Access Patterns and Integrity Constraints Revisited

Vince Bárány
Department of Mathematics
Technical University of Darmstadt
barany@mathematik.tu-darmstadt.de

Michael Benedikt
Department of Computer Science
Oxford University
michael.benedikt@cs.ox.ac.uk

Pierre Bourhis
Department of Computer Science
Oxford University
pierre.bourhis@cs.ox.ac.uk

We consider which queries are answerable in the presence of access restrictions and integrity constraints, and which portions of the schema are accessible in the presence of access restrictions and constraints. Unlike prior work, we focus on integrity constraint languages that subsume inclusion dependencies. We also use a semantic definition of answerability: a query is answerable if the accessible information is sufficient to determine its truth value. We show that answerability is decidable for the class of guarded dependencies, which includes all inclusion dependencies, and also for constraints given in the guarded fragment of first-order logic. We also show that answerable queries have "query plans" in a restricted language. We give corresponding results for extractability of portions of the schema. Our results relate querying with limited access patterns, determinacy-vs-rewriting, and analysis of guarded constraints.

1. INTRODUCTION

Querying under access patterns has been the object of considerable research over the past decade. Access patterns are restrictions in a schema, stating that a relation can only be queried by a lookup on certain attributes. The most fundamental problem considered is: given a schema with access patterns and a query $\mathcal{Q}$, is $\mathcal{Q}$ *answerable*: i.e. is it true that we can obtain the full answer to $\mathcal{Q}$ on every input database, making use of the given access patterns. A basic setting is where there is one access method per relation, and accesses are *exact*: they return all tuples. In this setting, Li characterized which conjunctive queries are answerable, showing that the problem of answerability is NP-complete [1]. Answerable queries in this setting can be answered in a simple way – there is an equivalent conjunctive query such that a naive evaluation of atoms in the order they appear in the query produces the complete query answer. For instance, a query $\mathcal{Q} = \exists x\ S(x) \wedge R(x, y)$ can be naively evaluated if there is unrestricted access to S but access to R only on the first position – the evaluation would first get all values for S, then plug the resulting set into R. In the terminology of [2] such a query is "executable" and a query equivalent to it is "feasible", [1] calls a query of this sort "orderable" and a query equivalent to it "stable".

Later work considered extensions to more complex settings, including richer query languages (e.g. [3, 4]). Of particular interest to us is the work of Deutsch, Ludäscher and Nash, which considers richer schemas, including integrity constraints in addition to access patterns [2]. They deal with a general class of integrity constraints, tuple-generating dependencies. One of their main results is that for classes of constraints that satisfy a certain condition (chase termination) one can determine whether a query has an executable rewriting over structures satisfying the constraints. Unfortunately, the chase termination condition is undecidable. Furthermore, many basic integrity constraints, such as referential constraints, do not lead to chase termination.

Here we will take another look at query answering with access patterns and integrity constraints. Our focus will be on inclusion dependencies and some natural (and wide-ranging) generalizations, including *guarded tuple-generating dependencies* and *guarded constraints*. This class of dependencies is important in practice, is orthogonal to chase termination, and allows (we believe) a cleaner theory of answering under constraints.

We will also distinguish several notions of data management under access patterns and constraints. We first look at a stronger query-independent property that we denote *super-extractability*. A schema is super-extractable if we can get at a superset of the values within it using the access patterns. More generally, we will consider which portions of the schema are super-extractable – a position in a relation is super-extractable if we can get at all the values that occur in that position using the access methods.

Example 1.1. *Consider a schema with ternary relations R and S, along with unary relation U. The constraints are inclusion dependencies $R[1,2] \longrightarrow S[1,2]$, $S[3] \longrightarrow U[1]$, $R[1] \longrightarrow U[1]$. $R[1,2] \longrightarrow S[1,2]$ denotes the fact that every tuple t in R has a tuple $s \in S$ such that $s.1 = t.1 \wedge s.2 = t.2$.*

There is an access method AcM_R on R using positions 1 and 2 as the input, an access method AcM_S on S using positions 1 and 3 as the input, and an input-free access method AcM_U to U.

We can extract all of relation R from any structure for the schema. To do this, we extract all of relation U using AcM_U, putting the result in a set U_0. We then bind all pairs of values from U_0 within AcM_S, resulting in a set S_0. Since U_0 must contain $R[1]$ and $S[3]$, we can see that S_0 must contain all tuples that are images of R under the inclusion

EDBT/IDBT'13 March 18-22 2013, Genoa, Italy

dependency from R to S, i.e. the projection of S_0 onto the first two components must contain $\pi_{\#1,\#2}R$. Thus if we take all pairs of values from S_0 and plug them into AcM_R, we will obtain the full value relation R. In our terminology we state that every position of R is super-extractable (see Section 3 for the precise definition of super-extractability).

Super-extractability is not necessary for query answering, but it is clearly a desirable property, and it allows certain issues in the interaction of constraints and access patterns to be neglected. We will also investigate a variant of super-extractability which states that we can determine exactly which values are in a given portion of the schema (we say that the corresponding position is *extractable*).

