The VLDB Journal (2016) 25:381–397
DOI 10.1007/s00778-016-0421-x

REGULAR PAPER

Mapping-equivalence and oid-equivalence of single-function object-creating conjunctive queries

Angela Bonifati[1] · **Werner Nutt**[2] · **Riccardo Torlone**[3] · **Jan Van den Bussche**[4]

Received: 11 March 2015 / Revised: 5 January 2016 / Accepted: 11 January 2016 / Published online: 30 January 2016

Abstract Conjunctive database queries have been extended with a mechanism for object creation to capture important applications such as data exchange, data integration, and ontology-based data access. Object creation generates new object identifiers in the result that do not belong to the set of constants in the source database. The new object identifiers can be also seen as Skolem terms. Hence, object-creating conjunctive queries can also be regarded as restricted second-order tuple-generating dependencies (SO-tgds), considered in the data exchange literature. In this paper, we focus on the class of single-function object-creating conjunctive queries, or sifo CQs for short. The single-function symbol can be used only once in the head of the query. We give a new characterization for oid-equivalence of sifo CQs that is simpler than the one given by Hull and Yoshikawa and places the problem in the complexity class NP. Our characterization is based on Cohen's equivalence notions for conjunctive queries with multiplicities. We also solve the logical entailment problem for sifo CQs, showing that also this problem belongs to NP. Results by Pichler et al. have shown that logical equivalence for more general classes of SO-tgds is either undecidable or decidable with as yet unknown complexity upper bounds.

Keywords Conjunctive query · Object creation · Oid · Equivalence · Logical entailment · SO-tgd · Sifo CQ · Nested tgd · Schema mapping

Work done while the author Angela Bonifati was affiliated with University of Lille 1 and INRIA Links.

✉ Jan Van den Bussche
jan.vandenbussche@uhasselt.be

Angela Bonifati
angela.bonifati@gmail.com

Werner Nutt
Werner.Nutt@unibz.it

Riccardo Torlone
torlone@dia.uniroma3.it

1 LIRIS, University of Lyon 1, Lyon, France

2 Free University of Bozen-Bolzano, Bolzano, Italy

3 Università Roma Tre, Rome, Italy

4 Hasselt University and Transnational University of Limburg, Hasselt, Belgium

1 Introduction

Conjunctive queries (CQs) form a natural class of database queries, which can be defined by combinations of selection, renaming, natural join, and projection. Much of the research on database query processing is focused on CQs; moreover, these queries are amenable to advanced optimizations because containment of CQs is decidable (though NP-complete). In this paper, we are interested in CQs extended with a facility for object creation.

Object creation, also called oid generation or value invention, has been repeatedly proposed and investigated as a feature of query languages. This has happened in several contexts: high expressiveness [4,5,11]; object orientation [3,10,22,24,29]; data integration [21]; semi-structured data and XML [1]; and data exchange [8,16,18]. In a logic-based approach, object creation is typically achieved through the use of Skolem functions [22,24,29].

In the present paper, we consider CQs extended with object creation through the use of a single Skolem function, which can be used only once in the head of the query. We refer to such a query as a 'sifo CQ' (for single-function object-creating). The following example of a sifo CQ uses a Skolem function f:

$$Q : Family(c, f(x, y)) \leftarrow Mother(c, x), Father(c, y).$$

The query introduces a new oid $f(x, y)$ for every pair (x, y) of a woman x and a man y who have at least one child together; all children c of x and y are linked to the new oid in the result of the query (a relation called *Family*). As an example, if $Mother(beth, anne)$ and $Father(beth, adam)$ are two facts in the underlying database, then the result of the query includes the fact $Family(beth, f(anne, adam))$, where $f(anne, adam)$ is the newly created oid. This oid will be shared by all the children having *anne* and *adam* as parents.

In this paper, we first revisit the problem of checking oid-equivalence of sifo CQs. Oid-equivalence has its origins in the theory of object-creating queries introduced by Abiteboul and Kanellakis [3]; it is the natural generalization of query equivalence in the presence of object creation.

Consider for instance the following sifo CQ:

$$Q' : Family(c, g(x, y, x)) \leftarrow Mother(c, x), Father(c, y).$$

It is not hard to see that the result of Q' has the same structure as the result of the query Q above. The query Q' links all children c of the parents x and y to the oid $g(x, y, x)$ that depends exactly on x and y. That is, two children in the result of Q are connected to the same oid if and only if they are connected to same oid in Q', although the oids will be syntactically different. Therefore, we can conclude that Q and Q' are oid-equivalent, which means that their results are identical on any input up to a simple isomorphism mapping the oids in one result to those in the other.

Hull and Yoshikawa [23] studied oid-equivalence (they called it 'obscured equivalence') for non-recursive ILOG programs; the decidability of this problem is a long-standing open question. Nevertheless, for the case of 'isolated oid creation,' to which sifo CQs belong, they have given a decidable characterization.

We give a new result relating oid-equivalence to equivalence of classical conjunctive queries under 'combined' bag–set semantics [14], which models the evaluation of CQs when query results and relations may contain duplicates of tuples. As a corollary, we obtain that oid-equivalence for sifo CQs belongs to NP, which does not follow from the Hull–Yoshikawa test. Obviously, then, oid-equivalence for sifo CQs is NP-complete, since equivalence of classical CQs without object creation is already NP-complete.

Object creation is receiving renewed interest in the context of schema mappings [8,18], which are formalisms describing how data structured under a source schema are to be transformed into data structured under a target schema. Hence, it is instructive to view sifo CQs as schema mappings, simply by interpreting them as implicational statements. As an example, we may view query Q above as an implicational statement that relates a query over relations *Mother* and *Father* in the source schema to the relation *Family* in the target schema.

For standard CQs without object creation, two queries are equivalent if and only if they are logically equivalent as schema mappings [17]. For sifo CQs, we show that oid-equivalence implies logical equivalence, while the converse is not true.

Sifo CQs viewed as schema mappings belong to the class of so-called 'nested dependencies' [8], which belong in turn to the class of formulas called second-order tuple-generating dependencies (SO-tgds [18]). For instance, consider again the sifo CQ Q above: It can be rewritten into the following SO-tgd:

$$\exists f \forall x \forall y \forall c \, (Mother(c, x) \wedge Father(c, y) \rightarrow Family(c, f(x, y))),$$

which is of second order because the function f is existentially quantified.

Although logical equivalence of SO-tgds is undecidable [19], logical implication of nested dependencies has recently been shown to be decidable [26]. We give a novel and elegant characterization of logical implication for sifo CQs which is simpler than the general implication test for nested dependencies. It turns out that the problem belongs to NP. Hence, logical implication for sifo CQs has no worse complexity than containment for standard CQs without object creation.

Summarizing, in this paper we provide the following contributions in the area of query languages with object creation:

1. We clarify the relationship between sifo CQs and other formalisms in the literature, notably the language ILOG [22], second-order tuple-generating dependencies [18], and nested tuple-generating dependencies [8].
2. We relate the problem of oid-equivalence for sifo CQs to the equivalence of classical conjunctive queries under combined bag–set semantics, which implies its NP-completeness.
3. We show that when sifo CQs are interpreted as schema mappings, oid-equivalence implies logical equivalence but not vice versa.
4. We provide a new characterization of logical implication for sifo CQs as object-creating queries showing that this problem has the same complexity as deciding containment for classical CQs.

This paper is organized as follows. In Sect. 2, we review some practical applications of sifo CQs. In Sect. 3, we formally define object-creating conjunctive queries. Section 4 is devoted to the results on oid equivalence. Section 5 is devoted to the results on logical entailment. In Sect. 6, we conclude by discussing related work and topics for further research.

2 Applications of sifo CQs

In this section, we discuss further applications of sifo CQs, which may constitute important components of many advanced database systems, spanning from information integration, and schema mapping engines along with their benchmarks, to several semantic Web tools. We believe this shows that the results in this article on equivalence and logical implication of sifo CQs are relevant and contribute to our understanding of how solutions for these applications can be optimized.

Global-as-view (*GAV*) schema mappings [20,27,33] relate a query over the source schema, represented by a body B of a CQ, to an atomic element of the global schema, represented by a head atom H of a CQ. More precisely, a GAV mapping can be written as follows:

$$T(\bar{x}) \leftarrow B$$

where we use a relation symbol T as the atomic head predicate.

GAV schema mappings have been used already in the 1990s in mediator systems like Tsimmis [30,33] or information manifold [28] for the integration of heterogeneous data sources. In both systems, source facts are related to facts over the global schema by means of queries.

Sifo CQs can naturally be seen as extensions of GAV mappings, when one of the attributes of the global schema carries newly created identifiers.

For instance, the sifo CQ Q from Sect. 1 can express a mapping from a source schema containing two relations *Mother* and *Father* to one relation *Family* of a global schema, with created identifiers for families appearing in the tuples in the result of the mapping. Thus, we can also interpret Q as an extended GAV schema mapping.

Another important application of sifo CQs is schema mapping benchmarks allowing the users to compare and evaluate schema mapping systems. In particular, the flexibility of the arguments of the Skolem functions used for object creation has been advocated as one of the desirable features in recent benchmarks for schema mapping and information integration, such as STBenchmark [6] and iBench [9].

More precisely, in the mapping primitives of iBench [9], an extension of STBenchmark [6] that supports SO-tgds, the users can choose among two different skolemization strategies to fill the arguments of the Skolem functions: *fixed*, where the arguments of the function are pre-defined in a native mapping primitive, or *variable*, where one can further choose among the options *All, Key*, and *Random*, which generate mappings where all variables, the variables in the positions of the primary key, or a random set of variables, respectively, are used as arguments of the function.

These skolemization strategies can be captured by sifo CQs as follows.

In the query below:

$$T(x, y, f(x, y, z, w)) \leftarrow B(x, y, z, w)$$

we can observe that the Skolem term uses all the source variables in the body B (option *All*). If the attribute in the position of x is a primary key for B, then the application of the option *Key* generates a mapping that can be expressed by the sifo CQ

$$T(x, y, f(x)) \leftarrow B(x, y, z, w).$$

Alternatively, choosing the option *Random* may lead the iBench to randomly select the attributes in the positions of x and z and then to generate the mapping represented by

$$T(x, y, f(x, z)) \leftarrow B(x, y, z, w).$$

It is also worth highlighting that three out of the seven mapping primitives in iBench that are novel with respect to STBenchmark, namely *ADD* (copy a relation and ADD new attributes), *ADL* (copy a relation, Add and DeLete attributes in tandem), and *MA* (merge and add new attributes) contain single Skolem functions. They correspond to the following sifo CQs, respectively:

$$T(x, y, f(x, y)) \leftarrow B(x, y)$$
$$T(x, f(x)) \leftarrow B(x, y)$$
$$T(x, y, z, f(x, y, z)) \leftarrow B(x, y), T(y, z).$$

A third significant application of sifo CQs is the Semantic Web, where sifo CQs can be envisioned in at least two scenarios, namely in systems for ontology-based data access (OBDA) and in direct mappings from the relational to the RDF data format, under development at W3C.[1] Indeed, newly created identifiers in the head of a sifo CQ can serve as generated keys, or simply as newly invented values needed to fill an attribute of a relation in the global schema. As such, sifo CQs can be seen as examples of mapping assertions from source schemas to a global ontology in OBDA [31]. Typically, OBDA mapping assertions relate facts in relational source schemas to RDF triples in a global ontology. The newly generated IRIs[2] in the RDF triples can be interpreted as skolemized values in the global ontology.

