Equivalence of Nested Queries with Mixed Semantics
(extended version)

David DeHaan

UNIVERSITY OF
Waterloo

Technical Report CS-2009-12
March 13, 2009

Equivalence of Nested Queries with Mixed Semantics

David DeHaan
David R. Cheriton School of Computer Science
University of Waterloo
Waterloo, Ontario, Canada
dedehaan@uwaterloo.ca

ABSTRACT

We consider the problem of deciding query equivalence for a conjunctive language in which queries output complex objects composed from a mixture of nested, unordered collection types. Using an encoding of nested objects as flat relations, we translate the problem to deciding the equivalence between encodings output by relational conjunctive queries. This *encoding equivalence* cleanly unifies and generalizes previous results for deciding equivalence of conjunctive queries evaluated under various processing semantics. As part of our characterization of encoding equivalence, we define a normal form for encoding queries and contend that this normal form offers new insight into the fundamental principles governing the behaviour of nested aggregation.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems—*query processing, relational databases*; H.2.3 [**Database Management**]: Languages—*query languages*

General Terms

Algorithms, Languages, Theory

Keywords

conjunctive queries, query equivalence, bag-set semantics, set semantics, normalized bags, aggregation

1. INTRODUCTION

Deciding equivalence between queries has long been of interest because of its relevance to query optimization [5], rewriting over views [22], maintenance of materialized views or integrity constraints [37, 17], and access control [31]. While query equivalence is well understood for simple query languages such as conjunctive queries (CQs) under both set and bag-set semantics, modern database systems routinely face workloads of complex queries built from nested query blocks that apply various aggregation functions, introducing an interleaving of different semantics within a single query. One well-known source of complex queries arises from decision-support applications (e.g. TPC-H, TPC-DS). More recently, with the advent of object-relational mapping technologies, application programmers with little or no knowledge of SQL can write seemingly simple programs that translate into very complex queries due to the reliance on logical views to enact object-relational mappings [28]. In short, the need for optimization techniques that handle complex queries can only be expected to grow.

In this paper we consider the problem of deciding equivalence between conjunctive queries that return nested structures. We generalize previous work by allowing arbitrary nesting of three collection types: sets, bags, and normalized bags. We show the equivalence problem to be NP-complete by reducing it to a relationship we call *encoding equivalence* between relational CQs. As part of our characterization, we define a normal form for queries that captures interactions between collection types in terms of query-implied multivalued dependencies. Although equivalence of conjunctive queries returning complex objects with nested sets has been considered before [25], the intricacy of the previous characterization makes it difficult to extend to either varied collection types or—as shown in Section 1.2—arbitrary nesting depths. In contrast, the elegance of our normal form makes clear how query structure interacts with the semantics of nested collections of arbitrary types and nesting depths.

1.1 Related Research

Optimization of nested SQL queries has long been of interest. One line of research focuses on algebraic transformations that change the nesting structure of the query, including both merging or decorrelating nested query blocks [20, 10, 2] and commuting aggregation with join or with other aggregation [40, 16]. An orthogonal line of work generalizes predicate pushdown and moves join predicates between or introduces semijoins into existing query blocks [24, 29]. Many such transformations have been incorporated into algorithms that rewrite complex queries over materialized views [35, 41, 12]. Unfortunately, the query transformation literature fails to provide a systematic understanding of the principles governing the interaction between nested query blocks.

For non-aggregated relational queries, the containment and equivalence problems are mutually reducible. An extensive body of literature characterizes the containment problem for CQs [5], queries with disjunction or negation [33], inequalities [21, 39], and schema constraints [19]. Chaudhuri

University of Waterloo Technical Report CS-2009-12.
This is an extended version of work published in
PODS'09, June 29–July 2, 2009, Providence, Rhode Island, USA.

and Vardi [6] and Ioannidis and Ramakrishnan [18] independently propose bag/bag-set semantics as a way to model the input to cardinality-sensitive aggregation functions. Cohen later proposes "combined semantics" as a generalization of bag-set semantics in which cardinality depends only on a specified subset of the query variables [7], while Grumbach et al. [15] propose "isomorphism modulo a product" as a relaxation of bag-set equivalence to model the input to aggregation functions such as AVG. Our study of equivalence for nested objects of mixed collection types generalizes all of these equivalence relations for CQs, as they all reduce to special cases of encoding equivalence (see Section 4). Containment is not known to be decidable under bag-set semantics, and so we restrict our attention to equivalence.

Equivalence of aggregation queries has been investigated previously [8, 9, 15], primarily so as to understand the behaviour of specific aggregation functions within an unnested context. Our abstraction of aggregation functions as collection constructors is comparatively primitive, but our work is orthogonal in that we seek to understand the effect of query nesting. Other authors have shown that constraints induced by nested aggregation functions easily yield undecidability in the presence of domain-specific knowledge [23, 32].

Early work on complex objects assumes a model of nested relations [1], including the well-known nested relational algebra of Thomas and Fischer [36]. More powerful models and languages encompassing other collection types have also been proposed—in particular, variations of the Nested Relational Calculus, which typically allow for the creation of objects with empty subcollections [30, 26, 4]—but this research mostly focuses on power of expression, rather than the query equivalence problem. To place our work in context, the query language we consider can be described informally as a bag semantic conjunctive algebra extended with three variants of the *nest* operator (for constructing different collection types), but with no *unnest* operator (we briefly discuss such an extension in Section 5.3), and no power to create empty subcollections. Transformation rules for the nested relational algebra have been defined [34, 27], but these do not characterize equivalence of arbitrary expressions.

Containment and equivalence of queries returning complex objects (nested sets only) is studied by Levy and Suciu [25], who consider "conjunctive OQL" (COQL) queries. Whereas containment of flat relations indisputably corresponds to set inclusion, there is no single definition for containment of nested sets. Levy and Suciu use an inductive definition previously proposed for Verso relations [3], and they reduce containment (under this definition) of COQL queries constructing objects with nesting depth d to testing a relationship between CQs that they call "simulation to depth d," defined as follows. Let $Q(\overline{\mathcal{I}}_1; \ldots; \overline{\mathcal{I}}_d; \overline{\mathcal{V}})$ be a CQ whose head has been annotated to distinguish d sets of *index variables*, and define $\overline{\mathcal{I}} := (\overline{\mathcal{I}}_1; \ldots; \overline{\mathcal{I}}_d)$. Given two such queries, Q *simulates* Q' *to depth* d—denoted $Q \preceq_d Q'$—iff over every database instance the following equation holds:

$$\forall \overline{\mathcal{I}}_1.\exists \overline{\mathcal{I}}'_1 \ldots \forall \overline{\mathcal{I}}_d.\exists \overline{\mathcal{I}}'_d.\forall \overline{\mathcal{V}} \left[Q(\overline{\mathcal{I}}; \overline{\mathcal{V}}) \Rightarrow Q'(\overline{\mathcal{I}}'; \overline{\mathcal{V}}) \right] \quad (1)$$

a condition characterized by the existence of a *simulation mapping*, and hence NP-complete to decide [25]. For COQL queries that cannot construct empty sets, containment reduces to a single simulation test. Levy and Suciu claim that arbitrary COQL containment reduces to testing an exponential number of simulation conditions; however, Dong et al. [13] point out that this is insufficient for implying containment.[1]

The containment relationship used by Levy and Suciu is not antisymmetric (mutual containment does not imply equivalence) and so they define a separate "strong simulation" relationship between CQs for testing COQL equivalence. Query Q *strongly simulates* Q' *to depth* d—denoted $Q \preccurlyeq_d Q'$—iff:

$$\forall \overline{\mathcal{I}}_1.\exists \overline{\mathcal{I}}'_1 \ldots \forall \overline{\mathcal{I}}_d.\exists \overline{\mathcal{I}}'_d.\forall \overline{\mathcal{V}} \left[Q(\overline{\mathcal{I}}; \overline{\mathcal{V}}) \iff Q'(\overline{\mathcal{I}}'; \overline{\mathcal{V}}) \right] \quad (2)$$

a condition which they claim is characterized by the existence of a *strong simulation mapping* [25], and hence still NP-complete to decide (although they define this mapping only for $d \leq 1$). While equivalence of general COQL queries is left open, they claim that equivalence for COQL queries that cannot construct empty sets reduces to testing a single strong simulation condition in each direction (Proposition 6.3 [25]). We demonstrate in Example 2 that this reduction of nested query equivalence to strong simulation is incorrect.

Finally, Van den Bussche et al. prove that the query equivalence problem is undecidable for the Positive-Existential fragment of the Nested Relational Calculus [38]. Although PENRC lacks the ability to explicitly test set-emptiness, Van den Bussche et al.'s proof of undecidability relies on the ability to construct objects containing empty subsets. As such, their result does not necessarily transfer to positive fragments of the nested relational algebra that are incapable of creating empty subobjects.

1.2 Two Motivating Examples

Our first example illustrates the weakness of current query rewriting algorithms that depend on sets of algebraic transformations that are sound but incomplete.

Example 1 Consider the following database schema, storing information about customer orders solicited by a company's agents. Assume that the schema includes the obvious primary and foreign key constraints.

Customer(<u>cid</u>, cname, ctype)
Order(<u>oid</u>, cid, date)
LineItem(<u>oid</u>, <u>lineno</u>, price, qty)
Agent(<u>aid</u>, aname)
OrderAgent(<u>oid</u>, <u>aid</u>)
Date(<u>date</u>, qtr)

The schema also contains a logical view defined by the following SQL query (we abbreviate relation names with capitals and use subscripts to distinguish repeated relations). Although the base relations do not contain duplicates, view AgentSales may (due to the bag semantics of SQL).

AgentSales(aid, aname, date, ctype, oval)
select aid, aname, date, ctype, sum(price * qty)
from C $\bowtie_{\text{cid}}$ O $\bowtie_{\text{oid}}$ LI $\bowtie_{\text{oid}}$ OA $\bowtie_{\text{aid}}$ A
group by aid, aname, date, ctype, oid

[1]Dong et al. [13] consider containment of a restricted class of COQL queries (corresponding to XQuery), showing it to be in co-NEXPTIME, but NP-complete or co-NP-complete for a variety of further restrictions. To the best of our knowledge, the complexity of the general COQL containment problem remains open.

The attribute `Customer.ctype` is code that classifies customers as either Residential or Corporate, and sales from the two sectors are always reported separately. Suppose that an end user wants a report that lists for each agent the quarterly average order value, with the Residential and Corporate metrics shown in separate columns. Equipped only with a reporting tool that generates single-block conjunctive SQL queries (with aggregation), the user could accomplish this report by generating the following query.

Q_1: select $\texttt{AS}_1$.aname, qtr,
avg($\texttt{AS}_1$.oval) as avgRsale,
avg($\texttt{AS}_2$.oval) as avgCsale
from $(\texttt{AS}_1 \bowtie_{\text{date}} \texttt{D}_1) \bowtie_{\{\text{aid, qtr}\}} (\texttt{AS}_2 \bowtie_{\text{date}} \texttt{D}_2)$
where $\texttt{AS}_1$.ctype = 'R' and $\texttt{AS}_2$.ctype = 'C'
group by aid, $\texttt{AS}_1$.aname, qtr

Suppose that the database system contains the following materialized views.

`OrderValues`(oid, oval)
select oid, sum(price * qty)
from `LI` group by oid

`AnnualAgentSales`(aid, qtr, ctype, avgOval)
select aid, qtr, ctype, avg(oval)
from $\texttt{C} \bowtie_{\text{cid}} \texttt{O} \bowtie_{\text{oid}} \texttt{OV} \bowtie_{\text{oid}} \texttt{OA} \bowtie_{\text{date}} \texttt{D}$
group by aid, qtr, ctype

The best rewriting of Q_1 found by any RDBMS that we tested uses schema information to push down the sum aggregate in `AgentSales` in order to rewrite over two occurrences of view `OrderValues`. However, no RDBMS could remove the problematic cartesian product between each agent's quarterly Residential and Corporate orders, and hence no rewritings of Q_1 over view `AnnualAgentSales` were found. In contrast, the following query Q_2 does not contain the problematic cartesian product, and our paper provides an algorithm proving that Q_2 is equivalent to Q_1 with respect to the given schema constraints (but not equivalent in general).

Q_2: select aname, qtr,
$\texttt{AAS}_1$.avgOval as avgRsale,
$\texttt{AAS}_2$.avgOval as avgCsale
from $\texttt{A} \bowtie_{\text{aid}} \texttt{AAS}_1 \bowtie_{\{\text{aid, qtr}\}} \texttt{AAS}_2$
where $\texttt{AAS}_1$.ctype = 'R' and $\texttt{AAS}_2$.ctype = 'C'
order by aname, qtr

Our second example illustrates why mutual strong-simulation does not imply equivalence of queries with nested sets.

Example 2 Consider a database containing a relation `E(P,C)` that denotes parent-child relationships, along with the following three queries (written in an SQL-like syntax that corresponds to empty-set-free COQL).

Q_3: { select { $u.C$ } from E as x,
(select $z.P$, { $z.C$ } as C from E as z
group by $z.P$) as u
where $x.C = u.P$ group by $x.C$ }

Q_4: { select { $u.C$ } from E as x, E as y,
(select $z.P$, { $z.C$ } as C from E as z
group by $z.P$) as u
where $x.C = u.P$ and $y.C = u.P$
group by $x.P$, $y.P$ }

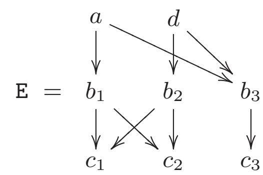


Figure 1: Database instance $\mathbb{D}_1$

Q'_3

A	B	C
a	b_1	c_1
a	b_1	c_2
a	b_3	c_3
d	b_2	c_1
d	b_2	c_2
d	b_3	c_3

Q'_4

AD	B	C
$a\ a$	b_1	c_1
$a\ a$	b_1	c_2
$a\ a$	b_3	c_3
$a\ d$	b_3	c_3
$d\ a$	b_3	c_3
$d\ d$	b_2	c_1
$d\ d$	b_2	c_2
$d\ d$	b_3	c_3

Q'_5

A	DB	C
a	$a\ b_1$	c_1
a	$a\ b_1$	c_2
a	$a\ b_3$	c_3
a	$d\ b_3$	c_3
d	$a\ b_3$	c_3
d	$d\ b_2$	c_1
d	$d\ b_2$	c_2
d	$d\ b_3$	c_3

Figure 2: Evaluating Q'_3, Q'_4, and Q'_5 over $\mathbb{D}_1$

Q_5: { select { C } from E as x,
(select $z.P$, { $z.C$ } as C
from E as y, E as z where $y.C = z.P$
group by $y.P$, $z.P$) as u
where $x.C = u.P$
group by $x.P$, }

Query Q_3 returns sets of related grandchildren, grouped first into sets with a common parent, and then into sets with a common grandparent. Query Q_4 is similar to Q_3, but the outer aggregation groups by pairs of grandparents. Query Q_5 is also similar to Q_3, but the inner aggregation groups by both parent and grandparent. Levy and Suciu's technique [25] associates Q_3, Q_4, and Q_5 with the following indexed CQs.

$$\begin{aligned}
Q'_3(\overbrace{A}^{\overline{\mathcal{I}}_1};\ \overbrace{B}^{\overline{\mathcal{I}}_2};\ \overbrace{C}^{\overline{\mathcal{V}}}) &:- \texttt{E}(A,B), \texttt{E}(B,C)\\
Q'_4(A,D;\ B\ ;\ C\ ) &:- \texttt{E}(A,B), \texttt{E}(B,C), \texttt{E}(D,B)\\
Q'_5(\ A\ ; D,B;\ C\ ) &:- \texttt{E}(A,B), \texttt{E}(B,C), \texttt{E}(D,B)
\end{aligned}$$

Consider the database $\mathbb{D}_1$ in Figure 1 and the corresponding query results in Figure 2 (index groups have been visually separated for clarity). The reader can verify that over database $\mathbb{D}_1$ all six strong simulation conditions $Q'_3 \lll_2 Q'_4$, $Q'_4 \lll_2 Q'_3$, $Q'_3 \lll_2 Q'_5$, $Q'_5 \lll_2 Q'_3$, $Q'_4 \lll_2 Q'_5$, and $Q'_5 \lll_2 Q'_4$ are satisfied (c.f. equation 2); in fact, we can show that they are all satisfied over any database. However, the queries are not all equivalent since over $\mathbb{D}_1$ queries Q_3 and Q_5 output the object $\{\{\{c_1, c_2\}, \{c_3\}\}\}$ while Q_4 outputs $\{\{\{c_1, c_2\}, \{c_3\}\}, \{\{c_3\}\}\}$. We show later that queries Q_3 and Q_5 are equivalent.

The remainder of the paper will proceed as follows. In Section 2 we formalize a data model for objects and a query language for constructing them. In Section 3 we describe an encoding of objects within flat relations and reduce equivalence of nested object queries to encoding equivalence between CQs. We propose a normal form for encoding queries in Section 4, where we use it to characterize encoding equivalence. Section 5 considers certain extensions of the basic technique, including the handling of schema dependencies. We summarize our results in Section 6 and suggest further possible extensions.

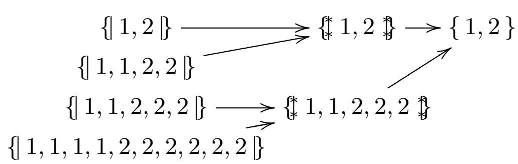
2. OBJECTS AND OBJECT QUERIES

We define a data model for complex objects, along with certain transformations between objects that we will find useful later. We then define a query language for constructing objects out of a database of flat relations.

2.1 Mixed-Type Objects

Our data model utilizes three different collection types: sets, bags, and normalized bags, which we denote with the delimiters $\{\cdot\}$, $\{\!|\cdot|\!\}$, and $\{\!\{\cdot\}\!\}$, respectively. A normalized bag is a special case of a bag in which the greatest common divisor of the element frequencies is one; this is useful for modelling the semantics of certain statistical functions such as average or standard deviation.

Example 3 The following four distinct bags correspond to two distinct normalized bags and a single set. The user can verify that the collections have four distinct sums, two distinct averages, and the same max or min.



Let dom denote a countably infinite set of atomic values. A *sort* is a finite instance of the following grammar

$$\tau := \texttt{dom} \mid \{\tau\} \mid \{\!|\tau|\!\} \mid \{\!\{\tau\}\!\} \mid \langle \tau, \ldots, \tau \rangle \qquad (3)$$

where the delimiters $\langle \cdot \rangle$ denote a tuple. We call a tuple sort *flat* if it is composed of atomic sorts only, and we say that a sort is a *chain sort* if it contains precisely one descendant tuple sort, and that tuple sort is flat. We define the *depth* of a sort as the maximum number of *collection* sorts occurring along any root-to-leaf path in its hierarchical definition.

We define three *semantic indicators* s, b, and n which are used to denote whether a collection is of type set, bag, or normalized bag, respectively. Any chain sort of depth d can be abbreviate by a pair $(\overline{\S}, k)$, where $\overline{\S}$ is a *signature* composed of d semantic indicators that indicates from left-to-right the type of successive descendant collection sorts, and k is the arity of the tuple at the leaf of the type. Given an arbitrary sort τ, we use $\textsc{Chain}(\tau)$ to denote the chain sort abbreviated as $(\overline{\S}, k)$, where $\overline{\S}$ records the semantic indicators of the collection sorts in τ in *preorder*, and k is the total number of atomic sorts in τ. If $\overline{\S} \neq \varnothing$ then $\S_i$ represents the i^{th} semantic indicator ($i \in [1, |\overline{\S}|]$).

Example 4 Consider the sorts depicted graphically in Figure 3 (collection types have been numbered for clarity). Sort τ_1 has depth three and is *not* a chain sort. Sort $\textsc{Chain}(\tau_1)$ is a chain sort of depth five that abbreviates as $(\texttt{bnbnb}, 6)$.

We use $[\![\tau]\!]$ to denote the (infinite) set of possible values conforming to sort τ, called the *interpretation of* τ. We define a *complex object* as a finite member of the set $\bigcup_\tau [\![\tau]\!]$.

We say that an object is *complete* if it does not contain any empty collections. We say that an object is *trivial* if it is either an empty collection or a tuple of trivial objects. We say that an object is a *chain object* if it conforms to a chain sort and it is either complete or trivial. Chain objects are useful because they are straightforward to encode within a single relation (Section 3.1).

Given any object $o \in [\![\tau]\!]$ that is either complete or trivial, we can transform o into a corresponding chain object $\textsc{Chain}(o) \in [\![\textsc{Chain}(\tau)]\!]$ by a recursive procedure that removes tuple branching by distributing copies of the right sub-object over the leaves of the left-subobject. The algorithm is given in Appendix A. This transformation is lossless in that given both τ and $\textsc{Chain}(o)$ the original object o can be reconstructed. Hence, given any sort τ and any two objects $o, o' \in [\![\tau]\!]$ that are each either complete or trivial, $o = o'$ iff $\textsc{Chain}(o) = \textsc{Chain}(o')$.

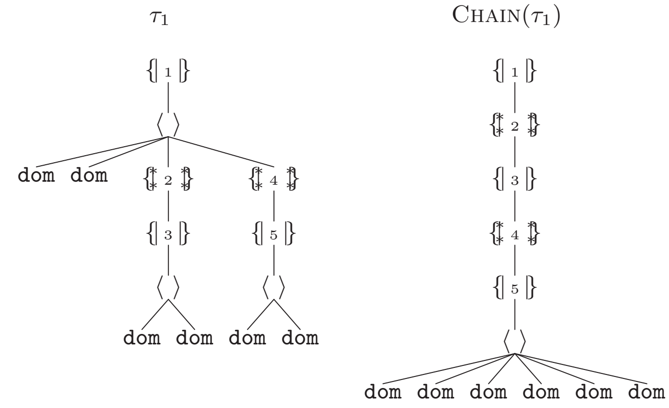


Figure 3: Sorts τ_1 and $\textsc{Chain}(\tau_1) = (\texttt{bnbnb}, 6)$

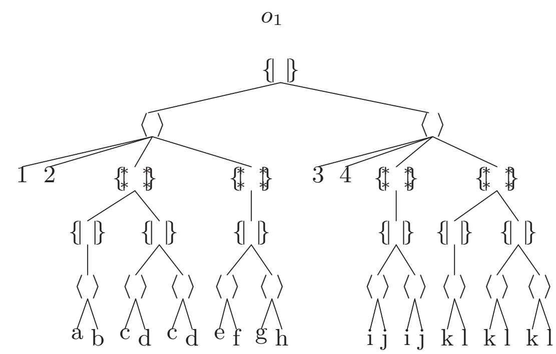


Figure 4: Object $o_1 \in [\![\tau_1]\!]$

Example 5 Figure 4 depicts object o_1 conforming to sort τ_1 from Figure 3. The transformation of o_1 into chain object $\textsc{Chain}(o_1)$ conforming to sort $\textsc{Chain}(\tau_1)$ is shown in Figure 5.

2.2 Object-Constructing Queries

We now specify a query language we call COCQL ("Conjunctive Object-Constructing Query Language") for constructing objects out of a database of flat relations (we consider nested inputs in Section 5.2). Our intent is to approximate the queries expressible using conjunctive SQL expressions with *non-scalar* aggregation and FROM-clause nesting (i.e., the

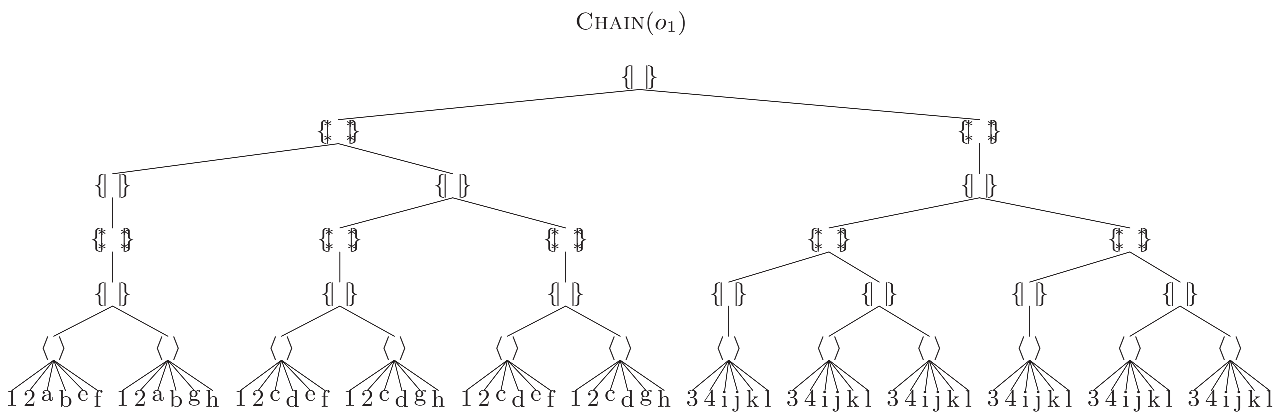


Figure 5: Object $\text{CHAIN}(o_1) \in [\![\, \text{CHAIN}(\tau_1)\,]\!]$

language of "stacked views" [12]). As such, COCQL corresponds to a conjunctive fragment of the *bag semantic* relational algebra extended with a grouping operator [14]. We let aname denote an infinite set of attribute names. We also define a set $\mathcal{F} = \{\text{SET}, \text{BAG}, \text{NBAG}\}$ of aggregation functions that each aggregate a collection of tuples into a set, bag, or normalized bag object.

A COCQL query is an expression conforming to the following grammar.

$$\begin{aligned} Q &:= \{\, E \,\} \;\mid\; \{\!|\, E \,|\!\} \;\mid\; \{\!\!|\, E \,|\!\!\} \\ E &:= R(\overline{A}) \;\mid\; \sigma_p(E) \;\mid\; E_1 \bowtie_p E_2 \;\mid\; \textstyle\prod^{\text{dup}}_{\overline{W}}(E) \;\mid \\ &\quad\; \textstyle\prod^{[Y=f(\overline{Z})]}_{\overline{X}}(E) \end{aligned}$$

Evaluating COCQL query Q over database $\mathbb{D}$ yields the object $(Q)^{\mathbb{D}}$, which is either a set, a bag, or a normalized bag constructed from the result of evaluating the algebraic subquery under *bag-set semantics* (i.e., bag semantics with the assumption that base relations are sets). Several comments pertain to the algebraic sub-language:

1. The base relation operator $R(\overline{A})$ requires $\overline{A}$ to be a tuple of "fresh" attributes names from aname. This notation should be perceived algebraically as enacting mandatory attribute renaming, rather than as introducing query variables (although we use it here to simplify later translation to variable-based CQ notation).
2. Predicate p is a conjunction of equality comparisons restricted to constants/attributes of *atomic sort*.
3. $\prod^{\text{dup}}_{\overline{W}}$ denotes *duplicate-preserving* projection. Tuple $\overline{W}$ is a sequence of constants/attributes of unrestricted sort.
4. $\prod^{[Y=f(\overline{Z})]}_{\overline{X}}$ denotes *generalized projection* with grouping list $\overline{X}$ and an optional aggregation expression [14, 16]. In this paper, we restrict $\overline{X}$ to containing *atomic sorts* (a restriction analogous to one in COQL [25]). Expression $Y = f(\overline{Z})$ requires that Y be a "fresh" attribute name from aname, $f \in \mathcal{F}$, and $\overline{Z}$ be a sequence of constants or attribute names. We note that the case $\overline{X} = \varnothing$ is treated with the same semantics as $\overline{X} \neq \varnothing$ and so, analogous to the *nest* operator [36], generalized projection cannot construct empty collection objects (in contrast to SQL, which switches between *scalar* and *non-scalar* aggregation).

Because the algebraic component of COCQL is not capable of constructing empty collection objects, the result of any COCQL query is always either a *complete* or a *trivial* object. A COCQL query is *satisfiable* there exists a database instance over which it outputs a non-trivial object. COCQL satisfiability is verifiable in polynomial time (identical to satisfiability of CQs with explicit equality), and so for the remainder of the paper we restrict our attention to satisfiable COCQL queries.

Example 6 Query Q_3 from Example 2 can be expressed in COCQL as follows. Queries Q_4 and Q_5 are similar.

$$Q_3\colon \{\, \textstyle\prod^{\mathsf{dup}}_{Y}(\prod^{Y=\mathsf{set}(X)}_{A}(\mathtt{E}(A, B') \bowtie_{B'=B} \prod^{X=\mathsf{set}(C)}_{B}(\mathtt{E}(B, C)))) \,\}$$

Because COCQL queries do not explicitly contain tuple constructors, we adopt a convention for the evaluation of COCQL queries that uses the minimal number of tuple constructors necessary (i.e., no unary tuples). For example, the query in Example 6 outputs results with sort $\{\, \{\, \{\, \mathtt{dom} \,\}\,\}\,\}$.

3. RELATIONAL ENCODINGS AND ENCODING QUERIES

In this section we first specify a relational encoding for complex objects. We then describe a translation from an arbitrary COCQL query Q to a conjunctive query $\text{ENCQ}(Q)$ such that whenever query Q outputs object o, query $\text{ENCQ}(Q)$ outputs an encoding of $\text{CHAIN}(o)$.

3.1 Encoding Relations

Because COCQL queries are incapable of constructing empty subcollections, we restrict out attention to objects that are either complete or trivial. In light of the CHAIN transformation previously defined, it suffices to encode chain objects, which we encode within relations by use of *indexes*. Figure 6 illustrates the basic idea—given a chain object o of depth d, to each member of each collection type we assign a locally-unique *index value* composed of one or more atomic values. Then for each leaf tuple $\langle\, \overline{x} \,\rangle \in o$, we generate one relational tuple $\langle\, i_1; \ldots; i_d; \overline{x} \,\rangle$ where $i_1; \ldots; i_d$ is the sequence of index values assigned along the path from the root to t.[2]

[2]There are two minor differences between our encoding

$$\{\!\!\{\; \underbrace{\{\!|\; \overbrace{\langle 1 \rangle}^{y_1} \;|\!\}}_{w_1 x_1},\; \underbrace{\{\!|\; \overbrace{\langle 1 \rangle}^{y_1} \;|\!\}}_{w_1 x_2},\; \underbrace{\{\!|\; \overbrace{\langle 2 \rangle}^{y_1}, \overbrace{\langle 2 \rangle}^{y_2} \;|\!\}}_{w_2 x_1} \;\}\!\!\}$$

R_1

W X	Y	Z
$w_1\, x_1$	y_1	1
$w_1\, x_2$	y_1	1
$w_2\, x_1$	y_1	2
$w_2\, x_1$	y_2	2

Figure 6: Encoding of a chain object

R_2

A	B C	D
a_1	$b_1\, c_1$	1
a_2	$b_1\, c_1$	1
a_2	$b_1\, c_2$	1
a_3	$b_1\, c_1$	1
a_4	$b_1\, c_1$	1
a_5	$b_1\, c_1$	2
a_5	$b_2\, c_1$	2
a_6	$b_2\, c_2$	2

$R_2[a_2]$

B C	D
$b_1\, c_1$	1
$b_1\, c_2$	1

$R_2[a_2 b_1 c_1]$

D
1

Figure 7: More encoding relations

More formally, we define an *encoding schema of depth d* ($d \geq 0$) as a relational schema with the following form.

$$R(\overline{\mathcal{I}}_1; \overline{\mathcal{I}}_2; \ldots; \overline{\mathcal{I}}_d; \overline{\mathcal{V}})$$

Each $\overline{\mathcal{I}}_i$ is a sequence of distinct attributes called the *index attributes at level i*, with $\mathcal{I}_i$ denoting the set of attributes in $\overline{\mathcal{I}}_i$, while $\overline{\mathcal{V}}$ is a sequence of *output attributes*, with $\mathcal{V}$ denoting the set of attributes in $\overline{\mathcal{V}}$. For convenience, we use $\overline{\mathcal{I}}_{[i,j]}$ to denote the sequence $\overline{\mathcal{I}}_i \overline{\mathcal{I}}_{i+1} \cdots \overline{\mathcal{I}}_j$, and $\mathcal{I}_{[i,j]}$ the corresponding set. Each attribute can occur as either an index attribute, an output attribute, or both; however, an attribute cannot occur as an index within multiple levels.

We define an *encoding relation* as an encoding schema paired with a relational instance over the attributes $\mathcal{I}_{[1,d]} \cup \mathcal{V}$ that satisfies the functional dependency $\mathcal{I}_{[1,d]} \rightarrow \mathcal{V}$. (When depicting encoding relations graphically, as in Figure 6, we separate index levels with a single rule and the index attributes from the output attributes with a double rule.) Given a relation R and any attribute $A \in (\mathcal{I}_{[1,d]} \cup \mathcal{V})$, we use $\texttt{adom}(A, R) \subset \texttt{dom}$ to denote the *active domain* of attribute A within relation R. Given any value $\overline{a} \in \texttt{adom}(\overline{\mathcal{I}}_{[1,l-1]}, R)$ we use $R[\overline{a}]$ to denote the sub-relation of R indexed by $\overline{a}$, which is itself an encoding relation with schema $R(\overline{\mathcal{I}}_l; \ldots; \overline{\mathcal{I}}_d; \overline{\mathcal{V}})$. For example, Figure 7 shows encoding relation R_2 with schema $R_2(A; B, C; D)$ along with sub-relations $R_2[a_2]$ and $R_2[a_2 b_1 c_1]$.

Consider encoding relation R_1 in Figure 6. By applying the "decoding" query $\{\!\!\{ \prod_A^{\text{dup}}(\prod_{WX}^{A=\text{BAG}(Z)}(R_1(W, X; Y; Z))) \}\!\!\}$ we re-obtain the object in Figure 6.[3] We call this object the $\texttt{nb}$*-decoding of* R_1, denoted $\textsc{Decode}(R_1, \texttt{nb})$. Because R_1 has depth two and one output attribute, for any signature $\overline{\S}$ with $|\overline{\S}| = 2$ we can define a similar decoding query that yields an object of sort $(\overline{\S}, 1)$. For example, the $\texttt{ss}$-decoding of R_1 is the object $\{\,\{\,\langle 1 \rangle\,\}, \{\,\langle 2 \rangle\,\}\,\}$.

method and the one used by Levy and Suciu [25]. First, they effectively convert arbitrary sorts into chain sorts by merging all collections at the same depth, whereas we increase the sort depth by marshalling types in preorder (which is required because merging collection types loses cardinality information). Second, because they only consider sets they do not need indexes for innermost collection (see Example 2), whereas we require them to retain element cardinalities.

Definition 1 (Encoding-Equality) Given a signature $\overline{\S}$ and two encoding relations R, R' of depth $|\overline{\S}|$, we say that R and R' are $\overline{\S}$*-equal*—denoted $R \doteq_{\overline{\S}} R'$—if $\textsc{Decode}(R, \overline{\S}) = \textsc{Decode}(R', \overline{\S})$.

Example 7 Consider the encoding relation R_2 in Figure 7. Fairly obviously, the $\texttt{nb}$-decoding of R_2 does *not* yield the object in Figure 6, and so $R_1 \not\doteq_{\texttt{nb}} R_2$. However, $R_1 \doteq_{\texttt{ns}} R_2$ because decoding either relation with signature $\texttt{ns}$ yields the object $\{\!\!\{\,\{\,\langle 1 \rangle\,\}, \{\,\langle 1 \rangle\,\}, \{\,\langle 2 \rangle\,\}\,\}\!\!\}$.

While Definition 1 captures the desired semantics of $\overline{\S}$-equality, the invocation of the $\textsc{Decode}$ procedure makes formal reasoning awkward. In Appendix B we define a mechanism called a $\overline{\S}$*-certificate* that allows us to characterize $\overline{\S}$-equality in a more declarative fashion, which is required for our proofs of the theorems in Section 4. A $\overline{\S}$-certificate is essentially a recursive log of one possible set of comparisons justifying the conclusion that two invocations of the $\textsc{Decode}$ procedure yield the same object.

3.2 Encoding Queries

Assuming standard rule-based syntax for CQs [1], we define a *conjunctive encoding query (CEQ) of depth d* as a CQ with a head resembling a depth-d encoding schema.

$$Q(\overline{\mathcal{I}}_1; \ldots; \overline{\mathcal{I}}_d; \overline{\mathcal{V}}) \coloneq R_1(\overline{X}_1), \ldots, R_n(\overline{X}_n) \qquad (4)$$

Each $\overline{\mathcal{I}}_i$ is a sequence of distinct variables called the *index variables at level i*, with $\mathcal{I}_i$ denoting the set of variables in $\overline{\mathcal{I}}_i$; we again assume that the index sets of different levels are disjoint. $\overline{\mathcal{V}}$ is a sequence of variables and constants, with $\mathcal{V}$ denoting the set of *variables* occurring in $\overline{\mathcal{V}}$. Finally, we use $\mathcal{B}$ to denote the variables occurring in the query body, and we require that $\mathcal{I}_{[1,d]} \cup \mathcal{V} \subseteq \mathcal{B}$. The result of evaluating query Q over a database $\mathbb{D}$ is an encoding relation $(Q)^{\mathbb{D}}$ whose encoding schema is deduced from the query head in a manner analogous to CQs.

Definition 2 (Encoding-Equivalence) Given a signature $\overline{\S}$ and two CEQs Q, Q' of depth $|\overline{\S}|$ over the same database schema, we say that Q and Q' are $\overline{\S}$*-equivalent*—denoted $Q \doteqdot_{\overline{\S}} Q'$—if over every database $\mathbb{D}$ the encoding relations $(Q)^{\mathbb{D}}$ and $(Q')^{\mathbb{D}}$ are $\overline{\S}$-equal.

Given a satisfiable $\texttt{COCQL}$ query Q with output sort τ, we construct the corresponding CEQ $\textsc{EncQ}(Q)$ as follows.

1. Create the body of $\textsc{EncQ}(Q)$ by collecting all of the base relation operators in Q (taking the assigned attribute names as query variables) and then introducing constants and shared variables to enact the join and selection predicates.
2. Construct the output list $\overline{\mathcal{V}}$ by enumerating the atomic sorts of τ in preorder, and emitting for each the corresponding query variable.

[3] Modulo introduction of unary tuple constructors, a technicality which we ignore.

3. Let $\tau_1, \ldots, \tau_d$ denote the *collection* sorts within τ listed in preorder. For each $i \in [1, d]$, calculate $\overline{\mathcal{I}}_i$ as follows.
   (a) Locate the query operator within Q that constructs collections corresponding to τ_i. When $i = 1$ this operator is the explicit collection constructor enclosing the algebraic expression; otherwise it is a generalized projection operator.
   (b) Let E be the algebraic sub-expression that inputs into the construction operator. Let E' be a copy of E with all duplicate-preserving projection operators deleted. Let S be the set of query variables corresponding to *atomic* attributes output by E'.
   (c) Define $\overline{\mathcal{I}}_i$ as any ordering of the set $S \setminus \mathcal{I}_{[1,i-1]}$.

Example 8 Consider queries Q_1 and Q_2 from Example 1. If we model the output of sum and avg as bags and normalized bags, respectively, then Q_1 and Q_2 have straightforward translations into COCQL queries with output sort τ_1 from Figure 3. (These translations into COCQL make use of a well-known technique of transforming an aggregation block with k aggregation expressions into a join of k such blocks, each with a single aggregation expression.) Figure 8 illustrates the CEQs $Q_6 := \textsc{EncQ}(Q_1)$ and $Q_7 := \textsc{EncQ}(Q_2)$. (Components of the queries have been labelled for the sake of clarity; the significance of the shaded attributes will be explained later.)

Proposition 1 *Given any database schema and any satisfiable COCQL query Q over that schema with output sort τ, let $(\overline{\S}, k)$ abbreviate* $\textsc{Chain}(\tau)$. *Then, for every database instance $\mathbb{D}$, the $\overline{\S}$-decoding of relation $(\textsc{EncQ}(Q))^{\mathbb{D}}$ yields object* $\textsc{Chain}((Q)^{\mathbb{D}})$.

Theorem 1 *Given two satisfiable COCQL queries Q, Q' with the same output sort τ, let $(\overline{\S}, k)$ abbreviate* $\textsc{Chain}(\tau)$. *Then, $Q \equiv Q'$ iff $\textsc{EncQ}(Q) \doteq_{\overline{\S}} \textsc{EncQ}(Q')$.*

4. EQUIVALENCE OF ENCODING QUERIES

We now consider how to determine encoding equivalence between CEQs, thereby providing an algorithm for COCQL query equivalence (cf. Theorem 1). In Section 4.1 we define a normal form for CEQs and prove that conversion to the normal form preserves encoding equivalence. Our main result is in Section 4.2, where we prove that testing encoding equivalence between queries in normal form is a simple generalization of CQ equivalence.

CEQs must yield encoding relations, meaning the query results always satisfy $\mathcal{I}_{[1,d]} \rightarrow \mathcal{V}$. We assume in this section that queries satisfy the syntactic constraint $\mathcal{V} \subseteq \mathcal{I}_{[1,d]}$ (a condition satisfied by all queries generated by procedure $\textsc{EncQ}(Q)$ in Section 3.2). Section 5.1 describes how to relax this assumption in the presence of schema dependencies.

Encoding equivalence is a relationship that is interesting in its own right, as the case $|\overline{\S}| = 1$ suffices to express CQ equivalence under various processing semantics. For example, given two CQs $Q(\overline{\mathcal{V}})$ and $Q'(\overline{\mathcal{V}}')$, testing $Q \equiv Q'$ under

- *set semantics* [5] reduces to $Q(\overline{\mathcal{V}}; \overline{\mathcal{V}}) \doteq_{\mathtt{s}} Q'(\overline{\mathcal{V}}'; \overline{\mathcal{V}}')$;
- *bag-set semantics* [6] reduces to $Q(\mathcal{B}; \overline{\mathcal{V}}) \doteq_{\mathtt{b}} Q'(\mathcal{B}'; \overline{\mathcal{V}}')$ where $\mathcal{B}$ and $\mathcal{B}'$ are the query body variables;
- *bag-set semantics modulo a product* [15] reduces to $Q(\mathcal{B}; \overline{\mathcal{V}}) \doteq_{\mathtt{n}} Q'(\mathcal{B}'; \overline{\mathcal{V}}')$; and
- *combined semantics* [7] reduces to $Q(\mathcal{V} \cup \mathcal{M}; \overline{\mathcal{V}}) \doteq_{\mathtt{b}} Q'(\mathcal{V}' \cup \mathcal{M}'; \overline{\mathcal{V}}')$ where $\mathcal{M}$ and $\mathcal{M}'$ are the specified *multi-set variables.*

4.1 Encoding Normal Form

In this section we define a normal form for CEQs which is based upon *multivalued dependencies (MVDs)* over relations [1]. Given an CQ Q that yields a relation over attribute set U, and a disjoint partitioning of U into three sets X, Y, Z, we say that Q implies $X \twoheadrightarrow Y$—denoted $Q \models X \twoheadrightarrow Y$—if for every database $\mathbb{D}$ the relation $(Q)^{\mathbb{D}}$ satisfies $X \twoheadrightarrow Y$. This implies the following equivalence by definition,

$$Q \equiv \prod_{XY}(Q) \bowtie \prod_{XZ}(Q) \qquad (5)$$

and so deciding CQ-implied MVDs reduces to CQ equivalence. We can reduce CQ containment to deciding CQ-implied MVDs, so deciding CQ-implied MVDs is NP-complete.

Equation 5—which follows directly from the definition of MVDs—has consequences for the structure of the query body. Define the *query hypergraph* $H^Q = (\mathcal{B}, E)$ as a pair where $\mathcal{B}$ is the set of variables in $\mathtt{body}_Q$ and E is a set of subsets of $\mathcal{B}$ such that for each subgoal $R_i(\overline{X}_i)$ in $\mathtt{body}_Q$, there exists a hyperedge $e_i \in E$ equal to the set of variables occurring in $\overline{X}_i$. We say that X is a *strong* (Y, Z)*-articulation set* in H^Q if by deleting the variables in X from H^Q we disconnect each variable in Y from each variable in Z. The following lemma can be shown to follow from equation 5.

Lemma 1 *Given CQ $Q(\overline{U})$ and a disjoint partitioning of the variables in $\overline{U}$ into three sets X, Y, Z, let $Q'(\overline{U})$ be an equivalent* minimal *CQ. Then Q implies $X \twoheadrightarrow Y$ iff X is a strong (Y, Z)-articulation set of $H^{Q'}$.*

Our normal form is calculated by recursively identifying the *core* indexes. Given a CEQ $Q(\overline{\mathcal{I}}_1; \ldots; \overline{\mathcal{I}}_d; \overline{\mathcal{V}})$ and a length-d signature $\overline{\S}$, define the *core indexes at level i relative to* $\overline{\S}$—denoted $\mathcal{I}_i^{\overline{\S}}$—as follows. For each $i \in [1, d]$ let Q_i be the following CQ.

$$Q_i(\mathcal{I}_{[1,i]}\mathcal{I}^{\overline{\S}}_{[i+1,d]}) :- \mathtt{body}_Q$$

Then, $\mathcal{I}_i^{\overline{\S}}$ is the smallest subset of $\mathcal{I}_i$ that satisfies the following conditions. In Appendix C.2 we show that Lemma 1 implies that a unique minimum set $\mathcal{I}_i^{\overline{\S}}$ always exists.

$\S_\mathtt{i}$	**Condition**
b	$\mathcal{I}_i \subseteq \mathcal{I}_i^{\overline{\S}}$
s	$\mathcal{I}_i \cap \mathcal{V} \subseteq \mathcal{I}_i^{\overline{\S}}$ and $Q_i \models (\mathcal{I}_{[1,i-1]} \cup \mathcal{I}_i^{\overline{\S}}) \twoheadrightarrow \mathcal{I}^{\overline{\S}}_{[i+1,d]}$
n	$\mathcal{I}_i \cap \mathcal{V} \subseteq \mathcal{I}_i^{\overline{\S}}$ and $Q_i \models \mathcal{I}_{[1,i-1]} \twoheadrightarrow \mathcal{I}^{\overline{\S}}_{[i,d]}$

Any non-core index variable is called *redundant.* A CEQ is converted to $\overline{\S}$*-normal form ($\overline{\S}$-NF)* by deleting all redundant index variables from the query head.

Theorem 2 $\overline{\S}$*-Normalization is NP-complete.*

Example 9 Consider the four CEQs in Figure 9. (Queries Q_8, Q_9, and Q_{10} correspond to $\textsc{EncQ}(Q_3)$, $\textsc{EncQ}(Q_4)$, and $\textsc{EncQ}(Q_5)$, respectively). With respect to signature sss, variable D is redundant in both Q_{10} and Q_{11}, but both Q_8 and Q_9 are in sss-NF. With respect to signature snn, variable D is redundant in Q_{11}, but the other three queries are in snn-NF.

$$Q_6(\overbrace{A,N,R}^{\overline{\mathcal{I}}_1};\ \overbrace{D_1,O_1,\boxed{N_2,D_2,O_2}}^{\overline{\mathcal{I}}_2};\ \overbrace{C_1,M_1,L_1,P_1,Y_1}^{\overline{\mathcal{I}}_3};\ \overbrace{\boxed{D_3,O_3,N_4},D_4,O_4}^{\overline{\mathcal{I}}_4};\ \overbrace{C_4,M_4,L_4,P_4,Y_4}^{\overline{\mathcal{I}}_5};\ \overbrace{N,R,P_1,Y_1,P_4,Y_4}^{\overline{\mathcal{V}}}) :-$$

$$\begin{array}{ll}
\texttt{C}(C_1,M_1,\texttt{'R'}),\ \texttt{O}(O_1,C_1,D_1),\ \texttt{LI}(O_1,L_1,P_1,Y_1),\ \texttt{OA}(O_1,A),\ \texttt{A}(A,N),\ \texttt{D}(D_1,R), & \}(\texttt{AS}_1 \bowtie \texttt{D}_1)\ \texttt{avgRsale} \\
\texttt{C}(C_2,M_2,\texttt{'C'}),\ \texttt{O}(O_2,C_2,D_2),\ \texttt{LI}(O_2,L_2,P_2,Y_2),\ \texttt{OA}(O_2,A),\ \texttt{A}(A,N_2),\ \texttt{D}(D_2,R), & \}(\texttt{AS}_2 \bowtie \texttt{D}_2)\ \texttt{avgRsale} \\
\texttt{C}(C_3,M_3,\texttt{'R'}),\ \texttt{O}(O_3,C_3,D_3),\ \texttt{LI}(O_3,L_3,P_3,Y_3),\ \texttt{OA}(O_3,A),\ \texttt{A}(A,N),\ \texttt{D}(D_3,R), & \}(\texttt{AS}_1 \bowtie \texttt{D}_1)\ \texttt{avgCsale} \\
\texttt{C}(C_4,M_4,\texttt{'C'}),\ \texttt{O}(O_4,C_4,D_4),\ \texttt{LI}(O_4,L_4,P_4,Y_4),\ \texttt{OA}(O_4,A),\ \texttt{A}(A,N_4),\ \texttt{D}(D_4,R) & \}(\texttt{AS}_2 \bowtie \texttt{D}_2)\ \texttt{avgCsale}
\end{array}$$

$$Q_7(\overbrace{A',N',R'}^{\overline{\mathcal{I}}'_1};\ \overbrace{C'_1,M'_1,O'_1,D'_1}^{\overline{\mathcal{I}}'_2};\ \overbrace{L'_1,P'_1,Y'_1}^{\overline{\mathcal{I}}'_3};\ \overbrace{C'_2,M'_2,O'_2,D'_2}^{\overline{\mathcal{I}}'_4};\ \overbrace{L'_2,P'_2,Y'_2}^{\overline{\mathcal{I}}'_5};\ \overbrace{N',R',P'_1,Y'_1,P'_2,Y'_2}^{\overline{\mathcal{V}}'}) :-$$

$$\begin{array}{ll}
\texttt{C}(C'_1,M'_1,\texttt{'R'}),\ \texttt{O}(O'_1,C'_1,D'_1),\ \texttt{LI}(O'_1,L'_1,P'_1,Y'_1),\ \texttt{OA}(O'_1,A'),\ \texttt{D}(D'_1,R'),\ \texttt{A}(A',N'), & \}\texttt{AAS}_1 \bowtie \texttt{A} \\
\texttt{C}(C'_2,M'_2,\texttt{'C'}),\ \texttt{O}(O'_2,C'_2,D'_2),\ \texttt{LI}(O'_2,L'_2,P'_2,Y'_2),\ \texttt{OA}(O'_2,A'),\ \texttt{D}(D'_2,R') & \}\texttt{AAS}_2
\end{array}$$

Figure 8: Encoding queries $Q_6 := \text{EncQ}(Q_1)$ and $Q_7 := \text{EncQ}(Q_2)$

$$\begin{array}{l}
Q_8(\overbrace{A}^{\overline{\mathcal{I}}_1};\overbrace{B}^{\overline{\mathcal{I}}_2};\overbrace{C}^{\overline{\mathcal{I}}_3};\overbrace{C}^{\overline{\mathcal{V}}}) :- \texttt{E}(A,B),\texttt{E}(B,C) \\
Q_9(A,D;\ B\ ;\ C\ ;\ C\ ) :- \texttt{E}(A,B),\texttt{E}(B,C),\texttt{E}(D,B) \\
Q_{10}(\ A\ ;D,B;\ C\ ;\ C\ ) :- \texttt{E}(A,B),\texttt{E}(B,C),\texttt{E}(D,B) \\
Q_{11}(\ A\ ;\ B\ ;C,D;\ C\ ) :- \texttt{E}(A,B),\texttt{E}(B,C),\texttt{E}(D,B)
\end{array}$$

Figure 9: Four sample CEQs

Example 10 Consider Figure 8. Converting query Q_6 to bnbnb-NF removes the shaded indexes from $\overline{\mathcal{I}}_4$ and $\overline{\mathcal{I}}_2$. Query Q_7 is already in bnbnb-NF.

We now describe the intuition behind the normal form. Bags are sensitive to changes in absolute cardinalities, which can be caused by deleting any index column; hence $\S_i = \texttt{b}$ requires $\mathcal{I}_i^{\overline{\S}} = \mathcal{I}_i$. Sets are only sensitive to changes in sub-object values, so the condition for $\S_i = \texttt{s}$ limits $\mathcal{I}_i^{\overline{\S}}$ to the index attributes that determine the contents of the sub-relations (inner core indexes + output variables). Finally, normalized bags are sensitive to changes in sub-object values or relative cardinalities, so when $\S_i = \texttt{n}$ an index attribute is redundant if it only serves to inflate the cardinalities of sub-objects by a multiplicative factor.

Theorem 3 *$\overline{\S}$-Normalization preserves $\overline{\S}$-equivalence.*

4.2 Testing Encoding Equivalence

We now fully characterize encoding equivalence by generalizing the traditional homomorphism test for CQs.

Definition 3 (Index-Covering Homomorphism) Given two CEQs $Q(\overline{\mathcal{I}}_1;\ldots;\overline{\mathcal{I}}_d;\overline{\mathcal{V}})$ and $Q'(\overline{\mathcal{I}}'_1;\ldots;\overline{\mathcal{I}}'_d;\overline{\mathcal{V}}')$, an *index-covering homomorphism from Q' to Q* is a mapping h from the variables of Q' to the variables and constants of Q satisfying (1) $h(\texttt{body}_{Q'}) \subseteq \texttt{body}_Q$, (2) $h(\overline{\mathcal{V}}') = \overline{\mathcal{V}}$, and (3) $\forall i \in [1,d]$: $\mathcal{I}_i \subseteq h(\mathcal{I}'_i)$.

Theorem 4 *Two CEQs are $\overline{\S}$-equivalent iff there exists index-covering homomorphisms in both directions between their $\overline{\S}$-normal forms.*

Corollary 1 *Deciding $\overline{\S}$-equivalence is NP-complete.*

Corollary 2 *Deciding COCQL equivalence is NP-complete.*

Example 11 Continuing Example 10, clearly no index-covering homomorphisms can exist between the normalized Q_6 and Q_7, and so $Q_6 \not\doteq_{\texttt{bnbnb}} Q_7$ which entails $Q_1 \not\equiv Q_2$ (for the COCQL versions of the queries, and also for the SQL versions assuming uninterpreted aggregation functions).

5. EXTENSIONS

In this section we discuss a few extensions to the technique presented so far—namely, adding schema dependencies, allowing nested inputs, and adding an unnest operator.

5.1 Schema Dependencies

In Sections 3 and 4 we reduced COCQL equivalence to a condition very close to relational CQ equivalence. Because of this similarity, we can adapt techniques for testing equivalence of CQs over database instances constrained by a set Σ of schema constraints (denoted $Q \equiv^\Sigma Q'$) to COCQL equivalence. For classes allowing a terminating chase procedure (e.g., FDs + JDs + acyclic INDs [1]), we can decide *encoding equivalence w.r.t.* Σ as follows. Prior to the conversion to $\overline{\S}$-NF, we pre-process CEQs by first chasing out the query bodies and then using FDs to expand out the index sets in the query head (deleting variables from inner index sets whenever they are added to outer index sets). The conversion to $\overline{\S}$-NF is unchanged, but the test for query-implied MVDs in equation 5 needs to use $\equiv^\Sigma$. Theorem 1 is then modified to say $Q \equiv^\Sigma Q'$ iff $\text{EncQ}(Q) \doteq^\Sigma_{\overline{\S}} \text{EncQ}(Q')$.

Example 12 Reconsider queries Q_6 and Q_7 from Figure 8. Chasing the query bodies with the primary and foreign key constraints from Example 1 does not introduce any new subgoals, but it does merge the variables N, N_2, N_4 in Q_6. Expanding the index sets in Q_6 yields the following head, with shading again indicating the redundant index columns that get removed by bnbnb-normalization.

$$\begin{array}{lll}
Q'_6(& A,N,R; & \}\overline{\mathcal{I}}_1 \\
 & D_1,O_1,C_1,M_1,\boxed{D_2,O_2,C_2,M_2}; & \}\overline{\mathcal{I}}_2 \\
 & L_1,P_1,Y_1; & \}\overline{\mathcal{I}}_3 \\
 & \boxed{D_3,O_3,C_3,M_3},D_4,O_4,C_4,M_4; & \}\overline{\mathcal{I}}_4 \\
 & L_4,P_4,Y_4; & \}\overline{\mathcal{I}}_5 \\
 & N,R,P_1,Y_1,P_4,Y_4) & \}\overline{\mathcal{V}}
\end{array}$$

The head of Q_7 is unchanged. The reader can verify that index-covering homomorphisms exist in both directions between Q'_6 and Q_7, implying $Q'_6 \doteq^\Sigma_{\texttt{bnbnb}} Q_7$ and therefore $Q_1 \equiv^\Sigma Q_2$.

5.2 Nested Inputs

Our results extend directly to databases containing collections of non-flat tuples. Consider database instance $\mathbb{D}$ of schema S containing collection R of tuples of sort $\langle \tau_1, \ldots, \tau_k \rangle$, as well as two COCQL queries Q_a, Q_b over S that reference R. Using a standard shredding of complex objects into flat relations [25], we can create a new database instance $\mathbb{D}'$ over flat relational schema S' and two new COCQL queries Q'_a, Q'_b over S' satisfying $(Q_a)^{\mathbb{D}} = (Q'_a)^{\mathbb{D}'}$ and $(Q_b)^{\mathbb{D}} = (Q'_b)^{\mathbb{D}'}$. As a consequence, $Q'_a \equiv Q'_b \implies Q_a \equiv Q_b$.

Not every instance $\mathbb{D}''$ of schema S' encodes a valid instance of schema S; for example, $\mathbb{D}''$ could encode duplicate elements within collection R when R is supposed to be a set. However, we can show that if $\mathbb{D}''$ is a counter-example proving $Q'_a \not\equiv Q'_b$, then there exists another instance of S' that is both a counter-example and encodes a valid instance of schema S. As a consequence, $Q_a \equiv Q_b \implies Q'_a \equiv Q'_b$.

5.3 Adding the Unnest Operator

Suppose that the algebraic sub-language of COCQL is extended with an unnest operator $\coprod^{Y \to \overline{Z}}(E)$ which flattens aggregated objects previously constructed by a generalized projection operator of the form $\prod_{\overline{X}}^{Y=f(\overline{Z}')}(E)$. Syntactically, we require $\overline{Z}$ to be a tuple of fresh attribute names satisfying $|\overline{Z}| = |\overline{Z}'|$.

Within the set-based nested-relational algebra, *unnest* is the right inverse of *nest* (but not vice versa) [1]; however, this is not the case when mixed collection types are considered. The aggregation functions SET and NBAG do not preserve information about absolute cardinality when constructing objects. This means that operators of the form $\prod_{\overline{X}}^{Y=f(\overline{Z}')}(E)$ with $f \in \{\text{SET}, \text{NBAG}\}$ do not, in general, have a right inverse under bag-set semantics (which is required for the algebraic sub-language of COCQL in order to allow construction of bag objects).

We can use this phenomenon to show that the unnest operator adds expressive power to COCQL. The duplicate-eliminating projection operator within COCQL is restricted to only allow atomic sorts within the grouping list $\overline{X}$. By using set construction followed by unnesting, we can effect duplicate-eliminating projection even when $\overline{X}$ contains attributes with complex sorts, as follows.

$$\prod_{\overline{X}}(E) \equiv \coprod^{Y \to \overline{Z}}(\prod_{\varnothing}^{Y=\text{SET}(\overline{X})}(E)) \qquad (6)$$

Of course, this does not prove that adding unnest necessarily makes the equivalence problem harder. It may be possible to adapt our reduction of equivalence to encoding equivalence of CQs. However, the construction of encoding query $\text{EncQ}(Q)$ and the subsequent identifying of "core indexes" needs to depend not only on the output sort, but also somehow on the transient intermediate sorts. Our investigation into this extension is still in the preliminary stages at this time.

6. CONCLUSIONS

Optimization of complex queries is a problem of very practical importance. Our work is the first to consider the general query equivalence problem for a language allowing both nesting and a mixture of collection types. In so doing, we generalize previous work on (un-nested) CQs under various semantics. We also generalize previous work on queries that construct nested sets. In contrast to the previous approach of adapting techniques for nested containment to the equivalence problem, our direct consideration of query equivalence yields a much simpler condition, which is crucial for extending it to mixed collection types. The normal form that we propose for encoding queries illumines the the possible interactions between nested components, and hence lays a foundation for understanding and thereby optimizing nested aggregation.

The problem we consider in this paper has many extensions that deserve future attention. The most obvious are standard extensions to CQs such as allowing (atomic) inequalities or some form of disjunction. Equivalence for queries that can construct empty objects is extremely interesting, as it is required to model *scalar aggregation* within SQL, although this is known to make the equivalence problem undecidable when the query language also contains disjunction [38]. Allowing higher-order comparisons—either explicitly within predicates or implicitly by grouping on aggregated values—has very practical significance, since these comparisons are very common in decision-support queries. This extension could also quickly lead to undecidability, but using uninterpreted aggregation values rather than identifiable collection values might allow a decidable fragment. Finally, it would be valuable to synthesize our work on nesting with more sophisticated models of aggregation functions.

Acknowledgements.

Thanks to my supervisor Frank Wm. Tompa for many helpful discussions during the course of this research, and to the anonymous reviewers for their thoughtful suggestions. Support for this work was provided in part by the University of Waterloo and the Natural Sciences and Engineering Research Council of Canada.

7. REFERENCES

[1] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[2] Vasilis Aggelis and Stavros Cosmadakis. Optimization of nested SQL queries by tableau equivalence. In *Proc. 7th DBPL*, volume 1949 of *LNCS*. Springer, 2000.

[3] Nicole Bidoit. The verso algebra or how to answer queries with fewer joins. *J. Comput. Syst. Sci.*, 35(3):321–364, 1987.

[4] Peter Buneman, Shamim A. Naqvi, Val Tannen, and Limsoon Wong. Principles of programming with complex objects and collection types. *Theor. Comput. Sci.*, 149(1):3–48, 1995.

[5] Ashok K. Chandra and Philip M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *Proc. 9th ACM STOC*, pages 77–90, 1977.

[6] Surajit Chaudhuri and Moshe Y. Vardi. Optimization of real conjuctive queries. In *Proc. 12th ACM PODS*, pages 59–70, 1993.

[7] Sara Cohen. Equivalence of queries combining set and bag-set semantics. In *Proc. 25th ACM PODS*, pages 70–79, 2006.

[8] Sara Cohen, Werner Nutt, and Yehoshua Sagiv. Deciding equivalences among conjunctive aggregate queries. *J. ACM*, 54(2):5, 2007.

[9] Sara Cohen, Yehoshua Sagiv, and Werner Nutt.

Equivalences among aggregate queries with negation. *ACM Trans. Comput. Logic*, 6(2):328–360, 2005.

[10] Umeshwar Dayal. Of nests and trees: A unified approach to processing queries that contain nested subqueries, aggregates, and quantifiers. In *Proc. 13th VLDB*, pages 197–208, 1987.