We then turn to query answering under schemas that include access patterns and constraints. We distinguish several notions of what it means to answer a query under a schema. The first is semantically defined: a query is *access-determined* if the information we can obtain by extracting all the accessible facts is sufficient to determine the truth value of the query. We will show that this property is decidable for a large class of constraints, and that when a query is access-determined we have an algorithm for answering it that has low complexity in the size of the database.

EXAMPLE 1.2. *Suppose our schema has a binary relation A, a ternary relation B and inclusion dependencies: $A[1,2] \longrightarrow B[1,2]$ and $B[2,3] \longrightarrow A[1,2]$. The access methods in the schema give the ability to freely query relation B, but only the ability to do membership tests on A.*

The query $\mathcal{Q} = \exists y A(x,y)$ is access-determined, since we can obtain the full picture of relation B, and using the first inclusion dependency and the membership test, determine all of relation A.

The second inclusion dependency is not critical for this reasoning. However, the two dependencies together are cyclic, so any algorithm based on "chasing until termination" – roughly, trying to enumerate all the consequences of the query under the dependencies (e.g. [2]) – will not terminate.

We compare this semantic notion to the syntactic notion of answerability. We define a general notion of "access-rewritability" for a query under access restrictions and constraints – a query is access-rewritable in a logic if it can be evaluated by running a query in the logic on the accessible data. We show that for guarded constraints, any query that is access-determined has an access-rewriting in first-order logic.

EXAMPLE 1.3. *Consider a schema with unary relations A, B, D and a binary relation C and the integrity constraint: $\forall x[A(x) \leftrightarrow (D(x) \wedge (\forall y\ C(x,y) \rightarrow B(y)))]$, which states that the predicate A is a view defined exactly by the formula $D(x) \wedge (\forall y\ C(x,y) \rightarrow B(y))$.*

There are input-free accesses on C, B and D, but on A there is only an access method requiring a value for the first position.

Consider the query $\mathcal{Q} = \exists x A(x)$. $\mathcal{Q}$ is access-determined, since we can access completely all relations in the definition of the view A, and thus use the constraint to get all values of $A(x)$. It is easy to see that $\mathcal{Q}$ is not equivalent to any executable conjunctive query or even to a union of conjunctive queries. In our sense, $\mathcal{Q}$ is rewritable as: $\exists x D(x) \wedge (\forall y\ C(x,y) \rightarrow B(y))$.

Our results on rewriting for guarded constraints will make use of tools developed for the recently-defined *guarded negation fragment* [5].

We also show that for restricted classes of constraints we get a rewriting as a positive existential first-order formula, which will correspond to the notion of rewritability used in all earlier work. For instance, our methods will allow us to determine that the query in Example 1.2 is access-rewritable, with the rewriting being $\exists xy B(x,y)$.

Putting all of our results on query answering together, we get a new method for showing that queries are stable in the presence of integrity constraints. We examine the complexity of finding the rewritings for each considered subclass.

In summary our contributions are:

- Algorithms for determining whether a portion of the schema can be extracted by using the given access methods, exploiting the constraints.
- Results showing that when a position can be extracted, there are very simple "static extraction plans" that can perform the extraction.
- Matching upper and lower bounds on the complexity of extraction for guarded constraints, guarded tuple-generating dependencies, linear guarded (aka LAV) dependencies, and inclusion dependencies.
- Algorithms for determining whether a query is "semantically-answerable" (access-determined), along with complexity bounds.
- Results that show that when a query is access-determined it has a rewriting in a restricted class.

Our results shed light on the relationship between semantic implication and rewriting in the context of access restrictions, and also relate work on access restrictions to recent work on interpolation and decidability for restricted logics [5, 6, 7].

Organization: Section 2 gives the basic definitions. Section 3 deals with extractability, while Section 4 deals with access-determined queries. Section 5 explains the connection with prior work.

2. DEFINITIONS AND PROBLEM STATEMENT

Schemas, structures, and access paths. Throughout this work we deal with schemas that consist of:

- relations, each with a given arity; a number bounded by the arity of relation R will be called a *position* of R. For simplicity we assume that each position of each relation can store data of some fixed infinite datatype, say integers.
- integrity constraints, to be described below
- access methods, each consisting of a relation symbol R and a set of positions of R, the *input positions of the method*. Unlike most prior work, we allow multiple methods per relation. We also allow the possibility that a relation has no accesses – i.e. we can learn about the values of such a relation only via integrity constraints with other relations.

A structure I for S is defined as usual ([8]), consisting of a set $dom(\mathsf{I})$ (the domain) along with a mapping taking each relation symbol R in the schema to a relation $\mathsf{I}(R)$ on D having the arity of R. Since our schemas include constraints, if we say I is an S-structure we will mean that the constraints in S hold in I. We will often use the basic terminology of model theory, including the various notations for saying that

a formula holds given a valuation and a structure. We say that a structure is a *model* of a sentence, meaning that it satisfies the sentence according to the usual semantics of first-order logic. The vocabulary of a structure is the set of relations it interprets. A *reduct* of a structure to a subset V_0 of its vocabulary is obtained by removing all of the predicate symbols outside the vocabulary. The *expansion* of a structure is formed by adding on relations for new symbols, leaving the interpretations in the original vocabulary the same.

When we talk about a structure for a schema S consisting of constraints and access restrictions, we will mean by default a finite structure. However, in our results we will consider only integrity constraint languages where the restriction to finite structures is not critical, and thus statements of the form "on every structure ...", can generally be interchanged with "on every finite structure ..." without changing the truth value of the results.

An *access* consists of an access method to some relation R of the schema and a binding of the input positions of the method to values. The output of such an access on a structure I is the set of all tuples in $\mathsf{I}(R)$ that project onto the binding. Given a structure I and a set of seed constants C_0, a *valid access path* (for S and C_0) is a sequence of accesses a_i and (alternating) outputs o_i such that all values in the input binding of a_i occur in the output of an earlier a_j or are constants of C_0. We often drop the qualifier "valid" for brevity, and just refer to an access path. The set of constants C_0 can be considered a parameter in accessibility problems. However it can be easily shown that adding constants other than those in the constraints and the queries under consideration (e.g. for query-answering problems) does not make any difference in the problems we consider, given our assumptions on the schema (only integer attribute datatypes). Hence *we will always take C_0 to be the constants in the constraints unioned with constants in the query (for query answerability).* It is easy to modify our results for schemas in which the attributes can be of enumerated type.

An access path is a way of getting information about a particular structure. We will be interested in the ability to perform some task (e.g. answering a query) that requires finding an access path in *every* structure satisfying the schema. An important question is: when this is possible, can we get some convenient static description of the access path that is required in each structure. Generalizing the notion of Chang and Li [9] to the case of multiple-accesses per relation, we define a conjunctive query $\mathcal{Q} = \exists \vec{x} \bigwedge_{i \leq n} A_i$ to be *executable* if there are access methods AcM_i on the relation of atom A_i such that for every $i \leq n$ and for each variable x that occurs in A_i within input positions of AcM_i, there is $j(x) < i$ such that x occurs in A_j in a non-input position of AcM_j. On any structure, we can generate an access path P_k that implements $\mathcal{Q}_k = \bigwedge_{i \leq k} A_i$ by induction on k, where in the inductive step we add on all accesses that bind input positions of AcM_i in which a variable x of A_i occurs with values from $\mathsf{P}_{j(x)}$, while positions where A_j has a constant c, c is put in the binding. An executable conjunctive query equivalent to $\mathcal{Q}(x_i) = \exists x_1 \dots x_{i-1} x_{i+1} \dots x_n \; R(x_1 \dots x_n)$ will be called a *static extraction plan* for $R[i]$.

A structure I' is *consistent* with an access path p of the form $a_1, o_1 \dots a_i, o_i$ and schema S if I' satisfies the constraints in the schema and for each access a_i in p the output o_i of a_i in p is the same as the output of a_i on I'. A boolean query $\mathcal{Q}$ is *certain* after access path p if it is true on every structure consistent with p. Likewise, we say $\mathcal{Q}$ is *inconsistent* after path p if it is false on every consistent structure.

Constraints. The most general class of constraints we will look at are sentences of the *guarded fragment* GF [10]: given any signature consisting of relations and constant symbols only, that is the fragment of first-order logic built up from atomic formulas via boolean connectives and guarded existential quantification $\exists \vec{y} \; R(\vec{x}, \vec{y}) \wedge \phi(\vec{x}, \vec{y})$, where $\vec{x}, \vec{y}$ contain all free variables of ϕ. From the definition, one easily sees that GF is also closed under guarded universal quantification $\forall \vec{y} \; R(\vec{x}, \vec{y}) \rightarrow \phi(\vec{x}, \vec{y})$. GF contains all inclusion dependencies, and also contains constraints with other quantifier alternation patterns, such as the constraint in Example 1.3.

The class of *guarded dependencies*, GTGD consists of sentences of the form:

$$\forall \vec{x} R(\vec{x}) \wedge \phi(\vec{x}) \rightarrow \exists \vec{y} \bigwedge_j R'_j(\vec{x}, \vec{y})$$

where ϕ is a conjunction of atoms (possibly empty) and each R and R'_j are atoms (possibly with repeated variables). *Linear-guarded dependencies*, $\mathsf{LinGDep}$, introduced in [11] represent a special case in which ϕ is empty: that is, the left-hand side of the implication is simply an atom.

The most restricted class of constraints we consider here is that of inclusion dependencies (ID), which are, of course, linear-guarded dependencies whose right-hand side has a single atom, with no repeated variables in atoms. Thus, $\mathsf{ID} \subset \mathsf{LinGDep} \subset \mathsf{GF}$.

We do not have $\mathsf{GTGD} \subseteq \mathsf{GF}$, but we can transform every GTGD sentence to a GF sentence that behaves similarly: we simply add an additional "dummy guard relation" covering the head of each clause. We will show directly that all the results we get for GF carry over to GTGD.

Super-Extractability. The first problem we study is, informally: what subset of the data can we obtain from a database in advance of querying? