{\mathsf{Loop}}\,(\vec{x} \in \vec{P})\,\underline{\mathsf{if}}\ C(\vec{x})\ \underline{\mathsf{then}}\ E(\vec{x})$ where [6]:

$$E ::= P \mid \underline{\mathsf{sng}}P \mid \underline{\mathsf{sng}}\langle A_1 : P_1, \ldots, A_n : P_n\rangle$$

An *embedded path-conjunctive dependency* (EPCD) has one of the equivalent forms (see section 11):

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

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

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

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

[6] Although we don't allow record constructors in paths, their equality is still representable, componentwise. Same for equality of <u>sng</u> expressions. We will make use of this without mentioning it.

For an EPCD as above we will also use the notation $dep(T, T')$, where T is as above and $T' = \{\vec{x} \in \vec{P_1}, \vec{y} \in \vec{P_2}(\vec{x}); C_1(\vec{x})$ $\underline{\textsf{and}}$ $C_2(\vec{x}, \vec{y})\}$. This is in the spirit with the notation for tuple generating dependencies using tableaux in [BV84a] and [BV84b]. Note however that our formalism doesn't necessarily distinguish between EPCDs and EGDs: any EGD can be written as $dep(T, T')$, where $T' = \{\vec{x} \in \vec{P_1}; C_1(\vec{x})$ $\underline{\textsf{and}}$ $\underline{\textsf{eq}}(\vec{P_2}(\vec{x}), \vec{P_3}(\vec{x}))\}$. For a PC query Q as above we will use the abbreviation $\underline{\textsf{Loop}}(T)E$.

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

1. A *simple* type is defined (inductively) as either a base type or a record type in which the types of the components are simple types (in other words, it doesn't involve set or dictionary types). Dictionary types $\sigma \times\!\!> \tau$ are restricted such that σ is a simple type. The paths P_i, P_i' appearing in path conjunctions must be of simple type. The expression E appearing in PC queries must be of simple type.
2. A *finite set type* is a type of the form $\{\tau\}$ where the only base type occurring in τ is $\textsf{bool}$ or $\langle\rangle$ (the empty record type). We do not allow in PC queries or EPCDs bindings of the form $x \in P$ such that P is of finite set type [7].
3. Additionally, we restrict our tableaux and paths (and consequently our EPCDs and PC queries) to be *well-defined* (see the next section). This condition restricts an expression of the form $x\,!\,P$ to be such that either it occurs in the scope of a binding $x \in \underline{\textsf{dom}}\,P$ or, more general[8], there exists a binding $x' \in S$ in T such that $\vec{x} \in \vec{P_1} \vdash C_1(\vec{x}) \wedge\!\!= \underline{\textsf{eq}}(x, x')$ $\underline{\textsf{and}}$ $\underline{\textsf{eq}}(S, \underline{\textsf{dom}}\,P)$ is a valid EGD (true in all instances) [9].

In some of our results, we will be able to drop the simple-type restriction in equalities or in the expression E in the body of a query. We will mention this explicitly whenever this is the case. However, the type of keys in dictionaries will always be of simple type.

Definition 5.1 *A valuation 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) $v(C(\vec{x})) = true$

We will show in the next section that the restriction to well-defined tableaux guarantees that no valuation ever fails. To be more precise, for a well-defined expression of the form $x\,!\,P$ over some well-defined tableau T and a valuation v from T into some arbitrary instance, $v(x)\,!\,v(M)$ doesn't fail because it is always the case that $v(x) \in \underline{\textsf{dom}}\ v(M)$.

We remark here that our syntax for path expressions does not allow for the typical navigation-style paths used in OO query languages, for example: $(\ldots((x_1\,!\,P_1)\,!\,P_2)\ldots!\,P_n)$. While apparently this is an important limitation, in most practical situations it is possible to break such paths into simple paths having only one dereferencing operation by introducing additional loops and explicit join conditions. For example, the query Q_1 (not a PC query):

$$\underline{\textsf{BigU}}\,(x \in \underline{\textsf{dom}}\,P_1)\,\underline{\textsf{sng}}\,(x\,!\,P_1)\,!\,P_2$$

can be written as Q_2 (a PC query):

$$\underline{\textsf{BigU}}\,(x \in \underline{\textsf{dom}}\,P_1)\,\underline{\textsf{BigU}}\,(z \in \underline{\textsf{dom}}\,P_2)\,\underline{\textsf{if}}\ \underline{\textsf{eq}}(z,\, x\,!\,P_1)\ \underline{\textsf{then}}\ \underline{\textsf{sng}}\ z\,!\,P_2$$

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

[8] Note that is a relaxation of the condition in [PT99].

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

The two queries are not equivalent: Q_1 can fail while Q_2 never fails! This is because the implicit constraint that one assumes when writing Q_1, namely $x \in \underline{\mathsf{dom}}\, P_1 \vdash \underline{\mathsf{true}} \wedge\!= \underline{\mathsf{Some}}\,(z \in \underline{\mathsf{dom}}\, P_2)\, \underline{\mathsf{eq}}(z,\, x\,!\,P_1)$, may not hold. On the other hand the constraint is explicit in Q_2: Q_2 is equivalent, by the idemloop law (see section 11), to

$$\underline{\mathsf{BigU}}\,(x \in \underline{\mathsf{dom}}\, P_1)\, \underline{\mathsf{if}}\ \underline{\mathsf{Some}}\,(z \in \underline{\mathsf{dom}}\, P_2)\, \underline{\mathsf{eq}}(z,\, x\,!\,P_1)\ \underline{\mathsf{then}}\ \underline{\mathsf{sng}}\ (x\,!\,P_1)\,!\,P_2$$

Clearly, Q_1 and Q_2 are equivalent under all instances satisfying the constraint: the first query is a pointer-based join while the second one is a value-based join [SC90, BK93]. We argue that a typical user has in mind the semantics of Q_2 even though she may write Q_1.

The notion of valuation is useful in giving meaning of expressions with free variables, i.e. path expressions $P(\vec{x})$ over a tableau T. Thus, we are able to express in terms of valuations the notion of satisfiability of an EPCD by an instance.

Definition 5.2 *Let I be an instance and $d = dep(T, T')$ an EPCD. Then I satisfies d, written as $I \models dep(T, T')$, if for any valuation $v : T \to I$ there exists a valuation $v' : T' \to I$ such that v' coincides with v on T (i.e. on variables $\vec{x}$ of T).*

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.

Proposition 5.3 *Let (d) be an EPCD over a schema $\vec{\mathtt{S}}$ whose roots have set type and suppose that $\vec{\mathtt{S}}$ is a* PC view*, that is, each $\mathtt{S}$ is expressed as a set-valued PC query over another schema $\vec{\mathtt{R}}$. Composing this view with (d) is provably equivalent to another EPCD, this one over $\vec{\mathtt{R}}$.*

6 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. 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). In the relational case this instance is isomorphic to the tableau itself. However, in our case, the construction is more complicated. The first reason is that we are dealing with complex objects and thus the instance has to be able to represent all reachable components of an object, and also we have to propagate equalities down to these components. Second, we want to ensure that every operation $x\,!\,P$, that we consider, is well defined (it doesn't fail) in the canonical instance. Finally, the third complication has to do with the extensionality property of sets: in the canonical instance, if two sets have the same elements then they must be equal, even though their equality may not be a consequence of $C(\vec{x})$.

For this section there are no restrictions in the type of equalities that can occur in $C(\vec{x})$. In particular, set/dictionary equality is allowed. The construction, which has two stages, is sketched next.

A non-extensional instance: *CInst*(T)**.** In the first stage we built, in parallel, a set G of path expressions over the variables $\vec{x}$, and a binary relation $\simeq\ \subseteq\ G \times G$ such that G will represent the set of all possible well-defined paths over T while $\simeq$ will be the congruence closure of the relation $\{(Q_1, Q_2) \mid \underline{\mathsf{eq}}(Q_1,\, Q_2) \text{ occurs in } T\}$. We will use as a notation $Q : G$ to assert that the path expression Q is an element of G. Then, G and $\simeq$ are defined to be the least set and binary relation that are closed under the below rules. The first group of rules, the *add rules*, specify when a path expression Q belongs to G:

$$(\text{prj-add})\ \frac{Q : G}{Q.\mathtt{A} : G} \qquad (\underline{\mathsf{dom}}\text{-add})\ \frac{Q : G}{\underline{\mathsf{dom}}\, Q : G} \qquad (\text{root-add})\ \frac{\mathtt{R} \text{ in the schema}}{\mathtt{R} : G}$$

$$(\text{var-add})\ \frac{x \in S \text{ in } T, \quad S : G}{x : G} \qquad (\underline{\mathsf{true}}\text{-add})\ \frac{}{\underline{\mathsf{true}} : G} \qquad (\underline{\mathsf{false}}\text{-add})\ \frac{}{\underline{\mathsf{false}} : G}$$

$$(\text{!-add})\ \frac{x \simeq x', \quad x' \in S \text{ in } T, \quad S \simeq \underline{\mathsf{dom}}\, Q}{x \,!\, Q : G}$$

We remark here that the (var-add) rule is not an axiom, in other words to infer that $x : G$ we need to make sure first that the set S to which x is bound is well-defined. This reflects the fact that in $\vec{x} \in \vec{P}$ we may have dependent bindings (for example: $y \in R, x \in y$). Also note that the last rule requires in the hypothesis two conditions involving $\simeq$. The second group of rules, the *equality* rules, specify when two paths Q_1 and Q_2 are to be considered equal as a consequence of $C(\vec{x})$:

$$(\text{eq})\ \frac{\underline{\mathsf{eq}}(Q_1, Q_2) \text{ occurs in } C(\vec{x}), \quad Q_1 : G, \quad Q_2 : G}{Q_1 \simeq Q_2}$$

$$(\text{refl})\ \frac{Q : G}{Q \simeq Q} \qquad (\text{sym})\ \frac{Q_1 \simeq Q_2}{Q_2 \simeq Q_1} \qquad (\text{trans})\ \frac{Q_1 \simeq Q_2 \quad Q_2 \simeq Q_3}{Q_1 \simeq Q_3}$$

$$(\text{prj-cong})\ \frac{Q \simeq Q'}{Q.\mathtt{A} \simeq Q'.\mathtt{A}} \qquad (\underline{\mathsf{dom}}\text{-cong})\ \frac{Q \simeq Q'}{\underline{\mathsf{dom}}\, Q \simeq \underline{\mathsf{dom}}\, Q'} \qquad (\text{rcd-ext})\ \frac{Q.\mathtt{A}_1 \simeq Q'.\mathtt{A}_1 \ \ldots \ Q.\mathtt{A}_n \simeq Q'.\mathtt{A}_n}{Q \simeq Q'}$$

$$(\text{!-cong})\ \frac{x \simeq y \quad Q_1 \simeq Q_2 \quad x \,!\, Q_1 : G \quad y \,!\, Q_2 : G}{x \,!\, Q_1 \simeq y \,!\, Q_2}$$

Observe that $Q_1 \simeq Q_2$ implies $Q_1 : G$ and $Q_2 : G$ (a simple induction on the derivation of $Q_1 \simeq Q_2$). In other words we only equate paths that we already know to be well-defined. The add rules and the equality rules are cyclic. As a consequence of equating paths via $\simeq$, more paths may become members of G (and therefore well-defined) and vice-versa.

The important observation now is that G and $\simeq$ are finite, and moreover, constructible in polynomial time in the size of the tableau T[10]. This is an immediate consequence of the fact that the set of *all* syntactically valid paths with variables in $\vec{x}$ is finite and polynomial in the size of T. To see why this is true it suffices to observe that all our operators (with the exception of !) are *deconstructors* that reduce the size of the input type, and that in the left-hand side of the ! operator only variables are allowed. An algorithm that computes G and $\simeq$ will always make sure that at each step produces at least one new path $Q : G$ or a new pair $Q_1 \simeq Q_2$. Since G and $\simeq$ are polynomial in the size of T the algorithm must terminate within polynomially many steps. It is easy to see that each step of the algorithm (mainly deciding which rule to be applied) can be implemented in polynomial time as well. We will refer to G and $\simeq$ as the *set of well-defined paths over* T and, respectively, the *congruence closure of* T.

Definition 6.1 (1) *A tableau T is a* well-defined tableau *if, for every Q that occurs in T, $Q : G$. A path Q with variables in $\vec{x}$ is a* well-defined path over T *if $Q : G$.*

[10]However, it may be exponential in the height of the schema. The height of the schema is the maximum number of levels of nesting of record or dictionary types. In this paper we always consider the schema fixed.

(2) *An EGD* $\vec{x} \in \vec{P} \ \vdash \ C(\vec{x}) \wedge\!= \underline{\mathsf{eq}}(Q,\, Q')$ *is well-defined if the tableau* $T = \{\vec{x} \in \vec{P}\ ;\ C(\vec{x})\}$ *is well-defined and* Q *and* Q' *are well-defined over* T.

(3) *An EPCD* $dep(T,T')$ *is well-defined if both tableaux* T *and* T' *are well-defined.*

Note that it is possible that a tableau T is well-defined and there are paths that are not well-defined over T (however, they do not occur in T). It is also possible to have well-defined paths over tableaux that are not well-defined. We will always work with well-defined tableaux and with well-defined paths over well-defined tableaux. The next lemma shows that a well-defined EGD is a special case of a well-defined EPCD.

Lemma 6.2 *Let* $T = \{\vec{x} \in \vec{P}\ ;\ C(\vec{x})\}$ *be a well-defined tableau. Then* $Q(\vec{x})$ *and* $Q'(\vec{x})$ *are well-defined paths over* T *if and only if the tableau* $T' = \{\vec{x} \in \vec{P}\ ;\ C(\vec{x})\ \underline{\mathsf{and}}\ \underline{\mathsf{eq}}(Q,\, Q')\}$ *is well-defined. Thus the EGD* $\vec{x} \in \vec{P} \ \vdash$ $C(\vec{x}) \wedge\!= \underline{\mathsf{eq}}(Q,\, Q')$ *is the same as the EPCD* $dep(T,T')$.

Proof. Let $\simeq$ and $\simeq'$ be, respectively, the corresponding relations for T and T'. Similarly, we have G and G'. For the first direction, assume that Q and Q' are well-defined over T, i.e $Q : G$ and $Q' : G$. An easy induction (by the derivation of $P : G$ or $P_1 \simeq P_2$) shows that $P : G$ implies $P : G'$ and $P_1 \simeq P_2$ implies $P_1 \simeq' P_2$. Thus Q and Q' are well-defined over T', and hence T' is well-defined.

Conversely, we prove by induction that for any $P : G'$, $P_1 \simeq' P_2$ with derivations that do not use $\underline{\mathsf{eq}}(Q,\, Q')$, we have $P : G$ and $P_1 \simeq P_2$. The interesting case is when the last rule applied in the derivation of $P_1 \simeq' P_2$ is (eq). It must be the case then that $\underline{\mathsf{eq}}(P_1,\, P_2)$ occurs in $C(\vec{x})$ since the derivation doesn't use $\underline{\mathsf{eq}}(Q,\, Q')$. Since $P_1 : G'$ and $P_2 : G'$, by the inductive hypothesis, we have $P_1 : G$ and $P_2 : G$. Thus, by (eq), $P_1 \simeq P_2$. The other cases are straightforward. We observe now that to derive $Q : G'$ and $Q' : G'$ we don't need $\underline{\mathsf{eq}}(Q,\, Q')$ ($Q : G'$ and $Q' : G'$ must be already derived before we can apply the (eq) rule with $\underline{\mathsf{eq}}(Q,\, Q')$). Thus, $Q : G$ and $Q' : G$. $\square$

Next, we build a directed acyclic labeled graph with G as the set of nodes. The edges are as follows: for any $Q : G$ and $Q.\mathtt{A} : G$, add an edge, labeled $.A$ from Q into $Q.A$. For any $Q : G$ and $\underline{\mathsf{dom}}\, Q : G$, add an edge labeled $\underline{\mathsf{dom}}$ from Q edges labeled !, one from x into $x\,!\,Q$ and one from Q into $x\,!\,Q$. Finally, we populate nodes corresponding to set values: for each binding $x \in S$ occuring in T, add an edge labeled with $\in$ from S into x.

Each $\simeq$-equivalence class becomes a node in a new graph, *CInst*(T). 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 with the same label from some P_i into some Q_j in $G(T)$. We will use the following notations: if P and Q are two nodes in *CInst*(T) such that there is an edge labeled $.A$ from P into Q then we denote the node Q by $P.A$. For a given node P (of record type) there is a unique node $P.A$ (since $\simeq$ is closed under (prj-cong). Similar, for edges labeled ! and $\underline{\mathsf{dom}}$. Thus, $.A$, ! and $\underline{\mathsf{dom}}$ can be viewed as operations on nodes of *CInst*(T). Also, observe that these operations are always defined. For nodes P and Q in *CInst*(T) such that there is an edge labeled $\in$ from Q into P we use the notation $P \in Q$.

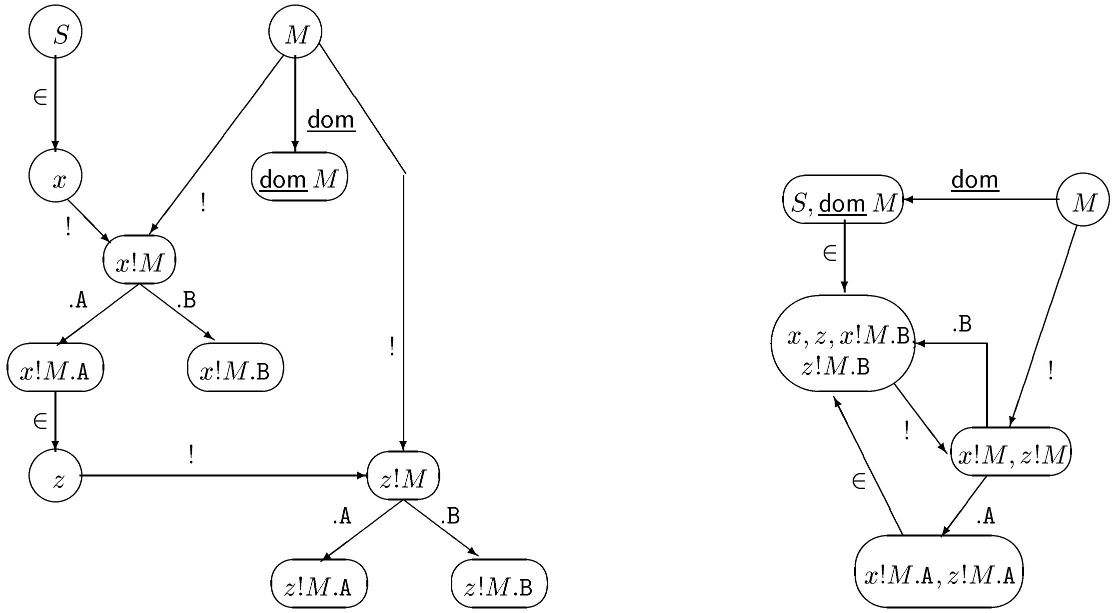
Example. Consider the tableau

$$T = \{x \in S, z \in x\,!\,M.\mathtt{A}\ ;\ \underline{\mathsf{eq}}(x\,!\,M.\mathtt{B},\, z)\ \underline{\mathsf{and}}\ \underline{\mathsf{eq}}(S,\, \underline{\mathsf{dom}}\, M)\ \underline{\mathsf{and}}\ \underline{\mathsf{eq}}(x\,!\,M.\mathtt{B},\, x)\}$$

where $M : \underline{\mathsf{b}} \times\!\!> \langle \mathtt{A} : \{\underline{\mathsf{b}}\}, B : \underline{\mathsf{b}} \rangle$, $S : \{\underline{\mathsf{b}}\}$, and $\underline{\mathsf{b}}$ is some base type. Then G is shown in figure 2a, while *CInst*(T) is shown in figure 2b. The relation $\simeq$ can be read from the nodes of *CInst*(T): each node is a $\simeq$-equivalence class. Note that $x\,!\,M$ is well-defined over T since $x \in S$ and $S \simeq \underline{\mathsf{dom}}\, M$. $z\,!\,M$ is well-defined over T as well, since $z \simeq x$. However, $(x\,!\,M.B)\,!\,M$ is not well-defined, even though $x\,!\,M.B \simeq x$, because it is not sintactically valid. Observe that all the well-defined paths over T appear in G, and T is a well-defined tableau.

Proposition 6.3 *Let* T *be a well-defined tableau and let* Q *be a well-defined path over* T. *Then, for any instance* I *and for any valuation* $v : T \to I$, $v(Q)$ *doesn't fail.*

Proof. We prove by simultaneous induction on the derivation of $Q : G$ or $Q_1 \simeq Q_2$ that $Q : G$ implies $v(Q)$ doesn't fail and $Q_1 \simeq Q_2$ implies that neither $v(Q_1)$ nor $v(Q_2)$ fails and $v(Q_1) = v(Q_2)$. The base



(a) G (b) *CInst*(T)

Figure 2: Defining G, $\simeq$ and *CInst*(T).

cases, (root-add), ($\underline{\mathsf{true}}$-add), ($\underline{\mathsf{false}}$-add) are obvious. For the inductive cases, (prj-add), ($\underline{\mathsf{dom}}$-add), (sym), (trans), (prj-cong), ($\underline{\mathsf{dom}}$-cong), and (rcd-ext) require a simple application of the inductive hypothesis and of the definition of a valuation. (var-add) is trivial. The first interesting case is (!-add). Then $v(x\,!\,Q) = v(x)\,!\,v(Q)$ and $v(x') \in v(S)$. But since $S \simeq \underline{\mathsf{dom}}\,Q$ it follows by the inductive hypothesis that $v(S) = v(\underline{\mathsf{dom}}\,Q) = \underline{\mathsf{dom}}\,v(Q)$. Similarly, $x \simeq x'$ implies $v(x) = v(x')$. Thus $v(x) \in \underline{\mathsf{dom}}\,v(Q)$ and therefore $v(x)\,!\,v(Q)$ doesn't fail. For the (eq) rule, by the inductive hypothesis, we have that $v(Q_1)$ and $v(Q_2)$ don't fail. Since v is a valuation it must be the case that $v(Q_1) = v(Q_2)$. In the case of (!-cong) we make use of the fact that $v(x\,!\,Q_1)$ and $v(y\,!\,Q_2)$ must be defined. The (refl) case is similar. □

We conclude the construction of *CInst*(T) and the discussion on well-definedness with an interesting remark. Suppose T is a tableau as before and $\underline{\mathsf{eq}}(Q_1,\, Q_1'), \ldots, \underline{\mathsf{eq}}(Q_n,\, Q_n')$ are, each of them, well-defined over T. Then the EPCD

$$d \;=\; \vec{x} \in \vec{P} \;\vdash\; C(\vec{x}) \wedge\!= \underline{\mathsf{eq}}(Q_1,\, Q_1') \;\underline{\mathsf{and}}\; \ldots \;\underline{\mathsf{and}}\; \underline{\mathsf{eq}}(Q_n,\, Q_n')$$

is obviously well-defined. Moreover it is equivalent to the set of EGDs:

$$D \;=\; \{\vec{x} \in \vec{P} \;\vdash\; C(\vec{x}) \wedge\!= \underline{\mathsf{eq}}(Q_i,\, Q_i') \;\mid\; i = 1, n\}$$

However, there are cases in which an well-defined EPCD of the same form as d is not equivalent with D simply because the EGDs in D may not be well-defined when taken individually. A simple example is to take d to be:

$$x \in S \;\vdash\; \underline{\mathsf{true}} \wedge\!= \underline{\mathsf{eq}}(S,\, \underline{\mathsf{dom}}\,M) \;\underline{\mathsf{and}}\; \underline{\mathsf{eq}}(x\,!\,M,\, x.A)$$

While d is well-defined, obviously the EGD $x \in S \;\vdash\; \underline{\mathsf{eq}}(x\,!\,M,\, x.A)$ is not well-defined. However, we will use the above equivalence without explicitly mentioning, whenever it is clear from the context that the well-definedness conditions are satisfied.

The canonical instance: *Inst*(T)**.** *CInst*(T) has all the properties to be a valid instance with one exception: there may be two distinct nodes of set type S_1, S_2, that have the same set of $\in$-successors, $\{e_1, \ldots, e_m\}$). Thus,

$CInst(T)$ does not satisfy the extensionality property of sets. In this case, we will identify S_1 and S_2. Similar, if two dictionaries M and N have the same domain, i.e. $\underline{\mathsf{dom}}\, M$ and $\underline{\mathsf{dom}}\, N$ are the same node, and for all $\in$-successors x of $\underline{\mathsf{dom}}\, M$, $x\,!\,M$ and $x\,!\,N$ are identical, then we identify the nodes M and N. Of course, we may need to propagate these identifications in the graph. More formally, we define an equivalence relation on nodes of $CInst(T)$ (which in turn consist of congruence classes with respect to $C(\vec{x})$), denote it by $\sim$, as the smallest relation closed under the following *extensionality* rules:

$$\text{(refl)}\ \frac{}{Q \sim Q} \qquad \text{(sym)}\ \frac{Q_1 \sim Q_2}{Q_2 \sim Q_1} \qquad \text{(trans)}\ \frac{Q_1 \sim Q_2 \quad Q_2 \sim Q_3}{Q_1 \sim Q_3}$$

$$\text{(set-ext)}\ \frac{\forall e \in S_1 \,.\, \exists e' \in S_2 \,.\, e \sim e' \qquad \forall e' \in S_2 \,.\, \exists e \in S_1 \,.\, e' \sim e}{S_1 \sim S_2}$$

$$\text{(dict-ext)}\ \frac{M \le N \qquad N \le M}{M \sim N} \qquad \text{(rcd-ext)}\ \frac{Q\,.\,A_1 \sim Q'\,.\,A_1 \quad \ldots \quad Q\,.\,A_n \sim Q'\,.\,A_n}{Q \sim Q'}$$

where $M \le N$ means: $\forall e \in \underline{\mathsf{dom}}\, M \,.\, \exists e' \in \underline{\mathsf{dom}}\, N \,.\, [e \sim e' \ \wedge\ e\,!\,M \sim e'\,!\,N]$. Note that $M \le N$ and $N \le M$ implies $\underline{\mathsf{dom}}\, M \sim \underline{\mathsf{dom}}\, N$.

It is obvious that $\sim$ exists and is unique. The graph consisting of $\sim$-equivalence classes is $Inst(T)$ (we add edges between $\sim$-equivalence classes in the same way we did for $CInst(T)$). $Inst(T)$ is an extensional instance, and one can see that its construction is in PTIME. We consider two canonical mappings, $\underline{\mathsf{cval}}_C : T \to CInst(T)$, and $\underline{\mathsf{cval}} : T \to Inst(T)$, associating to each path expression P occuring in T nodes in $CInst(T)$ and, respectively, $Inst(T)$. We denote by $\underline{\mathsf{collapse}}$ the function mapping nodes in $CInst(T)$ to their corresponding $\sim$-equivalence classes in $Inst(T)$. We have then $\underline{\mathsf{cval}} = \underline{\mathsf{collapse}} \circ \underline{\mathsf{cval}}_C$.

Example. Consider again the tableau T from the previous example. Since in $CInst(T)$ the nodes $[S, \underline{\mathsf{dom}}\, M]$ and $[x\,!\,M.\mathtt{A}]$ have the same $\in$-successor, it follows that $[S, \underline{\mathsf{dom}}\, M] \sim [x\,!\,M.\mathtt{A}]$. Thus they are collapsed in $Inst(T)$ (see figure 3). For simplicity, we represented the $\sim$-equivalence class consisting of the two nodes as the union of the elements in each of the two $\simeq$-equivalence classes. We note that, in $Inst(T)$, $x\,!\,M.\mathtt{A}$ and $\underline{\mathsf{dom}}\, M$ are equal ($\underline{\mathsf{cval}}\ (x\,!\,M.\mathtt{A}) = \underline{\mathsf{cval}}\ (\underline{\mathsf{dom}}\, M)$) even though their equality doesn't follow (by congruence) from the original equalities of T. Also there exist instances into which we can map T via valuations but still these instances don't satisfy the equality of $x\,!\,M.\mathtt{A}$ and $\underline{\mathsf{dom}}\, M$. Since our goal is to characterize dependencies that hold in all instances by checking their satisfiability in the canonical instance, this is apparently a problem. We will see in a little while how we can overcome this difficulty.

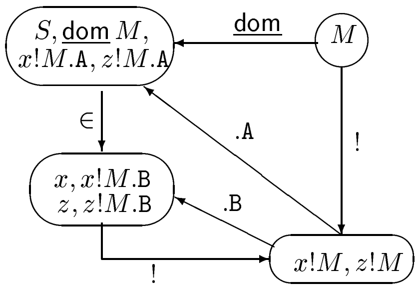


Figure 3: $Inst(T)$

We prove first some technical lemmas that ensure that our construction is well-defined, and are essential for the next results. The first one says that set nodes in $CInst(T)$ are identified by $Inst(T)$ *only as* a consequence of the extensionality rule. The same property holds for dictionaries. We will see how this fails if we allow keys of non-simple type to appear in dictionaries.

Lemma 6.4 *1. If S_1 and S_2 are set nodes in CInst(T) then the following is a derived rule:*

$$\text{(set-inv-ext)} \quad \frac{S_1 \sim S_2}{\forall e \in S_1 \,.\, \exists e' \in S_2 \,.\, e \sim e' \qquad \forall e' \in S_2 \,.\, \exists e \in S_1 \,.\, e' \sim e}$$

2. If M and N are dictionary nodes in CInst(T) then the following is a derived rule:

$$\text{(dict-inv-ext)} \quad \frac{M \sim N}{M \leq N \qquad N \leq M}$$

Proof. The only way to infer $S_1 \sim S_2$ other than by (refl), (sym) or (trans) is by (set-ext). A simple induction concludes the proof. Similar for the dictionary case. □

Lemma 6.5 *If Q_1 and Q_2 are of simple type such that $Q_1 \sim Q_2$ then $Q_1 = Q_2$.*

Lemma 6.6 *$\sim$ is a congruence relation, i.e. it is closed under rules* (prj-cong)*,* (!-cong) *and* $\underline{\mathsf{dom}}$-cong*.*

Proof. For (prj-cong) we use a similar observation as in the proof of Lemma 6.4. (dom-cong) is an immediate consequence of Lemma 6.4. The interesting case is (!-cong). Suppose $Q_1 \sim Q_2$ of simple type, and $M_1 \sim M_2$. We observe first that $Q_1 = Q_2$ (by the previos lemma), and second that $Q_1 \in \underline{\mathsf{dom}}\, M_1$ if and only if $Q_1 \in \underline{\mathsf{dom}}\, M_2$ (using the fact that $\underline{\mathsf{dom}}\, M_1 \sim \underline{\mathsf{dom}}\, M_2$ and Lemma 6.4 and, again, previous lemma). Suppose that $Q_1 \in \underline{\mathsf{dom}}\, M_1$ Since $M_1 \leq M_2$, there exists some $Q' \in \underline{\mathsf{dom}}\, M_2$ such that $Q_1 \sim Q'$ (and therefore $Q_1 = Q'$) and $Q_1 \,!\, M_1 \sim Q' \,!\, M_2$. Thus, we conclude $Q_1 \,!\, M_1 \sim Q_2 \,!\, M_2$. □

The last lemma allows us, as in the case of *CInst*(T), to define in a correct way, the operations $.A$, ! and $\underline{\mathsf{dom}}$ on nodes of *Inst*(T). We also observe that without the restriction on the type of keys of dictionaries, Lemma 6.6 fails. A simple counterexample to it is $T = \{x \in \underline{\mathsf{dom}}\, M, y \in \underline{\mathsf{dom}}\, N, z \in y \,!\, N \;;\; \underline{\mathsf{true}}\}$, where x and y are of the same *set* type. Here, we infer $\underline{\mathsf{cval}}_C x \sim \underline{\mathsf{cval}}_C y$, by (set-ext), since they are both empty, and $\underline{\mathsf{cval}}_C(\underline{\mathsf{dom}}\, M) \sim \underline{\mathsf{cval}}_C(\underline{\mathsf{dom}}\, N)$, by (set-ext) again, but there is no way to infer $\underline{\mathsf{cval}}_C(x \,!\, M) \sim \underline{\mathsf{cval}}_C(y \,!\, N)$ with only the rules listed in the definition of $\sim$. In this we would have to postulate explicitly the congruence rules as part of the definition of $\sim$. However, one can easily see that Lemma 6.4 fails in that case.

Lemma 6.7 *The mappings $\underline{\mathsf{cval}}_C$, $\underline{\mathsf{collapse}}$ and $\underline{\mathsf{cval}}$ are algebraic homomorphisms with respect to operations $.A$, ! and $\underline{\mathsf{dom}}$ on path expressions over T, nodes in CInst(T) and nodes in Inst(T).*

And this concludes our construction!

7 Trivial dependencies and query containment

7.1 Trivial EGDs

We show first that an EGD (with set/dictionary equality) is true in all (finite and unrestricted) instances (trivial EGD) if and only if it is satisfied in *CInst*(T) under the canonical mapping $\underline{\mathsf{cval}}_C$. Since the construction of *CInst*(T) can be carried out in PTIME , it follows that deciding triviality of EGDs is in PTIME as well.

Lemma 7.1 (EGD Lemma) *Let* $d = \vec{x} \in \vec{P} \vdash C(\vec{x}) \wedge\!= \underline{\mathsf{eq}}(Q(\vec{x}), Q'(\vec{x}))$ *be an EGD over* $T = \{\vec{x} \in \vec{P}\,; C(\vec{x})\}$*. T may have set/dictionary equality.*

1. *If* Q *and* Q' *are of simple type then the following are equivalent:*
   (a) d *is trivial (fin and/or unr)*
   (b) $\underline{\mathsf{cval}}_C Q = \underline{\mathsf{cval}}_C Q'$
   (c) $\underline{\mathsf{cval}} Q = \underline{\mathsf{cval}} Q'$
2. *If* Q *and* Q' *are allowed to have set/dictionary equality then the following are equivalent:*
   (a) d *is trivial (fin and/or unr)*
   (b) $\underline{\mathsf{cval}}_C Q = \underline{\mathsf{cval}}_C Q'$

Proof. Part 1: One direction, (b) implies (a), is true even for Q and Q' of non-simple type, and we already proved it in Proposition 6.3. Indeed, $\underline{\mathsf{cval}}_C Q = \underline{\mathsf{cval}}_C Q'$ means that $Q \simeq Q'$, and thus, for any instance I and for any valuation $v : T \to I$, $v(Q) = v(Q')$. Since we didn't make any assumption about I, it follows that d is trivial both in the finite and unrestricted case. The direction (a) implies (c) is obvious. For the last direction in part (1), (c) implies (b), assume $\underline{\mathsf{cval}} Q = \underline{\mathsf{cval}} Q'$. This means that $\underline{\mathsf{collapse}}(\underline{\mathsf{cval}}_C Q) = \underline{\mathsf{collapse}}(\underline{\mathsf{cval}}_C Q')$, or $\underline{\mathsf{cval}}_C Q \sim \underline{\mathsf{cval}}_C Q'$. If Q and Q' are of simple type then, by Lemma 6.5, $\underline{\mathsf{cval}}_C Q = \underline{\mathsf{cval}}_C Q'$.

Part 2: We need to show (a) implies (b) even when set/dictionary equality is allowed. We already proved *(b)* $\Rightarrow$ *(a)* in this case. Suppose d is trivial, Q and Q' are of set or dictionary type, and $\underline{\mathsf{cval}}_C Q \neq \underline{\mathsf{cval}}_C Q'$. We construct another trivial EGD $d' :\ \vec{x} \in \vec{P},\ \vec{y} \in \vec{R}\ \vdash\ C(\vec{x}) \wedge\!= \underline{\mathsf{eq}}(Q(\vec{x}), Q'(\vec{x}))$ for which we show that $\underline{\mathsf{cval}}' Q \neq \underline{\mathsf{cval}}' Q'$, where $\underline{\mathsf{cval}}'$ is the canonical valuation associated to $T' = \{\vec{x} \in \vec{P}, \vec{y} \in \vec{R}\ ;\ C(\vec{x})\}$. Thus, $\underline{\mathsf{cval}}'$ and *Inst*(T') provide a counterexample to the fact that d' is trivial.

First, it is easy to see that any d' of the above form is a logical consequence of d: for any instance $I \models d$, if v is a valuation from T' into I then v restricted to $\vec{x}$ is a valuation from T into I, thus $v(Q) = v(Q')$, and therefore $I \models d'$. Hence d trivial implies d' trivial.

We explain the construction of the new bindings $\vec{y} \in \vec{R}$ that are added to d. If Q is of set type, then we generate the first binding : $y \in Q$, where y is a new variable. Now y may generate new bindings as well. Here are all possible cases:

- y is of set type. Add $z \in y$ and continue recursively with z.
- y is of simple type. Then y doesn't generate any bindings any more.
- y is of dictionary type. Then add $z \in \underline{\mathsf{dom}}\, y$ and continue recursively with $z\,!\,y$. z itself must be of simple type, therefore it cannot generate any more bindings.
- y is of record type. Then for each attribute A such that $y.A$ is of set or dictionary type, continue recursively with $y.A$.

Similarly, if Q is of dictionary type, the first binding is $y \in \underline{\mathsf{dom}}\, Q$, and then we continue in the same way as above with y. Notice that this process terminates, because the type of Q is finite. It is easy to see that this construction reflects in the fact that *CInst*(T') differs from *CInst*(T) by having an additional tree rooted at y. This tree is entirely disjoint from *CInst*(T), since $C(\vec{x})$ doesn't involve any of the new variables. Also, this tree doesn't have any set nodes with no $\in$-successors (our construction rules out explicitly empty sets). This ensures that for any node Q'' in this tree, $Q'' \not\sim P$ for any node P outside of the tree. In particular, $\underline{\mathsf{cval}}'_C y \not\sim P$ for any P outside of the tree (here y is the first variable in $\vec{y}$, the one added in the first step to Q or $\underline{\mathsf{dom}}\, Q$). This, together with the fact that $\underline{\mathsf{cval}}_C Q \neq \underline{\mathsf{cval}}_C Q'$ (and therefore $\underline{\mathsf{cval}}'_C Q \neq \underline{\mathsf{cval}}'_C Q'$), implies that $\underline{\mathsf{cval}}'_C Q \not\sim \underline{\mathsf{cval}}'_C Q'$ (we make use here of Lemma 6.4). Thus, $\underline{\mathsf{cval}} Q \neq \underline{\mathsf{cval}} Q'$. □

Note that the the direction (c) $\Rightarrow$ (b) in part (1) fails when Q and Q' are of non-simple type. For example, $d = a \in A, b \in B \vdash \underline{\mathsf{eq}}(a, b) \wedge\!= \underline{\mathsf{eq}}(A, B)$ is satisfied in *Inst*(T) under the canonical valuation $\underline{\mathsf{cval}}$.

This is because the nodes corresponding to A and B are identified by the (set-ext) rule. Nonetheless d is not trivial (take an instance with $A = \{a, b\}, B = \{a, b'\}$). A similar counterexample using dictionaries: $a \in \underline{\mathsf{dom}}\, M \vdash \underline{\mathsf{eq}}(\underline{\mathsf{dom}}\, M,\, \underline{\mathsf{dom}}\, N)\ \underline{\mathsf{and}}\ \underline{\mathsf{eq}}(a\,!\,M,\, a\,!\,N) \wedge\!\!= \underline{\mathsf{eq}}(M,\, N)$. If we look at our example in the previous section, $\underline{\mathsf{cval}}(x\,!\,M.\mathtt{A}) = \underline{\mathsf{cval}}(\underline{\mathsf{dom}}\, M)$ but $\underline{\mathsf{cval}}_C(x\,!\,M.\mathtt{A}) \neq \underline{\mathsf{cval}}_C(\underline{\mathsf{dom}}\, M)$ and thus $x \in S, z \in x\,!\,M.\mathtt{A} \vdash \underline{\mathsf{eq}}(x\,!\,M.\mathtt{B},\, z)\ \underline{\mathsf{and}}\ \underline{\mathsf{eq}}(S,\, \underline{\mathsf{dom}}\, M)\ \underline{\mathsf{and}}\ \underline{\mathsf{eq}}(x\,!\,M.\mathtt{B},\, x) \wedge\!\!= \underline{\mathsf{eq}}(x\,!\,M.\mathtt{A},\, \underline{\mathsf{dom}}\, M)$ is not trivial, as expected.

It is easy to see that any 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 $\underline{\mathsf{cval}}_C Q = \underline{\mathsf{cval}}_C Q'$ implies that $Q \simeq Q'$ and thus $\underline{\mathsf{eq}}(Q,\, Q')$ follows from $C(\vec{x})$ using congruence rules (see section 11). To summarize:

Theorem 7.2 *An EGD (with equality at set/dictionary type) holds in all unrestricted instances iff it holds in all finite instances. Trivial EGDs are provable in CoDi and triviality is decidable in PTIME.*

7.2 Homomorphisms of tableaux

In the next subsection we will prove our first important results, Theorem 7.9 and 7.11, that relate trivial dependencies and query containment with existence of special mappings between tableaux that generalize the notion of homomorphism from the relational case. As we'll see, the completeness part of the theorem (the difficult direction) makes use of the fact that, given two tableaux T_1 and T_2, a valuation v from T_2 into $\mathit{Inst}(T_1)$ induces a homomorphism h from T_2 into T_1.

Definition 7.3 (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, when extended to path expressions over T' in the usual way (algebraic homomorphism), h satisfies the following conditions:*

(1) *h maps well-defined paths over T' to well-defined paths over T,*

(2) *for any $y_i \in R_i$ in T' and $x_j \in P_j$ in T, if $h(y_i) = x_j$ then the following EGD is trivial*

$$\vec{x} \in \vec{P}\ \vdash\ C(\vec{x})\ \wedge\!\!=\ \underline{\mathsf{eq}}(P_j,\, h(R_i)) \qquad (h \text{ is context-preserving})$$

(3) *for each $\underline{\mathsf{eq}}(Q(\vec{y}),\, Q'(\vec{y}))$ that occurs in $D(\vec{y})$ the following EGD is trivial*

$$\vec{x} \in \vec{P}\ \vdash\ C(\vec{x})\ \wedge\!\!=\ \underline{\mathsf{eq}}(Q(h(\vec{y})),\, Q'(h(\vec{y})))$$

It is easy to see that checking whether a mapping h is a homomorphism is in PTIME. Also, we observe that composing a valuation with a homomorphism yields a valuation. Using this, and the EGD lemma, one can easily verify the following proposition, which gives an equivalent and often convenient characterization of homomorphisms in terms of G and $\simeq$ of section 6. As an immediate application of this proposition, composition of two homomorphisms is a homomorphism.

Proposition 7.4 *Let $T = \{\vec{x} \in \vec{P}\ ;\ C(\vec{x})\}$ and $T' = \{\vec{y} \in \vec{R}\ ;\ D(\vec{y})\}$ be two tableaux, and let h be a type-preserving mapping from variables $\vec{y}$ into variables $\vec{x}$. Extend h to path expressions such that h is an algebraic homomorphism. Let G, $\simeq$, and G', $\simeq'$ be the set of well-defined paths and the congruence closure for T, respectively T'. Then h is a homomorphism if and only if the following conditions hold:*

(1) *for any $Q : G'$, $h(Q) : G$*

(2) *for any $y_i \in R_i$ in T' and $x_j \in P_j$ in T, if $h(y_i) = x_j$ then $P_j \simeq h(R_i)$*

(3) *for any $Q_1 \simeq' Q_2$, $h(Q_1) \simeq h(Q_2)$*

The next lemma shows that given a valuation from a tableau T_2 into $\mathit{Inst}(T_1)$ we can "shift" this to an "equivalent" valuation from T_2 into $\mathit{CInst}(T_1)$. However, this equivalence holds only with respect to simple type equalities.

Lemma 7.5 *Let* $T_1 = \{\vec{x} \in \vec{P_1} \; ; \; C_1(\vec{x})\}$ *and* $T_2 = \{\vec{y} \in \vec{P_2} \; ; \; C_2(\vec{y})\}$ *in which* C_1 *may have set/dictionary equality but* C_2 *is simple-type restricted. Then, for any valuation* $v : T_2 \to \mathit{Inst}(T_1)$ *there exists a context-preserving, algebraic homomorphism,* $v_c : T_2 \to \mathit{CInst}(T_1)$ *such that* $v = \underline{\mathsf{collapse}} \circ v_c$. *Moreover,* v *and* v_c *satisfy the same set of formulas* $\underline{\mathsf{eq}}(Q, Q')$ *over* T_2 *with* Q *and* Q' *of simple type.*

Proof Sketch[11]**.** We define v_c such that $v = \underline{\mathsf{collapse}} \circ v_c$ by inducting over $\vec{y} \in \vec{P_2}$. **Base case:** $y \in R$ where R must be a root name of set type. It must be the case that $v(y) \in v(R)$ in $\mathit{Inst}(T_1)$. Since $v(R) = R^{\mathit{Inst}(T_1)}$, $\underline{\mathsf{cval}}_C R$ must belong to the congruence class $v(R)$, i.e. $\underline{\mathsf{collapse}}(\underline{\mathsf{cval}}_C R) = v(R)$. We know that, by the construction of $\mathit{Inst}(T_1)$, there must be some node S in $\mathit{CInst}(T_1)$ such that $\underline{\mathsf{collapse}}(S) = v(R)$ and a node e in $\mathit{CInst}(T_1)$ such that $\underline{\mathsf{collapse}}(e) = v(y)$, and $e \in S$. But $S \sim \underline{\mathsf{cval}}_C R$, thus by Lemma 6.4, there is some e', $e' \sim e$ such that $e' \in \underline{\mathsf{cval}}_C R$. Define $v_c(y) = e'$, thus $\underline{\mathsf{collapse}}(v_c(y)) = v(y)$. We then extend v_c on paths over the variable y, so that v_c is an algebraic homomorphism. It will be the case that $v_c(R) = \underline{\mathsf{cval}}_C(R)$. We can verify that $v_c(y) \in v_c(R)$ in $\mathit{CInst}(T_1)$ and that $v = \underline{\mathsf{collapse}} \circ v_c$ on paths over y. The case when $y \in \underline{\mathsf{dom}}\, M$ with M a root name of dictionary type is handled similarly, making use again of Lemma 6.4 (the dictionary part).

Induction case: $y_n \in P_2(y_1, \ldots, y_{n-1})$. If P_2 involves only a root name, we define v_c on y_n as before. Suppose P_2 does depend on $y_1, \ldots, y_{n-1}$. Then, by the inductive hypothesis, $v_c(P_2)$ is defined and $v(P_2) = \underline{\mathsf{collapse}}(v_c(P_2))$. We know that $v(y) \in v(P_2)$, thus there must exist an element e (in $\mathit{CInst}(T_1)$) such that $\underline{\mathsf{collapse}}(e) = v(y)$ and an element S such that $\underline{\mathsf{collapse}}(S) = v(P_2)$ and $e \in S$. Applying Lemma 6.4, we infer the existence of some e', $e' \sim e$, such that $e' \in v_c(P_2)$. Define $v_c(y) = e'$. We can verify that $\underline{\mathsf{collapse}}(v_c(y)) = v(y)$. We then extend v_c on paths depending on y_n.

Finally, we need to check that $v(Q) = v(Q')$ iff $v_c(Q) = v_c(Q')$ whenever Q and Q' are of simple type. One direction is true even when Q and Q' are arbitrary: $v_c(Q) = v_c(Q')$ implies $v(Q) = v(Q')$. Conversely, $v(Q) = v(Q')$ means $\underline{\mathsf{collapse}}(v_c(Q)) = \underline{\mathsf{collapse}}(v_c(Q'))$. But, by Lemma 6.5, $\underline{\mathsf{collapse}}$ is injective on simple type nodes. Thus $v_c(Q) = v_c(Q')$, if Q and Q' are of simple type. □

We stop here to remark that for any mapping v_c as above, for any well-defined path Q over T_2, $v_c(Q)$ is defined in $\mathit{CInst}(T_1)$. The proof is essentially the one given in Proposition 6.3.

Lemma 7.6 *Let* $T_1 = \{\vec{x} \in \vec{P_1} \; ; \; C_1(\vec{x})\}$ *and* $T_2 = \{\vec{y} \in \vec{P_2} \; ; \; C_2(\vec{y})\}$ *in which* C_1 *and* C_2 *are allowed to have equality at set/dictionary type. Then, for any context-preserving, algebraic homomorphism* $v_c : T_2 \to \mathit{CInst}(T_1)$ *such that* $v_c(C_2(\vec{y})) = \underline{\mathsf{true}}$ *there exists a homomorphism* $h : T_2 \to T_1$ *such that* $\underline{\mathsf{cval}}_C \circ h = v_c$.

Proof Sketch. We define mapping h by induction on the bindings $\vec{y} \in \vec{P_2}$ such that h maps well-defined paths to well-defined paths, is context preserving and $\underline{\mathsf{cval}}_C \circ h = v_c$. **Base case:** $y \in R$ in T_2, where R is a root name of set type. We know that $v_c(y) \in v_c(R)$ in $\mathit{CInst}(T_1)$. Also, we must have $v_c(R) = \underline{\mathsf{cval}}_C R$ (v_c is an algebraic homomorphism). By the construction of $\mathit{CInst}(T_1)$, there exists at least one S in the congruence class $\underline{\mathsf{cval}}_C R$ and one x in the congruence class $v_c(y)$ such that $x \in S$ is a binding in T_1. Define $h(y) = x$, and extend h to well-defined paths with at most one variable, y. This includes as a limit case $h(R) = R$. Since $\underline{\mathsf{cval}}_C S = \underline{\mathsf{cval}}_C R$, it follows $\vec{x} \in \vec{P_1} \vdash C_1(\vec{x}) \wedge = \underline{\mathsf{eq}}(S, h(R))$ is trivial, by Lemma 7.1.

We prove now by induction on the size of Q that, for any well-defined $Q(y)$ over T_2, $h(Q(y))$ is well-defined over T_1 and $\underline{\mathsf{cval}}_C \circ h\ (Q(y)) = v_c(Q(y))$. **Base case.** $Q = y$**:** $h(y) = x$ is well-defined over T_1, by (var-add), and $\underline{\mathsf{cval}}_C(h(y)) = v_c(y)$. $Q = R$, where R is any root name : $h(R) = R$ is well-defined, by (root-var), and $\underline{\mathsf{cval}}_C(h(R)) = v_c(R)$. The cases $Q = \underline{\mathsf{true}}, \underline{\mathsf{false}}$ are obvious. **Induction case.** $Q = Q'.\mathtt{A}$**:** Q' must be well-defined over T_2 (the rule applicable is (proj-add)) and $h(Q) = h(Q').\mathtt{A}$. By the inductive hypothesis, $h(Q')$ is well-defined over T_1 and $\underline{\mathsf{cval}}_C(h(Q')) = v_c(Q')$. Then $h(Q)$ is well-defined by (proj-add) (this time over T_1) and it is easily verified that $\underline{\mathsf{cval}}_C(h(Q)) = v_c(Q)$. $Q = \underline{\mathsf{dom}}\, Q'$**:** similar with the previous case. Finally, the interesting case is $Q = y\,!\,Q'$. The general picture, including the way $h(y)$ was defined, is described in figure 4.

Since $y\,!\,Q'$ is well-defined, it must be the case that Q' is well-defined as well and, by the inductive hypothesis, $h(Q')$ is well-defined and $\underline{\mathsf{cval}}_C(h(Q')) = v_c(Q')$. By ($\underline{\mathsf{dom}}$-add), $\underline{\mathsf{dom}}\, h(Q')$ is well-defined and

[11] A more formal proof, along the same lines, uses induction on the derivation of $Q : G_2$ or $Q \simeq_2 Q'$. See the proof of Lemma 10.2

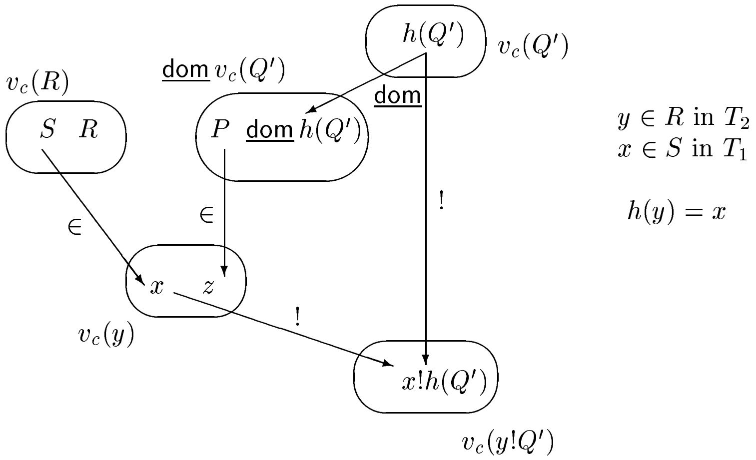


Figure 4: Proof of Lemma 7.6: well-definedness of h.

$\underline{\mathsf{cval}}_C(\underline{\mathsf{dom}}\, h(Q')) = v_c(\underline{\mathsf{dom}}\, h(Q'))$. We also know that $v_c(y\,!\,Q')$ must be defined in *CInst*(T_1), by the previous remark. Thus, it must be the case that $v_c(y) \in \underline{\mathsf{dom}}\, v_c(Q') = v_c(\underline{\mathsf{dom}}\, h(Q'))$ in *CInst*(T_1). Hence, there must exist some binding $z \in P$ in T_1 such that $\underline{\mathsf{cval}}_C z = v_c(y)$ (and therefore $z \simeq x$) and $\underline{\mathsf{cval}}_C P = v_c(\underline{\mathsf{dom}}\, h(Q'))$ (and therefore $P \simeq \underline{\mathsf{dom}}\, h(Q')$). Thus, by (!-add), $x\,!\,h(Q')$ is well-defined over T_1. Moreover, by (!-cong), $\underline{\mathsf{cval}}_C(x\,!\,h(Q')) = v_c(y\,!\,Q')$. But since $h(y\,!\,Q') = x\,!\,h(Q')$, we proved what we wanted.

For the case when $y \in \underline{\mathsf{dom}}\, M$ where M is a root name of dictionary type, h is defined in a similar way. The above proof regarding h still applies.

Induction case: $y_n \in P_2(y_1, \ldots, y_{n-1})$. If P_2 involves only a root name, we define h on y_n as before. Suppose P_2 does depend on $y_1, \ldots, y_{n-1}$. Then, by the inductive hypothesis, $h(P_2)$ is defined and $v_c(P_2) = \underline{\mathsf{cval}}_C(h(P_2))$. As with the base case there must be at least one P_1 over T_1 such that $\underline{\mathsf{cval}}_C P_1 = v_c(P_2)$ and at least one variable x_n in T_1 such that $\underline{\mathsf{cval}}_C x_n = v_c(y_n)$. Define $h(y_n) = x_n$ and extend it to paths over variables $y_1, \ldots, y_n$. We have that $v_c(y_n) = \underline{\mathsf{cval}}_C(h(y_n))$. Moreover, $\underline{\mathsf{cval}}_C P_1 = v_c(P_2) = \underline{\mathsf{cval}}_C(h(P_2))$, by the inductive hypothesis. Thus, $\vec{x} \in \vec{P_1} \vdash C_1(\vec{x}) \wedge{=} \underline{\mathsf{eq}}(P_1,\, h(P_2))$ is trivial. The previous proof regarding well-definedness of h for paths depending of $y_1, \ldots, y_n$ and the comutativity of $\underline{\mathsf{cval}}_C \circ h$ with v_c still applies.

Finally, we have to verify that $\vec{x} \in \vec{P_1} \;\vdash\; C_1(\vec{x}) \wedge{=}\, \underline{\mathsf{eq}}(Q(h(\vec{y})),\, Q'(h(\vec{y})))$ is trivial for any $\underline{\mathsf{eq}}(Q(\vec{y}),\, Q'(\vec{y}))$ that occurs in $C_2(\vec{y})$. By Lemma 7.1 it suffices to verify that $\underline{\mathsf{cval}}_C(Q(h(\vec{y}))) = \underline{\mathsf{cval}}_C(Q'(h(\vec{y})))$. But, since $\underline{\mathsf{cval}}_C \circ h = v_c$, this is equivalent to $v_c(Q) = v_c(Q')$ which we know is true. $\square$

Note that we used only one direction (the easy one) in the Lemma 7.1: $\underline{\mathsf{cval}}_C Q = \underline{\mathsf{cval}}_C Q'$ implies triviality. Putting together the last two lemmas we obtain the following:

Corollary 7.7 *Let* $T_1 = \{\vec{x} \in \vec{P_1}; C_1(\vec{x})\}$ *and* $T_2 = \{\vec{y} \in \vec{P_2}; C_2(\vec{y})\}$ *in which* C_1 *may have set/dictionary equality while* C_2 *is restricted to simple type equality only. Then, for any valuation* $v : T_2 \to$ *Inst*(T_1)*, there exists a homomorphism* $h : T_2 \to T_1$ *such that* $\underline{\mathsf{cval}} \circ h = v$.

7.3 Trivial EPCDs and query containment

We show here that query containment is reducible to triviality of EPCDs and vice-versa. Let Q_1 and Q_2 be two PC queries and define *meld*(-,-):

$$Q_1 \overset{\text{def}}{=} \underline{\mathsf{Loop}}\,(\vec{x} \in \vec{P_1})\,\underline{\mathsf{if}}\; C_1(\vec{x})\; \underline{\mathsf{then}}\; P_1'(\vec{x}) \qquad\qquad Q_2 \overset{\text{def}}{=} \underline{\mathsf{Loop}}\,(\vec{y} \in \vec{P_2})\,\underline{\mathsf{if}}\; C_2(\vec{y})\; \underline{\mathsf{then}}\; P_2'(\vec{y})$$

$$meld(Q_1, Q_2) \stackrel{\text{def}}{=} \underline{\mathsf{Loop}}\,(\vec{x} \in \vec{P_1})\, \underline{\mathsf{Loop}}\,(\vec{y} \in \vec{P_2})\; \underline{\mathsf{if}}\; C_1(\vec{x})\; \underline{\mathsf{and}}\; C_2(\vec{y})\; \underline{\mathsf{and}}\; \underline{\mathsf{eq}}(P_1'(\vec{x}),\, P_2'(\vec{y}))\; \underline{\mathsf{then}}\; P_1'(\vec{x}) \qquad \text{(or, by (eqcond), } P_2'(\vec{y}))$$

Recall (from section 3):

$$cont(Q_1, Q_2) \stackrel{\text{def}}{=} \vec{x} \in \vec{P_1} \;\vdash\; C_1(\vec{x}) \;\wedge\!\!=\; \underline{\mathsf{Some}}\,(\vec{y} \in \vec{P_2})\, C_2(\vec{y})\; \underline{\mathsf{and}}\; \underline{\mathsf{eq}}(P_1'(\vec{x}),\, P_2'(\vec{y}))$$

Clearly, $cont(Q_1, Q_2)$ proves by (idemloop) that $Q_1 = meld(Q_1, Q_2)$. If Q_1 and Q_2 are set-valued $\underline{\mathsf{BigU}}$ queries and $P_i' = \underline{\mathsf{sng}}(P_i'')$ then the meaning of $meld(Q_1, Q_2)$ is $Q_1 \cap Q_2$. Hence $Q_1 = meld(Q_1, Q_2)$ if and only if $Q_1 \subseteq Q_2$. We thus reduced query containment to query equivalence[12]. We also have that $cont(Q_1, Q_2)$ implies the containment. The next lemma, using just semantic arguments involving valuations and instances, shows that the other direction holds as well, thus, PC query containment is reducible to triviality of EPCDs. Similarly, triviality of EPCDs is reducible to query containment[13]. More precisely, let d be an EPCD: $\vec{x} \in \vec{P_1} \vdash C_1(\vec{x}) \wedge\!\!= \underline{\mathsf{Some}}\,(\vec{y} \in \vec{P_2}(\vec{x}))\, C_2(\vec{x}, \vec{y})$. Define:

$$front(d) \stackrel{\text{def}}{=} \underline{\mathsf{BigU}}\,(\vec{x} \in \vec{P_1})\, \underline{\mathsf{if}}\; C_1(\vec{x})\; \underline{\mathsf{then}}\; \underline{\mathsf{sng}}\langle \vec{\mathtt{A}} : \vec{x} \rangle \qquad (\vec{\mathtt{A}} \text{ are fresh labels})$$

$$back(d) \stackrel{\text{def}}{=} \underline{\mathsf{BigU}}\,(\vec{x} \in \vec{P_1})\, \underline{\mathsf{BigU}}\,(\vec{y} \in \vec{P_2}(\vec{x}))\, \underline{\mathsf{if}}\; C_1(\vec{x})\; \underline{\mathsf{and}}\; C_2(\vec{x}, \vec{y})\; \underline{\mathsf{then}}\; \underline{\mathsf{sng}}\langle \vec{\mathtt{A}} : \vec{x} \rangle$$

Lemma 7.8 (Reducibility) *Let Q_1 and Q_2 be two set-valued PC queries and let d be an EPCD, with no restriction to simple types. Then:*

1. *$Q_1 \subseteq Q_2$ if and only if $cont(Q_1, Q_2)$ is trivial*
2. *d is trivial if and only if $front(d) \subseteq back(d)$*

Theorem 7.9 (Containment) *Let*

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

be two set-valued PC queries (and T_1, respectively T_2, their tableaux). C_2, P_1' and P_2' are simple-type restricted, while C_1 maye have set/dictionary equality. Then the following are equivalent:

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

(b) *there exists homomorphism $T_1 \xleftarrow{h} T_2$ such that $\vec{x} \in \vec{P_1} \vdash C_1(\vec{x}) \wedge\!\!= \underline{\mathsf{eq}}(P_1'(\vec{x}),\, P_2'(h(\vec{y})))$*[14]

Proof: Suppose (a) is true. Then $\underline{\mathsf{sng}}\; \underline{\mathsf{cval}}(P_1'(\vec{x})) \in Q_1(Inst(T_1)) \subseteq Q_2(Inst(T_1))$. Thus, there must exist a valuation $v : T_2 \to Inst(T_1)$ such that $\underline{\mathsf{cval}}(P_1'(\vec{x})) = v(P_2'(\vec{y}))$. By Corollary 7.7 there exists a homomorphism $h : T_2 \to T_1$ such that $v(P_2'(\vec{y})) = \underline{\mathsf{cval}} \circ h(P_2'(\vec{y}))$. Hence, $\underline{\mathsf{cval}}(P_1'(\vec{x})) = \underline{\mathsf{cval}} \circ h(P_2'(\vec{y}))$. By Lemma 7.1, it follows that $\vec{x} \in \vec{P_1} \vdash C_1(\vec{x}) \wedge\!\!= \underline{\mathsf{eq}}(P_1'(\vec{x}),\, P_2'(h(\vec{y})))$ is trivial. Since $Inst(T_1)$ is finite, the same proof works for both (a1) $\Rightarrow$ (b) and (a2) $\Rightarrow$ (b).

Conversely, suppose (b) holds, and let I be an instance and $v : T_1 \to I$ a valuation. The result of Q_1 on I for valuation v is $v(P_1')$. Observe that $v \circ h$ is a valuation from T_2 into I. Moreover, $v \circ h(P_2') = v(P_1')$ since $\vec{x} \in \vec{P_1} \vdash C_1(\vec{x}) \wedge\!\!= \underline{\mathsf{eq}}(P_1'(\vec{x}),\, P_2'(h(\vec{y})))$ is trivial. Thus the result of Q_2 on I contains $v(P_1')$. Since I and v were chosen arbitrary, and I can be either finite or infinite, it follows that $Q_1 \subseteq Q_2$ in the finite and unrestricted. $\square$

Before we can allow non-simple type variables in $\vec{x}$ in theorem 7.11 we need to strengthen Lemma 7.5. It is the case then that Lemma 7.5 becomes a particular case of the following lemma when $\vec{x}$ is taken to be empty.

[12]This holds, in general, for *any* $\underline{\mathsf{BigU}}$ queries with bodies $\underline{\mathsf{sng}}$ expressions (as observed in section 3).

[13]The two reductions are basic properties that hold, in principle, for a larger class of queries and dependencies, not only path-conjunctive.

[14]Here P_1' and P_2' may be record expressions, and thus their equality is represented by a conjunction of equalities. However, since everything is well-defined, by our observation in section 6, this is equivalent to a set of EGDs, and thus its triviality is a PTIME condition.

Lemma 7.10 *Let* $T_1 = \{\vec{x} \in \vec{P_1}, \vec{z} \in \vec{P_1'}(\vec{x})\ ;\ C_1(\vec{x})\ \underline{\mathsf{and}}\ C_2(\vec{x}, \vec{z})\}$ *and* $T_2 = \{\vec{x} \in \vec{P_1}, \vec{y} \in \vec{P_2}(\vec{x})\ ;\ C_1(\vec{x})\ \underline{\mathsf{and}}\ C_3(\vec{x}, \vec{y})\}$ *in which* C_1 *and* C_2 *may have set/dictionary equality but* C_3 *is simple type restricted. Then, for any valuation* $v : T_2 \to Inst(T_1)$ *such that* $v(\vec{x}) = \underline{\mathsf{cval}}(\vec{x})$[15] *there exists a context-preserving, algebraic homorphism,* $v_c : T_2 \to CInst(T_1)$ *such that* $v = \underline{\mathsf{collapse}} \circ v_c$ *and* $v_c(\vec{x}) = \underline{\mathsf{cval}}_C(\vec{x})$*. Moreover,* v *and* v_c *satisfy the same set of formulas* $\underline{\mathsf{eq}}(Q, Q')$ *over* T_2 *with* Q *and* Q' *of simple type.*

Proof Sketch. The proof is similar with that of Lemma 7.5. However, $v_c(x)$ must be defined as $\underline{\mathsf{cval}}_C(x)$ in this case. In contrast, for $v_c(y)$ we had the liberty to *choose* an element e with $\underline{\mathsf{collapse}}(e) = v(y)$ such that the requirements for v_c were satisfied. Thus, we have to verify that indeed $v_c(x) \in \overline{v_c(P_1)}$ for any binding $x \in P_1$ (it is obvious that $\underline{\mathsf{collapse}}(v_c(\vec{x})) = v(\vec{x})$). We prove this by induction on $\vec{x} \in \vec{P_1}$.

Base case: $x \in R$ where R is a root name of set type. Then since $x \in R$ is also in T_1 it must be the case that $\underline{\mathsf{cval}}_C x \in \underline{\mathsf{cval}}_C R$ in $CInst(T_1)$. Also, we must have $v_c(R) = \underline{\mathsf{cval}}_C(R)$, thus $v_c(x) \in v_c(R)$. The dictionary case is similar.

Inductive case: The root name case is as before. Suppose that $x_n \in P_1(x_1, \ldots, x_{n-1})$. $v(x_n) \in v(P_1(x_1, \ldots, x_{n-1}))$ and by the inductive hypothesis $v_c(P_1)$ is defined and $v_c(P_1) = \underline{\mathsf{cval}}_C(P_1)$. Again, we use the fact that $x_n \in P_1$ is also in T_1 and therefore $\underline{\mathsf{cval}}_C(x_n) \in \underline{\mathsf{cval}}_C(P_1)$, thus $v_c(x_n) \in v_c(P_1)$.

On $\vec{y}$, v_c is defined in a similar manner as in the proof of Lemma 7.5. Finally, as in that proof, v_c and v satisfy the same set of simple-type formulas over T_2. □

Theorem 7.11 (Trivial dependencies) *Let* d *be an EPCD:* $\vec{x} \in \vec{P_1} \ \vdash\ C_1(\vec{x}) \wedge\!\!= \underline{\mathsf{Some}}\,(\vec{y} \in \vec{P_2}(\vec{x}))\, C_2(\vec{x}, \vec{y})$ *and* T_1 *and* T_2 *such that* $d = dep(T_1, T_2)$*.* C_2 *is simple-type restricted while* C_1 *may have set/dictionary equality. Then the following are equivalent:*

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

(b) *there exists homomorphism* $T_1 \xleftarrow{h} T_2$ *such that* $\vec{x} \in \vec{P_1} \ \vdash\ C_1(\vec{x}) \wedge\!\!= \underline{\mathsf{eq}}(\vec{x}, h(\vec{x}))$[16]

Proof: Suppose (a) holds. Then $front(d) \subseteq back(d)$. We cannot apply Theorem 7.9 because $\vec{x}$ may be of non-simple type [17], but we extend the proof of it. Reasoning like there we conclude that there must exist a valuation $v : T_2 \to Inst(T_1)$ such that $v(\vec{x}) = \underline{\mathsf{cval}}(\vec{x})$. By Lemma 7.10 (in which $\vec{z}$ and C_2 are empty) there exists $v_c : T_2 \to CInst(T_1)$ such that $v_c(\vec{x}) = \underline{\mathsf{cval}}_C(\vec{x})$ and $v_c(C_2(\vec{x}, \vec{y})) = \underline{\mathsf{true}}$ (provided that C_2 involves only simple-type equality). Then, by Lemma 7.6 there exists homomorphism $h : T_2 \to T_1$ such that $\underline{\mathsf{cval}}_C \circ h = v_c$. Therefore $\underline{\mathsf{cval}}_C(h(\vec{x}) = \underline{\mathsf{cval}}_C(\vec{x})$. Then we apply the EGD lemma (the non-simple type part) and conclude that $\vec{x} \in \vec{P_1} \ \vdash\ C_1(\vec{x}) \wedge\!\!= \underline{\mathsf{eq}}(\vec{x}, h(\vec{x}))$ is trivial. For (b) $\Rightarrow$ (a) the proof is similar with the proof of (b) $\Rightarrow$ (a) in Theorem 7.9. □.

We remark here that Theorem 7.11 is a stronger statement than Theorem 7.9. Indeed, there is a direct proof of the containment theorem from the triviality theorem that uses the reducibility lemma. In Section 11 we show that CoDi's axiomatization is complete for proving triviality of EPCDs and therefore containment/equivalence of path-conjunctive $\underline{\mathsf{BigU}}$ queries.

Corollary 7.12 *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]*).*

8 The PC chase

Before defining the chase we observe that the reducibility lemma holds in the presence of dependencies as well.

[15]Here $\underline{\mathsf{cval}}$ and $\underline{\mathsf{cval}}_C$ are the canonical mappings from T_1 into $Inst(T_1)$, respectively $CInst(T_1)$.

[16]Same observation as in the previous theorem.

[17]Nonetheless, the fact that we are being able to extend that proof to this case encourages the hope that the simple type restriction on P_1' and P_2' can be removed for the containment theorem.

Lemma 8.1 (Reducibility under dependencies) *Let Q_1 and Q_2 be two set-valued PC queries , let d be an EPCD, and let D be a set of EPCDs, all with no restriction to simple types. Then:*

1. $Q_1 \subseteq_D Q_2$ *if and only if* $D \models cont(Q_1, Q_2)$
2. $D \models d$ *if and only if* $front(d) \subseteq_D back(d)$

It suffices to define the chase on tableaux: chasing a query or a dependency means in fact chasing the underlying tableau.

Definition 8.2 (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 $T \xrightarrow{d} T'$.*

For a query $Q = \underline{\mathsf{Loop}}(T)P'$, if $T \xrightarrow{d} T'$, then we write $Q \xrightarrow{d} Q'$ where $Q' = \underline{\mathsf{Loop}}(T')P'$. Similarly, we may have $d' \xrightarrow{d} d''$ (if we express d' and d'' in the form with $\underline{\mathsf{All}}$).

The following lemma is easily verified (in fact we already proved a particular case of it in Lemma 6.2).

Lemma 8.3 *Let $T = \{\vec{x} \in \vec{P} \ ; \ C(\vec{x})\}$ and $T' = \{\vec{x} \in \vec{P}, \vec{y} \in \vec{S}(\vec{x}) \ ; \ C(\vec{x}) \ \underline{\mathsf{and}}\ D(\vec{x}, \vec{y})\}$ be two tableaux, and let $G, \simeq$, and $G', \simeq'$ be the set of well-defined paths and the congruence closure for T and, respectively, T'. Then $G \subseteq G'$ and $\simeq \subseteq \simeq'$.*

For the following lemma, assume that $T \xrightarrow{d} T'$, and $d = dep(T_r, T_{rs})$, T, T', and $h : T_r \to T$ are as in the definition of the chase step. Let $\underline{\mathsf{id}}_{\vec{s}}$ be the identity mapping on $\vec{s}$.

Lemma 8.4 *$h \ \cup \ \underline{\mathsf{id}}_{\vec{s}} : T_{rs} \to T'$ is a homomorphism.*

Proof. We will use G_{rs}, G', G_r, and G to denote the sets of well-defined paths for, respectively, T_{rs}, T', T_r and T. Similar notation for $\simeq$. Let $h' \stackrel{\text{def}}{=} h \ \cup \ \underline{\mathsf{id}}_{\vec{s}}$. First we prove, by induction on the derivation of $Q : G_{rs}$ or $Q_1 \simeq_{rs} Q_2$, that $Q : G_{rs}$ implies $h'(Q) : G'$ and $Q_1 \simeq_{rs} Q_2$ implies $h'(Q_1) \simeq' h'(Q_2)$. The interesting cases are (!-add) and (eq).

For (!-add) suppose Q is of the form $u\,!\,Q'$ where u is a variable in T_{rs} (either an r or an s). Then to derive $u\,!\,Q' : G_{rs}$, we must have had two possibilities. The first case:

$$\frac{u \simeq_{rs} r_i, \qquad r_i \in R_i \text{ in } T_{rs}, \qquad R_i \simeq_{rs} \underline{\mathsf{dom}}\, Q'}{u\,!\,Q' : G_{rs}}$$

Then, by the inductive hypothesis, we have $h'(u) \simeq' h'(r_i)$ and $h'(R_i) \simeq' \underline{\mathsf{dom}}\, h'(Q')$. On the other hand, $h'(r_i) = h(r_i) = x_k$ for some $x_k \in P_k$ in T. Using condition (2) of Proposition 7.4, since h is a homomorphism, we obtain that $P_k \simeq h(R_i)$ and thus $P_k \simeq h'(R_i)$. By Lemma 8.3, we must have $P_k \simeq' h'(R_i)$. Then, by (trans), $P_k \simeq' \underline{\mathsf{dom}}\, h'(Q')$. Putting it all together, the following instance of (!-add) is applicable:

$$\frac{h'(u) \simeq' h'(r_i), \qquad h'(r_i) \in P_k \text{ in } T', \qquad P_k \simeq' \underline{\mathsf{dom}}\, h'(Q')}{h'(u)\,!\,h'(Q') : G'}$$

thus concluding that $h'(u\,!\,Q') : G'$. The second case:

$$\frac{u \simeq_{rs} s_i, \quad s_i \in S_i \text{ in } T_{rs}, \quad S_i \simeq_{rs} \underline{\mathsf{dom}}\, Q'}{u\,!\,Q' : G_{rs}}$$

is simpler. Again, by the inductive hypothesis, $h'(u) \simeq' h'(s_i)$ and $h'(S_i) \simeq' \underline{\mathsf{dom}}\, h'(Q')$. But since $h'(s_i) = s_i$ and $h'(S_i) = S_i$ and $s_i \in S_i$ occurs in T', we can immediately apply (!-add) to conclude that $h'(u\,!\,Q') : G'$.

For the (eq) rule, we have again two cases. The first case is when $Q_1 \simeq_{rs} Q_2$ is obtained by:

$$\frac{\underline{\mathsf{eq}}(Q_1,\, Q_2) \text{ occurs in } B_1(\vec{r}), \quad Q_1 : G_{rs}, \quad Q_2 : G_{rs}}{Q_1 \simeq_{rs} Q_2}$$

It must be the case that $Q_1 = Q_1(\vec{r})$ and $Q_2 = Q_2(\vec{r})$, and $Q_1 : G_r$, $Q_2 : G_r$. Then, by (eq), $Q_1 \simeq_r Q_2$. Since h is a homomorphism, it follows by Proposition 7.4 that $h(Q_1) \simeq h(Q_2)$, and therefore, $h'(Q_1) \simeq h'(Q_2)$. By Lemma 8.3, $h'(Q_1) \simeq' h(Q_2)$ as well. The second case:

$$\frac{\underline{\mathsf{eq}}(Q_1,\, Q_2) \text{ occurs in } B_2(\vec{r}, \vec{s}), \quad Q_1 : G_{rs}, \quad Q_2 : G_{rs}}{Q_1 \simeq_{rs} Q_2}$$

Then, since $h'(B_2(\vec{r}, \vec{s})) = B_2(h(\vec{r}), \vec{s})$, we have that $\underline{\mathsf{eq}}(h'(Q_1),\, h'(Q_2))$ occurs in $B_2(h(\vec{r}), \vec{s})$. On the other hand, by the inductive hypothesis, $h'(Q_1) : G'$ and $\overline{h'(}Q_2) : G'$. Thus, we can apply (eq) to conclude that $h'(Q_1) \simeq' h'(Q_2)$.

We still need to show that h' satisfies condition (2) of Proposition 7.4. We already showed it in the above proof, but we repeat it. Suppose $r_i \in R_i$ is a binding in T_{rs}. Then $h'(r_i) = h(r_i) = x_k$ for some $x_k \in P_k$ in T. By Proposition 7.4, h is a homomorphism implies $h(R_i) \simeq P_k$ and thus $h'(R_i) \simeq' P_k$ (by Lemma 8.3). For the case $s_i \in S_i$ we use the fact that $h'(s_i) = s_i$ and that $h'(S_i) = S_i$, plus reflexivity. □

Lemma 8.5 (Chase properties) (1) *If $Q \xrightarrow{d} Q'$ then $Q = Q'$ is provable from d in CoDi .* (2) *If $Inst(T) \not\models d$ then d is applicable to T (Here, B_1 is simple-type restricted).*

Proof. Part (1), the soundness of chase, is shown in section 11, where we simulate the chase step via equational rewriting in CoDi. For the second part, we prove the contrapositive. Assume the same notations as in the definition of the chase step and let T_r and T_{rs} be the two tableaux such that $d = dep(T_r, T_{rs})$. Assume d is not applicable to T, and let $v : T_r \to Inst(T)$ an arbitrary valuation. We need to extend this to a valuation $v' : T_{rs} \to Inst(T)$ such that $v'(\vec{r}) = v(\vec{r})$ and $v'(B_2(\vec{r}, \vec{s})) = \underline{\mathsf{true}}$. By Corollary 7.7 there exists a homomorphism $h : T_r \to T$ such that $\underline{\mathsf{cval}} \circ h = v$. Since d is not applicable to T, it must be the case that there exists homomorphism $h' : T' \to T$ such that $\vec{x} \in \vec{P} \vdash C(\vec{x}) \wedge\!= \underline{\mathsf{eq}}(\vec{x},\, h'(\vec{x}))$. Now, observe that $h'' \stackrel{\text{def}}{=} h \cup \underline{\mathsf{id}}_{\vec{s}}$ is a homomorphism from T_{rs} into T', by Lemma 8.4. Take $v' \stackrel{\text{def}}{=} \underline{\mathsf{cval}} \circ h' \circ h''$.

It is easy to see that v' is context-preserving. Moreover, $v'(\vec{r}) = \underline{\mathsf{cval}}(h'(h(\vec{r})))$. Since $\vec{x} \in \vec{P} \vdash C(\vec{x}) \wedge\!= \underline{\mathsf{eq}}(\vec{x},\, h'(\vec{x}))$ is trivial, it follows by the EGD Lemma, part 2, that $\underline{\mathsf{cval}}(\vec{x}) = \underline{\mathsf{cval}}(h'(\vec{x}))$ and thus $\underline{\mathsf{cval}}(h(\vec{r})) = \underline{\mathsf{cval}}(\overline{h'}(h(\vec{r})))$. Therefore $v'(\vec{r}) = \underline{\mathsf{cval}}(h(\vec{r}))$, and hence $v'(\vec{r}) = v(\vec{r})$. Finally, $v'(B_2(\vec{r}, \vec{s})) = B_2(\underline{\mathsf{cval}}(h'(h(\vec{r}))), \underline{\mathsf{cval}}(h'(\vec{s})))$. Now, since $\vec{x} \in \vec{P} \vdash C(\vec{x}) \wedge\!= B_2(h'(h(\vec{r})), h'(\vec{s}))$ is trivial (h' is a homomorphism), the last term equals $\underline{\mathsf{true}}$ (again the EGD Lemma, part 2). We conclude that $Inst(T) \models d$. □

It is easy to find a counterexample to Lemma 8.5, part (2), for the case when $B_1(\vec{r})$ involves non-simple type equality. Let

$$d = a \in R, b \in S \;\vdash\; \underline{\mathsf{eq}}(R,\, S) \wedge\!= \underline{\mathsf{eq}}(a,\, b)$$

and

$$T = \{a \in R, a' \in R, b \in S, b' \in S \;;\; \underline{\mathsf{eq}}(a,\, b) \;\underline{\mathsf{and}}\; \underline{\mathsf{eq}}(a',\, b')\}.$$

Then it is easy to see that *Inst*$(T) \not\models d$, since R and S are equal in *Inst*(T) (see figure 5b), but still there are elements in R, respectively S that are not equal. Also, d is not applicable to T because there is no homomorphism to map the tableau $\{a \in R, b \in S \;\vdash\; \mathsf{eq}(R,\, S)\}$ to T. The problem seems to be related to the fact that R and S are equated "artificially" in *Inst*$(\overline{T})$ (as a result of our enforcement of extensionality) even though their equality does not follow as a logical consequence from $\mathsf{eq}(a,\, b)$ $\underline{\mathsf{and}}$ $\mathsf{eq}(a',\, b')$ (this is in fact the reason why d is not applicable to T). The conclusion is that there are limitations on how useful is *Inst*(T) in characterizing dependencies in the presence of non-simple type equality. We will relax later our notion of instance to *weak instance* (not necessarily extensional) and we will draw some interesting parallels between the two of them. In particular, *CInst*(T) is a weak instance and one can observe that *CInst*$(T) \models d$ (R and S are distinct, see figure 5a).

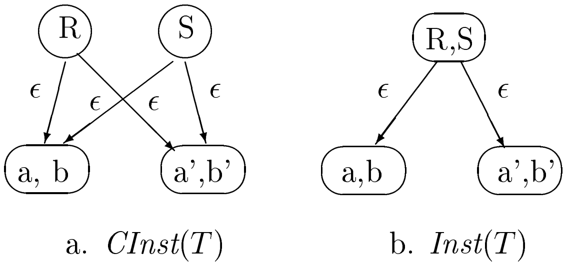


Figure 5: Counter-example to Lemma 8.5 part 2 for non-simple type equality in $B_1(\vec{r})$

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), *Inst*$(T_n) \models D$. We will see that this is essential for the proof of the main theorem of this section. The main idea there is that $D \models d$ implies *Inst*$(T_n) \models d$ and thus we are able to infer the existence of a certain homomorphism[18]. The fact that Lemma 8.5, part 2, fails when dependencies in D may allow for non-simple type equality in $B_1(\vec{r})$ implies that we cannot use the above mentioned idea in the proof of the theorem. However, it is not a counterexample to the theorem itself. Other means for proving the theorem might exist, at least, in principle. We will limit ourselves here to simple type equality in $B_1(\vec{r})$.

For any PC query $Q = \underline{\mathsf{Loop}}(T)P'$ we use the notation $chase_D(Q)$ for $\underline{\mathsf{Loop}}(T_n)P'$ (similar notation: $chase_D(d)$).

Theorem 8.6 (Terminating chase) *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 $chase_D(Q_1)$). Moreover, D is restricted as discussed above, while Q_1 and Q_2 are restricted as in Theorem 7.9. Then the following are equivalent:*

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

   (b1) $chase_D(Q_1) \subseteq^{unr} Q_2$ (b2) $chase_D(Q_1) \subseteq^{fin} Q_2$

   (c) *$Q_1 \subseteq Q_2$ 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. Moreover, D is restricted as discussed above, while d is restricted as in Theorem 7.11. Then 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)*

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

[18] The proof in the relational case uses the same idea [BV84b]

Proof. The provability in CoDi will be shown in section 11. We focus here on items (a) and (b) of the theorem. We prove part 2 first. If $chase_D(d)$ is trivial then $D \models d$, by repeatedly applying Lemma 8.5. For the interesting direction assume $D \models d$. Suppose $d = \vec{x} \in \vec{P} \;\vdash\; C(\vec{x}) \wedge\!\!= \underline{\mathsf{Some}}\,(\vec{y} \in \vec{R}(\vec{x}))\, D(\vec{x},\vec{y})$, let T_x and T_{xy} be such that $d = dep(T_x, T_{xy})$. Let the result of chasing T_x with D be the tableau $T = \{\vec{x} \in \vec{P}, \vec{z} \in \vec{Q}(\vec{x}) \;;\; C(\vec{x}) \;\underline{\mathsf{and}}\; B(\vec{x},\vec{z})\}$. Then $chase_D(d) = dep(T, T')$ where $T' = \{\vec{x} \in \vec{P}, \vec{z} \in \vec{Q}(\vec{x}), \vec{y} \in \vec{R}(\vec{x}) \;;\; C(\vec{x}) \;\underline{\mathsf{and}}\; B(\vec{x},\vec{z}) \;\underline{\mathsf{and}}\; D(\vec{x},\vec{y})\}$. To show that this is trivial, we show that there is a homomorphism from T' into T that is the identity on $\vec{x}, \vec{z}$.

$v = \underline{\mathsf{cval}} \,\circ\, \underline{\mathsf{id}}_{\vec{x}}$ is a valuation from T_x into $Inst(T)$. Since $Inst(T) \models D$ and $D \models d$, we have $Inst(T) \models d$, too. Thus there exists a valuation $v' : T_{xy} \to Inst(T)$ such that $v'(\vec{x}) = v(\vec{x})$ (see figure 6a). Thus, $v'(\vec{x}) = \underline{\mathsf{cval}}(\vec{x})$ where $\underline{\mathsf{cval}}$ is the canonical valuation corresponding to T. Therefore, we can apply Lemma 7.10 and conclude that there exists a mapping $v'_c : T_{xy} \to CInst(T)$ (see figure 6b) such that $v'_c(\vec{x}) = \underline{\mathsf{cval}}_C(\vec{x})$, $v' = \underline{\mathsf{collapse}} \,\circ\, v'_c$ and, making use of the fact that $D(\vec{x},\vec{y})$ is simple type restricted, $v'_c(D(\vec{x},\vec{y})) = \underline{\mathsf{true}}$.

$$
\begin{tikzcd}
T \arrow[d, "\underline{\mathsf{cval}}"'] & T_x \arrow[l, "\underline{\mathsf{id}}_{\vec{x}}"'] \arrow[dl, "v"'] & T_{xy} \arrow[dll, "v'"] \\
Inst(T) & &
\end{tikzcd}
$$

$$v'(\vec{x}) = v(\vec{x}) = \underline{\mathsf{cval}}(\vec{x})$$

a.

$$
\begin{tikzcd}
 & T' \arrow[dl, "h"'] \\
T \arrow[d, "\underline{\mathsf{cval}}_C"'] & T_{xy} \arrow[l, "h'"'] \arrow[dl, "v'_c"] \arrow[ddl, "v'"] \\
CInst(T) \arrow[d, "\underline{\mathsf{collapse}}"'] & \\
Inst(T) &
\end{tikzcd}
$$

$$v'_c(\vec{x}) = \underline{\mathsf{cval}}_C(\vec{x})$$

b.

Figure 6: Proof of the chase theorem

Then, by Lemma 7.6 there exists homomorphism $h' : T_{xy} \to T$ such that $\underline{\mathsf{cval}}_C \,\circ\, h' = v'_c$. Thus, $\underline{\mathsf{cval}}_C(h'(\vec{x})) = \underline{\mathsf{cval}}_C(\vec{x})$. Now, consider $h = h' \cup \underline{\mathsf{id}}_{\vec{z}}$. Obviously, h is a context-preserving mapping from T' into T. Moreover, $\underline{\mathsf{cval}}_C(h(\vec{x})) = \underline{\mathsf{cval}}_C(h'(\vec{x})) = \underline{\mathsf{cval}}_C(\vec{x})$, hence by the EGD Lemma part (2),

$$\vec{x} \in \vec{P}, \vec{z} \in \vec{Q}(\vec{x}) \;\vdash C(\vec{x}) \;\underline{\mathsf{and}}\; B(\vec{x},\vec{z}) \wedge\!\!= \underline{\mathsf{eq}}(h(\vec{x}),\, \vec{x})$$

is trivial. A similar triviality holds for $\vec{z}$ since h is defined to be the identity on $\vec{z}$. Finally, $\underline{\mathsf{cval}}_C(h(D(\vec{x},\vec{y}))) = \underline{\mathsf{cval}}_C(h'(D(\vec{x},\vec{y}))) = v'_c(D(\vec{x},\vec{y})) = \underline{\mathsf{true}}$. Again, by the EGD Lemma part (2),

$$\vec{x} \in \vec{P}, \vec{z} \in \vec{Q}(\vec{x}) \;\vdash C(\vec{x}) \;\underline{\mathsf{and}}\; B(\vec{x},\vec{z}) \wedge\!\!= D(h(\vec{x}), h(\vec{y}))$$

is trivial. (The case for $B(\vec{x},\vec{z})$ is immediate.) We conclude that h is a homomorphism from T' into T, identity on the variables of T. Thus, $chase_D(d)$ is trivial.

Part 1 follows from Part 2 by applying the two Reducibility lemmas. First, $Q_1 \subseteq_D Q_2$ iff $D \models cont(Q_1, Q_2)$, by Lemma 8.1. But, by part 1, $D \models cont(Q_1, Q_2)$ iff $chase_D(cont(Q_1, Q_2))$ is trivial. Now, we observe that $chase_D(cont(Q_1, Q_2)) = cont(chase_D(Q_1), Q_2)$. Then, $cont(chase_D(Q_1), Q_2)$ trivial iff $chase_D(Q_1) \subseteq Q_2$, by Lemma 7.8. □

9 Full EPCDs

Before we define full EPCDs we need to consider a slight extension of path expressions that takes into account record constructors. Formally, an *extended path* EP is described by the following:

$$EP \ ::= \ \langle A_1 : EP_1, \ldots, A_n : EP_n \rangle \ \mid \ P$$

where P is a path expression as defined before. Note that the record constructor and the constructors that occur in P are not orthogonal, i.e. cannot be composed in an arbitrary way. This restricted form suffices for our purposes. We also note that we represent equality between extended paths, componentwise, as conjunction of equalities between paths. We say that an extended path EP is well-defined (see section 6) over some tableau T if every P occuring in EP is well-defined over T.

Definition 9.1 (Full dependencies) [19] *Let* $d \stackrel{\text{def}}{=} \vec{r} \in \vec{R} \ \vdash \ B_1(\vec{r}) \wedge\!\!= \underline{\mathsf{Some}}\,(\vec{s} \in \vec{S}(\vec{r}))\ B_2(\vec{r}, \vec{s})$ *be an EPCD and* T_r *and* T_{rs} *be the two tableaux such that* $d = dep(T_r, T_{rs})$*. Then* d *is* full *if, for any path* $Q(\vec{r}, \vec{s})$ *over* T_{rs} *there exists an extended path* $EP_Q(\vec{r})$ *over* T_r *such that the following EGD is trivial:*

$$\vec{r} \in \vec{R}, \vec{s} \in \vec{S}(\vec{r}) \ \vdash \ B_1(\vec{r}) \ \underline{\mathsf{and}}\ B_2(\vec{r}, \vec{s}) \ \wedge\!\!= \ \underline{\mathsf{eq}}(Q(\vec{r}, \vec{s}), \, EP_Q(\vec{r}))$$

Informally, a full EPCD asserts the existence of elements $s_1, \ldots, s_n$ such that all the new paths that are well-defined over T_{rs} are determined (up to record constructors) in terms of $\vec{r}$. We remark that the new paths may include paths accesible from both $\vec{s}$ and $\vec{r}$. For example, if $\underline{\mathsf{eq}}(r,\, s)$ occurs in $B_2(\vec{r}, \vec{s})$ and $s \in \underline{\mathsf{dom}}\, M$ is a binding in the $\underline{\mathsf{Some}}$ part of d then $r\,!\,M$ becomes well-defined over T_{rs} although it may have not been so over T_r.

Checking whether an EPCD is full is in PTIME (in the size of the dependency). This is because, as we observed earlier, there are polynomially many paths over T_{rs} and triviality of EGDs is in PTIME. For each well-defined path over T_{rs}, if it is not a record, we only need to check whether its $\simeq$-equivalence class contains a path well-defined over T_r, while if it is a record, we may need to go recursively on each attribute of it and check its corresponding $\simeq$-equivalence class.

The next results are concerned with a step of chasing with a full EPCD d. Let $T = \{\vec{x} \in \vec{P} \ ; \ C(\vec{x})\}$ be a PC tableau. Let $T \xrightarrow{d} T'$ be a chase step, where $T' = \{\vec{x} \in \vec{P}, \vec{s} \in \vec{S}(h(\vec{r})) \ ; \ C(\vec{x}) \ \underline{\mathsf{and}}\ B_2(h(\vec{r}), \vec{s})\}$ for some homomorphism $T \xleftarrow{h} \{\vec{r} \in \vec{R}; B_1(\vec{r})\}$. Moreover, we assume that $C(\vec{x})$, $B_1(\vec{r})$ and $B_2(\vec{r}, \vec{s})$ are simple-type restricted. Then every well-defined path over T' is determined by a well-defined extended path over T. The proof consists of several simple observations.

Lemma 9.2 *Let* $T = \{\vec{x} \in \vec{P} \ ; \ C(\vec{x})\}$ *be any tableau such that* $C(\vec{x})$ *is simple type restricted. Then* $P \simeq \underline{\mathsf{dom}}\, Q$ *implies that* $P = \underline{\mathsf{dom}}\, P'$ *for some* P' *such that* $P' \simeq Q$*.*

Proof. Induction on the derivation of $P \simeq \underline{\mathsf{dom}}\, Q$ or $\underline{\mathsf{dom}}\, Q \simeq P$. We give the proof for the case $P \simeq \underline{\mathsf{dom}}\, Q$ (the other one is symmetric). The cases that may be possible, given the special form $\underline{\mathsf{dom}}\, Q$, are as follows. First, (eq): this is not possible since $C(\vec{x})$ is simple-type restricted. Second, ($\underline{\mathsf{dom}}$-cong): then P must be of the form $\underline{\mathsf{dom}}\, P'$ and $P' \simeq Q$. (refl) is trivial. For (trans), there exists some P_1 such that $P \simeq P_1$ and $P_1 \simeq \underline{\mathsf{dom}}\, Q$. Applying the inductive hypothesis for $P_1 \simeq \underline{\mathsf{dom}}\, Q$, it must be that $P_1 = \underline{\mathsf{dom}}\, P_1'$ with $P_1' \simeq Q$. Therefore we can use the inductive hypothesis again, this time for $P \simeq \underline{\mathsf{dom}}\, P_1'$ to obtain that $P = \underline{\mathsf{dom}}\, P'$ for some $P' \simeq P_1'$. Then we apply transitivity. For (sym) we use the inductive hypothesis (for $\underline{\mathsf{dom}}\, Q \simeq P$) and (sym). □

Coming back to the previous notations regarding the chase step, recall that, by Lemma 8.4, $h' = h \ \cup \underline{\mathsf{id}}_{\vec{s}}$ is a homomorphism from T_{rs} into T'. As we did before, we will use G_{rs}, G', G_r, and G to denote the sets of well-defined paths for, respectively, T_{rs}, T', T_r and T. Similar notation for $\simeq$.

[19] The definition given in [PT99] turned out to be erroneous by being too general. The definition for full EPCDs that we give here is the one for which we give the proofs of our theorems.

Lemma 9.3 *For any $Q(\vec{x},\vec{s}) : G'$ either $Q \simeq' Q_0$, for some $Q_0 : G$, or $Q \simeq' h'(Q_{rs})$, for some $Q_{rs} : G_{rs}$.*

Proof. Induction on the derivation of $Q : G'$. The base cases are trivial. For (var-add): if $Q = x$ then $Q : G$ and $Q \simeq Q$, therefore $Q \simeq' Q$. If $Q = s$ then $Q = h'(s)$ and $s : G_{rs}$. For (proj-add): $Q.\mathtt{A} : G'$ and $Q : G'$. We apply the inductive hypothesis for Q and we have two cases. First, suppose that $Q \simeq' Q_0$, for some $Q_0 : G$. Then, by (proj-add), $Q_0.\mathtt{A} : G$ and therefore $Q_0.\mathtt{A} : G'$. Hence, by (proj-cong), $Q.\mathtt{A} \simeq' Q_0.\mathtt{A}$. Second case, suppose that $Q \simeq' h'(Q_{rs})$, for some $Q_{rs} : G_{rs}$. Then $Q_{rs}.\mathtt{A} : G_{rs}$. By (proj-cong), $Q.\mathtt{A} \simeq' h'(Q_{rs}).\mathtt{A} = h'(Q_{rs}.\mathtt{A})$. The case ($\underline{\mathsf{dom}}$-add) is similar. Finally, the interesting case is (!-add):

$$\frac{u \simeq' v, \qquad v \in R \text{ in } T', \qquad R \simeq' \underline{\mathsf{dom}}\, Q}{u \,!\, Q : G'}$$

We know, by the previous lemma and by our simple-type restriction, that $R = \underline{\mathsf{dom}}\, R'$ for some $R' \simeq' Q$. Then, by (!-add), $v \,!\, R' : G'$ and, by (!-cong), $u \,!\, Q \simeq' v \,!\, R'$. Now, we have two possible cases for the variable v. First, $v = x$ and $x \in \underline{\mathsf{dom}}\, R'$ in T. Then $v \,!\, R' : G$. Second, $v = s$, $\underline{\mathsf{dom}}\, R'$ is of the form $\underline{\mathsf{dom}}\, S(h(\vec{r}))$, and $s \in \underline{\mathsf{dom}}\, S(\vec{r})$ in T_{rs}. Thus, $s \,!\, S(\vec{r}) : G_{rs}$ and, moreover, $v \,!\, R' = s \,!\, S(h(\vec{r})) = h'(s \,!\, S(\vec{r}))$. □

Lemma 9.4 *For any $Q(\vec{x},\vec{s})$ over T' there exists an $EP_Q(\vec{x})$ over T such that the following EGD is trivial:*

$$\vec{x} \in \vec{P}, \vec{s} \in \vec{S}(h(\vec{r})) \;\vdash\; C(\vec{x}) \;\underline{\mathsf{and}}\; B_2(h(\vec{r}),\vec{s}) \;\wedge\!\!=\; \underline{\mathsf{eq}}(Q(\vec{x},\vec{s}),\, EP_Q(\vec{x}))$$

Proof. This is an immediate application of the previous lemma and of Definition 9.1. If $Q \simeq' Q_0$, for some $Q_0 : G$ then take $EP_Q = Q_0$. If $Q \simeq' h'(Q_{rs})$ for some $Q_{rs} : G_{rs}$ then there must exist some extended path $EP_{Q_{rs}}(\vec{r})$ well-defined over T_r such that $EP_{Q_{rs}} \simeq_{rs} Q_{rs}$ (componentwise). It follows immediately (h' is a homomorphism) that $h'(EP_{Q_{rs}}) \simeq' h'(Q_{rs})$, and thus, $h'(EP_{Q_{rs}}) \simeq' Q$. Then take $EP_Q = h'(EP_{Q_{rs}})$[20]. □

The previous lemma generalizes to any sequence of chase steps, given that all the EPCDs involved are full. Let $T = \{\vec{x} \in \vec{P} \;;\; C(\vec{x})\}$ be a tableau and $T \xrightarrow{d_1} T_1 \xrightarrow{d_2} \ldots \xrightarrow{d_n} T_n$ be a chase sequence such that d_i is full, for any i. T_i must be of the form $\{\vec{x} \in \vec{P}, \vec{y_1} \in \vec{Q}_1(\vec{x}), \ldots, \vec{y_i} \in \vec{Q}_i(\vec{x}, \ldots, \vec{y}_{i-1}) \;;\; C(\vec{x}) \;\underline{\mathsf{and}}\; B_1(\vec{x},\vec{y_1}) \;\underline{\mathsf{and}}\; \ldots \;\underline{\mathsf{and}}\; B_i(\vec{x}, \ldots, \vec{y_i})\}$, for $i = 1, \ldots, n$.

Corollary 9.5 *For any path $Q(\vec{x},\vec{y_1},\ldots,\vec{y_n})$ over T_n there exists an extended path $EP_Q(\vec{x})$ over T such that the following EGD is trivial:*

$$\begin{aligned}\vec{x} \in \vec{P}, \ldots, \vec{y_n} \in \vec{Q}_n(\vec{x}, \ldots, \vec{y}_{n-1}) \;\vdash\;& B_1(\vec{x},\vec{y_1}) \;\underline{\mathsf{and}}\; \ldots \;\underline{\mathsf{and}}\; B_n(\vec{x}, \ldots, \vec{y_n}) \\ \wedge\!\!=\;& \underline{\mathsf{eq}}(Q(\vec{x},\vec{y_1},\ldots,\vec{y_n}),\, EP_Q(\vec{x}))\end{aligned}$$

9.1 The chase with full EPCDs is terminating

We prove first our termination result for full dependencies and then we give a complexity analysis. The main idea in the relational case is that there are only finitely many records built out of a finite number of possible values for their attributes, thus chasing with full TGDs is terminating ([BV84b]). We will also make use of this idea, although the proof is significantly more complicated in our nested/OO framework.

We will look at a single chase step $T_i \xrightarrow{d} T_{i+1}$ within a chase sequence with full EPCDs starting with a tableau $T = \{\vec{x} \in \vec{P} \;;\; C(\vec{x})\}$ (using the notations introduced in the previous subsection). By Corollary 9.5 paths over T_i (and T_{i+1}) are determined by extended paths EP over T. The first remark is that, in the case of paths of set type, EP must be paths as well. Second, we observe that the set of all possible extended paths $EP(\vec{x})$ over T that can determine paths over any T_i, for a given schema, is *finite*. This is true since, according to our earlier observation, any path Q has a type smaller than the type of some root name, thus any EP equated to Q must

[20] Here we suitably extend h' to extended paths.

satisfy the same property. But there are only finitely many EPs with variables in $\vec{x}$ that satisfy this upper bound on the size of the type (we will call them *bounded extended paths* over T). Let us denote by $\mathcal{E}$ the set of all bounded EPs over T and by $\mathcal{S}$ the set of all paths P over T of set type (note that $\mathcal{S}$ is a subset of $\mathcal{E}$).

We introduce a *measure* on a tableau T_i of the chase sequence, $m(T_i) = < p(T_i), s(T_i) >$, consisting of two components[21]:

$$\begin{array}{rcl} p(T_i) & = & \text{number of pairs } (EP, P) \text{ in } \mathcal{E} \times \mathcal{S} \text{ such that } EP : \tau,\ P : \{\tau\}, \text{ and either } \underline{\mathsf{cval}}_C EP \text{ is not in} \\ & & \mathit{CInst}(T_i) \text{ or, if it is, then there is no } \in\text{-edge from } \underline{\mathsf{cval}}_C P \text{ to } \underline{\mathsf{cval}}_C EP \text{ in } \mathit{CInst}(T_i) \\ s(T_i) & = & \text{number of nodes in } \mathit{CInst}(T_i) \end{array}$$

As we discussed above, $p(T_i)$ is finite. The ordering that we consider on pairs $m(T_i)$ is the lexicographical one. We show next that, under the simple type restriction, if $T_i \xrightarrow{d} T_{i+1}$ then $m(T_{i+1}) < m(T_i)$. This reflects our intuition that each chase step must either assert some membership condition between some EP in $\mathcal{E}$ and some P in $\mathcal{S}$ (a typical EPCD case) or, if not, must collapse some $\simeq$-equivalence classes in $\mathit{CInst}(T_i)$ (EGD case).

Lemma 9.6 *If $T_i \xrightarrow{d} T_{i+1}$, and T_i, T_{i+1}, and d are simple type restricted, then $m(T_{i+1}) < m(T_i)$.*

Proof. It is quite obvious that $p(T_{i+1}) \leq p(T_i)$. This is because if an EP is in $\mathit{CInst}(T_i)$ and participates in an $\in$-edge then it will be also in $\mathit{CInst}(T_{i+1})$ and it will participate in the same $\in$-edge (we use Lemma 8.3 here). If $p(T_{i+1}) < p(T_i)$ we are done. Now suppose that $p(T_{i+1}) = p(T_i)$. In other words, applying d doesn't produce any new $\in$-edges. We'll show that either some $\simeq_i$-equivalence classes are collapsed in $\mathit{CInst}(T_{i+1})$ and in this case $s(T_{i+1}) < s(T_i)$ and we are done, or $\mathit{CInst}(T_i)$ and $\mathit{CInst}(T_{i+1})$ are isomorphic. In the latter case, we can immediately infer the existence of a homomorphism from T_{i+1} into T_i, thus contradicting the definition of the chase step. The homomorphism comes, by Lemma 7.6, from the valuation obtained by composing $\underline{\mathsf{cval}}_C^{i+1}$ with the isomorphism.

Sublemma. If $p(T_{i+1}) = p(T_i)$ then for any path $Q : G_{i+1}$ there exists some path $Q_0 : G_i$ such that $Q \simeq_{i+1} Q_0$.

Proof of sublemma. Induction on the derivation of $Q : G_{i+1}$. The base cases are trivial. Now suppose $Q = y_{i+1}$ where $y_{i+1} \in Q_{i+1}$ is a binding in T_{i+1}. We know that $y_{i+1} \simeq_{i+1} EP(\vec{x})$ and $Q_{i+1} \simeq_{i+1} P(\vec{x})$ for some EP and P over T. Thus we have an $\in$-edge from P into EP in $\mathit{CInst}(T_{i+1})$. Since $p(T_{i+1}) = p(T_i)$ it must be the case that EP is also in $\mathit{CInst}(T_i)$ (with the corresponding $\in$-edge), i.e. there exists $Q_0 : G_i$ such that $Q_0 \simeq_i EP$. Therefore $Q_0 \simeq_{i+1} EP$ and, by transitivity, $y_{i+1} \simeq_{i+1} Q_0$. The case when $Q = u$ for some variable occurring in T_i is trivial.

(prj-add): $Q.\mathtt{A} : G_{i+1}$ for some $Q : G_{i+1}$. Applying the inductive hypothesis, there exists $Q_0 : G_i$ such that $Q \simeq_{i+1} Q_0$. Then, $Q_0.\mathtt{A} : G_i$ by (proj-add) and $Q.\mathtt{A} \simeq_{i+1} Q_0.\mathtt{A}$ by (prj-cong). The case (dom-add) is similar. The (!-add) case (we apply Lemma 9.2 first):

$$\frac{u \simeq_{i+1} v, \quad v \in \underline{\mathsf{dom}}\, R \text{ in } T_{i+1}, \quad R \simeq_{i+1} Q}{u\,!\,Q : G_{i+1}}$$

Case 1: v occurs in T_i. In this case, $v\,!\,R : G_i$ and by (!-cong) $u\,!\,Q \simeq_{i+1} v\,!\,R$. Case 2: v is bound only in T_{i+1}. We know that $v \simeq_{i+1} EP$ and $R \simeq_{i+1} P$ for some EP and P over T. Thus we have an $\in$-edge from $\underline{\mathsf{dom}}\, P$ into EP in $\mathit{CInst}(T_{i+1})$. We apply the hypothesis $p(T_{i+1}) = p(T_i)$ to conclude that EP must be in $\mathit{CInst}(T_i)$ and an $\in$-edge exists between $\underline{\mathsf{cval}}_C^i(EP)$ and $\underline{\mathsf{cval}}_C(\underline{\mathsf{dom}}\, P)$. Since $\in$-edges come from variables, it must be the case that there exists some $z \in \underline{\mathsf{dom}}\, P'$ in T_i such that $z \simeq_i EP$ and $P' \simeq_i P$ (here we made again use of Lemma 9.2). We obtain $z\,!\,P' : G_i$. Also, by transitivity and Lemma 8.3, $z \simeq_{i+1} u$ and $P' \simeq Q$. Hence, by (!-cong), $u\,!\,Q \simeq_{i+1} z\,!\,P'$. And this concludes the proof of the sublemma.

Under the assumption that $p(T_{i+1}) = p(T_i)$, a direct consequence of the sublemma is that $s(T_{i+1}) \leq s(T_i)$. This is because the mapping $f : \mathit{CInst}(T_i) \to \mathit{CInst}(T_{i+1})$, $f(\underline{\mathsf{cval}}_C^i(Q)) = \underline{\mathsf{cval}}_C^{i+1}(Q)$ is surjective, as it can be easily

[21] $\underline{\mathsf{cval}}_C$ is defined componentwise on a bounded extended path EP.

verified. If $s(T_{i+1}) < s(T_i)$ we are done. If $s(T_{i+1}) = s(T_i)$ then f is a bijection. Moreover, f preserves the structure (since cval^i_C and cval^{i+1}_C are algebraic homomorphisms, see Lemma 6.7), and the $\in$-edges. Finally, since $p(T_{i+1}) = p(T_i)$, $CInst(T_{i+1})$ doesn't have more $\in$-edges than $CInst(T_i)$. Thus, f is an isomorphism. As discussed above, this yields a contradiction with Definition 9.1. □

Since there cannot exist an infinitely decreasing sequence of $m(T_i)$, we proved the following:

Theorem 9.7 *If D is a set of full EPCDs and T is a tableau, both simple-type restricted, then any chase of T by D terminates.*

Corollary 9.8 *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.*

Complexity analysis. Relational full/total tgds are full EPCDs. Note that EGDs are always full, provided that they don't involve set/dictionary equality on the right-hand side. We show here that the complexity of the PC problem is the same as in the relational case ([BV84b, CLM81]), i.e. exponential. As we already said in section 6 we assume that the schema size is constant. However, some parameters of the schema will appear in our formulas.

Suppose $d = dep(T, T')$ is a full EPCD, D is a set of full EPCDs, both such that the restrictions of Theorem 9.7 and of Theorem 8.6 are satisfied, and our problem is to decide whether $D \models d$. The following parameters will be used in the analysis:

n	$=$	number of variables in the tableau T
m	$=$	number of variables in the tableau T' that are not in T
S	$=$	number of root names in the schema (including constants true and false)
h	$=$	maximum height of a record type or dictionary type in the schema
w	$=$	maximum width of a record type (number of attributes) in the schema
s	$=$	maximum number of variables in any EPCD in D
d	$=$	number of EPCDs in D

Lemma 9.9 (1) *The number of paths $P(\vec{x})$ over T is $N = O((nwS)^{c_0 h})$,*

(2) *The number of bounded extended paths $EP(\vec{x})$ over T is $|\ \mathcal{E}\ | \leq N + c_1 N^{w^h} = O((nwS)^{c_1' h w^h})$, where c_0, c_1 and c_1' are constants depending on the schema.*

Proof. For the first part, we observe that all paths can be organized in a stratified way: a) on the first level are variables and root names, b) paths on level i are obtained from paths from level $i-1$ by record projection, applying dom or dereferencing a dictionary P (with any of the n variables). There is no circularity between levels, due to our restriction that only variables can dereference dictionaries. The number of paths on level i is the number of paths on level $i-1$ multiplied by $nw+1$ (each $P : \langle A_1 : \tau_1, \ldots, A_k : \tau_k \rangle$ on level $i-1$ generates $k \leq w$ paths on level i, and each dictionary M on level i may generate at most n paths of the form $x\,!\,M$ on level i. dom contributes with multiplicity 1). The number of levels is bounded by h. Thus, the total number of paths can be estimated by:

$$(n+S)(1 + (nw+1) + \ldots + (nw+1)^h) = O((n+S)(nw+1)^{h+1}) = O((nwS)^{h+2})$$

For the second part, we observe that there is a constant number of possible record types in the schema, and for each of them the number of bounded extended paths obtained by applying the record constructor is at most N^{w^h} (there are at most w^h subexpressions which are not record expressions). □

By Corollary 9.5, the total number of nodes that can appear in any $\mathit{CInst}(T_i)$ in the chase sequence is at most $\mid \mathcal{E} \mid$. Thus the sequence $m(T) > m(T_1) > \ldots$ has length at most $\mid \mathcal{E} \times \mathcal{S} \mid + \mid \mathcal{E} \mid$. This is an upper bound on the number of chase steps. Recall that $\mathcal{S}$ is the number of paths over T of set type and therefore its size is bounded by N from Lemma 9.9. Hence, the number of chase steps is bounded by

$$(N+1) \mid \mathcal{E} \mid \leq \; (N+1)(N + c_1 N^{w^h})$$

and replacing N with its upper bound from Lemma 9.9 we obtain

Lemma 9.10 *The number of chase steps is $O((nwS)^{c_2 h w^h})$.*

Notice that this is polynomial in n when schema is fixed. We focus next on the complexity of a single chase step. To check whether a dependency $\mathit{dep}(T_r, T_{rs})$ applies to a tableau T_i in the sequence, we need to check the existence of a homomorphism from T_r into T_i. Since the number of variables in T_r is at most s and the number of variables in T_i is bounded by $\mid \mathcal{E} \mid$, there can be at most $\mid \mathcal{E} \mid^s = O((nwS)^{c_3 h w^h s})$ mappings. Checking whether any such mapping is actually a homomorphism takes time polynomial in the sizes of $\mathit{CInst}(T_r)$ and $\mathit{CInst}(T_i)$, which we can estimate as follows:

$$\text{size of } \mathit{CInst}(T_r) = O((swS)^{c_0 h})$$

$$\text{size of } \mathit{CInst}(T_i) = O(\mid \mathcal{E} \mid)$$

Any polynomial in the first term is dominated asymptotically by the expression for the number of mappings (s is at the exponent in the latter expression). Similarly, any polynomial in $\mid \mathcal{E} \mid$ is smaller than $\mid \mathcal{E} \mid^s$. We also observe that the time to check whether there is no homomorphism from T_{i+1} into T_i can be taken into account in the same manner (it is dominated by $\mid \mathcal{E} \mid^s$ as well). Finally, we have to multiply everything by d. Thus:

Lemma 9.11 *The complexity of a single chase step is $O(d(nwS)^{c_4 h w^h s})$.*

Notice that this is exponential in s. Hence, the most expensive part of the chase procedure is actually the chase step itself and not the total number of steps. To decide whether $D \models \mathit{dep}(T, T')$, at the end of the chase we still have to check whether $\mathit{chase}_D(d)$ is trivial. Here we need to consider $O(\mid \mathcal{E} \mid^m)$ mappings and we obtain, in a similar way with the chase step analysis, the upper bound $O((nwS)^{c_5 h w^h m})$. We conclude as follows:

Proposition 9.12 *Let D be a set of full EPCDs and $d = \mathit{dep}(T, T')$ an EPCD. Then:*

1. *The total number of chase steps and the size of $\mathit{chase}_D(T)$ (i.e. the number of variables in it) are both polynomial in n : $O(n^{k_1})$, respectively $O(n^{k_2})$,*

2. *The chase finishes in time exponential in s : $O(dn^{k_3 s})$,*

3. *Deciding whether $D \models d$[22] can be done in time exponential in s and m: $O(dn^{k_3 s}) + O(n^{k_4 m})$,*

where k_1, k_2, k_3, and k_4 are constants depending on the schema.

9.2 The chase with full EPCDs is Church-Rosser

Definition 9.13 *Given two instances I_1 and I_2, an* instance homomorphism *$ih : I_1 \to I_2$ is a type-preserving mapping from values of I_1 into values of I_2 such that:*

(1) ih is an algebraic homomorphism: $ih(R^{I_1}) = R^{I_2}$, for any root name, $ih(\underline{\mathsf{dom}}\, P) = \underline{\mathsf{dom}}\,(ih\, P)$, for any dictionary P in I_1, $ih(P.A) = ih(P).A$, for any record P in I_1 and any attribute A, and $ih(P_1 \,!\, P_2) = ih(P_1) \,!\, ih(P_2)$ for any dictionary P_2 in I_1 and any key $P_1 \in \underline{\mathsf{dom}}\, P_2$.

[22] The complexity formula given in [PT99] was erroneous because it didn't include the second term, exponential in m. The formula given there included only the first term, the time that the chase needs to finish.

(2) *ih is context preserving: for any* $P_1 \in P_2$ *in* I_1, $ih(P_1) \in ih(P_2)$ *in* I_2.

In the following, just for technical purposes, we will consider a slightly larger class of instances for which we don't require to be extensional. We call these instances *weak instances*. This class includes the normal instances considered until now. However, structures like $CInst(T)$ are in this class as well. The notions of valuation, dependency satisfaction and instance homomorphism are defined in the same way and we will call them, respectively, *weak valuation*, *weak satisfaction* (we use the same symbol, $\models$) and *weak instance homomorphism*. $\underline{\mathsf{cval}}_C$ is an example of a weak valuation. Also remark that any valuation is a weak valuation and any instance homomorphism is a weak instance homomorphism.

The first important observation is that Lemma 8.5, part (2), still holds when we replace $Inst(T)$ with $CInst(T)$. We also note that it would hold even in the presence of set/dictionary equality in d. We redo here the proof in the new context.

Lemma 9.14 *If* d *is not applicable to* T *then* $CInst(T) \models d$.

Proof. Assume $d = dep(T_r, T_{rs})$, and let $v_c : T_r \to CInst(T)$ be a weak valuation. By Lemma 7.6 there exists a homomorphism $h : T_r \to T$ such that $\underline{\mathsf{cval}}_C \circ h = v_c$. Since d is not applicable to T, there exists a homomorphism $h' : T' \to T$ such that $\vec{x} \in \vec{P} \vdash C(\vec{x}) \wedge\!= \underline{\mathsf{eq}}(\vec{x},\, h'(\vec{x}))$. As in the first version of the lemma, $h'' \stackrel{\text{def}}{=} h \cup \underline{\mathsf{id}}_{\vec{s}}$ is a homomorphism from T_{rs} into T'. Take $v'_c \stackrel{\text{def}}{=} \underline{\mathsf{cval}}_C \circ h' \circ h''$.

v'_c is obviously context preserving. Since $\vec{x} \in \vec{P} \vdash C(\vec{x}) \wedge\!= \underline{\mathsf{eq}}(\vec{x},\, h'(\vec{x}))$ is trivial, it follows by Lemma 7.1, part (2), that $\underline{\mathsf{cval}}_C(\vec{x}) = \underline{\mathsf{cval}}_C(h'(\vec{x}))$ and thus $\underline{\mathsf{cval}}_C(h(\vec{r})) = \underline{\mathsf{cval}}_C(h'(h(\vec{r})))$. Therefore, $v'_c(\vec{r}) = \underline{\mathsf{cval}}_C(h'(h(\vec{r}))) = \underline{\mathsf{cval}}_C(h(\vec{r}))$, and hence $v'_c(\vec{r}) = v_c(\vec{r})$. Finally, $v'_c(B_2(\vec{r},\vec{s})) = B_2(\underline{\mathsf{cval}}_C(h'(h(\vec{r}))), \underline{\mathsf{cval}}_C(h'(\vec{s})))$. Now, since $\vec{x} \in \vec{P} \vdash C(\vec{x}) \wedge\!= B_2(h'(h(\vec{r})), h'(\vec{s}))$ is trivial (h' is a homomorphism), the last term equals $\underline{\mathsf{true}}$, again by Lemma 7.1, part (2). We conclude that $CInst(T) \models d$. □

The next two lemmas show, the first one, that any weak valuation from a tableau T into a weak instance instance I is factored through a weak instance homomorphism from $CInst(T)$ into I, and, the second one, that instances and weak instances are indistinguishable with respect to trivial EGDs. We also remark that the first lemma fails if we try to strengthen it by replacing $CInst(T)$ with $Inst(T)$ and weak instance and weak valuation with instance and, respectively, valuation. This is one of the reasons why we need to develop several technical results with respect to weak instances and we cannot simply use what we have proved so far with respect to instances.

Lemma 9.15 *Let* $T = \{\vec{x} \in \vec{P} \;;\; C(\vec{x})\}$ *be a tableau and* I *a weak instance. Then, for any weak valuation* $v : T \to I$ *there exists a weak instance homomorphism* $ih_c : CInst(T) \to I$ *such that* $v = ih_c \circ \underline{\mathsf{cval}}_C$.

Proof. For any node Q in $CInst(T)$, pick any path expression P over T such that $\underline{\mathsf{cval}}_C P = Q$, and define $ih_c(Q) \stackrel{\text{def}}{=} v(P)$. This is well-defined because for any P_1 and P_2 over T, $\underline{\mathsf{cval}}_C P_1 = \underline{\mathsf{cval}}_C P_2$ implies $v(P_1) = v(P_2)$ (we already proved this in Proposition 6.3). Also, we have that $v = ih_c \circ \underline{\mathsf{cval}}_C$. It is easy to see that ih_c is context preserving and algebraic homomorphism. □

Lemma 9.16 *An EGD (with set/dictionary equality) is trivial if and only if it is trivial under all weak instances.*

Proof. We already observed in the previous lemma that if $\underline{\mathsf{cval}}_C P_1 = \underline{\mathsf{cval}}_C P_2$ then $v(P_1) = v(P_2)$ for any weak valuation $v : T \to I$. Thus, d trivial implies d weak trivial. The converse is obvious. □

Lemma 9.17 (Technical Lemma) *Let* $d = \vec{r} \in \vec{R} \vdash B_1(\vec{r}) \wedge\!= \underline{\mathsf{Some}}\,(\vec{s} \in \vec{S}(\vec{r}))\, B_2(\vec{r},\vec{s})$ *be an EPCD, and let* $T = \{\vec{x} \in \vec{P} \;;\; C(\vec{x})\}$ *be a tableau, where* C, B_1 *and* B_2 *are simple-type restricted. Assume that* $T \xrightarrow{d} T'$

where $T' = \{\vec{x} \in \vec{P}, \vec{s} \in \vec{S}(h(\vec{r}))\ ;\ C(\vec{x})\ \underline{\mathsf{and}}\ B_2(h(\vec{r}), \vec{s})\}$ *for some homomorphism* $T \xleftarrow{h} \{\vec{r} \in \vec{R}\ ;\ B_1(\vec{r})\}$. *Let* I *be a weak instance such that* $I \models d$ *and let* $v : T \to I$ *be a weak valuation. Then:*

(1) *there exists* $v' : T' \longrightarrow I$ *a weak valuation such that* $v'(\vec{x}) = v(\vec{x})$.

(2) *if, in addition,* d *is full, then for any* $Q(\vec{x}, \vec{s})$ *over* T'

$$v'(Q(\vec{x}, \vec{s})) = v(EP_Q(\vec{x})) \qquad (v' \textit{ is uniquely determined by } v)$$

where $EP_Q(\vec{x})$ *is from Lemma 9.4.*

Proof. Let T_r and T_{rs} such that $d = dep(T_r, T_{rs})$. Then $v \circ h$ is a weak valution from T_r into I. Since $I \models d$ there exists a weak valuation $v'' : T_{rs} \to I$ such that $v''(\vec{r}) = v \circ h(\vec{r})$. $h \cup \underline{\mathsf{id}}_{\vec{s}}$ is a homomorphism from T_{rs} into T'. We show that there exists $v' : T' \to I$ weak valuation such that $v'' = v' \circ (h \cup \underline{\mathsf{id}}_{\vec{s}})$. v' is defined as follows: $v'(\vec{x}) \stackrel{\text{def}}{=} v(\vec{x})$, $v'(\vec{s}) \stackrel{\text{def}}{=} v''(\vec{s})$, and then we extend it to paths over T' such that it is an algebraic homomorphism. Thus, $v''(\vec{r}) = v(h(\vec{r})) = v'(h(\vec{r})) = v' \circ (h \cup \underline{\mathsf{id}}_{\vec{s}})(\vec{r})$, and $v''(\vec{s}) = v' \circ (h \cup \underline{\mathsf{id}}_{\vec{s}})(\vec{s})$, and therefore, $v'' = v' \circ (h \cup \underline{\mathsf{id}}_{\vec{s}})$. It is easily verified that v' is context preserving.

Suppose now that d is full. By Lemma 9.4, for any $Q(\vec{x}, \vec{s})$ over T' there exists $EP_Q(\vec{x})$ over T such that

$$\vec{x} \in \vec{P}, \vec{s} \in \vec{S}(h(\vec{r}))\ \vdash\ C(\vec{x})\ \underline{\mathsf{and}}\ B_2(h(\vec{r}), \vec{s}) \wedge\!= \underline{\mathsf{eq}}(Q(\vec{x}, \vec{s}), EP_Q(\vec{x}))$$

is trivial, and therefore weak trivial, by Lemma 9.16. Hence, $v'(Q(\vec{x}, \vec{s})) = v'(EP_Q(\vec{x})) = v(EP_Q(\vec{x}))$. □

In the following we make use of Corollary 9.5, Lemma 9.14 and Lemma 9.17 to prove that chasing with full EPCDs satisfies the following confluence property:

Theorem 9.18 (Confluence) *For any two terminal chase sequences of* T *with a set* D *of full EPCDs,* $T \xrightarrow{d_1} T_1 \xrightarrow{d_2} \ldots \xrightarrow{d_n} T_n$ *and* $T \xrightarrow{d'_1} T'_1 \xrightarrow{d'_2} \ldots \xrightarrow{d'_m} T'_m$, *it must be the case that* $CInst(T_n)$ *and* $CInst(T'_m)$ *are isomorphic (and therefore* $Inst(T_n)$ *and* $Inst(T'_m)$ *are isomorphic as well).*

Proof. Assume $T = \{\vec{x} \in \vec{P}\ ;\ C(\vec{x})\}$, $T_n = \{\vec{x} \in \vec{P}, \vec{y} \in \vec{S}(\vec{x})\ ;\ C(\vec{x})\ \underline{\mathsf{and}}\ B_1(\vec{x}, \vec{y})\}$, and $T'_m = \{\vec{x} \in \vec{P}, \vec{y'} \in \vec{S'}(\vec{x})\ ;\ C(\vec{x})\ \underline{\mathsf{and}}\ B_2(\vec{x}, \vec{y'})\}$. Obviously, $\underline{\mathsf{id}}_{\vec{x}} : T \longrightarrow T'_m$ is a homomorphism. Thus, $\underline{\mathsf{cval}}'_C \circ \underline{\mathsf{id}}_{\vec{x}} : T \longrightarrow CInst(T'_m)$ is a weak valuation, where $\underline{\mathsf{cval}}'_C : T'_m \longrightarrow CInst(T'_m)$ is the canonical weak valuation on T'_m. By Lemma 9.14, $CInst(T'_m) \models D$. Then by repeatedly applying Lemma 9.17 for the chase sequence $T \xrightarrow{d_1} T_1 \xrightarrow{d_2} \ldots \xrightarrow{d_n} T_n$ we obtain a weak valuation $v_n : T_n \longrightarrow CInst(T'_m)$ such that $v_n(\vec{x}) = \underline{\mathsf{cval}}'_C \circ \underline{\mathsf{id}}(\vec{x})$. In addition, by Corollary 9.5, for any $Q(\vec{x}, \vec{y})$ over T_n there exists $EP_Q(\vec{x})$ over T such that

$$\vec{x} \in \vec{P}, \vec{y} \in \vec{S}(\vec{x})\ \ \vdash\ \ C(\vec{x})\ \underline{\mathsf{and}}\ B_1(\vec{x}, \vec{y})\ \wedge\!=\ \underline{\mathsf{eq}}(Q(\vec{x}, \vec{y}), EP_Q(\vec{x}))$$

is trivial, and therefore weak trivial. Thus $v_n(Q(\vec{x}, \vec{y})) = v_n(EP_Q(\vec{x})) = \underline{\mathsf{cval}}'_C(EP_Q(\vec{x}))$. Next, by Lemma 9.15, v_n induces a weak instance homomorphism $\theta : CInst(T_n) \longrightarrow CInst(T'_m)$ such that $\theta \circ \underline{\mathsf{cval}}_C = v_n$, where $\underline{\mathsf{cval}}_C : T_n \longrightarrow CInst(T_n)$ is the canonical weak valuation on T_n. Then, θ satisfies the following:

$$\theta(\underline{\mathsf{cval}}_C(Q(\vec{x}, \vec{y}))) = \underline{\mathsf{cval}}'_C(EP_Q(\vec{x})), \qquad \text{for any } Q(\vec{x}, \vec{y}) \text{ over } T_n$$

$$\theta(\underline{\mathsf{cval}}_C\ \vec{x}) = \underline{\mathsf{cval}}'_C \vec{x}$$

Similarly, we can show the existence of a weak instance homomorphism $\psi : CInst(T'_m) \longrightarrow CInst(T_n)$ satisfying similar properties. Then, for any $Q(\vec{x}, \vec{y})$ over T_n, we have:

$$\psi \circ \theta(\underline{\mathsf{cval}}_C\ Q(\vec{x}, \vec{y})) = \psi(\underline{\mathsf{cval}}'_C EP_Q(\vec{x})) = \underline{\mathsf{cval}}_C\ EP_Q(\vec{x})$$

But $\underline{\mathsf{cval}}_C\ Q(\vec{x}, \vec{y}) = \underline{\mathsf{cval}}_C\ EP_Q(\vec{x})$ by the above mentioned triviality! Thus, $\psi \circ \theta$ is the identity weak instance homomorphism on $CInst(T_n)$. Similarly, $\theta \circ \psi$ is the identity weak instance homomorphism on $CInst(T'_m)$. We conclude that we have an isomorphism between $CInst(T_n)$ and $CInst(T'_m)$. □

We will prove next that the final result of chase with full dependencies doesn't depend on the syntax of dependencies but rather on their semantics. Before doing that we need to go further into analyzing the behavior of dependencies with respect to weak instances. The first result generalizes Lemma 8.5, part (1), by stating the soundness of the chase step not only with respect to instances, but with respect to the larger class of weak instances. The proof could be done by observing that the part of the equational theory of CoDi that is used to simulate the chase step is sound with respect to weak instances. However, we choose to give a more direct proof.

Lemma 9.19 *If $Q \xrightarrow{d} Q'$ then $Q = Q'$ under any weak instance I such that $I \models d$.*

Proof. Let $d = dep(T_r, T_{rs})$ and suppose $T = \{\vec{x} \in \vec{P}\ ;\ C(\vec{x})\}$ and $T' = \{\vec{x} \in \vec{P}, \vec{s} \in \vec{S}(h(\vec{r}))\ ;\ C(\vec{x})\ \underline{\mathsf{and}}\ B_2(h(\vec{r}), \vec{s})\}$ for some homomorphism $h : T_r \to T$. We consider here only the case of $\underline{\mathsf{All}}$ queries (i.e. dependencies). The cases for $\underline{\mathsf{BigU}}$ and $\underline{\mathsf{Some}}$ are similar.

Suppose $Q = \underline{\mathsf{All}}\,(\vec{x} \in \vec{P})\,B(\vec{x})$ and $Q' = \underline{\mathsf{All}}\,(\vec{x} \in \vec{P})\,\underline{\mathsf{All}}\,(\vec{s} \in \vec{S}(h(\vec{r})))\,B(\vec{x})$ and let I be an arbitrary weak instance. The direction $I \models Q$ implies $I \models Q'$ is immediate. Indeed, assume $I \models Q$ and take $v' : T' \to I$ and arbitrary weak valuation. Then $v = v' \circ \underline{\mathsf{id}}_{\vec{x}}$ is a weak valuation from T into I. Since it must be the case that $B(v(\vec{x})) = \underline{\mathsf{true}}$, we obtain $B(v'(\vec{x})) = \underline{\mathsf{true}}$, and therefore, $I \models Q'$. Conversely, assume $I \models Q'$ and take $v : T \to I$ an arbitrary weak valuation. Then we can apply the Technical Lemma, part (1), to infer the existence of a weak valuation $v' : T' \to I$ such that $v'(\vec{x}) = v(\vec{x})$. Since $I \models Q'$ it follows that $B(v'(\vec{x})) = \underline{\mathsf{true}}$ and thus $B(v(\vec{x})) = \underline{\mathsf{true}}$. Hence, $I \models Q$. □

Proposition 9.20 (1) *An EGD (with set/dictionary equality) is trivial if and only if it is weak trivial.*

(2) *An EPCD (restricted as in Theorem 7.11) is trivial if and only if it is weak trivial.*

(3) *If D and d are restricted as in Theorem 8.6 and some chase of d by D terminates then:*

$$D \models d \ \text{ iff } \ D \models_{weak} d$$

Proof. (1) was already proved (Lemma 9.16). For (2), weak triviality implies obviously triviality. Now suppose $d = \vec{x} \in \vec{P_1} \ \vdash \ C_1(\vec{x}) \wedge\!= \underline{\mathsf{Some}}\,(\vec{y} \in \vec{P_2}(\vec{x}))\,C_2(\vec{x}, \vec{y})$ and T_1 and T_2 are such that $d = dep(T_1, T_2)$. If d is trivial then, by Theorem 7.11, there exists homomorphism $T_1 \xleftarrow{h} T_2$ such that the EGD $\vec{x} \in \vec{P_1} \ \vdash$ $C_1(\vec{x}) \wedge\!= \underline{\mathsf{eq}}(\vec{x}, h(\vec{x}))$ is trivial. Let I be an arbitrary instance and $v : T \to I$ an arbitrary weak valuation. Then $v \circ h : T' \to I$ is a weak valuation. Moreover, by (1), the above EGD is also weak trivial. Thus, $v(\vec{x}) = v(h(\vec{x}))$ and therefore $v(\vec{x}) = v'(\vec{x})$. We conclude that d is weak trivial.

For (3), again, one direction is immediate: $D \models_{\text{weak}} d$ implies $D \models d$. Now, suppose that $D \models d$. Then, by Theorem 8.6, $chase_D(d)$ is trivial, and therefore, by (2), weak trivial. Applying Lemma 9.19 for each chase step, we obtain that $D \models_{\text{weak}} d$. □

We can now observe that the confluence proof still works if one chase sequence uses dependencies from a set D and the other one uses dependencies from a different set D' provided that D and D' are weak equivalent, i.e. $D \models_{\text{weak}} D'$ and $D' \models_{\text{weak}} D$. But, by the previous proposition, weak implication and implication of full dependencies are the same. Thus, we obtain:

Theorem 9.21 (Semantic invariance) *Let D and D' be two equivalent sets of full EPCDs. Assume T_n and T_m are the resulting tableaux of two arbitrary terminal chase sequences of a tableau T with D and, respectively, D'. Then CInst(T_n) and CInst(T_m) are isomorphic.*

10 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 sequence 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 here the assumption that every EPCD that is applicable infinitely many times should be applied infinitely many times (non-starvation of dependencies).

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 a (countably) infinite tableau $T^\infty = \{\vec{x} \in \vec{P} \ ; \ C(\vec{x})\}$ that satisfies the following:

1. for any finite 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 two canonical instances of T^∞, $CInst(T^\infty)$ and $Inst(T^\infty)$ as follows. $CInst(T^\infty)$ is built as the limit of the sequence $(CInst(T_n))_{n\geq 0}$: the set of well-defined paths over T^∞ and the congruence closure of T^∞ are defined by:

$$G^\infty \stackrel{\text{def}}{=} \bigcup_{n\geq 0} G_n \qquad \simeq^\infty \stackrel{\text{def}}{=} \bigcup_{n\geq 0} \simeq_n$$

where G_n denotes the set of well-defined paths over T_n while $\simeq_n$ denotes the congruence closure of T_n. Recall (Lemma 8.3) that $G_n \subseteq G_{n+1}$ and $\simeq_n \subseteq \simeq_{n+1}$. It is easily verified that G^∞ and $\simeq^\infty$ are closed under the add and equality rules of the section 6. Similar with the construction of $CInst(T)$ in the case when T was finite, we define $CInst(T^\infty)$ by considering $\simeq^\infty$-equivalence classes and adding the appropriate edges. $CInst(T^\infty)$ is a weak *infinite* instance (however, countable). As in the finite case, we denote by $\underline{\mathsf{cval}}_C$ the canonical weak valuation.

$Inst(T^\infty)$ is defined out of $CInst(T^\infty)$ via a construction similar with the one given in section 6 for the finite case, using the same set of extensionality rules. However, because sets and dictionaries may be now infinite, a finite induction is not enough. In particular, the rules (set-ext) and (dict-ext) are the ones causing the difficulty. For example, in (set-ext), each pair $e \sim e'$ may have a finite length derivation but still the derivation $S_1 \sim S_2$ may be infinite. This is because S_1 and S_2 themselves may be infinite sets and the sup of an infinite set of finite numbers is not necessarily finite. Thus, $\sim$ is built by transfinite induction:

$$\begin{aligned}
\sim_0 \quad &\stackrel{\text{def}}{=} \quad \{(Q,Q) \mid Q \text{ node in } CInst(T^\infty)\} \\
\sim_{\lambda+1} \quad &\stackrel{\text{def}}{=} \quad \sim_\lambda \ \cup \\
&\qquad \{(Q,Q') \mid Q \sim Q' \text{ follows via some rule from (one or more) } P \sim P' \text{ with } (P,P') \in \sim_\lambda\} \\
\sim_\lambda \quad &\stackrel{\text{def}}{=} \quad \textstyle\bigcup_{\alpha<\lambda} \sim_\alpha \ , \qquad \text{if } \lambda \text{ is a limit ordinal}
\end{aligned}$$

Then we define $\sim \stackrel{\text{def}}{=} \bigcup_\alpha \sim_\alpha$ where the union is over all ordinals. Of course, since $CInst(T^\infty)$ is countable, $\sim$ must be countable as well, therefore the induction must terminate at some countable ordinal. In other words, there exists a countable β such that $\sim_{\beta+1} = \sim_\beta$. If β_0 is the least such β then $\sim_{\beta_0} = \bigcup_\alpha \sim_\alpha$ and hence $\sim \, = \, \sim_{\beta_0}$.

One can easily verify that $\sim$ is closed under the extensionality rules of section 6. By straightforward transfinite induction, one can show that Lemmas 6.4, 6.5 and 6.6 still hold. To exemplify, we prove here (set-inv-ext).

First case: $S_1 \sim_{\lambda+1} S_2$. Then either $S_1 \sim_\lambda S_2$, in which case we apply directly the inductive hypothesis, or $S_1 \sim_{\lambda+1} S_2$ follows by some rule application. For (sym) and (trans) we apply the inductive hypothesis. The other case possible is (set-ext), but this follows immediately since it must be the case that $\forall e \in S_1.\ \exists e' \in S_2.\ e \sim_\lambda e'$ and the symmetric condition are true. Second case: $S_1 \sim_\lambda S_2$ where λ is a limit ordinal. Then there must exist an $\alpha < \lambda$ such that $S_1 \sim_\alpha S_2$, in which case we apply the inductive hypothesis.

Thus, we can define $\mathit{Inst}(T^\infty)$ as in the finite case by taking $\sim$-equivalence classes and adding the necessary edges. We denote by $\underline{\mathsf{collapse}}$ the canonical mapping from nodes of $\mathit{CInst}(T^\infty)$ to $\mathit{Inst}(T^\infty)$. The first important result is that Lemma 7.5 in which we replace T_1 with T^∞ still holds. The proof is essentially the same (making use of Lemma 6.4).

Lemma 10.1 *Let $T_2 = \{\vec{y} \in \vec{P_2}\ ;\ C_2(\vec{y})\}$ in which C_2 is simple type restricted. Then, for any valuation $v : T_2 \to \mathit{Inst}(T^\infty)$ there exists a weak valuation $v_c : T_2 \to \mathit{CInst}(T^\infty)$ such that $v = \underline{\mathsf{collapse}} \circ v_c$. Moreover, v and v_c satisfy the same set of formulas $\underline{\mathsf{eq}}(Q, Q')$ over T_2 with Q and Q' of simple type.*

We remark that for any tableaux T_i, T_j in the chase sequence with $i < j$ the identity mapping on T_i is a homomorphism from T_i into T_j. We will denote it by $\underline{\mathsf{id}}_{ij}$. By Lemma 9.15 there exists an instance homomorphism $ih_{ij} : \mathit{CInst}(T_i) \to \mathit{CInst}(T_j)$ such that $ih_{ij} \circ \underline{\mathsf{cval}}^i_C = \underline{\mathsf{cval}}^j_C \circ \underline{\mathsf{id}}_{ij}$. These functions also exist when we replace T_j with T^∞. We will have in that case the identity mapping $\underline{\mathsf{id}}_i : T_i \to T^\infty$ and the weak instance homomorphism $ih_i : \mathit{CInst}(T_i) \to \mathit{CInst}(T^\infty)$ satisfying $ih_i \circ \underline{\mathsf{cval}}^i_C = \underline{\mathsf{cval}}_C \circ \underline{\mathsf{id}}_i$. Then we can prove the following lemma.

Lemma 10.2 *Let $T_r = \{\vec{r} \in \vec{R}\ ;\ B_1(\vec{r})\}$. Then, for any weak valuation $v_c : T_r \to \mathit{CInst}(T^\infty)$ there exists a finite n and a homomorphism $h : T_r \to T_n$ such that $v_c = \underline{\mathsf{cval}}_C \circ \underline{\mathsf{id}}_n \circ h$.*

Proof. For any $m \geq 0$ we define the following two sets:

$$\begin{aligned} G_r^m &= \{Q \mid Q : G_r \text{ with derivation length } \leq m\} \\ \simeq_r^m &= \{(Q, Q') \mid Q \simeq_r Q' \text{ with derivation length } \leq m\} \end{aligned}$$

We use the notations $Q : G_r^m$ for $Q \in G_r^m$ and $Q \simeq_r^m Q'$ for $(Q, Q') \in \simeq_r^m$. Note that, for any $m \geq 0$, G_r^m and $\simeq_r^m$ are finite. Also, there exists m_0 finite such that $G_r = G_r^{m_0}$ and $\simeq_r = \simeq_r^{m_0}$ (in other words all derivations are finite. We already argued in section 6 that in fact all derivations are of polynomial length.). We set the following **inductive hypothesis:**

For any $m \geq 0$ there exist $i_m \geq 0$ finite and $h_m : G_r^m \to G_{i_m}$ such that the following conditions are satisfied:

(1) *h_m maps variables of T_r into variables of T_{i_m}*

(2) *for any $Q : G_r^m$, $h_m(Q) : G_{i_m}$*

(3) *h_m is algebraic homomorphism on G_r^m*

(4) *h_m is context-preserving:*

for any $r \in R$ in T_r, if $h_m(r) = x$ with $x \in P$ in T_{i_m} then $P \simeq_{i_m} h_m(R)$

(5) *for any $Q \simeq_r^m Q'$, $h_m(Q) \simeq_{i_m} h_m(Q')$*

(6) $v_c = \underline{\mathsf{cval}}_C \circ \underline{\mathsf{id}}_{i_m} \circ h$

Then by taking m to be m_0 and n to be i_{m_0} we obtain the lemma. To prove the above statement we proceed by induction on m. The inductive step will consider all possible cases for the last rule applied in the derivation of $Q : G_r^m$ or $Q \simeq_r^m Q'$ and will extend in each case h_{m-1} to a new h'_m that will satisfy conditions (1) - (6) for m. A new $i'_m \geq i_{m-1}$ is obtained in each case. Then h_m will be the union of all h'_m while i_m will be taken as the maximum of all i'_m. The union, respectively, maximum, are taken over all Q and (Q, Q') that are in G_r^m, respectively $\simeq_r^m$, but not in G_r^{m-1}, respectively $\simeq_r^{m-1}$ (i.e. they have derivation length m). Since there sets are finite, i_m will be also finite. (Of course we assume that for each Q or (Q, Q') only one derivation is considered, in case there is more than one. This ensures that the above union is disjoint.)

Base case: $m = 0$. The possible cases are: (root-add), ($\underline{\mathsf{true}}$-add) and ($\underline{\mathsf{false}}$-add). Define $h_0(\mathtt{R}) = \mathtt{R}$, $h_0(\underline{\mathsf{true}}) = \underline{\mathsf{true}}$) and $h_0(\underline{\mathsf{false}}) = \underline{\mathsf{false}}$. Conditions (1)-(6) are trivially satisfied with $i_0 = 0$.

Induction step: $m > 0$. $h_m(Q) = h_{m-1}(Q)$, for all $Q : G_r^{m-1}$. For Q and $Q \simeq_r Q'$ with derivation length equal to m, we analyze the possible cases. First, (var-add):

$$\frac{r \in R \text{ in } T_r \ , \quad R : G_r^{m-1}}{r : G_r^m}$$

Since v_c is a weak valuation, $v_c(r) \in v_c(R)$ in $CInst(T^\infty)$. From the way $\in$-edges are added to $CInst(T^\infty)$ we can infer that there must exist $x \in P$ in some T_k such that $\underline{\mathsf{cval}}_C x = v_c(r)$ and $\underline{\mathsf{cval}}_C(P) = v_c(R)$. By the inductive hypothesis, $h_{m-1}(R) : G_{i_{m-1}}$ and $v_c(R) = \underline{\mathsf{cval}}_C \circ \underline{\mathsf{id}}_{i_{m-1}} \circ h_{m-1}\ (R) = \underline{\mathsf{cval}}_C(h_{m-1}(R))$. We infer that $\underline{\mathsf{cval}}_C(P) = \underline{\mathsf{cval}}_C(h_{m-1}(R))$ or $P \simeq^\infty h_{m-1}(R)$. Then, using the definition of $\simeq^\infty$, there exists some $j \geq \max\{i_{m-1}, k\}$ such that $h_{m-1}(R) \simeq_j P$. Take $h'_m(r) \stackrel{\text{def}}{=} x$ and $i'_m = j$. Condition (1) is automatically verified. Condition (2) is true: $h_m(r) : G_{i'_m}$, while (3) and (5) are satisfied by the inductive hypothesis for h_{m-1} and hence for h'_m (which only extends h_{m-1} on r). Condition (4) is true as well: $P \simeq_{i'_m} h'_m(R)$. Finally, for condition (6): $v_c(r) = \underline{\mathsf{cval}}_C x = \underline{\mathsf{cval}}_C(h'_m(r)) = \underline{\mathsf{cval}}_C \circ \underline{\mathsf{id}}_{i'_m} \circ h'_m\ (r)$.

The cases (proj-add), ($\underline{\mathsf{dom}}$-add) and (!-add) are all similar. We show here (!-add):

$$\frac{r \simeq_r^{m-1} r', \quad r' \in R \text{ in } T_r, \quad R \simeq_r^{m-1} \underline{\mathsf{dom}}\, Q}{r\,!\,Q : G_r^m}$$

Then, by the inductive hypothesis, $h_{m-1}(r) \simeq_{i_{m-1}} h_{m-1}(r')$ and they are both variables. Also, $h_{m-1}(R) \simeq_{i_{m-1}} h_{m-1}(\underline{\mathsf{dom}}\, Q) = \underline{\mathsf{dom}}\,(h_{m-1}(Q))$ (the last equality follows from the fact that h_{m-1} is an algebraic homomorphism). Now, since h_{m-1} is context preserving, we have $h_{m-1}(r') = x$ for some $x \in P$ in $T_{i_{m-1}}$ such that $P \simeq_{i_{m-1}} h_{m-1}(R)$. By (trans), $P \simeq_{i_{m-1}} \underline{\mathsf{dom}}\,(h_{m-1}(Q))$. Thus, we can apply (!-add):

$$\frac{h_{m-1}(r) \simeq_{i_{m-1}} h_{m-1}(r'), \quad h_{m-1}(r') \in P \text{ in } T_{i_{m-1}}, \quad P \simeq_{i_{m-1}} \underline{\mathsf{dom}}\,(h_{m-1}(Q))}{h_{m-1}(r)\,!\,h_{m-1}(Q) : G_{i_{m-1}}}$$

and then take $h'_m(r\,!\,Q) \stackrel{\text{def}}{=} h_{m-1}(r)\,!\,h_{m-1}(Q)$. We also set $i'_m = i_{m-1}$. Conditions (1)-(6) are easily verified.

Among the equality rules, the (eq) rule is the more interesting one:

$$\frac{\underline{\mathsf{eq}}(Q_1, Q_2) \text{ in } B_1(\vec{r}), \quad Q_1 : G_r^{m-1}, \quad Q_2 : G_r^{m-1}}{Q_1 \simeq_r^m Q_2}$$

By the inductive hypothesis, $h_{m-1}(Q_1) : G_{i_{m-1}}$ and $h_{m-1}(Q_2) : G_{i_{m-1}}$. Moreover, by the condition (6), $v_c(Q_1) = \underline{\mathsf{cval}}_C(h_{m-1}(Q_1))$ and $v_c(Q_2) = \underline{\mathsf{cval}}_C(h_{m-1}(Q_2))$. But v_c is a weak valuation, therefore $v_c(Q_1) = v_c(Q_2)$. It follows that $h_{m-1}(Q_1) \simeq^\infty h_{m-1}(Q_2)$. Then there must exist $j \geq i_{m-1}$ such that $h_{m-1}(Q_1) \simeq_j h_{m-1}(Q_2)$. Then we take $i'_m = j$ and $h'_m = h_{m-1}$. Condition (5) for the new pair (Q_1, Q_2) is satisfied.

The last cases, (refl), (sym), (trans), (prj-cong), ($\underline{\mathsf{dom}}$-cong), (rcd-ext) and (!-cong) are simple. We show here (prj-cong):

$$\frac{Q \simeq_r^{m-1} Q'}{Q.\mathtt{A} \simeq_r^m Q'.\mathtt{A}}$$

Applying the inductive hypothesis, $h_{m-1}(Q) \simeq_{i_{m-1}} h_{m-1}(Q') : G_{i_{m-1}}$. Therefore $h_{m-1}(Q).\mathtt{A} \simeq_{i_{m-1}} h_{m-1}(Q').\mathtt{A}$

by (prj-cong). Since h_{m-1} is an algebraic homomorphism, it follows that $h_{m-1}(Q.\mathtt{A}) \simeq_{i_{m-1}} h_{m-1}(Q'.\mathtt{A})$. Thus, it suffices to take $h'_m = h_{m-1}$ and $i'_m = i_m$. Condition (5) for the new pair $(Q.\mathtt{A}, Q'.\mathtt{A})$ is satisfied. □

Lemma 10.3 *CInst*$(T^\infty) \models D$.

Proof. Let $d = \vec{r} \in \vec{R} \vdash B_1(\vec{r}) \wedge\!\!= \underline{\mathsf{Some}}\,(\vec{s} \in \vec{S}(\vec{r}))\, B_2(\vec{r}, \vec{s})$ be an EPCD in D and let T_r and T_{rs} be such that $d = dep(T_r, T_{rs})$. Suppose $v_c : T_r \to CInst(T^\infty)$ is a weak valuation. Then, by Lemma 10.2, there exists some finite n and homomorphism $h : T_r \to T_n$ such that $v_c = \underline{\mathsf{cval}}_C \circ \underline{\mathsf{id}}_n \circ h$. Then for any $m \geq n$, $h_m = \underline{\mathsf{id}}_{nm} \circ h : T_r \to T_m$ is a homomorphism. If $T_m = \{\vec{x}_m \in \vec{P}_m \; ; \; C_m(\vec{x}_m)\}$ then let $T'_m = \{\vec{x}_m \in \vec{P}_m, \vec{s} \in \vec{S}(h_m(\vec{r})) \; ; \; C_m(\vec{x}_m) \; \underline{\mathsf{and}} \; B_2(h_m(\vec{r}), \vec{s})\}$. We claim that there exists a finite $m \geq n$ such that either (1) there exists a homomorphism $h'_m : T'_m \to T_m$, which is the identity (modulo trivial equalities) on $\vec{x}_m$, or (2) $T_{m+1} = T'_m$ (in the latter case $T_m \xrightarrow{d} T_{m+1}$). Indeed, if this is not the case then d is applicable to any T_m with $m \geq n$ and it is not applied, contradicting thus our earlier assumption regarding non-starvation of dependencies.

Let m be as above. We know that $h_m \cup \underline{\mathsf{id}}_{\vec{s}} : T_{rs} \to T'_m$ is a homomorphism (Lemma 8.4). In case (1), $g = h'_m \circ (h_m \cup \underline{\mathsf{id}}_{\vec{s}}) : T_{rs} \to T_m$ is a homomorphism. In case (2), $g = h_m \cup \underline{\mathsf{id}}_{\vec{s}} : T_{rs} \to T_{m+1}$ is a homomorphism. Thus, in both cases, we obtain a homomorphism g from T_{rs} into some tableau at finite index $k \geq m$ in the chase sequence. Let $T_k = \{\vec{x}_k \in \vec{P}_k \; ; \; C_k(\vec{x}_k)\}$. We claim that the following EGD is trivial:

$$\vec{x}_k \in \vec{P}_k \vdash C_k(\vec{x}_k) \wedge\!\!= \underline{\mathsf{eq}}(g(\vec{r}),\, h(\vec{r}))$$

Indeed, suppose we are in case (1). Then $\underline{\mathsf{cval}}^m_C(h'_m(h_m(\vec{r}))) = \underline{\mathsf{cval}}^m_C(h_m(\vec{r}))$ (h'_m is the identity on $\vec{x}_m$ and we apply the EGD Lemma). Therefore, $\underline{\mathsf{cval}}^k_C(g(\vec{r})) = \underline{\mathsf{cval}}^k_C(h'_m(h_m(\vec{r}))) = \underline{\mathsf{cval}}^k_C(h_m(\vec{r})) = \underline{\mathsf{cval}}^k_C(h(\vec{r})$ (we applied here Lemma 8.3 and the fact that $h_m = \underline{\mathsf{id}}_{nm} \circ h$). In case (2), $\underline{\mathsf{cval}}^k_C(g(\vec{r})) = \underline{\mathsf{cval}}^k_C(h_m(\vec{r})) = \underline{\mathsf{cval}}^k_C(h(\vec{r}))$. Hence, in both cases, using the EGD lemma, we conclude that the above EGD is trivial.

Next, define $v'_c : T_{rs} \to CInst(T^\infty)$ as $v'_c \stackrel{\text{def}}{=} \underline{\mathsf{cval}}_C \circ \underline{\mathsf{id}}_k \circ g$. It is easily verified using the definition of $\simeq^\infty$ and Lemma 8.3 that v'_c is a weak valuation. Moreover, $\underline{\mathsf{cval}}^k_C(g(\vec{r})) = \underline{\mathsf{cval}}^k_C(h(\vec{r}))$ implies $\underline{\mathsf{cval}}_C(g(\vec{r})) = \underline{\mathsf{cval}}_C(h(\vec{r}))$. Thus, $v'_c(\vec{r}) = \underline{\mathsf{cval}}_C(g(\vec{r})) = \underline{\mathsf{cval}}_C(h(\vec{r})) = v_c(\vec{r})$. We conclude that $CInst(T^\infty) \models d$, and since d is arbitrary in D, $CInst(T^\infty) \models D$. □

Lemma 10.4 *Inst*$(T^\infty) \models D$.

Proof. Let $d = dep(T_r, T_{rs})$ be an EPCD in D and let $v : T_r \to Inst(T^\infty)$ be an arbitrary valuation. Then by Lemma 10.1 there exists a weak valuation $v_c : T_r \to CInst(T^\infty)$ such that $v = \underline{\mathsf{collapse}} \circ v_c$. Since $CInst(T^\infty) \models D$ it follows that $CInst(T^\infty) \models d$. Thus there exists a valuation $v'_c : T_{rs} \to CInst(T^\infty)$ such that $v'_c(\vec{r}) = v_c(\vec{r})$. Define $v' : T_{rs} \to Inst(T^\infty)$ as $v' = \underline{\mathsf{collapse}} \circ v'_c$. Then v' is a valuation and $v'(\vec{r}) = \underline{\mathsf{collapse}}(v'_c(\vec{r})) = \underline{\mathsf{collapse}}(v_c(\vec{r})) = v(\vec{r})$. □

Theorem 10.5 (Non-terminating chase) *Let D be a set of EPCDs. In the following, D, d, Q_1 and Q_2 are restricted as in Theorem 8.6.*

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:*

   $$\textit{(1) } chase^m_D(d) \textit{ is trivial (unr) and/or (2) } chase^m_D(front(d)) \subseteq^{unr} back(d)$$

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

   (d) *d is provable from D*

Proof. We prove part 2 first. If $chase^m_D(d)$ is trivial for some m then $D \models d$, by repeatedly applying Lemma 8.5. For the interesting direction assume $D \models d$. Suppose $d = \vec{x} \in \vec{P} \ \vdash \ C(\vec{x}) \wedge\!\!= \underline{\mathsf{Some}}\,(\vec{y} \in \vec{R}(\vec{x}))\, D(\vec{x},\vec{y})$, and let T_x and T_{xy} be such that $d = dep(T_x, T_{xy})$. Let $T_x = T_0 \longrightarrow T_1 \longrightarrow \ldots \longrightarrow T_n \longrightarrow \ldots$ be an infinite chase sequence of T_x by D. Then $v = \underline{\mathsf{cval}} \circ \underline{\mathsf{id}}_0 : T_x \to Inst(T^\infty)$ is a valuation. We know by Lemma 10.4 that $Inst(T^\infty) \models D$, therefore, since $D \models^{unr} d$, $Inst(T^\infty) \models d$. Hence there exists a valuation $v' : T_{xy} \to Inst(T^\infty)$ such that $v'(\vec{x}) = v(\vec{x})$, and therefore $v'(\vec{x}) = \underline{\mathsf{cval}}(\vec{x})$. Now, by an analogous of Lemma 7.10 for the infinite case (the reader can verify it), there exists a weak valuation $v'_c : T_{xy} \to CInst(T^\infty)$ such that $v' = \underline{\mathsf{collapse}} \circ v'_c$ and $v'_c(\vec{x}) = \underline{\mathsf{cval}}_C(\vec{x})$. Applying Lemma 10.2 we infer the existence of a tableau T_n in the chase sequence and of a homomorphism $h_n : T_{xy} \to T_n$ such that $\underline{\mathsf{cval}}_C \circ \underline{\mathsf{id}}_n \circ h_n = v'_c$. Thus we have $\underline{\mathsf{cval}}_C(h_n(\vec{x})) = \underline{\mathsf{cval}}_C(\vec{x})$, or $h_n(\vec{x}) \simeq^\infty \vec{x}$. By the definition of $\simeq^\infty$ there must exist $m \geq n$ such that $h_n(\vec{x}) \simeq_m \vec{x}$, or $\underline{\mathsf{cval}}^m_C(h_n(\vec{x})) = \underline{\mathsf{cval}}^m_C(\vec{x})$.

We use the following notations for T_n and T_m (T_n is a subtableau of T_m because $m \geq n$):

$$\begin{aligned} T_n &= \{\vec{x}_n \in \vec{P}_n \ ; \ C_n(\vec{x}_n)\} \\ T_m &= \{\vec{x}_n \in \vec{P}_n, \vec{z} \in \vec{Q} \ ; \ C_n(\vec{x}_n) \ \underline{\mathsf{and}} \ B(\vec{x}_n, \vec{z})\} \end{aligned}$$

Recall also that $T_x = T_0$ and thus $T_x = \{\vec{x} \in \vec{P} \ ; \ C(\vec{x})\}$ is a subtableau of both T_n and T_m. Consider now the tableau T':

$$T' = \{\vec{x}_n \in \vec{P}_n, \vec{z} \in \vec{Q}, \vec{y} \in \vec{R}(\vec{x}) \ ; \ C_n(\vec{x}_n) \ \underline{\mathsf{and}} \ B(\vec{x}_n, \vec{z}) \ \underline{\mathsf{and}} \ D(\vec{x}, \vec{y})\}$$

Let h_m be the mapping from T' into T_m that is the same as h_n on $\vec{x}$ and $\vec{y}$ and the identity on the rest. Since h_n is a homomorphism, h_m is also a homomorphism (we've already used this idea in the proof of Theorem 8.6). We claim that the following is a trivial EGD:

$$\vec{x}_n \in \vec{P}_n, \vec{z} \in \vec{Q} \vdash C_n(\vec{x}_n) \ \underline{\mathsf{and}} \ B(\vec{x}_n, \vec{z}) \wedge\!\!= \underline{\mathsf{eq}}(\vec{x}_n, \, h_m(\vec{x}_n)) \ \underline{\mathsf{and}} \ \underline{\mathsf{eq}}(\vec{z}, \, h_m(\vec{z}))$$

The only part that we have to prove is the one concerning variables $\vec{x}$ because for the rest h_m is defined to be the identity. But then we know that $\underline{\mathsf{cval}}^m_C(h_n(\vec{x})) = \underline{\mathsf{cval}}^m_C(\vec{x})$, therefore $\underline{\mathsf{cval}}^m_C(h_m(\vec{x})) = \underline{\mathsf{cval}}^m_C(\vec{x})$ and we can apply the EGD lemma. To conclude, h_m proves, by Theorem 7.11, that $chase^m_D(d)$ is trivial. Part 1 follows from Part 2 by using the two Reducibility Lemmas. □

Corollary 10.6 *Let D and d be EPCDs restricted as in Theorem 10.5. Then $D \models^{unr}_{weak} d$ if and only if $D \models^{unr} d$ and both problems are r.e.*

10.1 Disjunction aggregates

Theorem 7.9, and parts (1) of 8.6 and 10.5 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 $Q = \underline{\mathsf{Some}}\,(\vec{r} \in \vec{\mathsf{R}})\, B(\vec{r})$ has a corresponding set-valued PC query $Q' = \underline{\mathsf{BigU}}\,(\vec{r} \in \vec{\mathsf{R}})\ \underline{\mathsf{if}}\ B(\vec{r})\ \underline{\mathsf{then}}\ \underline{\mathsf{sng}}\langle\rangle$ 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 $\underline{\text{idemloop}}$).

11 Equational axiomatization

In addition to the laws in figure 1, the axiomatization of the CoDi equational theory consists of the following axioms and inference rules:

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

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

2. **Record axioms**:

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

3. **Conditional axioms**:

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

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

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

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

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

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

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

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

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

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

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

6. **Implication rules**:

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

7. **Congruence rules:**

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

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

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

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

Some obvious rules such as symmetry and transitivity of equality are missing because they are derivable largely due to (eqcond). We already in mentioned in section 2 that the rule

(subst) $\quad \Gamma, x \in S \ \vdash\ E_1(x) = E_2(x) \quad$ and $\quad \Gamma\ \vdash\ \underline{\mathsf{member}}(E, S) = \underline{\mathsf{true}} \quad$ implies $\quad \Gamma\ \vdash\ E_1(E) = E_2(E)$

does not have to be axiomatically stipulated (as in first-order algebraic theories) because it is already derivable. Moreover, (subst) and (dict-β) prove that $\underline{\mathsf{member}}(K, D) = \underline{\mathsf{true}}$ implies $K\,!\,(\underline{\mathsf{key}}\ k\ \underline{\mathsf{in}}\ D\ \Rightarrow\ E(k)) = E(K)$.

This is the analogous for our calculus of the typical substitution rule found in equational proof systems. However, there is a main difference between the two in the sense that, here, before substituting E for $x \in S$ in both sides of the equation $E_1(x) = E_2(x)$, we need to ensure that E is an element of S. In many cases this is nothing more but a RIC while the other precondition is a key constraint!

The proof follows by applying mainly the idemloop law for sets. For simplicity we drop here the right-hand side of the equations.

$$
\begin{array}{rcll}
E_1(E) & = & \underline{\mathsf{if}}\ \underline{\mathsf{true}}\ \underline{\mathsf{then}}\ E_1(E) & \text{(cond-true)} \\
& = & \underline{\mathsf{if}}\ \underline{\mathsf{Some}}\,(x \in S)\,\underline{\mathsf{eq}}(E,\, x)\ \underline{\mathsf{then}}\ E_1(E) & \text{(membership condition)} \\
& = & \underline{\mathsf{Loop}}\,(x \in S)\,\underline{\mathsf{if}}\ \underline{\mathsf{eq}}(E,\, x)\ \underline{\mathsf{then}}\ E_1(E) & \text{(idemloop)} \\
& = & \underline{\mathsf{Loop}}\,(x \in S)\,\underline{\mathsf{if}}\ \underline{\mathsf{eq}}(E,\, x)\ \underline{\mathsf{then}}\ E_1(x) & \text{(eqcond)}
\end{array}
$$

Symmetrically, we have:

$$E_2(E) = \underline{\mathsf{Loop}}\,(x \in S)\,\underline{\mathsf{if}}\ \underline{\mathsf{eq}}(E,\, x)\ \underline{\mathsf{then}}\ E_2(x)$$

Now we can use the first condition equating $E_1(x)$ and $E_2(x)$ and the (cond-cong) and ($\underline{\mathsf{Loop}}$-cong) to obtain that $E_1(E) = E_2(E)$. $\Box$

Two forms for dependencies. We have claimed in section 3 that (form1) $\underline{\mathsf{All}}\,(\vec{x} \in \vec{S})\,\underline{\mathsf{if}}\ B(\vec{x})\ \underline{\mathsf{then}}\ C(\vec{x}) = \underline{\mathsf{true}}$ and (form2) $\vec{x} \in \vec{S} \vdash\ B(\vec{x}) \wedge\!\!= C(\vec{x})$ are derivable from each other in CoDi's equational theory. Indeed, (form2) follows from (form1) using (all) and an implication rule. To derive (form1) from (form2) it suffices by ($\underline{\mathsf{Loop}}$-cong) to show that $\underline{\mathsf{All}}\,(\vec{x} \in \vec{S})\,\underline{\mathsf{true}}\ =\ \underline{\mathsf{true}}$. This follows by (idemloop) from $\underline{\mathsf{if}}\ D\ \underline{\mathsf{then}}\ \underline{\mathsf{true}}\ \underline{\mathsf{else}}\ \underline{\mathsf{true}}\ =\ \underline{\mathsf{true}}$ which is a consequence of the implication, conditional and $\underline{\mathsf{and}}$ rules.

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

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

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), $\vec{x} \in \vec{P} \vdash \underline{\mathsf{Some}}\,(y_i \in P_j)\,\underline{\mathsf{eq}}(y_i,\, h(y_i)) = \underline{\mathsf{true}}$, for any y_i. Then we use the premises and, mainly, congruences, ($\underline{\mathsf{and}}$ -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 7.9.

Provability of the chase step. Let $Q = \underline{\mathsf{Loop}}\,(\vec{x} \in \vec{P})\,\underline{\mathsf{if}}\ C(\vec{x})\ \underline{\mathsf{then}}\ E$ and d and h as in the definition of the chase step (see section 8). Observe that h extended to be the identity on $\vec{x}$ is a homomorphism from $\{\vec{x} \in \vec{P}, \vec{r'} \in \vec{R}; \underline{\mathsf{eq}}(\vec{r'},\, h(\vec{r}))\}$ into $\{\vec{x} \in \vec{P}; \underline{\mathsf{true}}\}$ such that it satisfies the condition of Theorem 7.11. Therefore $\vec{x} \in \vec{P} \vdash \underline{\mathsf{true}} = \underline{\mathsf{Some}}\,(\vec{r'} \in \vec{R})\,\underline{\mathsf{eq}}(\vec{r'},\, h(\vec{r}))$ 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 $\underline{\mathsf{Loop}}\,(\vec{x} \in \vec{P})\,\underline{\mathsf{if}}\ C(\vec{x})\ \underline{\mathsf{then}}\ E$ to

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

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

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

Applying (cond-loop2) we move $\underline{\mathsf{eq}}(\vec{r'},\,h(\vec{r}))$ outside of the loop over $\vec{s}$ and apply (eqcond) to replace occurrences of $\vec{r'}$ with $h(\vec{r})$. Then rewriting with the trivial EPCD $\vec{x} \in \vec{P} \vdash \underline{\mathsf{true}} = \underline{\mathsf{Some}}\,(\vec{r'} \in \vec{R})\,\underline{\mathsf{eq}}(\vec{r'},\,h(\vec{r}))$, followed by replacing $C(\vec{x})$ $\underline{\mathsf{and}}$ $B_1(h(\vec{r}))$ with $C(\vec{x})$ yields $\underline{\mathsf{Loop}}\,(\vec{x} \in \vec{P})\,\underline{\mathsf{Loop}}\,(\vec{s} \in \vec{S}(h(\vec{r})))\,\underline{\mathsf{if}}\ C(\vec{x})\ \underline{\mathsf{and}}\ B_2(h(\vec{r}),\vec{s})\ \underline{\mathsf{then}}\ E$, the query Q' that we wanted. This proves lemma 8.5 (1)

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.

[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.

[BK93] Catriel Beeri and Yoram Kornatzky. Algebraic optimisation of object oriented query languages. *Theoretical Computer Science*, 116(1):59–94, August 1993.

[BNTW95] Peter Buneman, Shamim Naqvi, Val Tannen, and Limsoon Wong. Principles of programming with collection types. *Theoretical Computer Science*, 149:3–48, 1995.

[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.

[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.

[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.

[Fag82] Ronald Fagin. Horn clauses and database dependencies. *Journal of the ACM*, 29(4):952–985, 1982.

[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.

[LT97] Kazem Lellahi and Val Tannen. A calculus for collections and aggregates. In E. Moggi and G. Rosolini, editors, *LNCS 1290: Category Theory and Computer Science Proceedings of the 7th Int'l Conference, CTCS'97*, pages 261–280, Santa Margherita Ligure, September 1997. Springer-Verlag.

[MMS79] D. Maier, A. O. Mendelzon, and Y. Sagiv. Testing implications of data dependencies. *ACM Transactions on Database Systems*, 4(4):455–469, 1979.

[PT99] Lucian Popa and Val Tannen. An equational chase for path-conjunctive queries, constraints, and views. In *LNCS ????: Database Theory - ICDT'99, Proceedings of the 7th Int'l Conference*, pages ??–??, Jerusalem, 1999. Springer-Verlag.

[SC90] Eugene J. Shekita and Michael J. Carey. A performance evaluation of pointer-based joins. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 300–311, Atlantic City, New Jersey, May 1990.

[YP82] Mihalis Yannakakis and Christos Papadimitriou. Algebraic dependencies. *Journal of Computer and System Sciences*, 25:2–41, 1982.