[1] http://www.w3.org/TR/rdb-direct-mapping/.

[2] IRIs stand for internationalized resource identifiers and extend the syntax of uniform resource identifiers (URIs) to a much wider repertoire of characters. They naturally embody global identifiers that refer to the same resource on the Web and can be used across different mapping assertions to refer to that resource.

A related application is the direct translation of a relational schema into OWL, which uses as an important building block the creation of IRIs [32]. In contrast to the previous application, this application handles relational schemas that are not known in advance. For each relation r in a database schema, Datalog-like rules can be used to generate an IRI for the relation r and an IRI for each attribute a in r. We take an example of a translation from a relational schema into OWL, and we show that, actually, these Datalog-like rules can be viewed as sifo CQs, since they employ a single concatenation function to obtain such IRIs (exemplified as f). The corresponding sifo CQs are reported below:

$$T_1(r, f(b, r)) \leftarrow B_1(r)$$
$$T_2(a, r, f(b, r, a)) \leftarrow B_2(r, a),$$

where B_1 and B_2 are conjunctive query (CQ) bodies retrieving relation names r and attribute names a from the data dictionary of an underlying relational database and where b is a string representing a given *IRI base* (e.g., the string 'http://example.edu/db') for the same database to be translated. Thus, the first query creates a new IRI for the relation r, by concatenating b with the relation symbol r, while the second query returns the set of IRIs of the attributes a of r, by concatenating b with the relation symbol r and its attribute symbols a.

3 Preliminaries

In this section, we introduce our formalism for dealing with conjunctive queries and introduce the notion of object-creating CQ, adapted from the language ILOG [22].

3.1 Databases and conjunctive queries

From the outset, we assume a supply of *relation names*, where each relation name R has an associated arity $\mathrm{ar}(R)$. We also assume an infinite domain **dom** of atomic data elements called *constants*. A *fact* is of the form $R(a_1, \ldots, a_k)$ where $a_1, \ldots, a_k$ are constants and R is a k-ary relation name. We call R the *predicate* of the fact.

A *database schema* $\mathbf{S}$ is a finite set of relation names. An *instance* of $\mathbf{S}$ is a finite set of facts with predicates from $\mathbf{S}$. The set of all constants appearing in an instance I is called the *active domain* of I and denoted by $\mathrm{adom}(I)$.

We further assume an infinite supply of *variables*, disjoint from **dom**. An *atom* is of the form $R(x_1, \ldots, x_k)$ where $x_1, \ldots, x_k$ are variables and R is a k-ary relation name. As with facts, we call R the predicate of the atom.

We can now recall the classical notion of CQ [2,13]. Syntactically, a CQ over a database schema $\mathbf{S}$ is of the form

$$H \leftarrow B,$$

where B is a finite set of atoms with predicates from $\mathbf{S}$ and H is an atom with a predicate not in $\mathbf{S}$. The set B is called the *body*, and H is called the *head*. It is required that every variable occurring in the head also occurs in the body. We denote the set of variables occurring in a set of atoms B (or a single atom A) by $\mathrm{var}(B)$ (or $\mathrm{var}(A)$).

The semantics of CQs is defined in terms of valuations. A *valuation* is a mapping $\alpha : X \to \mathbf{dom}$ on some finite set of variables X. When A is an atom with $\mathrm{var}(A) \subseteq X$, we can apply α to A simply by applying α to every variable in A. This results in a fact and is denoted by $\alpha(A)$. When B is a set of atoms and α is a valuation on $\mathrm{var}(B)$, we can apply α to B by applying α to every atom in B. Formally, $\alpha(B)$ is defined as the instance $\{\alpha(A) \mid A \in B\}$.

When I is an instance and α is a valuation on $\mathrm{var}(B)$ such that $\alpha(B) \subseteq I$, we say that α is a *matching* of B in I, and denote this by $\alpha : B \to I$. Now when Q is a CQ $H \leftarrow B$ and I is an instance, the result of Q on I is defined as

$$Q(I) := \{\alpha(H) \mid \alpha : B \to I\}.$$

3.2 Object-creating conjunctive queries

Assume a finite vocabulary of *function symbols* of various arities. As with relation names, the arity of a function symbol f is denoted by $\mathrm{ar}(f)$.

Data terms are syntactical expressions built up from constants using function symbols. Formally, data terms are inductively defined as follows:

1. Every constant is a data term;
2. If f is a k-ary function symbol and $d_1, \ldots, d_k$ are data terms, then the expression $f(d_1, \ldots, d_k)$ is also a data term.[3]

An *extended fact* is defined just like a fact, except that it may contain data terms rather than only constants. Formally, an extended fact is of the form $R(d_1, \ldots, d_k)$, where $d_1, \ldots, d_k$ are data terms and R is a k-ary relation name. The active domain of an extended fact $e = R(d_1, \ldots, d_k)$ is defined as

$$\mathrm{adom}(e) := \{d_1, \ldots, d_k\}.$$

[3] Since constants are atomic data elements, no constant is allowed to be of the form $f(d_1, \ldots, d_k)$.

Table 1 Instances used in Example 1

Mother

beth	anne
ben	anne
eric	claire
emma	diana
dave	diana

Father

beth	adam
ben	adam
eric	carl
emma	carl

Family

beth	$f(\text{anne}, \text{adam})$
ben	$f(\text{anne}, \text{adam})$
eric	$f(\text{claire}, \text{carl})$
emma	$f(\text{diana}, \text{carl})$

An *extended instance* is a finite set of extended facts. The active domain of an extended instance J is defined as

$$\text{adom}(J) := \bigcup_{e \in J} \text{adom}(e).$$

Formula terms are defined in the same way as data terms, but are built up from variables rather than constants. *Extended atoms* are defined like atoms, but can contain formula terms in addition to variables. If t is a formula term and α is a valuation defined on all variables occurring in t, we can apply α to every variable occurrence in t, obtaining a data term $\alpha(t)$. Likewise, we can apply a valuation to an extended atom, resulting in an extended fact.

We are now ready to define the syntax and semantics of *object-creating conjunctive queries* (oCQ). Like a classical CQ, an oCQ is of the form $H \leftarrow B$. The only difference with a classical CQ is that H can be an extended atom; in particular, B is still a finite set of 'flat' atoms, not extended atoms. It is still required that $\text{var}(H) \subseteq \text{var}(B)$. The result of an oCQ $Q = H \leftarrow B$ on an instance I is now an extended instance, defined as

$$Q(I) := \{\alpha(H) \mid \alpha : B \rightarrow I\}.$$

Example 1 Recall the oCQ Q from Sect. 1:

$$\mathit{Family}(c, f(x, y)) \leftarrow \mathit{Mother}(c, x), \mathit{Father}(c, y).$$

If I is the instance consisting of the Mother and Father facts listed in Table 1, then $Q(I)$ is the extended instance consisting of the extended Family facts listed in the same table.

Example 2 For a more abstract example, consider the following oCQ Q:

$$T(x, f(y)) \leftarrow R(x, y, z).$$

If I is the instance consisting of the R-facts listed in Table 2, then $Q(I)$ consists of the extended T-facts listed in the same table.

Table 2 Instances used in Example 2

R

a	b	c
a	b	d
c	b	d
d	c	a

T

a	$f(b)$
c	$f(b)$
d	$f(c)$

3.3 The single-function case

In this paper, we focus on *single-function* oCQs (sifo CQs) that have exactly one occurrence of a function symbol in the head. Without loss of generality, we always place the function term in the last position of the head.

Definition 1 A sifo CQ over a database schema **S** is an oCQ over **S** of the form

$$T(\bar{x}, f(\bar{z})) \leftarrow B,$$

where T is the head predicate, f is a function symbol, B is the body, $\bar{x}$ is a tuple of (not necessarily distinct) variables from $\text{var}(B)$, called the *distinguished variables*, $\bar{z}$ is a tuple of (not necessarily distinct) variables from $\text{var}(B)$, called the *creation variables*; some creation variables may be distinguished; the elements of $\text{var}(B)$ that are not distinguished are called the *non-distinguished variables*.

Example 3 The queries in Examples 1 and 2 are both examples of sifo CQs.

3.4 Comparison with ILOG

Object-creating CQs can be considered to be the conjunctive query fragment of non-recursive ILOG [22]; our syntax exposes the Skolem functions, which are normally obscured in the standard ILOG syntax, and our semantics corresponds to what is called the 'exposed semantics' by Hull and Yoshikawa. Nevertheless, in the following section, we will consider oid-equivalence of sifo CQs, which does correspond to what has been called 'obscured equivalence' [23].

4 Characterization of oid-equivalence for sifo CQs

4.1 Oid-equivalence of oCQs

The result $Q(I)$ of an oCQ Q applied to an instance I is an extended instance. The data terms in $\text{adom}(Q(I))$ that are not constants play the role of created oids (also called invented values). Intuitively, it is clear that the actual form of the created oids does not matter.

Example 4 Recall the query Q from Example 1:

$$\mathit{Family}(c, f(x, y)) \leftarrow \mathit{Mother}(c, x), \mathit{Father}(c, y).$$

Table 3 Instance used in Example 4

	Family
beth	$g(\text{anne}, \text{adam}, \text{anne})$
ben	$g(\text{anne}, \text{adam}, \text{anne})$
eric	$g(\text{claire}, \text{carl}, \text{claire})$
emma	$g(\text{diane}, \text{carl}, \text{diane})$

As mentioned in Sect. 1, we could have used equivalently the following query Q':

$$\mathit{Family}(c, g(x, y, x)) \leftarrow \mathit{Mother}(c, x), \mathit{Father}(c, y).$$

Applying the above query to the mother and father facts from Table 1, results in the instance given in Table 3. Intuitively, this instance has exactly the same relevant properties as the family instance from Table 1: beth and ben are linked to the same family oid; eric is linked to another oid and emma to still another one.

We formalize this intuition in the following definitions.

Definition 2 Let J be an extended instance.

- The set $\text{adom}(J) - \mathbf{dom}$ is denoted by $\text{oids}(J)$;
- The set $\text{adom}(J) \cap \mathbf{dom}$ is denoted by $\text{consts}(J)$.

Definition 3 Let J be an extended instance and let ρ be a mapping from $\text{adom}(J)$ to the set of data terms. For any extended fact $e = R(d_1, \ldots, d_k)$ in J, we define $\rho(e)$ to be the extended fact $R(\rho(d_1), \ldots, \rho(d_k))$. We then define $\rho(J) := \{\rho(e) \mid e \in J\}$.