[11] David DeHaan. Equivalence of nested queries with mixed semantics. In *Proc. 28th ACM PODS*, 2009.

[12] David DeHaan, Per-Åke Larson, and Jingren Zhou. Stacked indexed views in Microsoft SQL Server. In *Proc. ACM SIGMOD*, pages 179–190, 2005.

[13] Xin Dong, Alon Y. Halevy, and Igor Tatarinov. Containment of nested XML queries. In *Proc. 30th VLDB*, pages 132–143, 2004.

[14] Hector Garcia-Molina, Jeffrey D. Ullman, and Jennifer Widom. *Database Systems: The Complete Book*. Prentice Hall, 2002.

[15] Stëphane Grumbach, Maurizio Rafanelli, and Leonardo Tininini. Querying aggregate data. In *Proc. 18th ACM PODS*, pages 174–184, 1999.

[16] Ashish Gupta, Venky Harinarayan, and Dallan Quass. Aggregate-query processing in data warehousing environments. In *Proc. 21st VLDB*, pages 358–369, 1995.

[17] Ashish Gupta, Yehoshua Sagiv, Jeffrey D. Ullman, and Jennifer Widom. Constraint checking with partial information. In *Proc. 13th ACM PODS*, pages 45–55, 1994.

[18] Yannis E. Ioannidis and Raghu Ramakrishnan. Containment of conjunctive queries: Beyond relations as sets. *ACM Trans. Database Syst.*, 20(3):288–324, 1995.

[19] D. S. Johnson and A. Klug. Testing containment of conjunctive queries under functional and inclusion dependencies. *J. Comput. Syst. Sci.*, 28(1):167–189, 1984.

[20] Won Kim. On optimizing an SQL-like nested query. *ACM Trans. Database Syst.*, 7(3):443–469, 1982.

[21] Anthony Klug. On conjunctive queries containing inequalities. *J. ACM*, 35(1):146–160, 1988.

[22] Alon Y. Levy, Alberto O. Mendelzon, Yehoshua Sagiv, and Divesh Srivastava. Answering queries using views (extended abstract). In *Proc. 14th ACM PODS*, pages 95–104, 1995.

[23] Alon Y. Levy and Inderpal Singh Mumick. Reasoning with aggregation constraints. In *Proc. 5th EDBT*, volume 1057 of *LNCS*, pages 514–534. Springer, 1996.

[24] Alon Y. Levy, Inderpal Singh Mumick, and Yehoshua Sagiv. Query optimization by predicate move-around. In *Proc. 20th VLDB*, pages 96–107, 1994.

[25] Alon Y. Levy and Dan Suciu. Deciding containment for queries with complex objects (extended abstract). In *Proc. 16th ACM PODS*, pages 20–31, 1997.

[26] Leonid Libkin and Limsoon Wong. New techniques for studying set languages, bag languages and aggregate functions. In *Proc. 13th ACM PODS*, pages 155–166, 1994.

[27] Hong-Cheu Liu and Jeffery X. Yu. Algebraic equivalences of nested relational operators. *Inf. Syst.*, 30(3):167–204, 2005.

[28] Sergey Melnik, Atul Adya, and Philip A. Bernstein. Compiling mappings to bridge applications and databases. *ACM Trans. Database Syst.*, 33(4):1–50, 2008.

[29] Inderpal Singh Mumick, Sheldon J. Finkelstein, Hamid Pirahesh, and Raghu Ramakrishnan. Magic conditions. In *Proc. 9th ACM PODS*, pages 314–330, 1990.

[30] G. Ozsoyoglu, Z. M. Ozsoyoglu, and V. Matos. Extending relational algebra and relational calculus with set-valued attributes and aggregate functions. *ACM Trans. Database Syst.*, 12(4):566–592, 1987.

[31] Shariq Rizvi, Alberto Mendelzon, S. Sudarshan, and Prasan Roy. Extending query rewriting techniques for fine-grained access control. In *Proc. ACM SIGMOD*, pages 551–562, 2004.

[32] Kenneth A. Ross, Divesh Srivastava, Peter J. Stuckey, and S. Sudarshan. Foundations of aggregation constraints. *Theor. Comput. Sci.*, 193(1-2):149–179, February 1998.

[33] Yehoshua Sagiv and Mihalis Yannakakis. Equivalences among relational expressions with the union and difference operators. *J. ACM*, 27(4):633–655, 1980.

[34] Marc H. Scholl. Theoretical foundation of algebraic optimization utilizing unnormalized relations. In *Proc. ICDT*, volume 243 of *LNCS*, pages 380–396. Springer, 1986.

[35] Divesh Srivastava, Shaul Dar, H. V. Jagadish, and Alon Y. Levy. Answering queries with aggregation using views. In *Proc. 22nd VLDB*, pages 318–329, 1996.

[36] Stan J. Thomas and Patrick C. Fischer. Nested relational structures. In Paris C. Kanellakis, editor, *Advances in Computing Research: The Theory of Databases*, pages 269–307. JAI Press, 1986.

[37] Frank Wm. Tompa and José A. Blakeley. Maintaining materialized views without accessing base data. *Inf. Syst.*, 13(4):393–406, 1988.

[38] Jan Van den Bussche, Dirk Van Gucht, and Stijn Vansummeren. Well-definedness and semantic type-checking for the nested relational calculus. *Theor. Comput. Sci.*, 371(3):183–199, March 2007.

[39] R. van der Meyden. The complexity of querying indefinite data about linearly ordered domains. In *Proc. 11th ACM PODS*, pages 331–345, 1992.

[40] Weipeng P. Yan and Per-Åke Larson. Performing group-by before join. In *Proc. 10th ICDE*, pages 89–100, 1994.

[41] Markos Zaharioudakis, Roberta Cochrane, George Lapis, Hamid Pirahesh, and Monica Urata. Answering complex SQL queries using automatic summary tables. In *Proc. ACM SIGMOD*, pages 105–116, 2000.

APPENDIX

A. CHAIN OBJECTS

Algorithm 1 computes the transformation from a complete or trivial object o to the corresponding chain object $\textsc{Chain}(o)$. Because any sort τ can be interpreted as a complete object (conforming to itself), Algorithm 1 can also be used to compute the chain sort $\textsc{Chain}(\tau)$; however, the method described in Section 2.1 for computing $\textsc{Chain}(\tau)$ is much simpler and logically equivalent for the special case of sorts. The reader can verify that applying Algorithm 1 to sort τ_1 in Figure 3 yields the sort $\textsc{Chain}(\tau_1)$ also shown in Figure 3.

Algorithm 1 Transforming objects into chains

```
Chain(o)
    ▷ Input: complete or trivial object o
    ▷ Output: chain object formed from o
if o is atomic
    then return ⟨ o ⟩
elseif o = { o_1, ..., o_n }
    then return { Chain(o_1), ..., Chain(o_n) }
elseif o = {| o_1, ..., o_n |}
    then return {| Chain(o_1), ..., Chain(o_n) |}
elseif o = {{| o_1, ..., o_n |}}
    then return {{| Chain(o_1), ..., Chain(o_n) |}}
elseif o = ⟨ ⟩
    then return o
elseif o = ⟨ o_1 ⟩
    then return Chain(o_1)
elseif o = ⟨ o_1, ..., o_n ⟩ and n > 1
    then return Distribute(Chain(o_1),
                           Chain(⟨ o_2, ..., o_n ⟩))

Distribute(o_a, o_b)
    ▷ Input: chain object o_a of sort (§̄^a, k)
             Assume that o_a is a tree whose m leaves are
             the k-ary tuples ⟨ a^1_1, ... a^1_k ⟩, ..., ⟨ a^m_1, ... a^m_k ⟩
    ▷ Input: chain object o_b of sort (§̄^b, l)
             Assume that o_b is a tree whose n leaves are
             the l-ary tuples ⟨ b^1_1, ... b^1_l ⟩, ..., ⟨ b^n_1, ... b^n_l ⟩
    ▷ Output: chain object of sort (§̄^a ∘ §̄^b, k + l)
             formed by distributing o_b over each leaf of o_a
             and pushing down atomic values
o ← copy of o_a
foreach i ∈ [1, m]
    do o^i ← copy of o_b
       foreach j ∈ [1, n]
           do substitute tuple ⟨ a^i_1, ..., a^i_k, b^j_1, ..., b^j_l ⟩
                 for tuple ⟨ b^j_1, ..., b^j_l ⟩ within o^i
       substitute o^i for tuple ⟨ a^i_1, ..., a^i_k ⟩ within o
return o
```

B. ENCODING EQUALITY REVISITED

In this section we provide a characterization of encoding equality that avoids the need to evaluate decoding queries. Consider Example 7 where we claimed that $R_1 \doteq_{\mathtt{ns}} R_2$. Verifying $\textsc{Decode}(R_1, \mathtt{ns}) = \textsc{Decode}(R_2, \mathtt{ns})$ involves two basic steps: (1) evaluate the two decoding queries, and (2) recursively compare the two constructed objects to verify that they are isomorphic. The first step implicitly performs a mapping of index values to sub-objects, while the second step explicitly maps between sub-objects. We now define a *certificate* that embodies the mappings necessary to conclude encoding equivalence. Clearly, the allowable mappings from index values to sub-objects depends upon the semantics of the enclosing collection type, and so the space of possible certificates depends upon the decoding signature.

Given a signature $\overline{\S}$ and two non-empty encoding relations $R(\overline{\mathcal{I}}_1; \ldots; \overline{\mathcal{I}}_d; \overline{\mathcal{V}})$ and $R'(\overline{\mathcal{I}}'_1; \ldots; \overline{\mathcal{I}}'_d; \overline{\mathcal{V}}')$ with depth $d = |\overline{\S}|$, we define a $\overline{\S}$-*certificate between R and R'* as a tree rooted by a *set node* if $\S_1 = \mathtt{s}$, a *bag node* if $\S_1 = \mathtt{b}$, a *normalized bag node* if $\S_1 = \mathtt{n}$, or a *tuple node* if $\overline{\S} = \varnothing$.

A *set node* $n^{\mathtt{s}}_{(R,R')}$ proves $R \doteq_{\mathtt{s}\overline{Y}} R'$ (for some signature $\overline{Y}$). It contains a function $f : \mathtt{adom}(\overline{\mathcal{I}}'_1, R') \to \mathtt{adom}(\overline{\mathcal{I}}_1, R)$ satisfying

$$\forall \overline{x}' \in \mathtt{adom}(\overline{\mathcal{I}}'_1, R') \left(R[f(\overline{x}')] \doteq_{\overline{Y}} R'[\overline{x}']\right) \qquad (7)$$

and an analogous function $f' : \mathtt{adom}(\overline{\mathcal{I}}_1, R) \to \mathtt{adom}(\overline{\mathcal{I}}'_1, R')$. For each pair $(\overline{x}, \overline{x}')$ such that either $\overline{x}' = f'(\overline{x})$ or $\overline{x} = f(\overline{x}')$, node $n^{\mathtt{s}}_{(R,R')}$ has a child $\overline{Y}$-certificate between $R[\overline{x}]$ and $R'[\overline{x}']$.

A *bag node* $n^{\mathtt{b}}_{(R,R')}$ proves $R \doteq_{\mathtt{b}\overline{Y}} R'$. It contains a *bijective* function $f : \mathtt{adom}(\overline{\mathcal{I}}'_1, R') \to \mathtt{adom}(\overline{\mathcal{I}}_1, R)$ satisfying the following equation.

$$\forall \overline{x}' \in \mathtt{adom}(\overline{\mathcal{I}}'_1, R') \left(R[f(\overline{x}')] \doteq_{\overline{Y}} R'[\overline{x}']\right) \qquad (8)$$

For each pair $(\overline{x}, \overline{x}')$ such that $\overline{x} = f(\overline{x}')$, node $n^{\mathtt{b}}_{(R,R')}$ has a child $\overline{Y}$-certificate between $R[\overline{x}]$ and $R'[\overline{x}']$.

A *normalized bag node* $n^{\mathtt{n}}_{(R,R')}$ proves $R \doteq_{\mathtt{n}\overline{Y}} R'$. It contains two *finite* domains D_1 and D_2, and two *surjective* functions $\rho : \mathtt{adom}(\overline{\mathcal{I}}_1, R) \to D_1$ and $\varrho : \mathtt{adom}(\overline{\mathcal{I}}'_1, R') \to D_2$ that satisfy the following equation.

$$\forall p \in D_1. \forall q \in D_2 \left[(\sigma_{\rho(\overline{\mathcal{I}}_1)=p}(R)) \doteq_{\mathtt{b}\overline{Y}} (\sigma_{\varrho(\overline{\mathcal{I}}'_1)=q}(R'))\right] \qquad (9)$$

For each pair $(p, q) \in D_1 \times D_2$, node $n^{\mathtt{n}}_{(R,R')}$ has a child $\mathtt{b}\overline{Y}$-certificate between $\sigma_{\rho(\overline{\mathcal{I}}_1)=p}(R)$ and $\sigma_{\varrho(\overline{\mathcal{I}}'_1)=q}(R')$.

A *tuple node* $n^{\mathtt{t}}_{(R,R')}$ proves $R \doteq_{\varnothing} R'$. A non-empty encoding relation of depth zero contains precisely one tuple (of only output values). Therefore, node $n^{\mathtt{t}}_{(R,R')}$ contains a single comparison of tuples.

Theorem 5 *Given a signature $\overline{\S}$ and two encoding relations R and R' of depth $|\overline{\S}|$, R and R' are $\overline{\S}$-equal iff there exists a $\overline{\S}$-certificate between R and R'.*

Proof. A simple induction on certificate height suffices. The base case is the tuple nodes, which are trivial. For the inductive case, it suffices to verify that each collection node correctly enforces the semantics of the appropriate collection constructor. For all of the collection nodes, equality of compared sub-objects follows by induction on the child certificates. For a set node, the two functions f and f' enforce mutual containment of the two sets of sub-objects, which is necessary and sufficient to conclude set equality. For a bag node, the bijective function enforces isomorphism of the two collections of sub-objects, which is necessary and sufficient to conclude bag equality. Finally, for a normalized bag node the functions ρ

and ϱ partition relations R and R', respectively, while the child $\mathtt{b}\overline{Y}$-certificates enforce that all of the partitions encode the same bag. This is necessary and sufficient to conclude normalized bag equality (the ratio $\frac{|D_1|}{|D_2|}$ captures the relative "inflation factors" of the two original bags). □

Figure 10 illustrates an ns-certificate proving $R_1 \doteq_{\mathtt{ns}} R_2$ with R_1 and R_2 shown in Figures 6 and 7, respectively.

C. PROOFS

C.1 Proof of Theorem 1

It is straightforward to verify that the transformation $\textsc{Chain}(o)$ shown in Algorithm 1 (Appendix A) is invertible, and so $o = o'$ iff $\textsc{Chain}(o) = \textsc{Chain}(o')$. Then, for any $\mathbb{D}$, object $(Q)^{\mathbb{D}} = (Q')^{\mathbb{D}}$ iff $\textsc{Chain}((Q)^{\mathbb{D}}) = \textsc{Chain}((Q')^{\mathbb{D}})$ iff (by Proposition 1)

$$\textsc{Decode}((\textsc{EncQ}(Q))^{\mathbb{D}}, \overline{\S}) = \textsc{Decode}((\textsc{EncQ}(Q'))^{\mathbb{D}}, \overline{\S})$$

and so the theorem follows immediately from Definitions 1 and 2.

Proposition 1 can be proven by a straightforward (but tedious) comparison of Algorithm 1 for constructing $\textsc{Chain}(o)$ with the algorithm in Section 3.2 for constructing $\textsc{EncQ}(Q)$. The crucial point is that the manner in which $\textsc{EncQ}(Q)$ chooses the index variables (via a preorder traversal of τ) emulates the behaviour of line 14 in Algorithm 1.

C.2 Uniqueness of $\overline{\mathcal{I}}_i^{\S}$

We prove here that the conditions given in Section 4.1 always determine a unique minimal set of core indexes $\mathcal{I}_i^{\overline{\S}} \subseteq \mathcal{I}_i$. First, let $Q'_i(\mathcal{I}_{[1,i]}\mathcal{I}^{\overline{\S}}_{[i+1,d]})$ be a minimal CQ equivalent to Q_i. Next, let a "candidate for $\mathcal{I}_i^{\overline{\S}}$" denote any set $X \subseteq \mathcal{I}_i$ such that if we choose $\mathcal{I}_i^{\overline{\S}} := X$ then all the conditions in Section 4.1 are satisfied. We now show that if $X_1, X_2 \subseteq \mathcal{I}_i$ are both candidates for $\mathcal{I}_i^{\overline{\S}}$, then $X_1 \cap X_2$ is also a candidate for $\mathcal{I}_i^{\overline{\S}}$.

Case $\S_i = \mathtt{b}$**:**
$\mathcal{I}_i \subseteq \mathcal{I}_i^{\overline{\S}}$ trivially implies $X_1 = X_2 = X_1 \cap X_2$.

Case $\S_i = \mathtt{s}$**:**
$\mathcal{I}_i \cap \mathcal{V} \subseteq X_1$ and $\mathcal{I}_i \cap \mathcal{V} \subseteq X_2$ trivially implies $\mathcal{I}_i \cap \mathcal{V} \subseteq (X_1 \cap X_2)$. Therefore, we need to prove the following MVD

$$Q_i \models \mathcal{I}_{[1,i-1]} \cup (X_1 \cap X_2) \twoheadrightarrow \mathcal{I}^{\overline{\S}}_{[i+1,d]} \quad (10)$$

which cannot be derived from axioms for MVDs [1], but can be reasoned from the query structure as follows.

1. By candidacy of X_1,
$$Q_i \models (\mathcal{I}_{[1,i-1]} \cup X_1) \twoheadrightarrow \mathcal{I}^{\overline{\S}}_{[i+1,d]}$$
and so by Lemma 1 $\mathcal{I}_{[1,i-1]} \cup X_1$ is a strong $(\mathcal{I}^{\overline{\S}}_{[i+1,d]}, (\mathcal{I}_i \setminus X_1))$-articulation set of $H^{Q'}$.
2. By candidacy of X_2,
$$Q_i \models \mathcal{I}_{[1,i-1]} \cup X_2 \twoheadrightarrow \mathcal{I}^{\overline{\S}}_{[i+1,d]}$$
and so by Lemma 1 $\mathcal{I}_{[1,i-1]} \cup X_2$ is a strong $(\mathcal{I}^{\overline{\S}}_{[i+1,d]}, (\mathcal{I}_i \setminus X_2))$-articulation set of $H^{Q'}$.
3. The two articulation sets together imply that deleting $\mathcal{I}_{[1,i-1]} \cup (X_1 \cap X_2)$ from $H^{Q'}$ causes the two sets $\mathcal{I}^{\overline{\S}}_{[i+1,d]}$ and $\mathcal{I}_i \setminus (X_1 \cap X_2)$ to occur in separate partitions of the remaining hypergraph. Equation 10 then follows from Lemma 1.

Case $\S_i = \mathtt{n}$**:**
$\mathcal{I}_i \cap \mathcal{V} \subseteq (X_1 \cap X_2)$ is the same as case $\S_i = \mathtt{s}$, so we need to prove the following MVD

$$Q_i \models \mathcal{I}_{[1,i-1]} \twoheadrightarrow (X_1 \cap X_2) \cup \mathcal{I}^{\overline{\S}}_{[i+1,d]} \quad (11)$$

which again cannot be derived from axioms for MVDs, but can be reasoned from the query structure.

1. By candidacy of X_1,
$$Q_i \models \mathcal{I}_{[1,i-1]} \twoheadrightarrow X_1 \cup \mathcal{I}^{\overline{\S}}_{[i+1,d]}$$
and so by Lemma 1 $\mathcal{I}_{[1,i-1]}$ is a strong $((X_1 \cup \mathcal{I}^{\overline{\S}}_{[i+1,d]}), (\mathcal{I}_i \setminus X_1))$-articulation set of $H^{Q'}$.
2. By candidacy of X_2,
$$Q_i \models \mathcal{I}_{[1,i-1]} \twoheadrightarrow X_2 \cup \mathcal{I}^{\overline{\S}}_{[i+1,d]}$$
and so by Lemma 1 $\mathcal{I}_{[1,i-1]}$ is a strong $((X_2 \cup \mathcal{I}^{\overline{\S}}_{[i+1,d]}), (\mathcal{I}_i \setminus X_2))$-articulation set of $H^{Q'}$.
3. The two articulation sets together imply that deleting $\mathcal{I}_{[1,i-1]}$ from $H^{Q'}$ causes the four sets $X_1 \setminus X_2$, $X_2 \setminus X_1$, $\mathcal{I}_i \setminus (X_1 \cup X_2)$, and $(X_1 \cap X_2) \cup \mathcal{I}^{\overline{\S}}_{[i+1,d]}$ to occur in separate partitions of the remaining hypergraph. Equation 11 then follows from Lemma 1.

C.3 Proof of Theorem 2

We will first prove that testing query-implied MVDs is NP-hard, by reduction from the NP-hard problem of deciding containment between boolean CQs. Let Q_a and Q_b be two boolean CQs whose bodies contain the disjoint sets of variables $\mathcal{B}_a$ and $\mathcal{B}_b$, respectively. Let A, Z be two fresh variables. Let $Q(\overline{\mathcal{V}})$ be a new conjunctive query whose output variables satisfy $\mathcal{V} = \mathcal{B}_a \cup \{A, Z\}$, and whose body is defined as follows.

$$\mathtt{body}_Q = \mathtt{body}_{Q_a} \cup \mathtt{body}_{Q_b} \cup \bigcup_{x \in \mathcal{B}_a \cup \mathcal{B}_b} \{R(A,x), R(x,Z)\}$$

Then, $Q_a \subseteq Q_b$ iff there exists a homomorphism $h : \mathcal{B}_b \to \mathcal{B}_a$ such that $h(\mathtt{body}_{Q_b}) \subseteq \mathtt{body}_{Q_a}$ iff Q implies $\mathcal{B}_a \twoheadrightarrow A$ (and $\mathcal{B}_a \twoheadrightarrow Z$). NP-hardness of $\overline{\S}$-normalization then follows directly from the definition of $\overline{\S}$-NF.

Identifying the core indexes at each level can be done in NP time using an algorithm that traverses query hypergraphs.

Case $\S_i = \mathtt{b}$**:** Trivial.

Case $\S_i = \mathtt{n}$**:** Minimize the body of Q_i, then construct hypergraph H^{Q_i}. Delete from H^{Q_i} all nodes corresponding to variables in the set $\mathcal{I}_{[1,i-1]}$. Identify $\mathcal{I}_i^{\overline{\S}}$ by traversing the connected components containing any variable in $(\mathcal{I}_i \cap \mathcal{V}) \cup \mathcal{I}_{[i+1,d]}$.

Case $\S_i = \mathtt{s}$**:** Minimize the body of Q', then construct hypergraph H^{Q_i}. Delete from H^{Q_i} all nodes corresponding to variables in the set $\mathcal{I}_{[1,i-1]} \cup (\mathcal{I}_i \cap \mathcal{V})$. Identify any non-output members of $\mathcal{I}_i^{\overline{\S}}$ incrementally by traversing the connected components containing $\mathcal{I}_{[i+1,d]}$ and deleting the "nearest" member of $\mathcal{I}_i$.

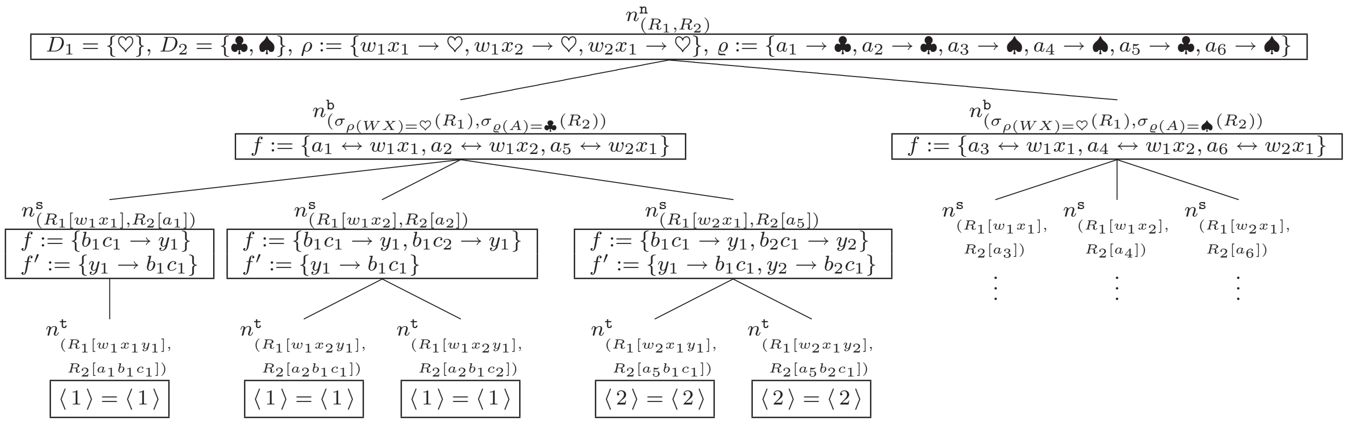


Figure 10: ns-Certificate proving $R_1 \doteq_{\mathtt{ns}} R_2$

C.4 Proof of Theorem 3

Let $Q(\overline{\mathcal{I}}_1;\ldots;\overline{\mathcal{I}}_d;\overline{\mathcal{V}})$ be any CEQ. For every $i \in [1, d+1]$, let Q^i denote the following CEQ.

$$Q^i(\overline{\mathcal{I}}_1;\ldots;\overline{\mathcal{I}}_{i-1};\overline{\mathcal{I}}_i^{\overline{\S}};\ldots;\overline{\mathcal{I}}_d^{\overline{\S}};\overline{\mathcal{V}}) :- \ \mathtt{body}_Q \tag{12}$$

Observe that Q^1 is the $\overline{\S}$-normal form of Q, which we prove $\overline{\S}$-equivalent to Q using induction on i. As a base case, Equation 12 implies $Q^{d+1} = Q$, and so $Q^{d+1} \doteqdot_{\overline{\S}} Q$ is trivial.

For the inductive step we need to show that for any database $\mathbb{D}$ that we can construct a $\overline{\S}$-certificate between $(Q^i)^{\mathbb{D}}$ and $(Q^{i+1})^{\mathbb{D}}$. W.l.o.g., assume that $\overline{\mathcal{I}}_i = \overline{\mathcal{I}}_i^{\overline{\S}} \cdot \overline{\mathcal{J}}$, where $\mathcal{J}$ is the set of redundant indexes in $\mathcal{I}_i$. Because Q^i and Q^{i+1} have the same body, relation $R^i := (Q^i)^{\mathbb{D}}$ is formed by a projection over $R^{i+1} := (Q^{i+1})^{\mathbb{D}}$ that retains all attributes except for $\mathcal{J}$.

Case $\S_i = \mathtt{b}$:
$\mathcal{J} = \varnothing$, and so $R^i \doteq_{\overline{\S}} R^{i+1}$ is trivial.

Case $\S_i = \mathtt{s}$:
For each value $\overline{a} \in \mathtt{adom}(\overline{\mathcal{I}}_{[1,i-1]}, R^i)$, let $C_{\overline{a}}$ be a $\overline{\S}_{[i,d]}$-certificate rooted by an initially empty set node. We will incrementally construct $C_{\overline{a}}$ until it proves the relationship $R^i[\overline{a}] \doteq_{\overline{\S}_{[i,d]}} R^{i+1}[\overline{a}]$. Because $\mathtt{adom}(\overline{\mathcal{I}}_{[1,i-1]}, R^i) = \mathtt{adom}(\overline{\mathcal{I}}_{[1,i-1]}, R^{i+1})$, it is then trivial to construct the upper levels of a $\overline{\S}$-certificate proving $R^i \doteq_{\overline{\S}} R^{i+1}$.

For each value $\overline{b} \in \mathtt{adom}(\overline{\mathcal{I}}_i^{\overline{\S}}, R^i[\overline{a}]) = \mathtt{adom}(\overline{\mathcal{I}}_i^{\overline{\S}}, R^{i+1}[\overline{a}])$, the sub-relation $R^i[\overline{a}\overline{b}]$ encodes an object of sort $(\overline{\S}_{[i+1,d]}, d-i)$ occurring in the set at level i. By definition of $\mathcal{I}_i^{\overline{\S}}$, relation R^i satisfies $\mathcal{I}_{[1,i-1]} \cup \mathcal{I}_i^{\overline{\S}} \twoheadrightarrow \mathcal{I}_{[i+1,d]}^{\overline{\S}}$, which implies that $R^{i+1}[\overline{a}]$ satisfies $\mathcal{I}_i^{\overline{\S}} \twoheadrightarrow \mathcal{I}_{[i+1,d]}^{\overline{\S}}$ (and $\mathcal{I}_i^{\overline{\S}} \twoheadrightarrow \mathcal{J}$). Let $\{\overline{c}_1, \ldots, \overline{c}_k\}$ be all of the values in set $\mathtt{adom}(\overline{\mathcal{J}}, R^{i+1}[\overline{a}])$. It follows from the MVD that all of the sub-relations $R^{i+1}[\overline{a}\overline{b}\overline{c}_1], \ldots, R^{i+1}[\overline{a}\overline{b}\overline{c}_k]$ are identical to each other and to the sub-relation $R^i[\overline{a}\overline{b}]$. For each $\overline{c}_j$ add to $C_{\overline{a}}$ the mapping $f(\overline{b}\overline{c}_j) := \overline{b}$, as well as a child $\overline{\S}_{[i+1,d]}$-certificate proving that $R^i[\overline{a}\overline{b}] \doteq_{\overline{\S}_{[i+1,d]}} R^{i+1}[\overline{a}\overline{b}\overline{c}_j]$ (which is trivial, because $R^i[\overline{a}\overline{b}] = R^{i+1}[\overline{a}\overline{b}\overline{c}_j]$). Then, add the mapping $f'(\overline{b}) := \overline{b} \cdot \overline{c}_1$ (we could choose any $\overline{c}_j$). Certificate $C_{\overline{a}}$ is complete when this has been performed for all values of $\overline{b}$.

Case $\S_i = \mathtt{n}$:
The proof is almost identical to case $\S_i = \mathtt{s}$, but uses the MVD $\mathcal{I}_{[1,i-1]} \twoheadrightarrow \mathcal{I}_{[i,d]}^{\overline{\S}}$ both to guarantee that the contents of inner encoding relations are identical (as in the case $\S_i = \mathtt{s}$), and to guarantee that for each value of $\overline{a} \in \mathtt{adom}(\overline{\mathcal{I}}_{[1,i-1]}, R^i) = \mathtt{adom}(\overline{\mathcal{I}}_{[1,i-1]}, R^{i+1})$ the multiplicative factor introduced by $\mathcal{J}$ is uniform across all $\overline{b} \in \mathtt{adom}(\overline{\mathcal{I}}_i^{\overline{\S}}, R^i[\overline{a}]) = \mathtt{adom}(\overline{\mathcal{I}}_i^{\overline{\S}}, R^{i+1}[\overline{a}])$.

C.5 Proof of Theorem 4

Assume without loss of generality that $Q(\overline{\mathcal{I}}_1;\ldots;\overline{\mathcal{I}}_d;\overline{\mathcal{V}})$ and $Q'(\overline{\mathcal{I}}'_1;\ldots;\overline{\mathcal{I}}'_d;\overline{\mathcal{V}}')$ are already in $\overline{\S}$-normal form (justified by Theorem 3). Assume also that the query bodies $\mathtt{body}_Q$ and $\mathtt{body}_{Q'}$ are minimal relative to the set of index and output attributes occurring in the query heads (in the sense of tableau minimization; justified by CQ equivalence).

If index-covering homomorphisms exist in both directions, then for any database $\mathbb{D}$ the encoding relations $(Q)^{\mathbb{D}}$ and $(Q')^{\mathbb{D}}$ differ at most by ordering of attributes within each index level. A $\overline{\S}$-certificate proving $(Q)^{\mathbb{D}} \doteq_{\overline{\S}} (Q')^{\mathbb{D}}$ is therefore straightforward to construct, since each node is simply an isomorphism between sub-relations modulo reordering of intra-index attributes; $Q \doteqdot_{\overline{\S}} Q'$ follows immediately.

The proof for the necessity of mutual index-covering homomorphisms is too long to reproduce in its entirety here, and so we include only an extended sketch. The full proof will appear within the author's Ph.D. dissertation (expected 2009).

The overall proof methodology for proving the existence of an index-covering homomorphisms follows the traditional proof for CQ equivalence.

1. Construct a canonical database $\mathbb{D}_Q$ from $\mathtt{body}_Q$.
2. Choose a particular embedding $\gamma : \mathtt{body}_Q \to \mathbb{D}_Q$ that yields a "canonical tuple" within $(Q)^{\mathbb{D}_Q}$.
3. Use the definition of encoding equivalence (specifically, the existence of a $\overline{\S}$-certificate between $(Q)^{\mathbb{D}_Q}$ and $(Q')^{\mathbb{D}_Q}$) to argue the existence of an embedding $\phi : \mathtt{body}_{Q'} \to \mathbb{D}_Q$ that yields a comparable tuple in $(Q')^{\mathbb{D}_Q}$.
4. Define mapping $h : Q' \to Q$ in terms of ϕ, and use both the definition of $\mathbb{D}_Q$ and the properties of the chosen canonical tuple to prove that h is an index-covering homomorphism from Q' to Q.
5. Repeat in the other direction using database $\mathbb{D}_{Q'}$.

The complication lies in the third step. An arbitrary certificate between $(Q)^{\mathbb{D}_Q}$ and $(Q')^{\mathbb{D}_Q}$ does not allow us to conclude that for each canonical tuple $\gamma(\overline{\mathcal{I}}_1;\ldots;\overline{\mathcal{I}}_d;\overline{\mathcal{V}})$ there exists an embedding ϕ satisfying

$$\gamma(\overline{\mathcal{I}}_1;\ldots;\overline{\mathcal{I}}_d;\overline{\mathcal{V}}) = \phi(\overline{\mathcal{I}}'_1;\ldots;\overline{\mathcal{I}}'_d;\overline{\mathcal{V}}')$$

because the mappings within set, bag, and normalized bag nodes do not require equality of index values. This makes it difficult in the fourth step to prove that the homomorphism h is index-covering.

Overcoming this requires construction of a very complex canonical database $\mathbb{D}_Q$ which depends heavily on the semantics of the nested collection types represented by $\overline{\S}$. The identification of certain tuples as "canonical" is then defined relative to the structure of $\mathbb{D}_Q$. Given an arbitrary $\overline{\S}$-certificate between $(Q)^{\mathbb{D}_Q}$ and $(Q')^{\mathbb{D}_Q}$, we use induction to show that the mappings in the certificate can be re-organized until for every canonical tuple $\gamma(\overline{\mathcal{I}}_1;\ldots;\overline{\mathcal{I}}_d;\overline{\mathcal{V}})$, there exists a path of nodes down the certificate such that at each level i, the certificate node maps $\gamma(\overline{\mathcal{I}}_i)$ to a permutation of itself. The induction starts at the leaves of the certificate and proceeds upward. Proving at each level that $\gamma(\overline{\mathcal{I}}_i)$ can be re-mapped to a permutation of itself requires exploiting the structure of $\mathbb{D}_Q$ which has been tailored for the type of mappings implied by the semantics of type $\S_i$. Unfortunately, for set nodes and normalized bag nodes, the argument based upon semantics of the mappings can only prove that $\gamma(\overline{\mathcal{I}}_i)$ maps that contains all of the value in $\gamma(\overline{\mathcal{I}}_i)$ (but could contain more values). For the induction hypothesis to be satisfied we require the stronger property that the mapping effects a permutation, in order to argue at the next level that the encoded sub-objects are equal. To deal with this, we actually need to perform the induction simultaneously in both directions (i.e., on one certificate proving $(Q)^{\mathbb{D}_Q} \doteq_{\overline{\S}} (Q')^{\mathbb{D}_Q}$ and simultaneously on another certificate proving $(Q)^{\mathbb{D}_{Q'}} \doteq_{\overline{\S}} (Q')^{\mathbb{D}_{Q'}}$) in order to establish that $|\overline{\mathcal{I}}_i| = |\overline{\mathcal{I}}'_i|$.

We will now describe the design of canonical database $\mathbb{D}_Q$. Because the construction combines three different techniques depending upon the type of certificate nodes at each level (i.e., depending upon $\overline{\S}$), we will illustrate the three techniques independently. In the full proof, both the formal definitions of the canonical database and the inductive arguments for the construction of the index-covering homomorphism are completely modular so that they can be interleaved to handle arbitrary encoding signatures.

C.5.1 Bag Nodes

The argument for bag nodes is an adaptation of Cohen et al.'s proof for equivalence of un-nested COUNT queries [8]. It relies upon an argument that if two multivariate polynomials of degree k over n variables are distinct, then there are an infinite number of points in $\mathbb{N}^n$ upon which they disagree.

Let $\mathcal{P}$ be an infinite palette of colours, each indexed by a positive integer:

$$\texttt{colour}_1\ \texttt{colour}_2\ \texttt{colour}_3\ \texttt{colour}_4 \ldots$$

Colour $\texttt{colour}_1$ is intentionally transparent.

Let $\mathcal{C}$ be any domain of n constants adhering to some arbitrary total ordering.

$$\mathcal{C} = \{\texttt{c}_1,\ldots,\texttt{c}_n\} \qquad \forall 1 \le i < j \le n : \texttt{c}_i < \texttt{c}_j$$

For each $\texttt{colour}_i \in \mathcal{P}$ satisfying $i \ge 2$, let $\mathcal{C}_i$ be a fresh set of constants isomorphic to $\mathcal{C}$, and let $\delta_i : \mathcal{C} \to \mathcal{C}_i$ be a function that "paints" each $\texttt{c}_j \in \mathcal{C}$ with colour $\texttt{colour}_i$ to yield the constant $\texttt{c}_j \in \mathcal{C}_i$. As implied by the transparency of $\texttt{colour}_1$, we define $\mathcal{C}_1 = \mathcal{C}$ and the painting function $\delta_1 : \mathcal{C} \to \mathcal{C}_1$ trivially as the identity function. Finally, because the different paintings of $\mathcal{C}$ are mutually disjoint, we define a single "whitewash" function δ^{-1} that is the inverse of all painting functions.

Given any point $\overline{r} \in \mathbb{N}^n$ and any tuple t over $\mathcal{C}$,

$$t = \langle\, \texttt{c}_{i_1}, \texttt{c}_{i_2}, \ldots, \texttt{c}_{i_m} \,\rangle$$

we define the *$\overline{r}$-inflation of t*, denoted $\Delta^{\overline{r}}(t)$, as the set of all possible "paintings" of t generated by independently choosing for each tuple component $\texttt{c}_{i_j}$ one of the first r_{i_j} colours in palette $\mathcal{P}$. The size $|\Delta^{\overline{r}}(t)|$ depends upon both $\overline{r}$ and the number of occurrences of each constant $\texttt{c}_i \in \mathcal{C}$ within the tuple. For a given tuple t, let $\#(t,\texttt{c}_i)$ denote the number of occurrences of $\texttt{c}_i$ within t. Then, the set $\Delta^{\overline{r}}(t)$ has size

$$|\Delta^{\overline{r}}(t)| = \prod_{\texttt{c}_i \in \mathcal{C}} r_i^{\#(t,\texttt{c}_i)} \tag{13}$$

which is a monomial over variables $r_1,\ldots,r_n$. with coefficient one and degree equal to the arity of t.

Given any set S of tuples over $\mathcal{C}$, we define $\Delta^{\overline{r}}(S) := \bigcup_{t\in S} \Delta^{\overline{r}}(t)$, and so

$$|\Delta^{\overline{r}}(S)| = f_S(\overline{r})$$

where f_S is a multivariate polynomial over variables $r_1,\ldots,r_n$ with degree equal to the maximum arity of tuples in S. Given any m sets $S_1,\ldots,S_m$ of tuples over $\mathcal{C}$ with maximum arity k, we show that there always exists a coordinate $\overline{r} \in \mathbb{N}^n$ such that for every $i,j \in [1,m]$,

$$f_{S_i}(\overline{r}) = f_{S_j}(\overline{r}) \iff f_{S_i} = f_{S_j} \iff S_i \approx S_j \tag{14}$$

where $S_i \approx S_j$ denotes that there exists a bijection between the tuples of S_i and S_j such that tuples are only mapped to permutations of themselves. When $S_1,\ldots,S_m$ includes *all possible* sets of tuples over $\mathcal{C}$ with maximum arity k, then we say that the coordinate $\overline{r}$ above is *k-distinguishing*.

We are now ready to define the canonical database $\mathbb{D}_Q$. Let $\mathcal{C}$ be the set of all constants and variables occurring in $\texttt{body}_Q$, and choose $\overline{r}$ to be any $(|\overline{\mathcal{I}}_{[1,d]}|+|\overline{\mathcal{I}}'_{[1,d]}|)$-distinguishing coordinate for $\mathcal{C}$. Then, define $\mathbb{D}_Q$ as follows.

$$\mathbb{D}_Q := \Delta^{\overline{r}}(\texttt{body}_Q)$$

Due to the "transparency" of $\texttt{colour}_1$, we guarantee that $\texttt{body}_Q \subseteq \mathbb{D}_Q$, and so $(Q)^{\mathbb{D}_Q}$ is guaranteed not to be empty.

Given any certificate between encoding relations $(Q)^{\mathbb{D}_Q}$ and $(Q')^{\mathbb{D}_Q}$, we consider each bag node at level i of the certificate. For each encoded sub-object o, we model the cardinality of o within the two encoding relations as the polynomials $f_{S_o}(\overline{r})$ and $f_{S'_o}(\overline{r})$. Set S_o is formed by taking the index values for $\overline{\mathcal{I}}_i$ that correspond to encodings of o, and restricting the tuples to non-output attributes; S'_o is analogous. Because bag equivalence entails $f_{S_o}(\overline{r}) = f_{S'_o}(\overline{r})$, we apply equation 14 to conclude that $S_o \approx S'_o$ (noting that f_{S_o} and $f_{S'_o}$ both have degree less than $(|\overline{\mathcal{I}}_{[1,d]}| + |\overline{\mathcal{I}}'_{[1,d]}|)$, and $\overline{r}$ was selected to be $(|\overline{\mathcal{I}}_{[1,d]}| + |\overline{\mathcal{I}}'_{[1,d]}|)$-distinguishing). It is easy to show that the mappings in the bag node must already agree on output attributes, and so we can re-arrange the mappings until the bijection maps each index value to a permutation of itself.

We can now choose any tuple $\gamma \in (Q)^{\mathbb{D}}$ satisfying

$$\delta^{-1} \circ \gamma(\overline{\mathcal{I}}_1; \dots; \overline{\mathcal{I}}_d; \overline{\mathcal{V}}) = \langle \overline{\mathcal{I}}_1; \dots; \overline{\mathcal{I}}_d; \overline{\mathcal{V}} \rangle$$

and by examining any path of nodes down the certificate tree leading to a tuple node containing γ, we can construct a homomorphism from Q' to Q that is guaranteed to be index-covering for all of the bag levels.

C.5.2 Normalized Bag Nodes

The proof is similar to bag nodes. The complicating factor is that normalized bag nodes do not enforce that a sub-object be encoded with the same absolute cardinality in both encoding relations. That is, for a given sub-object o we do not know that $f_{S_o}(\overline{r}) = f_{S'_o}(\overline{r})$, and hence we cannot apply equation 14 to conclude $S_o \approx S'_o$ as we did above for bag nodes.

Normalized bag nodes do enforce that two sub-objects be encoded with the same relative cardinalities. Therefore, given two sub-objects o_1, o_2,

$$\frac{f_{S_{o_1}}(\overline{r})}{f_{S_{o_2}}(\overline{r})} = \frac{f_{S'_{o_1}}(\overline{r})}{f_{S'_{o_2}}(\overline{r})}$$

and so

$$f_{S_{o_1}}(\overline{r}) \times f_{S'_{o_2}}(\overline{r}) = f_{S_{o_2}}(\overline{r}) \times f_{S'_{o_1}}(\overline{r})$$

follows. Because both sides of the second equation are polynomials of degree less than $(|\overline{\mathcal{I}}_{[1,d]}| + |\overline{\mathcal{I}}'_{[1,d]}|)$, we can apply equation 14 to conclude the following.

$$S_{o_1} \times S'_{o_2} \approx S_{o_2} \times S'_{o_1}$$

This is not necessarily useful, since it does not guarantee that every tuple in S_{o_1} has a permuted image in S'_{o_1}. However, by additional architecting of $\mathbb{D}_Q$ we produce particular "canonical sub-objects" for which we can show that the polynomials $f_{S_{o_1}}$ and $f_{S_{o_2}}$ have a greatest common divisor of degree zero (i.e. a constant). Using this information, we are able to show that either

- $S_{o_1} \approx S'_{o_1}$ and $S_{o_2} \approx S'_{o_2}$, or
- $\texttt{degree}(f_{S_{o_1}}) > \texttt{degree}(f_{S'_{o_1}})$ (which implies $|\mathcal{I}_i| > |\mathcal{I}'_i|$).

By simultaneously performing the induction on both $\mathbb{D}_Q$ and $\mathbb{D}_{Q'}$ we rule out the second case, after which the remaining construction of index-covering homomorphism is similar to the bag case.

We add additional structure to $\mathbb{D}_Q$ by combining multiple labelled copies of $\texttt{body}_Q$ together before performing $\overline{r}$-inflation. We define the following set of labels $\mathcal{L}$.

$$\mathcal{L} := \{c_1.c_2 \dots c_j \mid j \in [1,d] \wedge \forall i \in [1,j].(c_i \in \{1,2\})\}$$

For each $l \in \mathcal{L}$ we define the labelling function $\lambda^l : \mathcal{B} \to \mathcal{B}^l$, and we define a single de-labelling function $\lambda^{-1} : (\bigcup_{l \in \mathcal{L}} \mathcal{B}^l) \to \mathcal{B}$ which serves as an inverse for all of the labelling functions.

The labels in $\mathcal{L}$ with length d we call "sequences," while the labels with length less than d we call "prefixes." For every prefix $p \in \mathcal{L}$ with length $m < d$ we define the label-*generating* function $\theta_p : \mathcal{I}_{[1,m]} \to (\bigcup_{l \in \mathcal{L}} \mathcal{B}^l)$ as follows,

$$\theta_{c_1.c_2\dots c_m}(x) := \begin{cases} \theta_{c_1\dots c_{(m-1)}}(x) & \text{if } x \in I_{[1,(m-1)]} \\ \lambda^{c_1\dots c_m}(x) & \text{if } x \in I_m \end{cases}$$
$$= \begin{cases} \lambda^{c_1}(x) & \text{if } x \in \mathcal{I}_1 \\ \lambda^{c_1.c_2}(x) & \text{if } x \in \mathcal{I}_2 \\ \vdots & \vdots \\ \lambda^{c_1\dots c_{(m-1)}}(x) & \text{if } x \in I_{(m-1)} \\ \lambda^{c_1\dots c_m}(x) & \text{if } x \in I_m \end{cases}$$

while for every sequence $s \in \mathcal{L}$ we define the label-generating function $\theta_s : \mathcal{B} \to (\bigcup_{l \in \mathcal{L}} \mathcal{B}^l)$ as follows.

$$\theta_{c_1.c_2\dots c_d}(x) := \begin{cases} \theta_{c_1\dots c_{(d-1)}}(x) & \text{if } x \in I_{[1,(d-1)]} \\ \lambda^{c_1\dots c_d}(x) & \text{if } x \in I_d \\ \lambda^{c_1\dots c_d}(x) & \text{otherwise} \end{cases}$$
$$= \begin{cases} \lambda^{c_1}(x) & \text{if } x \in \mathcal{I}_1 \\ \lambda^{c_1.c_2}(x) & \text{if } x \in \mathcal{I}_2 \\ \vdots & \vdots \\ \lambda^{c_1\dots c_{(d-1)}}(x) & \text{if } x \in I_{(d-1)} \\ \lambda^{c_1\dots c_d}(x) & \text{if } x \in I_d \\ \lambda^{c_1\dots c_d}(x) & \text{otherwise} \end{cases}$$

We immediately extend functions θ_s and θ_p to tuples, sets, and subgoals and with identity on query constants. We also observe that λ^{-1} serves as an inverse for every θ_s and θ_p.

We now define the canonical database $\mathbb{D}_Q$ in two stages. First, we form the database $\mathbb{D}_Q^{\text{pre}}$ as follows.

$$\mathbb{D}_Q^{\text{pre}} := \bigcup_{c_1 \in \{1,2\}} \cdots \bigcup_{c_d \in \{1,2\}} \theta_{c_1\dots c_d}(\texttt{body}_Q)$$

Next, we let $\overline{r}$ be any $(|\mathcal{I}_{[1,d]}|+|\mathcal{I}'_{[1,d]}|)$-distinguishing coordinate for sets of tuples over $\texttt{adom}(\mathbb{D}_Q^{\text{pre}})$, and we use $\overline{r}$-inflation to define canonical database $\mathbb{D}_Q$ as we did with previously for bags.

$$\mathbb{D}_Q := \Delta^{\overline{r}}(\mathbb{D}_Q^{\text{pre}})$$

By whitewashing and de-labelling $\mathbb{D}_Q$, we re-obtain $\texttt{body}_Q$.

$$\lambda^{-1} \circ \delta^{-1}(\mathbb{D}_Q) = \texttt{body}_Q$$

Define $R := (Q)^{\mathbb{D}_Q}$ and $R' := (Q')^{\mathbb{D}_Q}$. For each label $l \in \mathcal{L}$ with length $|l| = j$, we define the canonical object o_l to be the object encoded by the sub-relation $R[\theta_l(\overline{\mathcal{I}}_{[1,j]})]$. We additionally define the canonical object $o_\varnothing$ to be the object encoded by R.

Now given any certificate between R and R' we restrict our attention to the normalized bag nodes that equate relations encoding canonical sub-objects. Consider any such a node occurring at level i of the certificate which encodes canonical object o_l with $|l| = i-1$. By combining the facts that $\texttt{body}_Q$ is minimal and that $\overline{\mathcal{I}}_i$ does not contain any redundant variables, we can prove that the polynomials $f_{S_{o_{l.1}}}$ and $f_{S_{o_{l.2}}}$ (which model the cardinalities of canonical sub-objects $o_{l.1}$ and $o_{l.2}$) have a GCD of degree zero, after which we conclude $S_{o_{l.1}} \approx S'_{o_{l.1}}$ and $S_{o_{l.2}} \approx S'_{o_{l.2}}$ (using simultaneous induction on $\mathbb{D}_{Q'}$ to establish that $|\mathcal{I}_i| = |\mathcal{I}'_i|$). Constructing the index-covering homomorphism is then identical to the bag case.

C.5.3 Set Nodes

The proofs for both bag and normalized bag nodes hinge upon applying equation 14 to translate from a counting argument to an argument that the mappings in the node can be re-organized so that (certain) index values map to permutations of themselves. Set equality ignores cardinality, so counting arguments are of no help. Instead, we architect $\mathbb{D}_Q$ so that the relation $(Q)^{\mathbb{D}_Q}$ encodes certain canonical objects which can only be constructed via indexing on a particular combination of values. To effect this, we construct $\mathbb{D}_Q$ by combining multiple labelled copies of $\texttt{body}_Q$, similar to the approach we used for normalized bags, but with a much more complicated labelling system that introduces much more symmetry into $\mathbb{D}_Q$.

Define integer $N = \max(|\mathcal{I}_{[1,d]}|, |\mathcal{I}'_{[1,d]}|) + 2$. The symmetry will be specified using mechanisms we will call *label-generating components, sequences,* and *prefixes.* For each level $i \in [1, d]$, let LGC_i denote the set of *label-generating components* at level i, defined as follows.

$$LGC_i := \{(\overline{y}_i, \overline{z}_i) \mid \overline{y}_i, \overline{z}_i \in [1, N]^{|\mathcal{I}_i|}\}$$

We say that component $c = (\overline{y}_i, \overline{z}_i)$ contains a *conflict at position $i.j$* if $y_{i.j} = z_{i.j}$, and we use $CF\text{-}LGC_i$ to denote the conflict-free subset of LGC_i.

Let LGS denote the set of *label-generating sequences*, composed out of label-generating components as follows.

$$LGS := \{c_1.c_2 \dots c_d \mid \forall i \in [1, d] : c_i \in LGC_i\}$$

We say that a sequence $s \in LGS$ is *conflict-free* if it is composed entirely of conflict-free components, and we use $CF\text{-}LGS$ to denote the conflict-free subset of LGS.

For each integer $m \in [1, d]$, let LGP_m denote the set of *label-generating prefixes of length m*, defined as follows.

$$\begin{aligned} LGP_m := \{c_1 \dots c_{(m-1)}.\overline{y}_m \mid \forall i \in [1, m-1].\, (c_i \in LGC_i) \\ \wedge \exists \overline{z}_m.\, ((\overline{y}_m, \overline{z}_m) \in LGC_m)\} \end{aligned}$$

Every sequence $s \in LGS$ corresponds to a unique prefix in each of $LGP_1, \dots, LGP_d$. Conversely, every prefix $p \in LGP_m$ with $m < d$ can be extended in $N^{|\mathcal{I}_m|+|\mathcal{I}_{(m+1)}|}$ different ways to yield a prefix in LGP_{m+1}, while every prefix $p \in LGP_d$ can be extended in $N^{|\mathcal{I}_d|}$ different ways to yield a complete sequence in LGS. We say that a prefix is *conflict-free* if it can be iteratively extended into a conflict-free sequence, and we use $CF\text{-}LGP_m$ to denote the conflict-free subset of LGP_m. Every conflict-free prefix $p \in CF\text{-}LGP_m$ with $m < d$ can be extended in $(N-1)^{|\mathcal{I}_m|} N^{|\mathcal{I}_{(m+1)}|}$ different ways to yield a conflict-free prefix in $CF\text{-}LGP_{m+1}$, while every conflict-free prefix $p \in CF\text{-}LGP_d$ can be extended in $(N-1)^{|\mathcal{I}_d|}$ different ways to yield a conflict-free sequence in $CF\text{-}LGS$.

Let $\mathcal{L}$ denote the following set of labels.

$$\begin{aligned} \mathcal{L} := \{\overline{z}_1 \dots \overline{z}_j.k \mid\ & j \in [1, d-1] \\ & \wedge \forall i \in [1, j].\exists \overline{y}_i.((\overline{y}_i, \overline{z}_i) \in LGC_i) \\ & \wedge k \in [1, N]\} \\ \cup \{\overline{z}_1 \dots \overline{z}_d \mid\ & \forall i \in [1, d].\exists \overline{y}.((\overline{y}_i, \overline{z}_i) \in LGC_i)\} \end{aligned}$$

For each $l \in \mathcal{L}$ we define the set $\mathcal{B}^l$, the labelling function λ^l, and the de-labelling function λ^{-1} as we did previously for normalized bags.

For each $m \in [1, d]$ and each label-generating prefix $p \in LGP_m$ we define the following label-generating function $\theta_p : \mathcal{I}_{[1,m]} \to (\bigcup_{l \in \mathcal{L}} \mathcal{B}^l)$ as follows.

$$\begin{aligned} &\theta_{(\overline{y}_1, \overline{z}_1) \dots (\overline{y}_{(m-1)}, \overline{z}_{(m-1)}).\overline{y}_m}(x) \\ &:= \begin{cases} \theta_{(\overline{y}_1, \overline{z}_2) \dots (\overline{y}_{(m-2)}, \overline{z}_{(m-2)}).\overline{y}_{(m-1)}}(x) & \text{if } x \in \mathcal{I}_{[1,m-1]} \\ \lambda^{\overline{z}_1 \dots \overline{z}_{(m-1)}.y_{m.j}}(x) & \text{if } \exists j \in [1, |\mathcal{I}_m|] \text{ such that } x = I_{m.j} \end{cases} \\ &= \begin{cases} \lambda^{y_{1.j}}(x) & \text{if } \exists j \in [1, |\mathcal{I}_1|] \text{ such that } x = I_{1.j} \\ \lambda^{\overline{z}_1.y_{2.j}}(x) & \text{if } \exists j \in [1, |\mathcal{I}_2|] \text{ such that } x = I_{2.j} \\ \lambda^{\overline{z}_1.\overline{z}_2.y_{3.j}}(x) & \text{if } \exists j \in [1, |\mathcal{I}_3|] \text{ such that } x = I_{3.j} \\ \vdots & \vdots \\ \lambda^{\overline{z}_1 \dots \overline{z}_{(m-1)}.y_{m.j}}(x) & \text{if } \exists j \in [1, |\mathcal{I}_m|] \text{ such that } x = I_{m.j} \end{cases} \end{aligned}$$

Similarly, for each label-generating sequence $s \in LGS$, we define the following label-generating function $\theta_s : \mathcal{B} \to (\bigcup_l \mathcal{B}^l)$ as follows.

$$\theta_{(\overline{y}_1, \overline{z}_1) \dots (\overline{y}_d, \overline{z}_d)}(x) := \begin{cases} \theta_{(\overline{y}_1, \overline{z}_2) \dots (\overline{y}_{(d-1)}, \overline{z}_{(d-1)}).\overline{y}_d}(x) & \text{if } x \in \mathcal{I}_{[1,d]} \\ \lambda^{\overline{z}_1 \dots \overline{z}_d}(x) & \text{otherwise} \end{cases}$$

We immediately extend functions θ_p and θ_s to tuples, sets, and subgoals and with identity on constants in $\mathcal{C}$. We also observe that λ^{-1} serves as an inverse for every θ_p and θ_s.

Suppose that $s \in LGS$ contains a conflict at position $i.j$. Given any tuple t containing both variable $I_{i.j}$ and some variable $I \in \mathcal{B} \setminus \mathcal{I}_{[1,i]}$ (i.e. a non-index variable or a member of $\mathcal{I}_{[i+1,d]}$), the labelled tuple $\theta_s(t)$ *evidences the conflict at $i.j$*. That is, because $\theta_s(I_{i.j}) = I_{i.j}^{\overline{z}_1 \dots \overline{z}_{(i-1)}.y_{i.j}}$ and $\theta_s(I)$ is assigned a label that starts with $\overline{z}_1 \dots \overline{z}_i$, from tuple $\theta_s(t)$ we can infer that $y_{i.j} = z_{i.j}$ and so conclude that sequence s has a conflict at position $i.j$.

We now define the canonical database $\mathbb{D}_Q$ as follows.

$$\mathbb{D}_Q := \bigcup_{s \in CF\text{-}LGS} \theta_s(\texttt{body}_Q)$$

Because variable s ranges over only conflict-free label-generating sequences, database $\mathbb{D}_Q$ does not contain any tuple that evidences any conflicts. By de-labelling $\mathbb{D}_Q$ we re-obtain $\texttt{body}_Q$.

$$\lambda^{-1}(\mathbb{D}_Q) = \texttt{body}_Q$$

Define $R := (Q)^{\mathbb{D}_Q}$ and $R' := (Q')^{\mathbb{D}_Q}$. For each $m \in [1, d]$ and $p \in LGP_m$ we define the canonical object o_p to be the object encoded by the sub-relation $R[\theta_p(\overline{\mathcal{I}_{[1,j]}})]$. We additionally define the canonical object $o_\varnothing$ to be the object encoded by R. By combining the definition of canonical objects with the fact that $\mathbb{D}_Q$ was only generated from conflict-free sequences, we can prove the following lemma.

Lemma 2 *Given any integer $m \in [2, d]$ satisfying $|\mathcal{I}_m| > 0$, any prefix $p \in CF\text{-}LGP_{(m-1)}$, and any prefix $q \in LGP_m$ that extends p; canonical object o_p contains canonical object o_q as a sub-object iff q is conflict-free.*

From Lemma 2 we can show that for any $m \in [1, d]$ and prefix $p \in CF\text{-}LGP_m$, by examining the canonical object o_p we can identify all of the values in the set $\theta_p(\mathcal{I}_m)$. (Prove this requires using the fact that $\mathcal{I}_m$ only contains core indexes, and hence each index in $\mathcal{I}_m$ is either an output variable or is related to an inner index variable as per the definition of core indexes in Section 4.1.) We can then prove the following lemma.

Lemma 3 *Given any integer $m \in [1, d]$, any prefix $p \in$ CF-LGP$_m$, and any sub-relation $R'[\overline{a}'_{[1,m]}]$ that encodes canonical object o_p; index tuple $\overline{a}'_{[1,m]}$ must contain all of the values in $\theta_p(\mathcal{I}_m)$.*

PROOF. For each $I_{m.j} \in \mathcal{I}_m$, $\theta_p(I_{m.j}) = I_{m.j}^{\overline{z}_1 \dots \overline{z}_{(m-1)} \cdot y_{m.j}}$. By the symmetry in the construction of $\mathbb{D}_Q$, every database constant that co-occurs with $\theta_p(I_{m.j})$ also co-occurs symmetrically with at least $N - 2$ other constants of the form $I_{m.j}^{\overline{z}_1 \dots \overline{z}_{(m-1)} \cdot n}$. Therefore, in order for index tuple $\overline{a}'_{[1,m]}$ to uniquely determine the constant $\theta_p(I_{m.j})$, either $\theta_p(I_{m.j})$ must occur in $\overline{a}'_{[1,m]}$, or $\overline{a}'_{[1,m]}$ must contain at least $N - 1 > |\mathcal{I}'_{[1,d]}|$ different database constants, which is a contradiction. Hence, $\overline{a}'_{[1,m]}$ must contain the value $\theta_p(I_{m.j})$. □

We can now choose any tuple $\gamma \in (Q)^{\mathbb{D}_Q}$ satisfying

$$\gamma(\overline{\mathcal{I}}_1; \dots; \overline{\mathcal{I}}_d; \overline{\mathcal{V}}) = \theta_s(\overline{\mathcal{I}}_1; \dots; \overline{\mathcal{I}}_d; \overline{\mathcal{V}})$$

for any sequence $s \in$ *CF-LGS*. We can choose any path of nodes down the certificate tree leading to a tuple node containing γ, and by applying Lemma 3 inductively along the path we can prove that each set node at level i must map the index value $\gamma(\overline{\mathcal{I}}_i)$ to a tuple of values $\overline{a}'_i$ that contains all of the same values. Using a simultaneous induction in the opposite direction (on $\mathbb{D}_{Q'}$), we conclude $|\mathcal{I}_i| = |\mathcal{I}'_i|$ and therefore $\overline{a}'_i$ must be a permutation of $\gamma(\overline{\mathcal{I}}_i)$. By composing the de-labelling function λ^{-1} with the embedding $\phi : Q' \rightarrow \mathbb{D}_Q$ that generated index tuple $\overline{a}'_{[1,d]} \in \mathtt{adom}(\overline{\mathcal{I}}'_{[1,d]}, R')$, we obtain an index-covering homomorphism from Q' to Q.