Querying Data Sources That Export Infinite Sets of Views

Bogdan Cautis
Telecom ParisTech
cautis@telecom-paristech.fr

Alin Deutsch*
UC San Diego
deutsch@cs.ucsd.edu

Nicola Onose*
UC San Diego
nicola@cs.ucsd.edu

ABSTRACT

We study the problem of querying data sources that accept only a limited set of queries, such as sources accessible by Web services which can implement very large (potentially infinite) families of queries. We revisit a classical setting in which the application queries are conjunctive queries and the source accepts families of conjunctive queries specified as the expansions of a (potentially recursive) Datalog program.

We say that query Q is *expressible* by the program $\mathcal{P}$ if it is equivalent to some expansion of $\mathcal{P}$. Q is *supported* by $\mathcal{P}$ if it has an equivalent rewriting using some finite set of $\mathcal{P}$'s expansions. We present the first study of expressibility and support for sources that satisfy integrity constraints, which is generally the case in practice.

1. INTRODUCTION

The recent proliferation of data sources accessible via Web services has renewed interest in the problem of querying sources with restricted querying capabilities [20, 15, 22, 23]. One reason is that, due to commercial, load-control or privacy considerations, Web sources do not typically accept arbitrary application queries against their schema. Instead, they allow only a (potentially infinite) family of parameterized queries implemented by the Web services. For instance, Amazon provides a service that takes an author name as parameter and returns the corresponding books, but will not allow queries that list all the available books. We refer to the queries accepted by a source as *views*.

In this setting, an application query issued against the source schema can experience two levels of service. It can be fully answerable at the source when the query is equivalent to some view exported by the source (provided the right view can be identified). In many cases, the set of answerable queries is extended by a *source wrapper* [20], which intercepts client queries and answers them by automatically identifying a series of relevant views, issuing the corresponding Web service calls and post-processing their results locally.

In this paper, we revisit the setting of [15, 23], in which the application queries are conjunctive queries and the source accepts families of possibly parameterized conjunctive queries specified as the expansions of a (potentially recursive) Datalog program. The program is said to *generate* these views. As argued in [15, 23] and illustrated below, the choice of Datalog as the view specification formalism enables concise yet expressive descriptions of large (even infinite) sets of views over a given schema.

We say that query Q is *expressible* by the program $\mathcal{P}$ if it is equivalent to some view generated by $\mathcal{P}$. Expressible queries can therefore be evaluated at the source, requiring no post-processing at the wrapper. Q is *supported* by $\mathcal{P}$ if it has an equivalent rewriting R using some finite set $\mathcal{V}$ of views generated by $\mathcal{P}$. Note that finding such R and $\mathcal{V}$ witnessing support enables the following execution plan at the wrapper: call the Web services implementing the queries in $\mathcal{V}$, materialize their results locally and run query R over the materialized database.

The problem of deciding support is also of interest for implementing security policies. For security reasons, a source would only allow data access via a set of *authorized views*, which are meant to enforce security policies and check user credentials [18, 21]. This type of access control is provided in particular by the so-called "non-Truman" access control model [21], in which the only allowed queries are those that are equivalent to authorized views or a combination thereof. The difference with respect to the previous scenario is that the system does not actually need to build a rewriting, as it will run the original query, provided that support holds.

The challenge in deciding expressibility and support lies in the fact that the family of views to pick from can be very large or even infinite. This renders infeasible any systematic enumeration of views. Remarkably, the two problems were previously shown to be decidable [15], however only when ignoring any knowledge of constraints satisfied by the source. In this work, we investigate the effect of source constraints.

The following example shows that source constraints generate new opportunities for detecting support, calling for algorithms which exploit them. (Example 1.1 illustrates a limited-query-capability setting and will be our running example in this paper.)

EXAMPLE 1.1. *Consider a travel information source conforming to the following schema:*

flight(origin, destination) *shuttle(origin, destination)*
train(origin, destination) *bus(origin, destination).*

*Partially funded by an Alfred P. Sloan fellowship and NSF grants IIS-0705589, IIS-0415257 and IIS-0347968.

ICDT 2009, March 23–25, 2009, Saint Petersburg, Russia.

The source admits only views concerning arbitrary-length itineraries by plane, such that Paris is reachable by train or bus from the destination airport. This family of views is described as the set of all expansions of the distinguished IDB predicate ans *in program $\mathcal{P}$ below:*

$$\begin{aligned} ans(A,B) &:- f(A,C), ind(C,B) \\ ind(C,B) &:- f(C,B), b(B, \text{“}Paris\text{”}) \\ ind(C,B) &:- f(C,C'), ind(C',B) \\ ind(C,B) &:- f(C,B), t(B, \text{“}Paris\text{”}) \end{aligned}$$

Consider a query that asks for 2-leg itineraries ending in an airport from which Paris is reachable by train, bus and shuttle.

$$\begin{aligned} Q: \quad q(A,B) :- \; & f(A,C), f(C,B), t(B, \text{“}Paris\text{”}), \\ & b(B, \text{“}Paris\text{”}), s(B, \text{“}Paris\text{”}) \end{aligned}$$

Clearly, Q is neither expressible nor supported by $\mathcal{P}$ because the views generated by $\mathcal{P}$ do not even mention shuttle information. However, suppose we knew the following constraint to hold on the source (stating that any city pair connected by train and bus is also connected by shuttle):

$$\forall A, S \quad t(A,S) \wedge b(A,S) \longrightarrow s(A,S). \tag{1}$$

Then we would like the wrapper to find the rewriting

$$(R) \qquad r(A,B) :- V_1^b(A,B), V_1^t(A,B)$$

where $\{V_i^b\}_{i\geq 1}$ (resp. $\{V_i^t\}_{i\geq 1}$) are families of views generated by $\mathcal{P}$, returning endpoints of itineraries of i flight legs where the destination has a bus link (resp. a train link) to Paris. Indeed, it can be checked that R is equivalent to Q on all databases satisfying (1). Therefore Q is supported by $\mathcal{P}$ when (1) holds.

Contributions. In this paper, we carry out the (to the best of our knowledge) first study of the problems of expressibility and support under source constraints. In particular, our contributions include:

Most permissive restrictions for decidability. We identify practically relevant restrictions on the program which ensure decidability under a mix of key and weakly acyclic foreign key constraints and beyond. The restrictions are particularly useful as they enable decidability via a reduction to the constraint-free case, which allows one to modularly “plug in” any existing algorithm to this end (such as those in [15, 22, 23] or the one we propose here for an improved upper bound). We show that these restrictions are as permissive as possible, since their slightest relaxation leads to undecidability in the presence of even a single key constraint. This result is counter-intuitive, since the existence of a rewriting of a conjunctive query using a finite set of non-parameterized conjunctive query views under key constraints (and beyond) is known to be decidable in NP.

A widely-applicable sound test. It is unsatisfactory in practice to refuse to test support and expressibility when the decidability restrictions are violated. A more useful approach consists in devising an algorithm which functions as a decision procedure under these restrictions, yielding only a best-effort “approximation” otherwise. One pragmatic articulation of what “approximation” could mean in this context is the following: the algorithm should be *sound* (i.e. no false positives) yet it may return false negatives (i.e. is not *complete*) for inputs that do not obey the decidability restrictions. We present such an algorithm for both expressibility and support, applicable to arbitrary programs under weakly acyclic sets of embedded dependencies [1], which are sufficiently expressive to capture key and foreign key constraints and beyond. The algorithm runs in deterministic exponential time in the size of the query, the size of the program and the maximum size of a constraint, which is as good as the best algorithm for rewriting queries using a finite list of views.

As a side-effect of our investigation, we *settle two open problems* left from prior work in the constraint-free setting.

Improved, practically tight upper bounds. We improve the previously best known upper bounds for deciding support in the constraint-free case: from *non-deterministic exponential* time in [23] and *doubly-exponential* time in [15], to *deterministic exponential* time in combined query and program size. Notice that in a practical implementation, the non-deterministic exponential time upper bound of [23] would still result in a doubly-exponential algorithm. The improvement is achieved using the sound algorithm mentioned above, which provably acts as an exponential-time decision procedure in the absence of constraints. We show our algorithm to be optimal in the program size (we give a deterministic EXPTIME lower bound for fixed query) and optimal for practical purposes in the query size (we give an NP lower bound for fixed program). The question of the tightness of this NP lower bound remains open. An interesting consequence of our new upper bound is that, in practical implementations, rewriting using an infinite set of views is no more expensive than using finitely many views listed individually (still deterministic exponential time).

The relationship between expressibility and support. We establish that expressibility and support are inter-reducible in PTIME in both the absence and the presence of constraints. This enables us to characterize the complexity of expressibility as well, and to employ the same algorithm for solving both problems. The result comes as a pleasant surprise, since prior work reports distinct upper bounds for these problems, suggesting (in line with intuition) that finding a rewriting of the query using program expansions is harder than finding a single equivalent expansion.

A one-size-fits-all solution. It is remarkable (and practically appealing) that all our upper bound results are based on the *same* algorithm for support, which serves simultaneously as (i) an essentially optimal decision procedure in the constraint-free case, improving prior upper bounds, (ii) a decision procedure under constraints in all known decidable cases, (iii) a sound procedure in general, and (iv) all of the above for the problem of expressibility, due to our inter-reducibility result.

Parameters. For presentation simplicity, in this paper we ignore the presence of parameters in the views generated by the program, handling parameters in the extended version [4].

Paper outline. After introducing preliminary concepts, results and notation in Section 2, in Section 3 we establish the PTIME inter-reducibility of expressibility and support. Section 4 presents decidable restrictions and Section 5 contains a sound algorithm in the case of general constraints. We also show there the improved upper bounds for the constraint-free setting (Section 5.1). We map the boundaries of decidability in Section 6. We discuss related work in Section 7 and conclude in Section 8. The extended version [4] contains the proofs.

2. PRELIMINARIES

We denote with CQ the language of conjunctive queries.

Constraints. We consider constraints ξ of the form

$$\forall \bar{u} \forall \bar{w}\ \phi(\bar{u}, \bar{w}) \longrightarrow \exists \bar{v}\, \psi(\bar{u}, \bar{v})$$

where ϕ (the *premise*) and ψ (the *conclusion*) are conjunctions of relational or equality atoms. Such constraints are known as *embedded dependencies* and are sufficiently expressive to specify all usual integrity constraints, such as keys, foreign keys, inclusion, join, multivalued dependencies, EGDs, TGDs etc. [1]. We call ϕ the *premise* and ψ the *conclusion*. If $\bar{v}$ is empty, then ξ is a *full dependency*. If ψ consists only of equality atoms, then ξ is an *equality-generating dependency (EGD)*. If ψ consists only of relational atoms, then ξ is a *tuple-generating dependency (TGD)*. If the premise and conclusion of a TGD contain one atom each, we call it an *inclusion dependency (IND)*. An IND in which the variables $\bar{u}$ appear precisely in the key attributes of the relation mentioned in the conclusion is a *foreign key* constraint. A *key constraint* on relation R can be expressed by the EGD $\forall \bar{u}, \bar{v}_1, \bar{v}_2\ R(\bar{u}, \bar{v}_1) \wedge R(\bar{u}, \bar{v}_2) \longrightarrow \bar{v}_1 = \bar{v}_2$. We write $A \models \mathcal{C}$ if the instance A satisfies all the constraints in $\mathcal{C}$.

Containment and Equivalence. Query Q_1 is contained in query Q_2 under the set $\mathcal{C}$ of constraints (denoted $Q_1 \sqsubseteq_{\mathcal{C}} Q_2$) iff $Q_1(D) \subseteq Q_2(D)$ for every database $D \models \mathcal{C}$, where $Q(D)$ denotes the result of Q on D. Q_1 is equivalent to Q_2 under $\mathcal{C}$ (denoted $Q_1 \equiv_{\mathcal{C}} Q_2$) iff $Q_1 \sqsubseteq_{\mathcal{C}} Q_2$ and $Q_2 \sqsubseteq_{\mathcal{C}} Q_1$.

Mappings. A *partial mapping* from CQ query Q_1 to CQ query Q_2 is a function h from the variables and constants of Q_1 to the variables and constants of Q_2 such that (i) h is the identity mapping on all constants, and (ii) for every relational atom (also called subgoal) $R(\bar{X})$ of Q_1, if h is defined for all variables in $(\bar{X})$, then $R(h(\bar{X}))$ is a subgoal of Q_2. A *homomorphism* from a set of subgoals C_1 to a set of subgoals C_2 is a partial mapping from the query $Q_1() :- C_1$ to the query $Q_2() :- C_2$ which is defined on all variables of Q_1. A *containment mapping* from CQ query Q_1 with tuple of head variables $\bar{X}_1$ to CQ query Q_2 with tuple of head variables $\bar{X}_2$ is a homomorphism h from Q_1 to Q_2 such that $h(\bar{X}_1) = \bar{X}_2$. We represent mappings as sets of pairs associating variables with either variables or constants, and use the notation $X : Y$ for the pair (X, Y). The *union* of two mappings is simply the union of their sets of pairs. A mapping is *consistent* if it does not map the same variable to two distinct values. A set of mappings is *compatible* if their union is consistent. Composition of mappings is the standard function composition, denoted by the operator $\circ$.

Expansion using views. Given a CQ query R formulated in terms of a set of view names $\mathcal{V}$(where the views are also CQs), the *expansion* of query R w.r.t. the views in $\mathcal{V}$ (denoted $expand_{\mathcal{V}}(R)$) is the query E obtained as follows: every subgoal $V(\bar{X})$ in R is replaced by a copy of the body of V, in which the head variables of V are renamed to $\bar{X}$ and all other variables are replaced by variables occurring in no other view bodies introduced during the expansion. It is easy to see that this variable renaming defines a homomorphism h from V into the expansion E, which we refer to as the *expansion homomorphism*.

Rewriting using views. We say that a conjunctive query R formulated in terms of view names $\mathcal{V}$ is a rewriting of a query Q using $\mathcal{V}$ under a set $\mathcal{C}$ of dependencies iff $Q \equiv_{\mathcal{C}} expand_{\mathcal{V}}(R)$.

Equivalence under views and constraints. Given queries R_1, R_2 formulated in terms of the view names in $\mathcal{V}$ and a set of dependencies $\mathcal{C}$, we say that R_1 is equivalent to R_2 under $\mathcal{V}$ and $\mathcal{C}$, denoted $R_1 \equiv_{\mathcal{C}}^{\mathcal{V}} R_2$, iff $expand_{\mathcal{V}}(R_1) \equiv_{\mathcal{C}} expand_{\mathcal{V}}(R_2)$.

The chase. We will use the classical *chase* procedure for rewriting conjunctive queries using a set of embedded dependencies [1]. For arbitrary sets $\mathcal{C}$ of dependencies, the chase is not guaranteed to terminate. The least restrictive condition on $\mathcal{C}$ known to date which is sufficient to ensure termination of the chase with $\mathcal{C}$ regardless of the query Q is called *weak acyclicity* [10] (see also [9]). Weak acyclicity of $\mathcal{C}$ implies termination of the chase of Q with $\mathcal{C}$ in time polynomial in the size of Q and exponential in the size of $\mathcal{C}$. Assuming termination of the chase, we denote with $chase_{\mathcal{C}}(Q)$ the query obtained by chasing conjunctive query Q with $\mathcal{C}$ to termination (this query is unique up to equivalence). Besides introducing new variables (for instance due to chasing with TGDs), the chase may equate the original variables of Q to constants or to each other (for instance due to chasing with key constraints) [1]. Denoting this variable renaming with r, it is a well-known fact that r is a homomorphic mapping from Q into $chase_{\mathcal{C}}(Q)$, also called the *chase homomorphism* [1].

Datalog expansions. A finite expansion (in short "expansion") of an IDB predicate p of a Datalog program $\mathcal{P}$ is a CQ query with head $p(\bar{X})$ and body obtained as follows: initialize the body to $body := p(\bar{X})$, then apply the following expansion step a finite number of times until no more IDBs are left in the body: for every IDB goal g_i in the body, pick a rule r_i in $\mathcal{P}$ defining g_i and collect all picked rules in a list $\mathcal{V}$. Treating $\mathcal{V}$ as views, replace *body* with $expand_{\mathcal{V}}(body)$, where each g_i is expanded using r_i. The set of expansions of $\mathcal{P}$ is infinite if $\mathcal{P}$ is recursive.

Convention. *In the remainder of this paper, unless explicitly stated otherwise, all queries and views are conjunctive queries, all programs are Datalog programs, and all dependencies are embedded dependencies.*

3. EXPRESSIBILITY VERSUS SUPPORT

We say that a view V is *generated* by program $\mathcal{P}$ if V is a CQ expansion of $\mathcal{P}$.

DEFINITION 3.1. *Given a Datalog program $\mathcal{P}$, a conjunctive query Q and a set of embedded dependencies $\mathcal{C}$, we say that*

1. *Q is* supported *by $\mathcal{P}$ under $\mathcal{C}$ (denoted $\text{SUPP}^{\mathcal{C}}_{\mathcal{P}}(Q)$), iff there is a* finite *set of views $\mathcal{V}$ generated by $\mathcal{P}$ and a conjunctive query rewriting of Q using $\mathcal{V}$ under $\mathcal{C}$.*

2. *Q is* expressible *by $\mathcal{P}$ under $\mathcal{C}$ (denoted $\text{EXPR}^{\mathcal{C}}_{\mathcal{P}}(Q)$), iff Q is equivalent under $\mathcal{C}$ to some view V generated by $\mathcal{P}$.*

In previous work, the problems of support and expressibility were introduced separately (in [15], respectively [23]). They were shown to be decidable, yet their reported complexity upper bounds were different even in the absence of constraints: doubly-exponential deterministic time for support [15], and EXPTIME for expressibility [23]. These results seemed to follow the intuition that finding a rewriting of the query using some expansions of the program is harder than finding a single equivalent expansion.

We establish a counter-intuitive relationship between the two problems, showing them to be inter-reducible in polynomial time even in the presence of dependencies.

THEOREM 3.1. *Let $\mathcal{C}$ be a weakly acyclic set of embedded dependencies. Then there is a reduction from the problem of support of a query Q by a program $\mathcal{P}$ under $\mathcal{C}$ to an instance of the expressibility problem, which is in PTIME in the size of Q and $\mathcal{P}$ and in EXPTIME in the size of $\mathcal{C}$.*

COROLLARY 3.1. *If the size of the schema (with dependencies) is bounded by a constant, then there is a PTIME reduction from support to expressibility provided the set of embedded dependencies is weakly acyclic.*

COROLLARY 3.2. *In the absence of dependencies, there is a PTIME reduction from support to expressibility.*

The next result shows the existence of a polynomial-time reduction in the other direction, requiring no restrictions on the embedded dependencies.

THEOREM 3.2. *Expressibility reduces in PTIME to support.*

In particular, since dependency-free support is known to be decidable [15], Theorem 3.2 implies decidability of dependency-free expressibility, with the same complexity.

4. DECIDABLE CASES

In this section, we give restrictions under which the problems of expressibility and support are decidable under constraints. As will be seen in Section 6, the restrictions are needed because the two problems are in general undecidable, and they are fairly tight, in the sense that even slight relaxations thereof lead to undecidability.

Because it is interesting in its own right, we show a particular route to decidability based on reducing to the dependency-free setting, which is known to be decidable [15]. However, this does not yet provide the improved upper bound, which requires improving prior results for the dependency-free case. We shall do so in Section 5, obtaining a more general result: a novel algorithm that does not rely on reduction to the dependency-free case, but serves as an optimal decision procedure when dependencies are absent or when they satisfy the restrictions presented in this section, and gracefully degenerates to a sound procedure otherwise.

We introduce properties of the program and of the views it generates that suffice for our reduction to the dependency-free case. The idea is to pre-process the program to explicitly incorporate into it the knowledge about the dependencies, so that these can then be ignored, thus reducing the problem to dependency-free expressibility and support for the new program. The pre-processing technique relies on the *chase* procedure. This was a natural choice, as the chase tool has been traditionally employed successfully to reduce classical decision problems (such as query equivalence or implication of dependencies [1]) from the presence of dependencies to their absence. We start with expressibility.

Given a Datalog program $\mathcal{P}$, we denote with $chase_{\mathcal{C}}(\mathcal{P})$ the program obtained by chasing each rule of $\mathcal{P}$ with $\mathcal{C}$.

DEFINITION 4.1 ($\mathcal{C}$-LOCAL PROGRAM). *Let $\mathcal{C}$ be a weakly acyclic set of dependencies. We say that a program $\mathcal{P}$ is* $\mathcal{C}$-local *iff for every view V generated by $\mathcal{P}$ there is a view W generated by $chase_{\mathcal{C}}(\mathcal{P})$, and for every view W generated by $chase_{\mathcal{C}}(\mathcal{P})$ there is a view V generated by $\mathcal{P}$, such that $chase_{\mathcal{C}}(V)$ is equivalent to W even in the absence of dependencies.*

The intuition behind $\mathcal{C}$-locality is as follows. Recall that when checking expressibility under $\mathcal{C}$, one needs to exhibit some view V generated by $\mathcal{P}$, such that $Q \equiv_{\mathcal{C}} V$. By the chase theorem [1, 17], if the chase terminates, the equivalence under $\mathcal{C}$ reduces to the following equivalence in the absence of dependencies (i.e. under the empty set of dependencies): $chase_{\mathcal{C}}(Q) \equiv_{\emptyset} chase_{\mathcal{C}}(V)$. $\mathcal{C}$-locality ensures that the chase of view V can be avoided by simply searching among the views generated by $chase_{\mathcal{C}}(\mathcal{P})$. These must include some W with $W \equiv_{\emptyset} chase_{\mathcal{C}}(V)$, so the existence of V as above is equivalent to the existence of W generated by $chase_{\mathcal{C}}(\mathcal{P})$, with $chase_{\mathcal{C}}(Q) \equiv_{\emptyset} W$. This in turn is by definition dependency-free expressibility of query $chase_{\mathcal{C}}(Q)$ by program $chase_{\mathcal{C}}(\mathcal{P})$. Indeed, we can show the following.

THEOREM 4.1. *Let Q be a conjunctive query, $\mathcal{C}$ a weakly acyclic set of dependencies, and $\mathcal{P}$ a $\mathcal{C}$-local program. Then $\text{EXPR}^{\mathcal{C}}_{\mathcal{P}}(Q)$ holds iff $\text{EXPR}^{\emptyset}_{chase_{\mathcal{C}}(\mathcal{P})}(chase_{\mathcal{C}}(Q))$ holds.*

The reduction of support to the dependency-free case requires an additional restriction on the views generated by the program. In this case, we need to exhibit a set $\mathcal{V}$ of views generated by $\mathcal{P}$ and a rewriting R of Q in terms of $\mathcal{V}$. Again by the chase theorem [1, 17], this is equivalent (provided the chase terminates) to exhibiting $\mathcal{V}$ and R such that $chase_{\mathcal{C}}(Q) \equiv_{\emptyset} chase_{\mathcal{C}}(expand_{\mathcal{V}}(R))$. The idea behind the reduction is to require the views to be such that no matter how they are used in R, chasing R's expansion gives the same result as first chasing each view individually and then expanding R with the chased views: $chase_{\mathcal{C}}(expand_{\mathcal{V}}(R)) \equiv_{\emptyset} expand_{\{chase_{\mathcal{C}}(V_1),\ldots,chase_{\mathcal{C}}(V_n)\}}(R)$. Now if $\mathcal{P}$ is $\mathcal{C}$-local, then the chased views are equivalent to some views $\mathcal{W} = \{W_1, \ldots, W_n\}$ generated by $chase_{\mathcal{C}}(\mathcal{P})$, and we have $chase_{\mathcal{C}}(Q) \equiv_{\emptyset} chase_{\mathcal{C}}(expand_{\mathcal{W}}(R))$, which is the definition of dependency-free support of $chase_{\mathcal{C}}(Q)$ by $chase_{\mathcal{C}}(P)$. We formalize this intuition next.

DEFINITION 4.2 ($\mathcal{C}$-INDEPENDENT VIEW SET). *Let $\mathcal{C}$ be a weakly acyclic set of dependencies. We say that a set of views $\mathcal{V} = \{V_1, \ldots, V_n\}$ is* $\mathcal{C}$-independent *iff, for every query R' formulated in terms of $\mathcal{V}$, there exists query R also formulated in terms of $\mathcal{V}$, such that*

(i) $R' \equiv^{\mathcal{V}}_{\mathcal{C}} R$,

(ii) and such that

$$chase_{\mathcal{C}}(expand_{\{V_1,\ldots,V_n\}}(R))$$

is equivalent even in the absence of dependencies to

$$expand_{\{chase_{\mathcal{C}}(V_1),\ldots,chase_{\mathcal{C}}(V_n)\}}(R).$$

Notice that we do not require property (ii) in Definition 4.2 to hold for all queries R' over $\mathcal{V}$, since there are potentially many equivalent forms of R'. It sufficies if one of them satisfies (ii). In that case, we can show the following.

THEOREM 4.2. *Let Q be a conjunctive query, $\mathcal{C}$ a weakly acyclic set of dependencies, and $\mathcal{P}$ a $\mathcal{C}$-local program. Then, if the views generated by $\mathcal{P}$ are $\mathcal{C}$-independent, then* $\text{SUPP}^{\mathcal{C}}_{\mathcal{P}}(Q)$ *iff* $\text{SUPP}^{\emptyset}_{chase_{\mathcal{C}}(\mathcal{P})}(chase_{\mathcal{C}}(Q))$.

We next provide various syntactic restrictions on the dependencies in $\mathcal{C}$ and on $\mathcal{P}$ to guarantee $\mathcal{C}$-independence and $\mathcal{C}$-locality.

THEOREM 4.3. *Let $\mathcal{C}$ be a weakly acyclic set of inclusion dependencies. Then any Datalog program $\mathcal{P}$ is $\mathcal{C}$-local and every finite subset of its generated views is $\mathcal{C}$-independent.*

Theorems 4.1, 4.2 and 4.3 immediately imply that for weakly acyclic sets of inclusion dependencies, expressibility and support reduce to the dependency-free versions:

COROLLARY 4.1. *If $\mathcal{C}$ is a weakly acyclic set of inclusion dependencies, then for any program $\mathcal{P}$ and query Q,* $\text{EXPR}^{\mathcal{C}}_{\mathcal{P}}(Q)$ *iff* $\text{EXPR}^{\emptyset}_{chase_{\mathcal{C}}(\mathcal{P})}(chase_{\mathcal{C}}(Q))$ *and* $\text{SUPP}^{\mathcal{C}}_{\mathcal{P}}(Q)$ *iff* $\text{SUPP}^{\emptyset}_{chase_{\mathcal{C}}(\mathcal{P})}(chase_{\mathcal{C}}(Q))$.

EXAMPLE 4.1. *Consider a source for travel data using the following schema:*

train(origin, destination, operator)
bus(origin, destination,operator)

where each origin-destination pair is connected by a non-stop leg. It accepts queries for train itineraries with arbitrary many legs in which the same operator is used. It returns the origin, the destination, one intermediary stop and the operator. This family of queries is described by program $\mathcal{P}$:

$$\begin{array}{lrll}(\mathcal{P}) & ans(A,B,C,O) & :- & ind(A,B,O), ind(B,C,O)\\ & ind(B,C,O) & :- & t(B,B',O), ind(B',C,O)\\ & ind(B,C,O) & :- & t(B,C,O)\end{array}$$

Let Q be an application query searching for a one-way trip with connection in Paris, such that starting from Paris one can either continue the trip by bus, and stay with the first operator, or take another train with any available operator.

$$(Q)\quad q(A,B) :- t(A,C,O_1), b(C,B,O_1), t(C,B,O_2), C = \text{“}Paris\text{”}$$

Notice that Q is not supported by $\mathcal{P}$ in the absence of constraints (the source does not even allow views mentioning the bus predicate): $\text{SUPP}^{\emptyset}_{\mathcal{P}}(Q)$ *does not hold.*

Assume that the source satisfies $\mathcal{C}$ which contains the inclusion dependency (2) below, stating that an operator will also cover by bus any leg important enough to be covered by train.

$$\forall X,Y,O \qquad t(X,Y,O) \longrightarrow b(X,Y,O) \tag{2}$$

Since $\mathcal{C}$ is (trivially) a weakly acyclic set of INDs, by Corollary 4.1 $\text{SUPP}^{\mathcal{C}}_{\mathcal{P}}(Q)$ *holds if and only if so does* $\text{SUPP}^{\emptyset}_{chase_{\mathcal{C}}(\mathcal{P})}(chase_{\mathcal{C}}(Q))$.

Chase steps apply on the extensional parts of the second and third rules of $\mathcal{P}$, yielding the new rules (we underline the newly added tuples):

$$\begin{array}{lrll}ind(B,C,O) & :- & t(B,B',O), \underline{b(B,B',O)}, ind(B',C,O)\\ ind(B,C,O) & :- & t(B,C,O), \underline{b(B,C,O)}\end{array}$$

The new program $chase_{\mathcal{C}}(\mathcal{P})$ generates the views V_{ij} denoting the expansion with i legs from the origin to the intermediary point and j legs from the intermediary point to the destination. This includes the view V_{11}, which gives the shortest itineraries:

$$(V_{11})\quad v(A,B,C,O) :- t(A,B,O), b(A,B,O), t(B,C,O), b(B,C,O)$$

By chasing also the query, we obtain $Q' = chase_{\mathcal{C}}(Q)$:

$$\begin{array}{lrll}(Q')\ q(A,B) & :- & t(A,C,O_1), \underline{b(A,C,O_1)}, b(C,B,O_1),\\ & & t(C,B,O_2), \underline{b(C,B,O_2)}, C = \text{“}Paris\text{”}\end{array}$$

Observe that $\text{SUPP}^{\emptyset}_{chase_{\mathcal{C}}(\mathcal{P})}(chase_{\mathcal{C}}(Q))$ *(and* $\text{SUPP}^{\mathcal{C}}_{\mathcal{P}}(Q)$*) still does not hold because all the views V_{ij} require that only one operator be used. To enforce this requirement on Q', one would need a constraint enforcing that the subgoals $b(C,B,O_1)$ and $b(C,B,O_2)$ from Q' refer to the same operator, making the equality $O_1 = O_2$ hold.*

Key safety. We next introduce the notion of a program being “key-safe”, which guarantees $\mathcal{C}$-locality and $\mathcal{C}$-independence in the presence of key constraints.

Let R be a relation with an n-attribute composite key and let $\bar{P} = (p_1,\ldots,p_k)$ be an ordered sequence of k distinct values in the range 1 to n. We say that a rule of $\mathcal{P}$ *outputs the key of R, by positions $\bar{P}$, into the sequence of head variables* $\bar{X} = (X_{i_1},\ldots,X_{i_k})$ if $\bar{X}$ appears in the rule body either

- in the positions $p_1,\ldots,p_k$ of the key attribute sequence of some R-subgoal, with the remaining $n-k$ positions (if any) of the key being bound to constant values, or
- in the positions $j_1,\ldots,j_k$ of some p-subgoal, where p is an IDB predicate with at least one rule that in turn outputs the key of R by key positions $\bar{P}$ into the sequence of head variables with indices $j_1,\ldots,j_k$.

We say that a subgoal g outputs the key of R, by positions $\bar{P} = (p_1,\ldots,p_k)$, into the sequence of variables $\bar{X} = (X_{i_1},\ldots,X_{i_k})$ if

- g uses EDB predicate R and $\bar{X}$ appears in positions $p_1,\ldots,p_k$ in the key attributes of g, with the remaining $n-k$ positions (if any) of the key being bound to constant values, or

- g uses IDB predicate p and there exists some rule defining p which outputs the key of R, by the key positions $\bar{P}$, into variables $\bar{X}$.

We say that a rule is *safe* for the key constraint on R if whenever one of its IDB subgoals outputs the key of R by some sequence of k key positions $\bar{P}$ into k variables $\bar{X} = (X_{i_1}, \ldots, X_{i_k})$, no other subgoal does the same (for the same key positions $\bar{P}$). Notice that several EDB subgoals may output the key of the same R by the same key positions and into the same sequence of variables $\bar{X}$, as long as no IDB goal does.

EXAMPLE 4.2. *Suppose that, in Example 4.1, $\mathcal{C}$ contains also a key constraint on the b table, stating that bus operators cover disjoint legs:*

$$\forall X, Y, O \qquad b(X, Y, O), b(X, Y, O') \longrightarrow O = O' \qquad (3)$$

Notice that $chase_{\mathcal{C}}(\mathcal{P})$ is the same as in Example 4.1 because no chase step applies with the key constraint.

The rules in $chase_{\mathcal{C}}(\mathcal{P})$ are safe. Indeed, in the second rule, b outputs the key into the sequence B, B', while ind outputs it into B', C. The two subgoals in the first rule also output the key, but into different sequences: A, B and B, C respectively.

Intuitively, safety of the rules in a program $\mathcal{P}$ is designed to guarantee $\mathcal{C}$-locality. It disallows two IDB goals in a rule from outputting the key of some EDB R into the same variables because this could lead, in the expansion of the rule, to two R goals agreeing on the key attributes and thus triggering a chase step with the key constraint. Since the R goals would come from the expansion of distinct IDB goals in the rule, the effect of this chase would not be reproducible by chasing the program rules in isolation (as in the definition of $chase_{\mathcal{C}}(P)$).

We now give a condition ensuring that every set of views generated by $\mathcal{P}$ is $\mathcal{C}$-independent. This requires additional restrictions on the rules of the distinguished predicates.

DEFINITION 4.3. *A* program $\mathcal{P}$ is key-safe *for a set of key constraints $\mathcal{K}$ if*

1. *each rule is safe for all key constraints in $\mathcal{K}$, and*
2. *for all distinguished predicates* ans *of $\mathcal{P}$, all defining rules r of* ans*, and all relational symbols R in the schema, if r outputs the key attributes $\bar{A}$ (as defined above) of some goal $R(\bar{A}, \bar{B})$, it also outputs all non-key attributes $\bar{B}$ (by the same definition that applied to the key attributes).*

If $\mathcal{I}$ is a set of weakly acyclic INDs, we say that $\mathcal{P}$ is key-safe for $\mathcal{C} = \mathcal{K} \cup \mathcal{I}$ if $chase_{\mathcal{I}}(\mathcal{P})$ is key-safe for $\mathcal{K}$.

Note that key-safety can be checked in PTIME in the size of $\mathcal{P}$ and $\mathcal{K}$.

EXAMPLE 4.3. *Continuing Example 4.2, we observe that distinguished predicate ans outputs the pairs of key attributes A, B and B, C, but it also outputs O, the only non-key attribute. Therefore, $\mathcal{P}$ is key-safe.*

Intuitively, the key safety condition on the distinguished predicates ensures that, given query R' in terms of some views $\mathcal{V}$ generated by $\mathcal{P}$, there is query $R \equiv^{\mathcal{V}}_{\mathcal{C}} R'$ such that no chase step with a key constraint will apply to $expand_{\mathcal{V}}(R)$. This is because, if two view atoms in R' happen to output the key of some EDB goal G into the same variables $\bar{A}$, then by key-safety they each must also output all non-key attributes of G, say in variables $\bar{B}_1$, respectively $\bar{B}_2$. But then there is a query R, equivalent to R', obtained by adding to R' the equalities $\bar{B}_1 = \bar{B}_2$. This equality is preserved in $expand_{\mathcal{V}}(R)$, so the chase step with the key constraint does not apply on $expand_{\mathcal{V}}(R)$. More formally, we can show the following.

THEOREM 4.4. *Let $\mathcal{C}$ consist of key constraints and an acyclic set of inclusion dependencies. Any Datalog program $\mathcal{P}$ that is key-safe for $\mathcal{C}$ is also $\mathcal{C}$-local and all views generated by it are $\mathcal{C}$-independent.*

COROLLARY 4.2. *If $\mathcal{C}$ consists of key constraints and an acyclic set of INDs and $\mathcal{P}$ is key-safe for $\mathcal{C}$, then for any query Q,* $\textsc{Expr}^{\mathcal{C}}_{\mathcal{P}}(Q)$ *iff* $\textsc{Expr}^{\emptyset}_{chase_{\mathcal{C}}(\mathcal{P})}(chase_{\mathcal{C}}(Q))$ *and* $\textsc{Supp}^{\mathcal{C}}_{\mathcal{P}}(Q)$ *iff* $\textsc{Supp}^{\emptyset}_{chase_{\mathcal{C}}(\mathcal{P})}(chase_{\mathcal{C}}(Q))$.

EXAMPLE 4.4. *Continuing Example 4.3, a chase step with* (3) *applies on Q', introducing the equality atom $O_1 = O_2$. With this,* $\textsc{Supp}^{\emptyset}_{chase_{\mathcal{C}}(\mathcal{P})}(chase_{\mathcal{C}}(Q))$ *holds, as witnessed by the rewriting*

$$q(A, B) \ :- \ V_{11}(A, \text{“}Paris\text{”}, B, O).$$

Remarks. The definition of key-safety described above is over-conservative: it considers all constants as being equatable in a chase step. This is because it only keeps track of the positions bound to constants, ignoring the actual constant values. We describe in Appendix B a refined version of key-safety that takes into account these values. This refined notion of key-safety is implied by the one presented here and detects strictly more decidable cases, but, for ease of presentation, it is omitted from the main text.

According to the results presented so far in this section, and in Section 3, under the decidability restrictions ($\mathcal{C}$-independence and $\mathcal{C}$-locality), we can solve expressibility under $\mathcal{C}$ even by using our favorite solver for dependency-free support (first reduce to dependency-free expressibility, then reduce to dependency-free support). Symmetrically, we can solve support under $\mathcal{C}$ using any solver for dependency-free expressibility. It turns out that the same cross-use of solvers can be achieved by first reducing from expressibility under $\mathcal{C}$ to support under $\mathcal{C}$ (using Theorem 3.2), and then to dependency-free support (using Theorem 4.1) (and symmetrically for support), as the reductions preserve restrictions for decidability. The formal results are presented in [4].

5. A WIDELY APPLICABLE SOUND TEST

We next present a sound algorithm for testing support, applicable to any program and set of weakly acyclic dependencies. It is a decision procedure (no false negatives) under the decidability restrictions of Section 4, and in the dependency-free case (where it provides an exponentially better upper bound than previous work).

Our solution is based on the following overall strategy. Since a systematic enumeration of all (potentially infinitely many) views generated by a program $\mathcal{P}$ is infeasible, we instead “describe the behavior” (in a sense formalized shortly) of any view generated by $\mathcal{P}$ w.r.t. a decision procedure (described below) for the existence of a rewriting under $\mathcal{C}$ using *finitely* many views. This description will abstract away

from the view body, focusing on how the view behaves in essential tests performed by this decision procedure. As it will turn out, under our decidability restrictions, there are only *finitely* many distinct behaviors, each exhibited by a possibly infinite set of views. It suffices therefore to find one representative view from each set, thus reducing the problem of checking support by $\mathcal{P}$ to checking the existence of a rewriting using the finitely many representatives. This problem is known to be decidable under weakly acyclic dependencies (Lemma 5.1 below). We start by describing the associated decision procedure.

Canonical Rewriting Candidate. Given a finite set of views $\mathcal{V}$, an acyclic set of constraints $\mathcal{C}$, and a query Q, call the *canonical rewriting candidate* of Q using $\mathcal{V}$ under $\mathcal{C}$, denoted $CRC_{\mathcal{V}}^{\mathcal{C}}(Q)$, the query obtained as follows: (i) it has the same head variables as Q, and (ii) its body is constructed by evaluating each view $V \in \mathcal{V}$ over the body of $chase_{\mathcal{C}}(Q)$ (viewed as a symbolic database, also known as the canonical instance [1]) and adding the subgoal $V(t)$ for every tuple t in the result of the evaluation.

We show next that the canonical rewriting candidate yields a decision procedure for the existence of a rewriting. This result reformulates a theorem in [9] (see also [8])[1]:

LEMMA 5.1 (COROLLARY OF [9]). *Q has a rewriting using $\mathcal{V}$ under $\mathcal{C}$ iff $CRC_{\mathcal{V}}^{\mathcal{C}}(Q)$ is one. Moreover, this in turn holds iff (a) $CRC_{\mathcal{V}}^{\mathcal{C}}(Q)$ is safe (its head variables appear in its body), and (b) there is a containment mapping from Q into the result of chasing with $\mathcal{C}$ the expansion of $CRC_{\mathcal{V}}^{\mathcal{C}}(Q)$: $chase_{\mathcal{C}}(expand_{\mathcal{V}}(CRC_{\mathcal{V}}^{\mathcal{C}}(Q))) \sqsubseteq Q$.*

EXAMPLE 5.1. *Revisiting Example 1.1, consider the following set of views $\mathcal{V} = \{V_1, V_2\}$:*

$$(V_1) \quad ans^1(Z_1, Z_2) \;:-\; f(Z_1, X), f(X, Z_2), t(Z_2, \text{“}Paris\text{”})$$
$$(V_2) \quad ans^2(Z_1, Z_2) \;:-\; f(Z_1, Y), f(Y, Z_2), b(Z_2, \text{“}Paris\text{”})$$

generated (among others) by $\mathcal{P}$. We will follow, step by step, the rewriting algorithm from [9]. The first step consists in finding mappings from the view queries into the body of Q and adding, to Q, atoms corresponding to the head of the view query. V_1 is mapped into Q by $m_1 = \{Z_1 : A; X : C; Z_2 : B\}$, which leads to adding $ans^1(A, B)$. Similarly, for V_2 we discover the mapping $m_2 = \{Z_1 : A; Y : C; Z_2 : B\}$ and add $ans^2(A, B)$. We stop here, since no more mappings can be inferred. The result is an expanded query

$$U: \quad q(A,B) :- f(A,C), f(C,B), t(B, \text{“}Paris\text{”}), b(B, \text{“}Paris\text{”}), s(B, \text{“}Paris\text{”}), \underline{ans^1(A,B)}, \underline{ans^2(A,B)}$$

in which the newly added atoms are underlined. U is called the universal plan *in [9], and it is guaranteed that any exact rewriting of Q is a subquery of U.*

$R = CRC_{\mathcal{V}}^{\mathcal{C}}(Q)$ is then obtained from U by keeping only the atoms from the view schema:

$$R(A,B) :- ans^1(A,B), ans^2(A,B).$$

R is equivalent to Q under dependency (1), as can be verified by first constructing the expansion $E = expand_{\mathcal{V}}(CRC_{\mathcal{V}}^{\mathcal{C}}(Q))$ as:

$$E(A,B) :- f(A,X'), f(X',B), t(B, \text{“}Paris\text{”}), f(A,Y'), f(Y',B), b(B, \text{“}Paris\text{”})$$

which chases with (1) to query (cE):

$$cE(A,B) :- f(A,X'), f(X',B), t(B, \text{“}Paris\text{”}), f(A,Y'), f(Y',B), b(B, \text{“}Paris\text{”}), \underline{s(B, \text{“}Paris\text{”})}$$

into which there is a containment mapping from Q, $cm_q = \{A : A, B : B, C : X'\}$. The reverse containment also holds, as witnessed by the containment mapping from cE into Q, $cm_e = \{A : A, B : B, X' : C, Y' : C\}$, hence R is indeed a rewriting.

Note that both views contribute to the rewriting, since both t and b atoms are needed as images of the t and b atoms from Q. The contribution of V_1 consists in m_{v1}, a partial mapping of Q into cE, obtained by restricting the domain of cm_q to the first three atoms of Q:

$$m_{v1} = \{A : A, B : B, C : X'\}.$$

In this case, the image of m_{v1}, E^1, is the entire expansion of ans^1:

$$E^1 = f(A,X'), f(X',B), t(B, \text{“}Paris\text{”}).$$

The contribution of V_2 is enabled by a partial mapping

$$m_{v2} = \{B : B\}$$

from (the b atom of) Q into the expansion of ans^2, with the image

$$E^2 = b(B, \text{“}Paris\text{”}).$$

m_{v1} and m_{v2} agree on the common B variable, and, since together they cover the whole of the body of Q, we obtain by combining them the containment mapping cm_q that maps the entire Q into cE.

Redundant views *Let us add now to program $\mathcal{P}$ a new rule, corresponding to the definition of the view V_3 given below:*

$$(V_3) \quad ans^3(Z_1, Z_3) :- f(Z_1, T), f(T, Z_2), b(Z_3, \text{“}Paris\text{”}).$$

Running the same rewriting algorithm as above on the set $\mathcal{V}' = \{V_1, V_2, V_3\}$, we discover that V_3 maps into Q by $m_3 = \{Z_1 : A, T : C, Z_2 : B, Z_3 : B\}$, which leads to a rewriting candidate $CRC_{\mathcal{V}'}^{\mathcal{C}}(Q)$ of the form

$$R'(A,B) :- ans^1(A,B), ans^2(A,B), ans^3(A,B).$$

V_3 does not modify the way in which the expansion query (which already had t and b atoms) chases, hence the resulting chased expansion of R' is:

$$cE'(A,B) :- f(A,X'), f(X',B), t(B, \text{“}Paris\text{”}), f(A,Y'), f(Y',B), b(B, \text{“}Paris\text{”}), f(A,T'), f(T',T''), b(B, \text{“}Paris\text{”}), \underline{s(B, \text{“}Paris\text{”})}$$

We can argue here that V_2 and V_3 are mutually redundant w.r.t. finding a rewriting of Q. The partial mapping $m_{v3} = \{B : B\}$ from Q into the expansion of ans^3, with the image $b(B, \text{“}Paris\text{”})$, is isomorphic to the partial mapping m_{v2}

[1]Lemma 5.1 is a corollary of [9], where it is also proven that there are only finitely many rewritings of Q using $\mathcal{V}$ that are minimal under $\mathcal{C}$, and that all of them are subqueries of $CRC_{\mathcal{V}}^{\mathcal{C}}(Q)$.

from Q into the expansion of ans^2. To this, add the fact that both mappings from the bodies of the two views into Q, v_2 and v_3, agree on the images of the distinguished variables, mapping them into variables A and B of Q. Without going into further details, this would be enough to allow us to discard one of the two views and to obtain as a rewriting either $ans^1(A,B), ans^2(A,B)$ or $ans^1(A,B), ans^3(A,B)$.

According to Lemma 5.1 and the observations above, in order for a view to contribute to the rewritability of Q

(i) it must generate a subgoal g of the canonical rewriting candidate
e.g. V_1 generates $ans^1(A,B)$, introduced by the mapping m_1 from V_1 into Q;

(ii) g's expansion may participate in the chase with $\mathcal{C}$ of the expansion E of the canonical rewriting candidate
e.g. the expansion E^1 of $ans^1(A,B)$ contains the atom $t(B, \text{“}Paris\text{”})$, which, together with the expansion of V_2, $E^2 = b(B, \text{“}Paris\text{”})$, allows a chase step with dependency (1) to apply;

(iii) since Q maps into the chase of E, the expansion of g must include (after the chase) the image of a partial map from Q
e.g. E^1 is the image of m_{v1}.

We shall therefore describe a view V with respect to its behavior for (i), (ii) and (iii), using the notion of *descriptor*.

Normalized program. For uniformity of treatment, we will assume from now on w.l.o.g. that the program $\mathcal{P}$ is normalized as follows. For every k-ary IDB predicate p, every rule for p has the head variables $\bar{Z} = Z_1, \ldots, Z_k$, in that order. Furthermore, for every EDB predicate e, introduce a new IDB e', replace each occurrence of e in $\mathcal{P}$ with e', and add the rule $e'(\bar{Z}) :- e(\bar{Z})$. The normalized program has only two kinds of rules: those whose bodies consist of a single EDB subgoal (called *EDB rules*), or solely of IDB subgoals (called *IDB rules*). For technical reasons, we additionally compute (as in [15]), the *closure* of the program, which consists in adding for every rule r in $\mathcal{P}$ all rules obtained from r by systematically equating in all possible ways the head variables of r with each other and with the constants in Q.

Definition 5.1 (Descriptors). *For a query Q and a program $\mathcal{P}$, $E^{(p(t),fr)}$ is called a* descriptor *w.r.t Q and $\mathcal{P}$ iff*

- *p is an IDB predicate from $\mathcal{P}$,*
- *E is a conjunctive query body over EDBs from $\mathcal{P}$,*
- *$\mathcal{P}$ generates as expansion of p a query of head variables $\bar{Z}$, $p(\bar{Z}) :- body$,*
- *there is a homomorphism* to : $body \rightarrow chase_{\mathcal{C}}(Q)$ *s.t.* $\text{to}(\bar{Z}) = t$,
- fr *is a partial variable mapping from Q into $chase_{\mathcal{C}}(body)$ such that the image of Q under* fr *is E.*

We call E the expansion fragment *described by the descriptor, and $(p(t), fr)$ the* adornment *of E. We call variables $\{Z_1, \ldots, Z_k\}$ (where k is the arity of p) the* distinguished *variables of the descriptor, while all other variables in the range of fr are* hidden.

In the following, when referring to a descriptor we will omit the program $\mathcal{P}$ and the query Q if they are obvious from the context.

Example 5.2. *In the setting of Example 5.1, $d_1 = E_1^{(p_1(t_1),fr_1)}$ and $d_2 = E_2^{(p_2(t_2),fr_2)}$ below are descriptors for the views V_1 and V_2, respectively:*

$$\begin{array}{lll} d_1 & : & E_1 = [f(Z_1,X), f(X,Z_2), t(Z_2, \text{“}Paris\text{”})], \\ & & p_1(t_1) = ans(A,B), fr_1 = \{A:Z_1, C:X, B:Z_2\} \\ d_2 & : & E_2 = [b(Z_2, \text{“}Paris\text{”})], \\ & & p_2(t_2) = ans(A,B),\ fr_2 = \{B:Z_2\} \end{array}$$

Note that, though the two views contribute the same $ans(A,B)$ goal to the canonical rewriting candidate, the two descriptors distinguish among V_1 and V_2 by the images of Q into the view bodies (E_1 includes the image of Q's t and two f goals, E_2 only the b goal).

Before explaining in detail how descriptors are found, we show how they can be used to soundly infer support. Intuitively, a descriptor represents the fragment of a chased view generated by $\mathcal{P}$ that serves as image of the partial mapping from Q. Our goal is to put together such fragments in a consistent way to create (if it exists) the image of Q under a *containment mapping*.

Partial rewriting candidate. More formally, consider a finite set of descriptors w.r.t. to query Q, program $\mathcal{P}$ and dependencies $\mathcal{C}$: $\mathcal{D} = \{E_i^{(p_i(t_i),fr_i)}\}_{1 \le i \le n}$, where all p_i are (not necessarily distinct) distinguished IDBs of $\mathcal{P}$. Introduce for each predicate p_i a fresh predicate p_i^i (using the rank i of the predicate in an arbitrary ordering of the descriptor set) such that $p_i^i \neq p_j^j$ for all $1 \le i, j \le n, i \neq j$. Assuming w.l.o.g. that Q's tuple of head variables is $\bar{X}$, we call the query

$$R(\bar{X}) :- p_1^1(t_1), \ldots, p_n^n(t_n)$$

the *partial rewriting candidate* described by $\mathcal{D}$. The set $\mathcal{V} := \{VF_i : p_i^i(\bar{Z}) :- E_i\}_{1 \le i \le n}$ is called the *view fragments* described by $\mathcal{D}$. The view fragments VF_i are not necessarily safe queries, if not all the head variables serve as image of the partial mapping fr_i.

Example 5.3. *For the set of descriptors $\mathcal{D} = \{d_1, d_2\}$ from Example 5.2, the fresh view goals are ans^1, ans^2 respectively. The partial rewriting candidate described by $\mathcal{D}$ is*

$$R(A,B) :- ans^1(A,B), ans^2(A,B)$$

(it happens to coincide with the canonical rewriting candidate shown in Example 5.1). The view fragments are

$$\begin{array}{llll} (VF_1) & ans^1(Z_1,Z_2) & :- & f(Z_1,X), f(X,Z_2), t(Z_2, \text{“}Paris\text{”}) \\ (VF_2) & ans^2(Z_1,Z_2) & :- & b(Z_2, \text{“}Paris\text{”}). \end{array}$$

Notice how VF_1's, VF_2's bodies are isomorphic to fragments of the bodies of V_1, respectively V_2 from Example 5.1. Also, VF_2 is not safe as variable Z_1 does not appear in the body.

The following result allows us to test support, as in Lemma 5.1, but using descriptors instead of explicit views. The key idea is to use the partial rewriting candidate instead of the canonical rewriting candidate.

Corollary 5.1 (of Lemma 5.1). *Let $\mathcal{D}$ be a finite set of descriptors w.r.t. query Q, program $\mathcal{P}$ and dependencies $\mathcal{C}$: $\mathcal{D} = \{E_i^{(p_i(t_i),fr_i)}\}_{1 \le i \le n}$. Denote with*

- *R the partial rewriting candidate described by $\mathcal{D}$,*

- $\mathcal{V}$ *the view fragments described by* $\mathcal{D}$,
- E *the expansion* $expand_{\mathcal{V}}(R)$.

If (a) R is safe and (b) there exists a containment mapping cfr from Q into $chase_{\mathcal{C}}(E)$, then Q is supported by $\mathcal{P}$ under $\mathcal{C}$.

We say that any set $\mathcal{D}$ as in Corollary 5.1 *witnesses support*. Notice that conditions (a) and (b) in Corollary 5.1 reformulate the corresponding conditions from Lemma 5.1 in terms of descriptors.

EXAMPLE 5.4. *The set of descriptors $\mathcal{D}$ in Example 5.3 witnesses support for the query, program and dependency in our running Example 1.1. Indeed, if we apply the test of Corollary 5.1 to the partial rewriting candidate R and the view fragments VF_1 and VF_2 described by $\mathcal{D}$(shown in Example 5.3), we obtain*

- *the expansion*

$$EF(A,B) \quad :- \quad f(A,X'), f(X',B), t(B,\text{“Paris”}), b(B,\text{“Paris”})$$

- *the result (cEF) of chasing EF with dependency (1),*

$$cEF(A,B) \quad :- \quad f(A,X'), f(X',B), t(B,\text{“Paris”}), b(B,\text{“Paris”}), \underline{s(B,\text{“Paris”})}$$

Notice that EF and cEF are fragments of E, respectively cE from Example 5.1. Let cfr be the mapping $\{A:A, B:B, C:X'\}$. Observe that (a) R is safe; and (b) cfr is a containment mapping from Q into cEF, thus satisfying the conditions of Corollary 5.1.

The number of descriptors is infinite due to the unbounded set of hidden variables, but there are only finitely many isomorphism types of descriptors modulo renaming of the hidden variables, in the following sense:

DEFINITION 5.2 (SIMILARITY). *Two descriptors $E_1^{(p_1(t_1),fr_1)}$ and $E_2^{(p_2(t_2),fr_2)}$ are* similar *iff $p_1 = p_2$ (and hence the distinguished variables of the descriptors are the same), $t_1 = t_2$, and there is an isomorphism i between the ranges of fr_1 and fr_2 which is the identity on the distinguished variables, and i witnesses the isomorphism of E_1 and E_2.*

Intuitively, the condition on fr_1 and fr_2 ensures that the partial containment mapping of Corollary 5.1, restricted to the view fragment, is the same for both descriptors. It is easy to see that similarity is an equivalence relation, and that there are only finitely many equivalence classes of descriptors under similarity. Indeed in $E^{(p(t),fr)}$, p is a predicate from $\mathcal{P}$; t a tuple of variables and constants from $chase_{\mathcal{C}}(Q)$, thus the number of distinct values it can take is polynomial in the size of $chase_{\mathcal{C}}(Q)$ and exponential in the arity of p; the number of distinct (up to isomorphism) partial mappings fr is exponential in the number of variables in Q.

Similarity plays a key role in our support test. Indeed we can show that any representative of a similarity equivalence class is as good as any member of the class for the purpose of witnessing support, in the following sense:

(†) if descriptor d_1 is similar to d_2, then for any set $\mathcal{D}$ of descriptors, $\mathcal{D} \cup \{d_1\}$ is a support witness if and only if $\mathcal{D} \cup \{d_2\}$ is one.

Algorithm findDescriptors. We next present a bottom-up algorithm for computing representatives of descriptor equivalence classes under similarity. The algorithm **findDescriptors** consists in initializing a set of descriptors $\mathcal{D}$ to the empty set, then repeatedly carrying out the rule steps described below until $\mathcal{D}$ reaches a fixpoint (under similarity), finally returning $\mathcal{D}$.

EDB rule step. Consider an EDB rule

$$e'(Z_1,\ldots,Z_k) :- e(Z_1,\ldots,Z_k)$$

For every variable mapping *to* from $Z_1,\ldots,Z_k$ into Q's variables and constants, such that the goal $e(to(Z_1),\ldots,to(Z_k))$ appears in $chase_{\mathcal{C}}(Q)$; and every partial variable mapping fr from the variables of Q to $\{Z_1,\ldots,Z_k\}$ (including the empty-domain one), add to $\mathcal{D}$ the descriptor $E^{(e(to(\bar{Z})),fr)}$, where $E = e(\bar{Z})$. Note that descriptors with empty-domain mappings capture the situation when none of the query goals maps into the described e goal[2].

IDB rule step. Consider an IDB rule

$$p(\bar{X}) :- p_1(\bar{X}_1),\ldots,p_n(\bar{X}_n)$$

If there exists a homomorphism h from the rule body into $chase_{\mathcal{C}}(Q)$, and a set of descriptors

$$E_1^{(p_1(h(\bar{X}_1)),fr_1)},\ldots,E_n^{(p_n(h(\bar{X}_n)),fr_n)}$$

in $\mathcal{D}$, then:

Construct the views $V_i : p_i(\bar{Z}_i) :- E_i$. Denote with E the expansion of the rule body using these views, and with xh_i the corresponding expansion homomorphism $xh_i : E_i \to E$ (i.e. the variable renaming performed on each V_i during expansion). Chase E with $\mathcal{C}$ and denote with ch the corresponding chase homomorphism $ch : E \to chase_{\mathcal{C}}(E)$. If the set $\{ch \circ xh_i \circ fr_i\}_{1\le i\le n}$ of partial mappings from Q into $chase_{\mathcal{C}}(E)$ is compatible, construct the combined mapping $cfr := \bigcup_{i=1}^{n} ch \circ xh_i \circ fr_i$, otherwise exit the rule step. For every partial mapping fr from Q into $chase_{\mathcal{C}}(E)$ which extends cfr (including the trivial extension $fr = cfr$) by mapping additional variables of Q into fresh variables added during the chase, compute the descriptor $d = F^{(p(h(\bar{X})),fr)}$, where F is the image under fr of all goals in Q such that fr is defined on all their variables. If d is not similar to any descriptor in $\mathcal{D}$, add it to $\mathcal{D}$.

EXAMPLE 5.5. *We next illustrate the rule steps of algorithm* **findDescriptors** *for Example 1.1 showing how descriptors d_1 and d_2 from Example 5.2 are derived. First, observe that no chase step applies on Q, so $Q = chase_{\mathcal{C}}(Q)$.*

For brevity, we work on the unnormalized program $\mathcal{P}$. Applications of EDB rule steps produce (among others) the fol-

[2]Technically, descriptors for EDB rule IDBs using empty-domain partial mappings do not fully conform to Definition 5.1 as the expansion fragment contains a goal that is not the image under the partial mapping. As seen in the IDB rule step, the definition holds for all other IDBs, which are the pre-normalization IDBs.

lowing descriptors:

$$d_3 = [f(Z_1, Z_2)]^{(f(A,C),\{A:Z_1,C:Z_2\})}$$
$$d_4 = [f(Z_1, Z_2)]^{(f(A,C),\{\})}$$
$$d_5 = [f(Z_1, Z_2)]^{(f(C,B),\{C:Z_1,B:Z_2\})}$$
$$d_6 = [f(Z_1, Z_2)]^{(f(C,B),\{\})}$$
$$d_7 = [t(Z_1, \text{“}Paris\text{”})]^{(t(B,\text{“}Paris\text{”}),\{B:Z_1\})}$$
$$d_8 = [b(Z_1, \text{“}Paris\text{”})]^{(b(B,\text{“}Paris\text{”}),\{B:Z_1\})}.$$

Notice that for the same match of EDB goal $f(Z_1, Z_2)$ into goal $f(A, B)$ of $chase_{\mathcal{C}}(Q)$, several partial mappings from the query are considered. We show only two here (in descriptors d_3 and d_4, where the latter uses the empty mapping, meaning that no query variable is mapped into its fragment).
An IDB rule step for the fourth $\mathcal{P}$ rule combines the descriptors d_5 and d_7 yielding a new descriptor:

$$(d_9) \quad [f(Z_1, Z_2), t(Z_2, \text{“}Paris\text{”})]^{(ind(C,B),\{C:Z_1,B:Z_2\})}$$

which combines with d_3 using the first rule of $\mathcal{P}$, yielding d_1.
Descriptors d_6 and d_8 combine via an IDB rule step with the third rule in $\mathcal{P}$ to

$$(d_{10}) \quad [b(Z_2, \text{“}Paris\text{”})]^{(ind(C,B),\{B:Z_2\})}$$

which combines with d_4 using the first rule of $\mathcal{P}$, yielding d_2.

We next prove that the inflationary process for descriptor discovery implemented by algorithm **findDescriptors** always terminates for weakly acyclic sets of constraints.

LEMMA 5.2. *If $\mathcal{C}$ is weakly acyclic, then algorithm* **findDescriptors** *is guaranteed to*
(a) terminate in time exponential in the sizes of $\mathcal{P}$, $\mathcal{C}$, and Q.
(b) output only descriptors, which are all pairwise dissimilar.

Algorithm testSupport. Our algorithm for testing support amounts to deciding if the descriptors computed by algorithm **findDescriptors** give a support witness (in the sense of Corollary 5.1). According to Corollary 5.1, the existence of such a witness is sufficient for support, but, due to our undecidability results, when the program is unrestricted (see Section 6), it is not always a necessary condition. That is why algorithm *testSupport* is in general only sound.

```
algorithm testSupport
input: query Q, program P, set of dependencies C;
begin
  N := the normalization of P;
  D := findDescriptors(Q, N, C);
  D' := all descriptors from D pertaining to
        distinguished predicates of N;
  if D' witnesses support (tested as in in Corollary 5.1)
    then return true;
    else return false;
  end
```

Algorithm **testSupport** satisfies the following properties.

THEOREM 5.1. *If $\mathcal{C}$ is weakly acyclic, the following hold: (1) algorithm* **testSupport** *is sound for testing support, and (2) it runs in time exponential in the size of $\mathcal{P}$, $\mathcal{C}$, and Q.*

Algorithm **testSupport** produces strictly less false negatives than the approach of reducing away dependencies described in Section 4. First, it is a decision procedure whenever the reduction succeeds:

THEOREM 5.2. *If $\mathcal{C}$ is weakly acyclic and $\mathcal{P}$ is a $\mathcal{C}$-local program generating $\mathcal{C}$-independent views, then algorithm* **testSupport** *is a decision procedure for support.*

COROLLARY 5.2. *If $\mathcal{C}$ is a weakly acyclic set of key and foreign key constraints, and $chase_{\mathcal{C}}(P)$ is safe for the keys in $\mathcal{C}$, then* **testSupport** *is a decision procedure for support.*

Second, the setting of Example 1.1 exhibits a case in which the restrictions required in Section 4 for reduction to the dependency-free case do not apply (they involve keys and foreign keys, while dependency (1) is neither). Indeed, it is easy to check that the chased program does not support the chased query in the absence of dependencies. We therefore need a qualitatively better approach, which is provided by algorithm **testSupport**: Example 5.5 shows that the call to **findDescriptors** yields (among others) the descriptors d_1, d_2, which, according to Example 5.4, witness support.

Algorithm testExpressibility. While we could use the reduction from expressibility to support used in Theorem 3.2, the following variation on **testSupport** constitutes a direct test: call **findDescriptors**, keep only the descriptors for distinguished IDB predicates, and perform the test of Corollary 5.1 only on singleton sets of descriptors.

Remark. In a setting in which sources expose limited query capabilities it is not enough to decide support: one would need to also find the actual witness for support and to minimize it. These can be done by simple bookkeeping extensions to our algorithm. The idea is to carry along with the descriptors an arbitrarily chosen view that represents the equivalence class of views described by the same descriptor. Details are explained in the extended version [4].

5.1 Revisiting the Dependency-free Case

Based on algorithm **testSupport**, we now improve the previously best-known upper bound for checking support in the dependency-free setting. [15] reported a deterministic doubly-exponential upper bound in the size of the query and program. We obtain an exponentially improved upper bound, implied by Theorem 5.2 and Theorem 5.1:

COROLLARY 5.3. *In the absence of dependencies, algorithm* **testSupport**
(a) is a decision procedure for support of a query Q by an arbitrary program $\mathcal{P}$, and
(b) runs in deterministic EXPTIME in the sizes of $\mathcal{P}$ and Q.

We next show that this upper bound is tight in the program size, and tight for practical purposes in the query size.

THEOREM 5.3. *$\text{SUPP}^{\emptyset}_{\mathcal{P}}(Q)$ is NP-hard in the size of Q and EXPTIME-complete in the size of $\mathcal{P}$.*

6. BOUNDARIES OF DECIDABILITY

We next justify the restrictions of Section 4 by exploring the boundaries of decidability for the problems of expressibility and support. To calibrate our results, we start with

the following: allowing unrestricted sets of constraints immediately leads to undecidability even if the program expresses a single view. This result is unsurprising given that unrestricted sets of embedded dependencies notoriously lead to undecidability of many fundamental database decision problems, such as equivalence of queries and implication of dependencies [1]:

THEOREM 6.1. *If $\mathcal{C}$ contains arbitrary embedded dependencies,* $\text{EXPR}^{\mathcal{C}}_{\mathcal{P}}(Q)$ *and* $\text{SUPP}^{\mathcal{C}}_{\mathcal{P}}(Q)$ *are undecidable even if $\mathcal{P}$ expresses a single view.*

Theorem 6.1 shows that decidability requires the set of constraints to conform at least to the restrictions yielding decidability in the single-view case. The most permissive restriction known to date requires $\mathcal{C}$ to be a weakly acyclic set of embedded dependencies [9, 10]. As we show below, weak acyclicity turns out to be too generous for sets of views described by unrestricted programs.

Indeed, it turns out that the interaction of recursion in the program and the presence of dependencies leads to undecidability even under strong restrictions on the dependencies and on the program which are known to lead to decidability in many classical decision problems as long as recursion and dependencies are mutually exclusive. For instance, query rewritability using finitely many views (listed explicitly, not described by a program) is known to be decidable under weakly acyclic dependencies [9], in particular under only functional dependencies (which include key constraints), or only full TGDs. In the absence of dependencies, expressibility and support for arbitrary recursive programs is decidable [15]. Moreover, many classical undecidable Datalog-related problems, such as containment and boundedness (undecidable by [12]) are known to become decidable for recursive *monadic* programs [6]. However when considering recursion and dependencies together, we obtain surprisingly strong undecidability results.

Recall that a program is *monadic* if all its IDB predicates have arity 1, and it is *linear* if each rule body contains at most one intentional subgoal.

THEOREM 6.2. *If $\mathcal{P}$ is recursive and not key-safe, then* $\text{EXPR}^{\mathcal{C}}_{\mathcal{P}}(Q)$ *is undecidable even if $\mathcal{C}$ consists of a single key constraint, and $\mathcal{P}$ is a linear monadic program.*

This justifies our key-safety restriction, showing that it is maximally permissive. Theorems 6.2 and 3.2 immediately yield:

COROLLARY 6.1. *If $\mathcal{P}$ is recursive and not key-safe, then* $\text{SUPP}^{\mathcal{C}}_{\mathcal{P}}(Q)$ *is undecidable even if $\mathcal{C}$ consists of a single key constraint and $\mathcal{P}$ is a linear monadic program.*

Sets of full TGDs are trivially weakly acyclic, and yet we have:

THEOREM 6.3. *If $\mathcal{P}$ is recursive, then* $\text{EXPR}^{\mathcal{C}}_{\mathcal{P}}(Q)$ *is undecidable even if $\mathcal{C}$ contains only full TGDs and $\mathcal{P}$ is a monadic program.*

COROLLARY 6.2. *If $\mathcal{P}$ is recursive, then* $\text{SUPP}^{\mathcal{C}}_{\mathcal{P}}(Q)$ *is undecidable even if $\mathcal{C}$ contains only full TGDs and $\mathcal{P}$ is monadic.*

Since INDs are a particular case of TGDs, it is interesting to contrast Theorem 6.3 and Corollary 4.3. Notice that there is no contradiction here, as weakly acyclic sets of INDs and sets of full TGDs have incomparable expressive power: weakly acyclic sets of INDs can express non-full TGDs, but INDs allow only one atom in the premise, while full TGDs allow multiple atoms.

7. RELATED WORK

The necessity of describing infinite families of views exported by the source was first argued in [20] and the problem of deciding support first solved (in the absence of constraints) in [14, 15]. [15] pioneers the idea of reducing support to rewriting the query using finitely many views. Views generated by the program are compared for *interchangeability*: V_1 and V_2 are interchangeable if in *every* rewriting R of Q, by replacing the V_1 goals with V_2 goals we still obtain a rewriting. [15] shows that interchangeability induces finitely many equivalence classes on the set of all views generated by the program, and gives an algorithm to find one representative of each class. This finite set of representative views is then used to check for a rewriting. The resulting algorithm runs in doubly-exponential deterministic time. We can show however that interchangeability under dependencies yields infinitely many equivalence classes, thus precluding the reduction from [15] (see Example A.1 in Appendix A). Even in the absence of dependencies, we observe that interchangeability is unnecessarily strong, leading to a refinement of the view equivalence classes that yields exponentially more representatives than truly needed. Intuitively, instead of interchangeability in *every* rewriting of Q, the descriptor similarity condition (†) from Section 5 detects interchangeability with respect to only the *canonical* rewriting. This allows us to manipulate mapping/partial mapping pairs rather than *sets* thereof as in [15], which yields the upper bound improvement from doubly-exponential to single-exponential time.

In the dependency-free setting, [23] improves the upper bound for support of [15] to non-deterministic exponential time in the combined query and program size. However for practical purposes this still yields implementations that run in doubly-exponential time. In addition to the extension to constraints, our solution improves on [23] even in the dependency-free case, by achieving an exponentially better upper bound, proven to be essentially tight.

The problem of support strictly extends that of rewriting queries using finitely many views. The latter was treated in depth in the literature, considering various extensions pertaining to the language of queries and views [13, 3, 2, 5], to adding limited access patterns for the views [11, 19], to adding constraints (see the references in [8]), and to mixing such extensions [7]. The problem is NP-complete in the size of the query and views, in practice leading to deterministic exponential-time implementations, which is no better than for support. Prior work on information integration [16] studied answering queries using a finite set of views with limited access patterns with a different goal, namely finding maximally contained answers.

8. CONCLUSION

In this paper, we revisit the problem of deciding support and expressibility of a conjunctive query by (possibly parameterized) views generated as the expansions of a Datalog program, investigating for the first time the effect of source constraints.

We identify practically relevant restrictions on the program which lead to decidability for the most prevalent constraints in practice (weakly acyclic sets of keys and foreign keys). Moreover, we show that even slight relaxations to our restrictions lead to undecidability. We present an algorithm which is applicable to unrestricted programs and

weakly acyclic sets of embedded dependencies, yielding a decision procedure in all known decidable cases, and a sound test in general.

We also settle two problems left open by work on the constraint-free case. First, we show that in the absence of constraints our algorithm is a decision procedure which improves the previously known upper bounds for support in the absence of constraints (from 2-EXPTIME [14] and NEXPTIME [23] to EXPTIME in the query and program size). We also give practically tight lower bounds, showing EXPTIME-hardness for fixed query and NP-hardness for fixed program. Second, we show that expressibility and support are interreducible in PTIME (even under constraints), which allows us to use essentially the same algorithm for solving them.

Note that the support problem discussed in this paper and in prior work decides whether a user query can be handled by the source or not by testing the existence of an *exact rewriting* using the generated views. This indeed represents the fundamental functionality one may expect in a limited-query-capability setting. However, when no equivalent rewriting exists, a user may also accept a best-effort approach in which instead of the exact answer she obtains its tightest *approximations*, either from below (contained in the answer) or from above (containing the answer). These approximations are known in the literature on view-based query rewriting as the *minimally containing* and *maximally contained* rewritings of the query. We leave for future research the problem of answering approximately a query using a potentially infinite family of views.

9. REFERENCES

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[2] F. N. Afrati, R. Chirkova, M. Gergatsoulis, and V. Pavlaki. Finding equivalent rewritings in the presence of arithmetic comparisons. In *EDBT*, pages 942–960, 2006.

[3] F. N. Afrati, C. Li, and P. Mitra. Answering queries using views with arithmetic comparisons. In *PODS*, 2002.

[4] B. Cautis, A. Deutsch, and N. Onose. Querying data sources that export infinite sets of views. Technical Report CS2007-0886, UCSD, 2007. Available at http://db.ucsd.edu/index.jsp?pageStr=publications.

[5] S. Cohen, W. Nutt, and Y. Sagiv. Rewriting queries with arbitrary aggregation functions using views. *ACM Trans. Database Syst. (TODS)*, 31(2):672–715, 2006.

[6] S. Cosmadakis, H. Gaifman, P. Kanellakis, and M. Vardi. Decidable optimization problems for database logic programs. In *STOC*, pages 477–490, New York, NY, USA, 1988. ACM Press.

[7] A. Deutsch, B. Ludaescher, and A. Nash. Rewriting queries using views with access patterns under integrity constraints. In *ICDT*, 2005.

[8] A. Deutsch, L. Popa, and V. Tannen. Query reformulation with constraints. *SIGMOD Record*, 35(1):65–73, 2006.

[9] A. Deutsch and V. Tannen. Reformulation of XML queries and constraints. In *ICDT*, 2003.

[10] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data exchange: Semantics and query answering. In *ICDT*, 2003.

[11] D. Florescu, A. Y. Levy, I. Manolescu, and D. Suciu. Query optimization in the presence of limited access patterns. In *SIGMOD*, pages 311–322, 1999.

[12] H. Gaifman, H. G. Mairson, Y. Sagiv, and M. Y. Vardi. Undecidable optimization problems for database logic programs. *Journal of the ACM (JACM)*, 40(3):683—713, 1993.

[13] A. Y. Levy, A. O. Mendelzon, Y. Sagiv, and D. Srivastava. Answering queries using views. In *PODS*, pages 95–104, 1995.

[14] A. Y. Levy, A. Rajaraman, and J. D. Ullman. Answering queries using limited external processors. In *PODS*, pages 227–237, 1996.

[15] A. Y. Levy, A. Rajaraman, and J. D. Ullman. Answering queries using limited external query processors. *J. Comput. Syst. Sci.*, 58(1):69–82, 1999.

[16] C. Li and E. Y. Chang. Query planning with limited source capabilities. In *ICDE*, pages 401–412, 2000.

[17] A. Maier, A. Mendelzon, and Y. Sagiv. Testing implications of data dependencies. In *PODS*, 1979.

[18] A. Motro. An access authorization model for relational databases based on algebraic manipulation of view definitions. In *ICDE*, pages 339–347. IEEE Computer Society, 1989.

[19] A. Nash and B. Ludäscher. Processing first-order queries under limited access patterns. In *PODS*, 2004.

[20] Y. Papakonstantinou, A. Gupta, H. Garcia-Molina, and J. D. Ullman. A query translation scheme for rapid implementation of wrappers. In *DOOD*, pages 161–186, 1995.

[21] S. Rizvi, A. O. Mendelzon, S. Sudarshan, and P. Roy. Extending query rewriting techniques for fine-grained access control. In *SIGMOD*, pages 551–562, 2004.

[22] V. Vassalos and Y. Papakonstantinou. Describing and using query capabilities of heterogeneous sources. In *VLDB*, pages 256–265, 1997.

[23] V. Vassalos and Y. Papakonstantinou. Expressive capabilities description languages and query rewriting algorithms. *J. Log. Program.*, 43(1):75–122, 2000.

APPENDIX

A. INTERCHANGEABILITY IS UNHELPFUL UNDER DEPENDENCIES

The following example shows that under dependencies, there are infinitely many equivalence classes of views with respect to interchangeability. This precludes the reduction described in [15] from the problem of support to that of rewriting using finitely many views, as it involves focusing on representatives of the equivalence classes.

EXAMPLE A.1. *We have a program $\mathcal{P}$ that produces unary views as follows:*

$$\begin{aligned} V(X) &:- e(X,a,Y), C_r(Y) \\ C_r(X) &:- e(X,a,Y), C_r(Y) \\ C_r(X) &:- e(X,b,Y), e(Y',b,Y), e(Y',a,Y'), \\ &\quad e(Y,up,Z) \\ C_r(X) &:- e(X,b,Y), e(Y',b,Y), e(Y',a,Y'), \\ &\quad e(Y,down,Z) \end{aligned}$$

Expansions are chains of a-labeled edges ending with a b-labeled edge and one of up *or* down.

Consider the query Q:

$$\begin{aligned} Q() &:- e(D,a,D), e(D,b,A), \\ &\quad e(A,up,B), e(A,down,C) \end{aligned}$$

The source obeys also one key constraint for each $l \in \{a,b\}$:

$$\forall X, Y', Y''\ e(X,l,Y'), e(X,l,Y'') \longrightarrow Y' = Y''.$$

We write V_n for the expansion with n a-labeled edges and ending with up. *We write U_n for the expansion with n a-labeled edges and ending with* down.

We can see that, for any n, the rewriting R_n defined as

$$R_n() :- V_n(X), U_n(X)$$

is an equivalent rewriting of Q.

However, replacing in R_n the V_n goal with any other view (V_i or U_i) would not yield another equivalent rewriting. So each V_n (and each U_n) is in its own equivalence class w.r.t. interchangeability in rewritings for Q. There are therefore infinitely many such equivalence classes.

B. REFINED VERSION OF KEY-SAFETY

The notion of key-safety presented in the main text keeps only track of the positions bound to constants, ignoring the actual constant values that may appear in these positions. As a consequence, it may fail to detect decidable instances of expressibility or support, where the constraints can still be ignored after the program and the query have been chased. We give in this section a refined definition for the key-safety restriction that addresses this problem, allowing us to solve strictly more cases by reduction to the dependency-free case.

Let R be a relation with an n-attribute composite key. By a *template of constants* (in short, template) for the key of R we denote a sequence of values $T = (v_1, \ldots, v_n)$, where each v_i can be either a constant value or a special value denoted *blank*. By the variable positions of T we denote the ordered sequence of positions $\bar{P}_T = (p_1, \ldots, p_k)$ of T that are occupied by *blank*.

We say that a rule of $\mathcal{P}$ *outputs the key of R, by template T, into the sequence of head variables $\bar{X} = (X_{i_1}, \ldots, X_{i_k})$* if $\bar{X}$ appears in the rule body either

- in the positions $\bar{P}_T = (p_1, \ldots, p_k)$ of the key attribute sequence of some R-subgoal, with the remaining $n-k$ positions (if any) of the key being bound to the constant values given in T.
- in the positions $j_1, \ldots, j_k$ of some p-subgoal, where p is an IDB predicate with at least one rule that in turn outputs the key of R by the template T, into the sequence of head variables with indices $j_1, \ldots, j_k$.

We say that a subgoal g outputs the key of R, by template T, into the sequence of variables $\bar{X} = (X_{i_1}, \ldots, X_{i_k})$ if

- g uses EDB predicate R and $\bar{X}$ appears in positions $\bar{P}_T = (p_1, \ldots, p_k)$ in the key attributes of g, with the remaining $n-k$ positions (if any) of the key being bound to the constant values given in T, or
- g uses IDB predicate p and there exists some rule defining p which outputs the key of R, by the template $\bar{T}$, into variables $\bar{X}$.

We say that a *rule is safe* for the key constraint on R if whenever one of its IDB subgoals outputs the key of R by some template of constants T into k variables $\bar{X} = (X_{i_1}, \ldots, X_{i_k})$, no other subgoal does the same (for the same template T). Notice that several EDB subgoals may output the key of the same R by the same template and into the same sequence of variables $\bar{X}$, as long as no IDB goal does. A *program $\mathcal{P}$ is key-safe* for a set of key constraints $\mathcal{K}$ if

- each rule is safe for all key constraints in $\mathcal{K}$, and
- for all distinguished predicates *ans* of $\mathcal{P}$, all defining rules r of *ans*, and all relational symbols R in the schema, if r outputs the key attributes $\bar{A}$, by some template, of some goal $R(\bar{A}, \bar{B})$, it also outputs all non-key attributes $\bar{B}$, by some template (using the same definition that applied to the key attributes).

If $\mathcal{I}$ is a set of weakly acyclic INDs, we say that $\mathcal{P}$ is key-safe for $\mathcal{C} = \mathcal{K} \cup \mathcal{I}$ if $chase_{\mathcal{I}}(\mathcal{P})$ is key-safe for $\mathcal{K}$. Notice that this new definition of key-safety can still be checked in PTIME in the size of $\mathcal{P}$ and $\mathcal{K}$.

EXAMPLE B.1. *Assuming the schema and constraints of Example 4.3, consider the following modified program $\mathcal{P}'$*

$$\begin{aligned} (\mathcal{P}')\ ans(A,B,C,O) &:- ind(A,B,O), ind'(B,O) \\ ind(B,C,O) &:- t(B,B',O), ind(B',C,O) \\ ind(B,C,O) &:- t(B,C,O) \\ ind'(B,O) &:- ind_P(B,O), ind_{SD}(B,O) \\ ind_P(B,O) &:- t(B, \text{“}Paris\text{”}, O) \\ ind_{SD}(B,O) &:- t(B, \text{“}SanDiego\text{”}, O) \end{aligned}$$

Notice that $\mathcal{P}'$ is not key-safe under the weaker restriction, since the rule defining ind' is not safe. But we can easily see that the two constants appearing in the second attribute of the key cannot be equated during the chase, and $\mathcal{P}'$ is indeed key-safe under the refined definition.

More precisely, in the sixth rule, b outputs the key into the sequence of variables B, by the template

$$T_1 = (blank, \text{“}SanDiego\text{”}).$$

Similarly, in the fifth rule, b outputs the key into the sequence of variables B, by the template

$$T_2 = (blank, \text{“}Paris\text{”}).$$

Since T_1 and T_2 are different, the rule defining ind' is safe. Then, in the first rule, the ind' subgoal outputs the key in B, by any of these two templates. Finally, ans outputs the key attributes in A, B (by the ind subgoal) and in B (by the ind' subgoal) but in both cases it also outputs O, the non-key attribute.