Definition 4 Let J_1 and J_2 be extended instances. Then J_1 and J_2 are called *oid-isomorphic* if there exists a bijection $\rho : \text{adom}(J_1) \rightarrow \text{adom}(J_2)$ such that

- ρ is the identity on $\text{consts}(J_1)$;
- ρ maps $\text{oids}(J_1)$ to $\text{oids}(J_2)$;
- $\rho(J_1) = J_2$.

Such a bijection ρ is called an *oid-isomorphism* from J_1 to J_2.

The above definition implies that oid-isomorphic instances have the same constants. Formally, if J_1 and J_2 are oid-isomorphic, then $\text{consts}(J_1) = \text{consts}(J_2)$.

Definition 5 Let Q and Q' be two oCQs with the same head predicate and over the same database schema $\mathbf{S}$. Then Q and Q' are called *oid-equivalent* if for every instance I over $\mathbf{S}$; the results $Q(I)$ and $Q'(I)$ are oid-isomorphic.

Example 5 The queries in Example 4 are oid-equivalent. For example, for the instance I of Table 1, the oid-isomorphism from $Q(I)$ to $Q'(I)$ is as follows:

Table 4 Instances used in Example 6

I	$Q(I)$	$Q'(I)$
$a\ b\ c$	$a\ f(b)$	$a\ f(a,b)$
$d\ b\ e$	$d\ f(b)$	$d\ f(d,b)$

Table 5 Instances used in Example 7

I	$Q(I)$	$Q'(I)$
$a\ b\ c$	$a\ f(a)$	$a\ f(a,b,c)$
$a\ d\ e$		$a\ f(a,d,e)$

$$f(\text{anne}, \text{adam}) \mapsto g(\text{anne}, \text{adam}, \text{anne})$$
$$f(\text{claire}, \text{carl}) \mapsto g(\text{claire}, \text{carl}, \text{claire})$$
$$f(\text{diane}, \text{carl}) \mapsto g(\text{diane}, \text{carl}, \text{diane}).$$

Example 6 Recall the query Q from Example 2:

$$T(x, f(y)) \leftarrow R(x, y, z)$$

Also consider the following variation Q' of Q:

$$T(x, f(x, y)) \leftarrow R(x, y, z)$$

Then Q and Q' are not oid-equivalent, as given by the simple instances in Table 4. Indeed, there cannot be an oid-isomorphism from $Q(I)$ to $Q'(I)$ because $Q(I)$ contains only one distinct oid while $Q'(I)$ contains two distinct oids.

Example 7 As a variant of Example 6, consider the following two oCQs:

$$Q = T(x, f(x)) \leftarrow R(x, y, z)$$
$$Q' = T(x, f(x, y, z)) \leftarrow R(x, y, z)$$

Again these two oCQs are not oid-equivalent, as shown by the counterexample instances in Table 5.

4.2 Homomorphisms and containment of conjunctive queries

The characterizations we will give for oid-equivalence of sifo CQs depend on the classical notions of homomorphism and containment between conjunctive queries. Let us briefly recall these notions now [2,13].

A *variable mapping* is a mapping h from a finite set X of variables to another finite set Y of variables. If A is an atom with variables in X, then we can apply h to each variable occurrence in A to obtain an atom with variables in Y, which we denote by $h(A)$. If B is a set of atoms with $\text{var}(B) \subseteq X$, then we naturally define $h(B) := \{h(A) \mid A \in B\}$.

For two sets B and B' of atoms, a variable mapping $h : \text{var}(B) \rightarrow \text{var}(B')$ is called a *homomorphism from B to B'*

if $h(B) \subseteq B'$. This is denoted by $h : B \to B'$. The notion of homomorphism is extended to conjunctive queries $Q = H \leftarrow B$ and $Q' = H' \leftarrow B'$ as follows. A homomorphism from Q to Q' is a homomorphism $h : B \to B'$ such that $h(H) = H'$. This is denoted by $h : Q \to Q'$.

A classical result relates homomorphisms between conjunctive queries to containment. Let Q and Q' be two conjunctive queries over a common database schema $\mathbf{S}$. We say that Q' *is contained in* Q if for every instance I of $\mathbf{S}$, we have $Q'(I) \subseteq Q(I)$. The classical result states that Q' is contained in Q if and only if there exists a homomorphism $h : Q \to Q'$.

Two queries Q and Q' are *equivalent* if for every instance I of $\mathbf{S}$, we have $Q(I) = Q'(I)$. Since equivalence amounts to containment in both directions, two conjunctive queries are equivalent if and only if there exist homomorphisms between them in both directions.

4.3 A normal form for oid-equivalence problems

In this subsection, we consider two arbitrary sifo CQs Q, Q' with the same head predicate:

$$Q = T(\bar{x}, f(\bar{z})) \leftarrow B$$
$$Q' = T(\bar{x}', f'(\bar{z}')) \leftarrow B'.$$

Then $\bar{x}$ and $\bar{x}'$ have equal length. Note that $\bar{x}$ and $\bar{z}$ as well as $\bar{x}'$ and $\bar{z}'$ may have variables in common.

Our aim is to show that oid-equivalence between arbitrary sifo CQs Q and Q' can be reduced to the case where the heads

$$T(\bar{x}, f(\bar{z})) \quad \text{and} \quad T(\bar{x}', f'(\bar{z}'))$$

have identical arguments, that is, where $\bar{x} = \bar{x}'$ and $\bar{z} = \bar{z}'$.

As a first lemma, we state that rearranging the creation variables of a query does not affect oid-equivalence.

Lemma 1 (Rearranging creation variables) *Let Q be a sifo CQ written as above. Let $\bar{u}$ be a tuple with exactly the same variables as $\bar{z}$, but possibly with different repetitions and a different ordering, and let g be a function symbol whose arity is equal to the length of $\bar{u}$. Then the sifo CQ $P = T(\bar{x}, g(\bar{u})) \leftarrow B$ is oid-equivalent to Q.*

Proof Let I be an instance. We define an oid-isomorphism from $Q(I)$ to $P(I)$ as follows. Any oid o in $Q(I)$ is of the form $f(\alpha(\bar{z}))$ for some matching $\alpha : B \to I$; we define $\rho(o) := g(\alpha(\bar{u}))$. This is well defined, i.e., independent of the choice of α. Indeed, if the data terms $f(\alpha_1(\bar{z}))$ and $f(\alpha_2(\bar{z}))$ are equal, then the tuples $\alpha_1(\bar{z})$ and $\alpha_2(\bar{z})$ are equal, which implies that α_1 and α_2 agree on every variable appearing in $\bar{z}$. Since exactly the same variables appear in $\bar{u}$, also the tuples $\alpha_1(\bar{u})$ and $\alpha_2(\bar{u})$ are equal, whence $g(\alpha_1(\bar{u})) = g(\alpha_2(\bar{u}))$.

That $\rho : \text{oids}(Q(I)) \to \text{oids}(P(I))$ is injective is shown by an analogous argument. The surjectivity of ρ, as well as the equality $\rho(Q(I)) = P(I)$, is clear. □

By the above lemma, we can remove all duplicates from $\bar{z}$ and $\bar{z}'$ in the heads of Q and Q', respectively. So, from now on we may assume $\bar{z}$ and $\bar{z}'$ have no duplicates.

In the following, let Z equal the set of variables occurring in $\bar{z}$, let X equal the set of variables occurring in $\bar{x}$, and let Z' and X' be defined similarly.

We next show that two sifo CQs can only be oid-equivalent if they have identical patterns of distinguished variables, up to renaming.

Lemma 2 (Renaming distinguished variables) *If Q and Q' are oid-equivalent, then there exists a bijective variable mapping $\sigma : X \to X'$ such that $\sigma(\bar{x}) = \bar{x}'$.*

Proof Certainly, if Q and Q' are oid-equivalent, then the conjunctive queries $Q_0 = T_0(\bar{x}) \leftarrow B$ and $Q'_0 = T_0(\bar{x}') \leftarrow B'$, where T_0 is a new predicate symbol, are equivalent. So, there are homomorphisms $h : Q_0 \to Q'_0$ and $h' : Q'_0 \to Q_0$. In particular, $h(\bar{x}) = \bar{x}'$ and $h'(\bar{x}') = \bar{x}$. We define σ to be the restriction of h to X. The claim $\sigma(\bar{x}) = \bar{x}'$ and the surjectivity of σ are then clear. So it remains to show that σ is injective. Thereto, consider $h'(\sigma(\bar{x})) = h'(h(\bar{x})) = h'(\bar{x}') = \bar{x}$. We see that $h' \circ \sigma$ is the identity on X and thus injective. Hence, σ must be injective as well. □

By the above lemma, if there does *not* exist a renaming σ as in the lemma, certainly Q and Q' are not oid-equivalent. If there exists such a renaming, then by renaming the variables in one of the two queries; we can now assume without loss of generality that $\bar{x} = \bar{x}'$ and in particular that $X = X'$.

The next step is to show that oid-equivalent queries must have the same distinguished variables among the creation variables, that is, $X \cap Z = X \cap Z'$.

Lemma 3 (Distinguished creation variables) *If $X \cap Z \neq X \cap Z'$, then Q and Q' are not oid-equivalent.*

Proof Either there exists some $x \in X \cap Z$ but not in Z' or vice versa. By symmetry, we may assume the first possibility.

We construct an instance I from B'. In doing this, to keep our notation simple, we consider the variables in B' to be constants. The instance I is obtained from B' by duplicating x to some new element x_2. Formally, consider the mapping d on $\text{var}(B')$ that is the identity everywhere except that x is mapped to x_2; then $I = B' \cup d(B')$.

First, let us look at $Q'(I)$. Using the identity matching that maps every variable to itself, we obtain the extended fact $T(\bar{x}, f'(\bar{z}')) \in Q'(I)$. Using the matching d defined above, we obtain the extended fact $T(\bar{x}_2, f'(d(\bar{z}')))$ in $Q'(I)$. Here, $\bar{x}_2$ denotes $d(\bar{x})$, i.e., $\bar{x}_2$ is obtained from $\bar{x}$ by replacing x

with x_2. Since x does not belong to Z', we have $d(\bar{z}') = \bar{z}'$, so $T(\bar{x}_2, f'(\bar{z}')) \in Q'(I)$.

On the other hand, in $Q(I)$ consider any two extended facts $T(\alpha_1(\bar{x}), f(\alpha_1(\bar{z})))$ and $T(\alpha_2(\bar{x}), f(\alpha_2(\bar{z})))$, with matchings $\alpha_1 : B \to I$ and $\alpha_2 : B \to I$, such that $\alpha_1(\bar{x}) = \bar{x}$ and $\alpha_2(\bar{x}) = \bar{x}_2$. Then in particular $\alpha_1(x) = x$ and $\alpha_2(x) = x_2$. Since α_1 and α_2 differ on x and x is in Z, also $\alpha_1(\bar{z})$ and $\alpha_2(\bar{z})$ are different. Hence, the two last components $f(\alpha_1(\bar{z}))$ and $f(\alpha_2(\bar{z}))$ are different. Thus, we see that in $Q(I)$ it is impossible to have two extended atoms $T(\bar{x}, o)$ and $T(\bar{x}_2, o)$ with the same oid o. But we have seen this is possible in $Q'(I)$, so $Q(I)$ and $Q'(I)$ are not oid-isomorphic and Q and Q' cannot be oid-equivalent. □

By the above lemma, we now assume $X \cap Z = X \cap Z'$. The last step is to show that $Z - X$ and $Z' - X$, the sets of non-distinguished creation variables, need to have the same cardinality.

Lemma 4 (Non-distinguished creation variables) *If $Z - X$ and $Z' - X$ have different cardinality, then Q and Q' are not oid-equivalent.*

Proof As in the proof of Lemma 3, we consider B as an instance, viewing variables as constants.

Let k and k' be the cardinalities of $Z - X$ and $Z - X'$, respectively. By symmetry, we may assume that $k > k'$. Now, for any natural number n, let I_n be the instance obtained from B by independently multiplying each variable $z \in Z - X$ into n fresh copies $z^{(1)}, \ldots, z^{(n)}$. Formally, for any function $d : Z - X \to \{1, \ldots, n\}$, let $\hat{d}$ be the valuation on $\mathrm{var}(B)$ that maps each $z \in Z - X$ to $z^{(d(z))}$ and that is the identity on all other variables. Then

$$I_n = \bigcup_{d : Z - X \to \{1, \ldots, n\}} \hat{d}(B).$$

There are n^k different functions $d : Z - X \to \{1, \ldots, n\}$. Each corresponding valuation $\hat{d}$ is a matching of B in I_n; all these matchings are the identity on $\bar{x}$ but are pairwise different on $\bar{z}$. Thus, there are at least n^k different extended facts in $Q(I_n)$ of the form $T(\bar{x}, o)$.

On the other hand, consider any set S of valuations from $X \cup Z'$ to $\mathrm{adom}(I_n)$ that are pairwise different on $Z' - X$ but that all agree on X. The cardinality of $Z' - X$ is k'. The cardinality of $\mathrm{adom}(I_n)$ is $O(n)$ (although the cardinality of I_n itself is larger). Hence, such a set S can be of cardinality at most $O(n^{k'})$. Consequently, since $k > k'$, for n large enough, $Q'(I_n)$ cannot possibly contain n^k different extended facts of the form $T(\bar{x}, o)$. But we saw that this is possible in $Q(I_n)$. So, $Q(I_n)$ and $Q'(I_n)$ are not oid-isomorphic and Q and Q' cannot be oid-equivalent. □

By the above lemma and after renaming the variables in $Z' - X$ and reordering the variables in $\bar{z}'$, we may now indeed assume that $\bar{z}$ and $\bar{z}'$ are identical.

4.4 Characterization of oid-equivalence

According to the results of the preceding subsection, we are now given two sifo CQs as follows:

$$Q = T(\bar{x}, f(\bar{z})) \leftarrow B \tag{1}$$

$$Q' = T(\bar{x}, f'(\bar{z})) \leftarrow B'. \tag{2}$$

Note that Q and Q' have identical tuples $\bar{x}$ and $\bar{z}$ of distinguished and creation variables; moreover, $\bar{z}$ contains no variable more than once. As before, we denote the sets of distinguished and creation variables as X and Z, respectively.

We will show that Q and Q' are oid-equivalent if and only if there are homomorphisms between B and B' in both directions that (i) keep $\bar{x}$ fixed and (ii) possibly permute the variables in $\bar{z}$. To make this formal, we associate with each query a classical CQ without function symbols.

Definition 6 Fix a new relation symbol $\mathring{T}$ of arity the sum of the lengths of $\bar{x}$ and $\bar{z}$. The *flattening* of Q is the query $\mathring{Q} = \mathring{T}(\bar{x}, \bar{z}) \leftarrow B$. The query $\mathring{Q}'$ is defined similarly.

Let π be a permutation of the set $Z - X$. We extend π to $\mathrm{var}(B)$ by defining it to be the identity outside $Z - X$. We now define $\mathring{Q}^\pi$ to be the CQ obtained from $\mathring{Q}$ by permuting the variables in $\bar{z}$, that is

$$\mathring{Q}^\pi = \mathring{T}(\bar{x}, \pi(\bar{z})) \leftarrow B.$$

This notion allows us to formulate the following natural sufficient condition for oid-equivalence.

Proposition 1 *If there exists a permutation π of $Z - X$ such that $\mathring{Q}^\pi$ and $\mathring{Q}'$ are equivalent, then Q and Q' are oid-equivalent.*

Proof Let I be an instance. We define an oid-isomorphism ρ from $Q(I)$ to $Q'(I)$ as follows. Any oid o in $Q(I)$ is of the form $f(\alpha(\bar{z}))$ for some matching $\alpha : B \to I$; we define $\rho(o) := f'(\alpha(\pi(\bar{z})))$. This is well defined, i.e., independent of the choice of α. Indeed, if the data terms $f(\alpha_1(\bar{z}))$ and $f(\alpha_2(\bar{z}))$ are equal, then the tuples $\alpha_1(\bar{z})$ and $\alpha_2(\bar{z})$ are equal, and consequently, the permuted tuples $\alpha_1(\pi(\bar{z}))$ and $\alpha_2(\pi(\bar{z}))$ are equal. Hence, $f'(\alpha_1(\pi(\bar{z}))) = f'(\alpha_2(\pi(\bar{z})))$.

The injectivity of $\rho : \mathrm{oids}(Q(I)) \to \mathrm{oids}(Q'(I))$ is shown by an analogous argument. The surjectivity of ρ and the equality $\rho(Q(I)) = Q'(I)$ follow readily from the equality $\mathring{Q}^\pi(I) = \mathring{Q}'(I)$. □

We next prove that the sufficient condition given by the above Proposition is actually also necessary for oid-equivalence. The key idea for proving this is to show that oid-equivalence of sifo CQs depends only on the *number* of oids generated for any binding of the distinguished variables.

Formally, for any instance I and any tuple $\bar{c}$ of elements from $\mathrm{adom}(I)$, we define

$$\#_{\bar{c}}(Q, I) := \#\{\, o \mid T\,(\bar{c}, o) \in Q(I)\,\},$$

that is, $\#_{\bar{c}}(Q, I)$ denotes the number of distinct oids o that occur together with $\bar{c}$ in $Q(I)$. We will show that Q and Q' are oid-equivalent if and only if $\#_{\bar{c}}(Q, I) = \#_{\bar{c}}(Q', I)$ for all instances I and tuples $\bar{c}$. The only-if direction of this statement is obvious, but the if-direction is not so obvious.

For our proof, we rely on work by Cohen [14] who studied queries with multiset variables that are evaluated under so-called combined semantics, a semantics that combines set and multiset semantics. Cohen characterized equivalence of such queries in terms of homomorphisms.

Queries with multiset variables (MV queries) have the form Q_0, M where Q_0 is a standard CQ and M is some set of variables of Q_0 that do not appear in the head of Q_0. The elements of M are called the multiset variables. Evaluating an MV query Q_0, M on an instance I results in a multiset (bag) of facts, where the number of times a fact occurs is related to the number of different possible assignments of values to the multiset variables.

Let us define the combined semantics formally. Let Q_0 be of the form $H_0 \leftarrow B_0$ and let I be an input instance. Recall that $Q_0(I)$ according to the classical semantics equals

$$\{\alpha(H_0) \mid \alpha : B_0 \to I\}\,.$$

Let W be the set of variables appearing in H_0. Then the result of evaluating the MV query Q_0, M on instance I is defined to be the multiset with ground set $Q_0(I)$, where for each fact $e \in Q_0(I)$; the multiplicity of e in the multiset is defined to be

$$\#\{\gamma|_M \mid \gamma : B_0 \to I \quad \text{and} \quad \gamma(H_0) = e\}\,.$$

That is, given a fact $\alpha(H_0) \in Q_0(I)$, there may be many different matchings γ that agree with α on H_0. The multiplicity of $\alpha(H_0)$ is defined to be not the *total* number of different such matchings γ, but rather the number of different *restrictions* one obtains when restricting these matchings γ to M.[4]

Two MV queries are *equivalent* if they evaluate to the same multiset on every input instance. Equivalence of MV queries can be characterized using the notion of multiset homomorphism [14]. A *multiset homomorphism* from MV query Q_0, M to MV query Q_0', M' is a homomorphism $h : Q_0 \to Q_0'$ such that h is injective on M and $h(M) \subseteq M'$. Cohen showed the following:

Theorem 1 ([14, Thm 5.3]) *Two MV queries are equivalent if and only if there are multiset homomorphisms between them in both directions.*

To leverage this result on MV equivalence, we associate two MV queries with our given sifo CQs in the following way.

Definition 7 Fix a new relation symbol T_0 of arity the length of $\bar{x}$. The MV queries $\tilde{Q}$ and $\tilde{Q}'$ are defined as $Q_0, (Z - X)$ and $Q_0', (Z - X)$, respectively, where

$$Q_0 = T_0\,(\bar{x}) \leftarrow B$$
$$Q_0' = T_0\,(\bar{x}) \leftarrow B'$$

The following proposition now relates oid-equivalence to MV equivalence:

Proposition 2 *If Q and Q' are oid-equivalent, then the MV queries $\tilde{Q}$ and $\tilde{Q}'$ are equivalent.*

Proof Let I be an instance. We must show that the multisets $\tilde{Q}(I)$ and $\tilde{Q}'(I)$ are equal. Since Q and Q' are oid-equivalent, the ground sets $Q_0(I)$ and $Q_0'(I)$ of $\tilde{Q}(I)$ and $\tilde{Q}'(I)$ are already equal. We must show that the element multiplicities are the same as well.

Let $T_0(\bar{c})$ be an arbitrary element of $Q_0(I)$. By the semantics of oCQs, we have the following equalities:

$$\#_{\bar{c}}(Q, I) = \#\{\gamma|_{X\cup Z} \mid \gamma : B \to I \quad \text{and} \quad \gamma\,(\bar{x}) = \bar{c}\}$$
$$\#_{\bar{c}}\left(Q', I\right) = \#\left\{\gamma|_{X\cup Z} \mid \gamma : B' \to I \quad \text{and} \quad \gamma\,(\bar{x}) = \bar{c}\right\}$$

Since $Q(I)$ and $Q'(I)$ are oid-isomorphic, the left-hand sides of the above two equalities are equal. Hence, the right-hand sides are equal as well. But these are precisely the multiplicities of $T_0(\bar{c})$ in $\tilde{Q}(I)$ and $\tilde{Q}'(I)$, respectively. □

The following proposition further relates MV equivalence to equivalence of the flattenings up to permutation:

Proposition 3 *If the MV queries $\tilde{Q}$ and $\tilde{Q}'$ are equivalent, then there exists a permutation π of $Z - X$ such that $\mathring{Q}^{\pi}$ and $\mathring{Q}'$ are equivalent.*

Proof By Theorem 1, there exist a multiset homomorphism h from $\tilde{Q}$ to $\tilde{Q}'$ and a multiset homomorphism h' from $\tilde{Q}'$ to $\tilde{Q}$. Since Theorem 1 also implies that h is injective on $Z - X$ and that $h(Z - X) \subseteq Z - X$, we can conclude that h acts as a permutation on $Z - X$. Moreover, h is the identity on X. The same two properties hold for h'.

Now put $\pi = (h|_{Z-X})^{-1}$. Then $h : \mathring{Q}^{\pi} \to \mathring{Q}'$. So it remains to find a homomorphism $h'' : \mathring{Q}' \to \mathring{Q}^{\pi}$. Thereto,

[4] The motivation for MV queries was to model the semantics of positive SQL queries with nested `EXISTS` subqueries. While queries under standard SQL semantics return multisets of tuples, only the relations mentioned in the top level SQL block contribute to the multiplicities of answers, whereas relations mentioned in the subquery do not.

note that $h'h$ acts as a permutation on $Z - X$. Since $Z - X$ is finite, there exists a nonzero natural number m such that $(h'h)^m$ is the identity on $Z - X$. Equivalently, $(h'h)^{m-1}h'$ equals π on $Z - X$. We conclude that $(h'h)^{m-1}h'$ is the desired homomorphism h''. □

We summarize the three preceding Propositions in the following.

Theorem 2 *Consider two sifo CQs*

$$Q = T(\bar{x}, f(\bar{z})) \leftarrow B \quad (3)$$
$$Q' = T(\bar{x}, f'(\bar{z})) \leftarrow B' \quad (4)$$

where Q and Q' have identical tuples $\bar{x}$ and $\bar{z}$ of distinguished and creation variables and where $\bar{z}$ contains no variable more than once. Denote the sets of distinguished and creation variables by X and Z, respectively.

The following are equivalent:

1. *The sifo CQs Q and Q' are oid-equivalent;*
2. *The MV queries $\tilde{Q}$ and $\tilde{Q}'$ are equivalent;*
3. *There is a permutation π of $Z - X$ such that the classical CQs $\mathring{Q}^\pi$ and $\mathring{Q}'$ are equivalent.*

4.5 Computational complexity

The results of this section imply the following:

Corollary 1 *Testing oid-equivalence of sifo CQs is NP-complete.*

Proof Assume given sifo CQs Q and Q' with the same head predicate:

$$Q = T(\bar{x}, f(\bar{z})) \leftarrow B$$
$$Q' = T(\bar{x}', f'(\bar{z}')) \leftarrow B'.$$

Let X, X', Z, and Z' denote the sets of variables occurring in $\bar{x}$, $\bar{x}'$, $\bar{z}$ and $\bar{z}'$, respectively.

To test oid-equivalence, we begin by removing duplicates in $\bar{z}$ and $\bar{z}'$, as justified by Lemma 1. Note that $\bar{x}$ and $\bar{x}'$ have the same length k, because of the fixed arity of T. So we can write $\bar{x} = x_1, \ldots, x_k$ and $\bar{x}' = x'_1, \ldots, x'_k$. Consider the mapping $\sigma = \{(x_1, x'_1), \ldots, (x_k, x'_k)\}$. We test whether σ is a bijection from X to X'; if not, then Q and Q' are not oid-equivalent by Lemma 2. If σ is a bijection, we can safely replace every variable x' in X' by $\sigma^{-1}(x')$, which yields a sifo CQ that is oid-equivalent to Q'. Hence, from now on we may assume that $\bar{x} = \bar{x}'$ and in particular $X = X'$.

Next, we test whether $X \cap Z = X \cap Z'$ and whether $Z - X$ and $Z' - X$ have the same cardinality; if one of the two tests fails then Q and Q' are not oid-equivalent by Lemmas 3 and 4. Otherwise, we can rename the variables in $Z' - X$, so that we may assume that $\bar{z} = \bar{z}'$.

We are now left in the situation where Q and Q' are in the general forms (3) and (4) from Sect. 4.4, to which Theorem 2 applies. By the third statement of this theorem, we can test oid-equivalence of Q and Q' in NP by guessing a permutation π and two homomorphisms between $\mathring{Q}^\pi$ and $\mathring{Q}'$ in both directions.

NP-hardness follows immediately because the problem has equivalence of classical CQs as a special case, which is well known to be NP-hard. Indeed, oid-equivalence of sifo CQs Q and Q' in the special case where the creation functions are nullary amounts to classical equivalence when we ignore the function terms in the heads.

5 Logical entailment of sifo CQs interpreted as schema mappings

Object-creating CQs, and sifo CQs in particular, can also be interpreted alternatively as schema mappings rather than as queries. Specifically, consider a sifo CQ Q of the general form $T(\bar{x}, f(\bar{z})) \leftarrow B$ over the database schema $\mathbf{S}$. Let $\bar{v}$ be the sequence of all variables used in B. Then we may view Q as a second-order implicational statement over the augmented schema $\mathbf{S} \cup \{T\}$, as follows:

$$\exists f \forall \bar{v}(B \rightarrow H)$$

where H is the head and B is conveniently used to stand for the conjunction of its elements. Note that this formula is second order because it existentially quantifies a function f; we denote the above formula by sotgd(Q). This formula belongs to the well-known class of second-order tuple-generating dependencies (SO-tgds). More specifically, it is a *plain* SO-tgd [7].

Syntactically, the plain SO-tgds coming from sifo CQs in this manner form a restricted class of SO-tgds, defined by the following restrictions:

- Plain SO-tgd may consist of multiple rules; sifo CQs consist of a single rule.
- The head of a plain SO-tgd may consist of multiple atoms; the head of a sifo CQ consists of a single atom (this is similar to GAV mappings [12,27], although the classical notion of GAV mapping does not use function symbols).
- There is only one function symbol, which moreover can be applied only once in the head.

When interpreting a sifo CQ Q as an SO-tgd, the semantics becomes that of a schema mapping. Specifically, let I be an instance over $\mathbf{S}$, considered as a source instance, and let J be an instance over $\{T\}$, considered as a target instance.

Table 6 Instances J_1 and J_2 from Example 8

Family	
beth	jones
ben	jones
eric	simpson
emma	smith

Family	
beth	jones
ben	jones
eric	jones
emma	jones

Table 7 Instance J_3 from Example 8

Family	
beth	jones
ben	murphy
eric	simpson
emma	smith

Then (I, J) together form an instance over the augmented schema $\mathbf{S} \cup \{T\}$. Now we say that (I, J) satisfies Q, denoted by $(I, J) \models Q$, if the structure $(\text{adom}(I) \cup \text{adom}(J), I, J)$ satisfies $\text{sotgd}(Q)$ under the standard semantics of second-order logic, using $\text{adom}(I) \cup \text{adom}(J)$ as the universe of the structure.

The following example and remark illustrate that the semantics of sifo CQs as SO-tgds is quite different from their semantics as object-creating queries.

Example 8 Let us consider again our query from Example 1. As we have mentioned in Sect. 1, we can now write it as an SO-tgd as follows:

$$\exists f \forall x \forall y \forall c \, (\mathit{Mother}(c, x) \wedge \mathit{Father}(c, y) \rightarrow \mathit{Family}(c, f(x, y)))$$

Take the instance I consisting of the Mother and Father facts listed in Table 1, and take the instances J_1 and J_2 consisting of the Family facts listed in Table 6 left and right, respectively. Then both pairs (I, J_1) and (I, J_2) satisfy the SO-tgd. For J_1, this is witnessed by the following function f:

x	y	$f(x, y)$
anne	adam	jones
claire	carl	simpson
diana	carl	smith

For J_2, this is witnessed by the function that simply maps everything to jones.

In contrast, for J_3 consisting of the Family facts listed in Table 7, the pair (I, J_3) does not satisfy the SO-tgd. Indeed, suppose there would exist a function f witnessing the truth of the formula on (I, J_3). Since beth has anne as mother and adam as father, the fact

$$\text{Family}(\text{beth}, f(\text{anne}, \text{adam}))$$

must belong to J_3. The only family fact with beth in the first position is

$$\text{Family}(\text{beth}, \text{jones}),$$

so we conclude

$$f(\text{anne}, \text{adam}) = \text{jones}.$$

Furthermore, since ben also has anne as mother and adam as father, the fact

$$\text{Family}(\text{ben}, f(\text{anne}, \text{adam}))$$

must be in J_3. The only family fact with ben in the first position is

$$\text{Family}(\text{ben}, \text{murphy}),$$

however, we must conclude that

$$f(\text{anne}, \text{adam}) = \text{murphy},$$

which is in contradiction with the previous conclusion.

Remark 1 Note that, by the purely implicational nature of SO-tgds, if (I, J) satisfies an SO-tgd and $J \subseteq J'$, then also (I, J') satisfies the SO-tgd. Hence, continuing the previous example, for any instance J' obtained by J_1 or J_2 by adding some more Family facts, the pair (I, J') would still satisfy the SO-tgd from the example.

The above example and remark show that given a source instance I, there are in general multiple possible target instances J such that $(I, J) \models Q$. This is in contrast to the semantics of Q as an oCQ, where $Q(I)$ is an extended instance that is uniquely defined. Still, there is a connection between the oCQ semantics and the SO-tgd semantics. Specifically, $Q(I)$ can be viewed as a target instance in a canonical manner, using *oid-to-constant assignments* (oc-assignments for short) defined as follows.

Definition 8 Let I be a source instance and let J be an extended instance over $\{T\}$ such that $\text{consts}(J) \subseteq \text{adom}(I)$. An *oc-assignment* for J with respect to I is an injective mapping $\rho : \text{oids}(J) \rightarrow \mathbf{dom}$ so that the image of ρ is disjoint from $\text{adom}(I)$.

Thus, ρ assigns to each non-constant data term from J a different constant that is not in $\text{adom}(I)$.

We now observe the following obvious property giving a connection between the oCQ semantics and the SO-tgd semantics:

Proposition 4 *Let I be a source instance and let ρ be an oc-assignment for $Q(I)$ with respect to I. Then $(I, \rho(Q(I))) \models Q$.*

In fact, $Q(I)$ corresponds to what Fagin et al. [18] call the *chase of I with* sotgd(Q).

5.1 Nested dependencies

We have introduced sifo CQs as a restricted class of plain SO-tgds. But actually, sifo CQs can also be considered as a restricted form of so-called nested tgds [8]. Thereto, consider again a sifo CQ of the general form $T(\bar{x}, f(\bar{z})) \leftarrow B$. Let $\bar{u}$ be the sequence of all variables from B, except for the creation variables (the variables from $\bar{z}$). Furthermore, let w be a fresh variable not occurring in B, and let H' be the atom $T(\bar{x}, w)$. We can now associate with Q the following implicational statement, denoted by ntgd(Q):

$$\forall \bar{z} \exists w \forall \bar{u} \left(B \rightarrow H'\right)$$

Note that ntgd(Q) is now a first-order formula, but it is clear that ntgd(Q) is logically equivalent to sotgd(Q). Hence, the schema mappings arising from sifo CQs are not essentially second order in nature.

5.2 Logical entailment

In Sect. 4, we have shown that equivalence of sifo CQs as object-creating queries is decidable. Now that we have seen that sifo CQs can also be given a semantics as schema mappings; we may again ask if equivalence under this alternative semantics is decidable. The answer is affirmative; we have seen in the previous subsection that sifo CQ mappings belong to the class of nested dependencies, and logical implication of nested dependencies has recently been shown to be decidable [26]. When this general implication test for nested dependencies is applied specifically to sifo CQ schema mappings, it can be implemented in non-deterministic polynomial time. Hence, logical entailment (and also logical equivalence) of sifo CQ schema mappings is NP-complete.

In the present section, we present a specialized logical entailment test for sifo CQ schema mappings which is much simpler and more elegant and provides more insight into the problem by relating it to testing implication of a join dependency by a CQ (Theorem 3). Interestingly, there is a striking correspondence between the general implication test when applied to sifo CQs and the strategy we use to prove our theorem. An in-depth comparison will be given in Sect. 6, after we have stated the theorem formally and have seen its proof.

Formally, given two schema mappings $\mathcal{M}$ and $\mathcal{M}'$ from a source schema $\mathbf{S}$ to a target schema $\{T\}$, we say that $\mathcal{M}$ *logically entails* $\mathcal{M}'$ if the following implication holds for every instance I over $\mathbf{S}$ and every instance J over $\{T\}$:

(I, J) satisfies $\mathcal{M} \Rightarrow (I, J)$ satisfies $\mathcal{M}'$.

Referring to the view of sifo CQs as SO-tgds introduced above, we now define:

Definition 9 Let Q and Q' be two sifo CQs with the same head predicate and over the same database schema. We say that Q *logically entails* Q' if sotgd(Q) logically entails sotgd(Q').

Example 9 Recall the sifo CQs Q and Q' from Example 6:

$$Q = T(x, f(y)) \leftarrow R(x, y, z)$$
$$Q' = T\left(x, f'(x, y)\right) \leftarrow R(x, y, z)$$

It is clear that Q logically entails Q'. Indeed, if there exists a function f witnessing the truth of sotgd(Q), then we can easily define a function f' witnessing the truth of sotgd(Q') by defining $f'(x, y) := f(y)$.

Conversely, however, Q' does not logically entail Q. Indeed, Table 8 shows (I, J) where $(I, J) \models Q'$ but $(I, J) \not\models Q$.

Table 8 Instances used in Example 9

I			J	
a_1	b	c	a_1	d_1
a_2	b	c	a_2	d_2

Example 10 Recall the sifo CQs Q and Q' from Example 7:

$$Q = T(x, f(x)) \leftarrow R(x, y, z)$$
$$Q' = T\left(x, f'(x, y, z)\right) \leftarrow R(x, y, z)$$

Although Q and Q' are not oid-equivalent, they are logically equivalent: They logically entail each other. The logical entailment of Q' by Q is again clear. To see the converse direction, assume f' witnesses the truth of sotgd(Q'). Then we define $f(x)$ for any x as follows: If there exists a pair (y, z) such that $R(x, y, z)$ holds, we fix one such pair (y, z) arbitrarily and define $f(x) := f'(x, y, z)$. If no such y and z exist, we may define $f(x)$ arbitrarily. It is now clear that this f witnesses the truth of sotgd(Q).

Example 11 Consider the sifo CQs:

$$Q = T(x, f(z_1)) \leftarrow R(z_1, x), R(z_1, z_2)$$
$$Q' = T\left(x, f'(z_1, z_2)\right) \leftarrow R(z_1, x), R(z_1, z_2)$$

Also here, Q and Q' logically entail each other. The logical entailment of Q' by Q is again clear. To see the converse

direction, we can use a reasoning similar to that used in Example 10. Assume f' witnesses the truth of sotgd(Q'). Then we define $f(z_1)$ for any z_1 as follows: If there exists z_2 such that $R(z_1, z_2)$ holds, we fix one such z_2 arbitrarily and define $f(z_1) := f'(z_1, z_2)$. If no such z_2 exists, we may define $f(z_1)$ arbitrarily. The function f thus defined witnesses the truth of sotgd(Q).

Note that the kind of reasoning used here and in Example 10 does not work in the case of Example 9. In Theorem 3, we will characterize formally when this kind of reasoning is correct.

Example 10 shows that logical equivalence (logical entailment in both directions) does not imply oid-equivalence of sifo CQs. We will see in Theorem 4 that the other direction does hold.

5.3 Join dependencies and tableau queries

In our characterization of sifo CQ logical entailment, we use a number of concepts from classical relational database theory [2], which we recall here briefly.

Recall that a *relation scheme* is a finite set of elements called *attributes*. It is customary to denote the union of two relation schemes X and Y by juxtaposition, thus writing XY for $X \cup Y$.

A *tuple* over a relation scheme U is a function from U to **dom**. A *relation* over U is a finite set of tuples over U.

Let t be a tuple over U and let $X \subseteq U$. The restriction of t to X is denoted by $t[X]$. The *projection* $\pi_X(r)$ of a relation r over U equals $\{\, t[X] \mid t \in r \,\}$.

We now turn to tableau queries, which are an alternative formalization of conjunctive queries so that the result of a query is a set of tuples rather than a set of facts. Let **S** be a database schema, and let B be a finite set of atoms with predicates from **S**, as would be the body of a CQ over **S**. Let $V = \mathrm{var}(B)$. For any $U \subseteq V$, the pair (B, U) is called a *tableau query* over **S**. When applied to an instance I over **S**, this tableau query returns a relation over U in the following manner. Let $\mathrm{Mat}(B, I)$ be the set of all matchings of B in I. Using variables for attributes, V can be viewed as a relation scheme. Under this view, every valuation on V is a tuple over V, and thus, $\mathrm{Mat}(B, I)$ is a relation over V. We now define the result of (B, U) on input I to be $\pi_U(\mathrm{Mat}(B, I))$. This result is denoted by $(B, U)(I)$.

We finally recall join dependencies. Let t_1 and t_2 be tuples over the relation schemes U_1 and U_2, respectively. If t_1 and t_2 agree on $U_1 \cap U_2$, the union $t_1 \cup t_2$ (where we take the union of two functions, viewed as sets of pairs) is a well-defined tuple over the relation scheme U_1U_2. The *natural join* $r_1 \bowtie r_2$, for relations r_1 and r_2 over U_1 and U_2, respectively, then equals

$$\{t_1 \cup t_2 \mid t_1 \in r_1 \;\&\; t_2 \in r_2 \;\&\; t_1[U_1 \cap U_2] = t_2[U_1 \cap U_2]\}.$$

Consider now any relation r over some relation scheme U. Let U_1 and U_2 be subsets of U (not necessarily disjoint) such that $U = U_1U_2$. Then r satisfies the *join dependency* (JD) $U_1 \bowtie U_2$ if $r = \pi_{U_1}(r) \bowtie \pi_{U_2}(r)$. Note that the containment from left to right is trivial, so one only needs to verify the containment $\pi_{U_1}(r) \bowtie \pi_{U_2}(r) \subseteq r$.

The logical implication of JDs by tableau queries is well understood and can be solved by the chase procedure with NP complexity [2,25]. Formally, a tableau query $Q = (B, U)$ over **S** is said to *imply* a JD over U if for every instance I over **S**, the relation $Q(I)$ satisfies this JD.

5.4 Decidability of sifo CQ logical entailment

We consider two sifo CQs Q and Q' with the same head predicate:

$$Q = T(\bar{x}, f(\bar{z})) \leftarrow B$$
$$Q' = T(\bar{x}', f'(\bar{z}')) \leftarrow B'$$

Remark 2 We assume Q and Q' to have their function symbol in the same position in the head (here taken to be the last position). This is justified because otherwise Q could never logically entail Q'. In proof, suppose the function symbol in the head of Q' would not be in the last position. Then we have a variable x' from B' in the last position. Now consider an instance I such that both $Q(I)$ and $Q'(I)$ are non-empty (such an instance could be constructed by taking the disjoint union of B and B' and substituting constants for variables). Let ρ by an oc-assignment for $Q(I)$ with respect to I. By Proposition 4, we have $(I, \rho(Q(I))) \models Q$. In $\rho(Q(I))$, none of the elements in the last position of a T-fact belongs to $\mathrm{adom}(I)$. But then $(I, \rho(Q(I)))$ cannot satisfy Q'. Indeed, since $Q'(I)$ is non-empty, there is a matching $\alpha' : B' \to I$. In any J' such that $(I, J') \models Q'$, there needs to be a T-fact with $\alpha'(x')$ in the last position, and $\alpha'(x') \in \mathrm{adom}(I)$. We conclude that Q does not logically entail Q'.

In what follows we use X, Z and Z' to denote the sets of variables appearing in the tuples $\bar{x}$, $\bar{z}$ and $\bar{z}'$, respectively.

We establish:

Theorem 3 *Q logically entails Q' if and only if there exists a homomorphism $h : B \to B'$ satisfying the following conditions:*

1. $h(\bar{x}) = \bar{x}'$;
2. $h(X \cap Z) \subseteq Z'$;
3. *Let* $Y_h := h^{-1}(Z')$, *i.e.,*

$$Y_h = \left\{ y \in \mathrm{var}(B) \mid h(y) \in Z' \right\}.$$

Then the tableau query (B, XY_hZ) implies the join dependency $XY_h \bowtie Y_hZ$.

5.4.1 Proof of sufficiency

Let $(I, J) \models Q$, witnessed by the function f. We must show $(I, J) \models Q'$. This means finding a function f' witnessing the truth of sotgd(Q') in (I, J).

Call any two matchings $\alpha_1, \alpha_2 \in \text{Mat}(B, I)$ equivalent if they agree on Y_h. This is denoted by $\alpha_1 \equiv \alpha_2$. Let ρ be any function from $\text{Mat}(B, I)$ to $\text{Mat}(B, I)$ with the two properties, first, that $\rho(\alpha) \equiv \alpha$ and, second, that $\alpha_1 \equiv \alpha_2$ implies $\rho(\alpha_1) = \rho(\alpha_2)$. Thus, ρ amounts to choosing a representative out of each equivalence class. We denote the application of ρ by subscripting, writing $\rho(\alpha)$ as ρ_α.

Let us define f' as follows. Take any matching $\beta : B' \to I$. Then we put $f'(\beta(\bar{z}')) := f(\rho_{\beta\circ h}(\bar{z}))$. To see that this is well defined, recall that $h(Y_h) \subseteq Z'$. Hence, $\beta_1(\bar{z}') = \beta_2(\bar{z}')$ implies that $\beta_1 \circ h \equiv \beta_2 \circ h$, so $\rho_{\beta_1\circ h} = \rho_{\beta_2\circ h}$.

We now show that this interpretation of f' satisfies the requirements. Specifically, let $\beta : B' \to I$ be a matching. We must show that $T(\beta(\bar{x}'), f'(\beta(\bar{z}'))) \in J$. Consider the valuations $\beta_1 = \beta \circ h$ and $\beta_2 = \rho_{\beta\circ h}$, both belonging to $\text{Mat}(B, I)$, and viewed as tuples over the relation scheme $\text{var}(B)$. Since these two tuples agree on Y_h, also the two restrictions $\beta_1[Y_h X]$ and $\beta_2[Y_h Z]$ agree on Y_h. Since $X \cap Z \subseteq Y_h$, the union $\beta_1[Y_h X] \cup \beta_2[Y_h Z]$ is a well-defined tuple over $XY_h Z$. Since $\pi_{XY_h Z}(\text{Mat}(B, I))$ satisfies the JD $Y_h X \bowtie Y_h Z$, the union belongs to $\pi_{XY_h Z}(\text{Mat}(B, I))$. Hence, there exists a valuation $\gamma \in \text{Mat}(B, I)$ that agrees with $\beta \circ h$ on X, and with $\rho_{\beta\circ h}$ on Z. Since $(I, J) \models Q$, we have $T(\gamma(\bar{x}), f(\gamma(\bar{z}))) \in J$. By the preceding, $\gamma(\bar{x}) = \beta(h(\bar{x}))$ and $\gamma(\bar{z}) = \rho_{\beta\circ h}(\bar{z}) = g(\beta(\bar{z}'))$. We conclude that $T(\beta(\bar{x}'), g(\beta(\bar{z}'))) \in J$ as desired.

5.4.2 Proof of necessity

Let $V' = \text{var}(B')$, and let n be the arity of f. For each $l \in \{0, 1, \ldots, n\}$ and each $u \in V' - Z'$, we introduce a fresh copy of u, denoted by u^l. We say that this fresh copy is 'colored' with color l. For each variable $u \in Z'$, we simply define u^l to be u itself. We say that the variables in Z' are 'colored white.'

For any tuple of variables $\bar{u} = (u_1, \ldots, u_p)$ in V', we denote the tuple $(u_1^l, \ldots, u_p^l)$ by $\bar{u}^l$. In this tuple, all variables are colored l or white. We then define $B'^l = \{ R(\bar{u}^l) \mid R(\bar{u}) \in B' \}$ and view it as an instance, i.e., the variables u^l are considered to be constants.

Now define the instance $I = \bigcup_{l=0}^{n} B'^l$, and construct the instance $J = Q(I)$. By Proposition 4, $(I, J) \models Q$, where we omit the oc-assignment for the sake of clarity. Since Q logically entails Q', also $(I, J) \models Q'$. Hence, there exists a function f' such that for each color l, using the matching $\text{id}^l : B' \to I, u \mapsto u^l$, the fact $T(\bar{x}'^l, f'(\bar{z}'^l)) = T(\bar{x}'^l, f'(\bar{z}'))$ belongs to J.

Since $J = Q(I)$, we have $f'(\bar{z}') = f(\bar{w})$ for some tuple $\bar{w}$ of colored variables in V'. Since the arity of f is n and there are $n + 1$ distinct colors, some color does not appear in $\bar{w}$. Without loss of generality we may assume that this is the color 0.

Let us now focus on the fact $T(\bar{x}'^0, f(\bar{w}))$ in J. Like any T-fact in J, this fact has been produced by some matching $k : B \to I$ such that $T(\bar{x}'^0, f(\bar{w})) = k(T(\bar{x}, f(\bar{z})))$, so

(a) $k(\bar{x}) = \bar{x}'^0$ and
(b) $k(\bar{z}) = \bar{w}$.

Let s denote the mapping that removes colors, i.e., $s(u^l) = u$ for every $u \in V'$ and every $l \in \{0, 1, \ldots, n\}$. Since $s(I) \subseteq B'$, we have a homomorphism $s \circ k : B \to B'$. We now define $h := s \circ k$ and show that it satisfies the conditions required by the Theorem. The first condition is clear since $h(\bar{x}) = s(k(\bar{x})) = s(\bar{x}'^0) = \bar{x}'$.

For the second condition, let $x \in X \cap Z$. By (a), $k(x)$ is colored 0 or white. By (b), $k(x)$ is colored nonzero or white. Hence, $k(x)$ is colored white, i.e., $k(x) \in Z'$, so $h(x) = s(k(x)) = k(x) \in Z'$ as desired.

Finally, to show that $(B, XY_h Z)$ implies $XY_h \bowtie Y_h Z$, we must establish the query containment

$$(B, XY_h) \bowtie (B, Y_h Z) \subseteq (B, XY_h Z)\,.$$

Treating tableau queries as conjunctive queries and using the well-known containment criterion for conjunctive queries, this amounts to showing the existence of a certain homomorphism. More specifically, we express the query $(B, XY_h) \bowtie (B, Y_h Z)$ by the CQ with the body $B_2 = B_0 \cup B_1$ defined as follows. The body B_0 is obtained from B by replacing each variable u not in Y_h by a fresh copy u^0. For each $u \in Y_h$, we define u^0 simply as u itself. The body B_1 is obtained from B by replacing each variable not in Y_h by a fresh copy u^1. Again, for each $u \in Y_h$ we define u^1 simply as u itself. To show the containment, we now must find a homomorphism m from B to B_2 such that each $u \in X - Y_h$ is mapped to u^0; each $u \in Y_h$ is mapped to u; and each $u \in Z - Y_h$ is mapped to u^1.

Thereto, we define the following mapping m:

- if $k(u)$ is colored 0, then $m(u) := u^0$;
- if $k(u)$ is colored l for some $l > 0$, then $m(u) := u^1$;
- if $k(u)$ is colored white, then $m(u) := u$.

Let us verify that $m : B \to B_2$ is a homomorphism. Consider an atom $R(\bar{u})$ in B; we must show $R(m(\bar{u})) \in B_2$. Since $k : B \to I$, we know that $R(k(\bar{u})) \in I$. By definition of I, this means that $R(k(\bar{u})) = R(\bar{v}^l)$ for some atom $R(\bar{v})$ in B' and some color l. So, for each variable u in $\bar{u}$, the color of $k(u)$ is either l or white. We now distinguish two cases.

- If $k(u)$ is colored white, then $h(u) = k(u) \in Z'$ so $u \in Y_h$. Hence, in this case, $m(u) = u = u^0 = u^1$.
- If $k(u)$ is colored l, then by definition $m(u) = u^0$ when $l = 0$, and $m(u) = u^1$ when $l > 0$.

We conclude that $R(m(\bar{u})) = R(\bar{u}^0) \in B_0$ when $l = 0$, and $R(m(\bar{u})) = R(\bar{u}^1) \in B_1$ when $l > 0$. Hence, since $B_2 = B_0 \cup B_1$, we always have $R(m(\bar{u})) \in B_2$ as desired.

It remains to verify that m maps the variables in XY_hZ correctly. If $u \in Y_h$, then $h(u) = k(u) \in Z'$ so $k(u)$ is colored white and $m(u) = u$ as desired. If $u \in X - Y_h$, then by (a), $k(u)$ is colored 0 so $m(u) = u^0$ as desired. Finally, if $u \in Z - Y_h$, then by (b), $k(u)$ is colored $l > 0$ so $m(u) = u^1$ as desired.

As a corollary, we obtain that the complexity of deciding logical entailment for sifo CQs is not worse than that of deciding containment for classical CQs:

Corollary 2 *Testing logical entailment of sifo CQs is NP-complete.*

Proof Membership in NP follows from Theorem 3; as a witness for logical entailment, we can use a homomorphism h satisfying the first two conditions of the theorem, together with a homomorphism h_0 from the query (B, XY_hZ) to the query $(B, XY_h) \bowtie (B, Y_hZ)$ witnessing the third condition of the theorem. NP-hardness follows because the problem has containment of classical CQs as a special case, which is well known to be NP-hard. Indeed, logical entailment of a sifo Q' by a sifo Q, in the special case where the creation functions of Q and Q' are nullary, amounts to classical containment of Q in Q' when we ignore the function terms in the heads. □

5.5 From oid-equivalence to logical entailment

Let Q and Q' be sifo CQs of the general forms (3) and (4) from Sect. 4.4. From our main Theorems 2 and 3, we can conclude the following.

Theorem 4 *If Q and Q' are oid-equivalent, then Q logically entails Q'.*

Proof By Theorem 2, there exists a permutation π of $Z - X$ such that $\mathring{Q}^\pi$ and $\mathring{Q}'$ are equivalent. Hence there is a homomorphism $h : \mathring{Q}^\pi \to \mathring{Q}'$. Clearly $h : B \to B'$. We verify that h satisfies the conditions of Theorem 3, thus showing that Q logically entails Q'.

1. Since h maps the head of $\mathring{Q}^\pi$ to the head of $\mathring{Q}'$, we have $h(\bar{x}) = \bar{x}$ and $h(\pi(\bar{z})) = \bar{z}$. Since $\bar{x}' = \bar{x}$, we have $h(\bar{x}) = \bar{x}'$ as desired.
2. Since h is the identity on X, we have $h(X \cap Z) = X \cap Z \subseteq Z = Z'$ as desired.
3. Since $h(\pi(\bar{z})) = \bar{z}$ and $\pi(Z) = Z$, we have $h(Z) = Z = Z'$. Hence, $Z \subseteq Y_h$. But then the join dependency $XY_h \bowtie Y_hZ$ becomes $XY_h \bowtie Y_h$ which trivially holds.

6 Discussion

The results in this paper provide an understanding of the notions of oid-equivalence and logical entailment for sifo CQs. Sifo CQs, however, form a very simple subclass of oCQs. Moreover, oCQs themselves are rather limited; for example, they consist of a single rule and the rule can have only one atom in the head. Thus, there are at least three natural directions for further research: (i) allowing more than one function in the head; (ii) allowing more than one atom in the head; (iii) allowing more than one rule.

6.1 Containment

Furthermore, in addition to oid *equivalence* of oCQs, it would be natural to also investigate a notion of oid-*containment*. There are actually at least two reasonable ways to define such a notion. The situation is similar to that in research on CQs with counting or bag semantics [14,15]. Most of the known results are for equivalence only, with the extension to containment typically an open problem. Indeed, our characterization of oid-equivalence for sifo CQs relies on equivalence of CQs with bag semantics. An extension to oid-containment will likely need a similar advance on containment of CQs with bag semantics.

6.2 Sifo CQs and ILOG

In the introduction, we mentioned that sifo CQs, and oCQs in general, are a fragment of ILOG without recursion [22]. Sifo CQs belong to the subclass of the class of recursion-free ILOG programs 'with isolated oid creation' [23]. For this class, oid-equivalence was already known to be decidable. This was shown by checking all finite instances up to some exponential size. Hence, our NP-completeness result for oid-equivalence of sifo CQs does not follow from the previous work. More generally, the decidability of oid-equivalence for general recursion-free ILOG programs, or already of oCQs for that matter, is a long-standing open question. Various interesting examples showing the intricacies of this problem have already been given by Hull and Yoshikawa [23].

6.3 Sifo CQs and nested dependencies

In Sect. 5.1, we also presented sifo CQs, now viewed as schema mappings, as a very simple subclass of nested tgds. The implication problem for general nested tgds was shown to be decidable by Kolaitis et al. [26] in work done

independently from the present paper. Nevertheless, our characterization of implication for sifo CQs, given by Theorem 3, does not follow from the general decision procedure for nested tgds. Instead, the general procedure, when applied to two sifo CQs, is strikingly similar to our proof of necessity of our theorem. Using the notation from that proof, the general procedure applied to test implication of sifo CQ Q' by sifo CQ Q would amount to testing for the existence of a homomorphism h from $\{T(\bar{x}'^{l}, f'(\bar{z}')) \mid l = 0, \ldots, n\}$ to $Q(I)$. Since $Q(I) = \{T(\alpha(\bar{x}), f(\alpha(\bar{z}))) \mid \alpha : B \to I\}$, this can be implemented by guessing h and $n+1$ matchings $\alpha_l : B \to I$ such that $(h(\bar{x}'^{l}), f'(h(\bar{z}'))) = (\alpha_l(\bar{x}), f(\alpha_l(\bar{z})))$ for $l = 0, \ldots, n$. In contrast, as explained in Corollary 2, our characterization involves guessing just two homomorphisms.

6.4 Sifo CQs and plain SO-tgds

As described in Sect. 5, sifo CQs are a very simple subclass of plain SO-tgds. For plain SO-tgds, deciding logical equivalence is again an open problem. Also, the notion of oid-equivalence, defined in this paper for oCQs, can be readily extended to plain SO-tgds. We illustrate some difficulties involved in allowing multiple functions in the head, which is indeed allowed in plain SO-tgds. First, consider the oid-equivalence problem. For sifo CQs, we have shown in Sect. 4.4 of this paper that as far as oid-equivalence is concerned, only the *counts* of generated oids per tuple are important. Now consider the following pair of oCQs:

$$Q = T\,(x, f(y), g(x, z)) \leftarrow R(x, y), R(x, z)$$
$$Q' = T\,(x, f(y), g(x, y)) \leftarrow R(x, y), R(x, z)$$

Both queries create the same number of new f- and g-oids per x-value, but now it also becomes important how these oids are paired. In Q, more pairs are generated for each x, and the two queries are not oid-equivalent. So, in the case of multiple functions, also the interaction between the multiple terms needs to be taken into account in some way.

A similar comment applies to the problem of logical equivalence. It is not immediately clear how the join dependency condition of Theorem 3 should be generalized in the presence of multiple functions. Consider, for example, the following:

$$\begin{aligned} Q &= T\,(x, f_1\,(z_1, y_1)\,, f_2\,(z_2, y_2)) \\ &\quad \leftarrow R\,(x, z_1, z_2)\,, S\,(z_1, y_1)\,, S\,(z_2, y_2) \\ Q' &= T\,(x, g_1(u), g_2(u)) \\ &\quad \leftarrow R(x, u, x), R(x, x, u), S(u, v_1), S(x, v_2) \end{aligned}$$

The f_1-part of Q (ignoring the third component in the head) logically entails the g_1-part of Q', and likewise, the f_2-part of Q (ignoring the second component in the head) logically entails the g_2-part of Q'. Globally, however, Q does not logically entail Q'; this can be seen by the instances given in Table 9, which satisfy Q but not Q'.

Table 9 Instances used to illustrate logical entailment in the presence of multiple functions

R	S	T
2 1 2	1 4	2 7 8
2 2 1	2 5	2 8 7
3 1 3	3 6	3 7 9
3 3 1		3 9 7

A related interesting question then is whether Theorem 4 that oid-equivalence implies logical entailment, still holds for plain SO-tgds. When we allow nested function terms in the head (which goes beyond plain SO-tgds) the implication breaks down, as shown by the following example [17, Example3.8]:

$$Q = T\,(x, f(x), g(f(x))) \leftarrow S(x)$$
$$Q' = T\,(x, f(x), g(x)) \leftarrow S(x)$$

where Q and Q' are oid-equivalent, and Q logically entails Q', but Q' does not logically entail Q.

Acknowledgments We thank the anonymous referees for their careful comments which helped improve the presentation of the paper. The work by Angela Bonifati has been partially supported by the ANR through the grant Datacert: Coq deep specification of security aware data integration (ANR-15-CE39-0009). The work by Werner Nutt has been partially supported by the grant CANDy of the Free University of Bozen-Bolzano.

References

1. Abiteboul, S., Buneman, P., Suciu, D.: Data on the Web: From Relations to Semistructured Data and XML. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2000)
2. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley, Reading (1995)
3. Abiteboul, S., Kanellakis, P.: Object identity as a query language primitive. J. ACM **45**(5), 798–842 (1998)
4. Abiteboul, S., Vianu, V.: Procedural languages for database queries and updates. J. Comput. Syst. Sci. **41**(2), 181–229 (1990)
5. Abiteboul, S., Vianu, V.: Datalog extensions for database queries and updates. J. Comput. Syst. Sci. **43**(1), 62–124 (1991)
6. Alexe, B., Tan, W.C., Velegrakis, Y.: STBenchmark: towards a benchmark for mapping systems. Proc. VLDB Endow. **1**(1), 230–244 (2008)
7. Arenas, M., Pérez, J., Reutter, J., Riveros, C.: The language of plain SO-TGDS: composition, inversion and structural properties. J. Comput. Syst. Sci. **79**(6), 737–1002 (2013)
8. Arocena, P., Glavic, B., Miller, R.: Value invention in data exchange. In: Proceedings of the SIGMOD Conference, pp. 157–168. ACM (2013)
9. Arocena, P.C., Ciucanu, R., Glavic, B., Miller, R.J.: Gain control over your integration evaluations. Proc. VLDB Endow. **8**(12), 1960–1971 (2015)

10. Van den Bussche, J., Paredaens, J.: The expressive power of complex values in object-based data models. Inf. Comput. **120**, 220–236 (1995)
11. Van den Bussche, J., Van Gucht, D., Andries, M., Gyssens, M.: On the completeness of object-creating database transformation languages. J. ACM **44**(2), 272–319 (1997)
12. ten Cate, B., Kolaitis, P.: Structural characterizations of schema-mapping languages. Commun. ACM **53**(1), 101–110 (2010)
13. Chandra, A., Merlin, P.: Optimal implementation of conjunctive queries in relational data bases. In: Proceedings of the 9th ACM Symposium on the Theory of Computing, pp. 77–90. ACM (1977)
14. Cohen, S.: Equivalence of queries that are sensitive to multiplicities. VLDB J. **18**, 765–785 (2009)
15. Cohen, S., Nutt, W., Sagiv, Y.: Containment of aggregate queries. In: Calvanese, D., Lenzerini, M., Motwani, R. (eds.) Database Theory—ICDT 2003. Lecture Notes in Computer Science, vol. 2572, pp. 111–125. Springer, Berlin (2003)
16. Fagin, R., Haas, L., M. Hernández, R.M., Popa, L., Velegrakis, Y.: Clio: schema mapping creation and data exchange. In: Borgida, A., Chaudhuri, V., Giorgini, P., Yu, E. (eds.) Conceptual Modeling: Foundations and Applications. Lecture Notes in Computer Science, vol. 5600, pp. 198–236. Springer, Berlin (2009)
17. Fagin, R., Kolaitis, P., Nash, A., Popa, L.: Towards a theory of schema-mapping optimization. In: Proceedings of the 27th ACM Symposium on Principles of Database Systems, pp. 33–42 (2008)
18. Fagin, R., Kolaitis, P., Popa, L.: Composing schema mappings: second-order dependencies to the rescue. ACM Trans. Database Syst. **30**(4), 994–1055 (2005)
19. Feinerer, I., Pichler, R., Sallinger, E., Savenkov, V.: On the undecidability of the equivalence of second-order tuple generating dependencies. Inf. Syst. **48**, 113–129 (2015)
20. Friedman, M., Levy, A.Y., Millstein, T.D.: Navigational plans for data integration. In: AAAI/IAAI, pp. 67–73 (1999)
21. Garcia-Molina, H., Papakonstantinou, Y., Quass, D., Rajaraman, A., Sagiv, Y., Ullman, J., Vassalos, V., Widom, J.: The TSIMMIS approach to mediation: data models and languages. J. Intell. Inf. Syst. **8**(2), 117–132 (1997)
22. Hull, R., Yoshikawa, M.: ILOG: declarative creation and manipulation of object identifiers. In: McLeod, D., Sacks-Davis, R., Schek, H. (eds.) Proceedings of the 16th International Conference on Very Large Data Bases, pp.455–468. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1990)
23. Hull, R., Yoshikawa, M.: On the equivalence of database restructurings involving object identifiers. In: Proceedings of the 10th ACM Symposium on Principles of Database Systems, pp. 328–340. ACM Press, New York (1991)
24. Kifer, M., Wu, J.: A logic for programming with complex objects. J. Comput. Syst. Sci. **47**(1), 77–120 (1993)
25. Klug, A., Price, R.: Determining view dependencies using tableaux. ACM Trans. Database Syst. **7**, 361–380 (1982)
26. Kolaitis, P., Pichler, R., Sallinger, E., Savenkov, V.: Nested dependencies: structure and reasoning. In: Proceedings of the 33rd ACM Symposium on Principles of Database Systems (2014)
27. Lenzerini, M.: Data integration: A theoretical perspective. In: Proceedings 21st ACM Symposium on Principles of Database Systems, pp. 233–246 (2002)
28. Levy, A.Y., Rajaraman, A., Ordille, J.J.: Querying heterogeneous information sources using source descriptions. In: Vijayaraman, T., Buchmann, A., Mohan, C., Sarda, N. (eds.) Proceedings of the 22nd International Conference on Very Large Data Bases, pp. 251–262. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1996)
29. Maier, D.: A logic for objects. In: Workshop on Foundations of Deductive Databases and Logic Programming, pp. 6–26 (1986)
30. Papakonstantinou, Y., Garcia-Molina, H., Widom, J.: Object exchange across heterogeneous information sources. In: ICDE, pp. 251–260 (1995)
31. Poggi, A., Lembo, D., Calvanese, D., De Giacomo, G., Lenzerini, M., Rosati, R.: Linking data to ontologies. J. Data Semant. **10**, 133–173 (2008)
32. Sequeda, J.F., Arenas, M., Miranker, D.P.: On directly mapping relational databases to RDF and OWL. In: International Conference on World Wide Web (WWW), pp. 649–658 (2012). doi:10.1145/2187836.2187924
33. Ullman, J.D.: Information integration using logical views. Theor. Comput. Sci. **239**(2), 189–210 (2000)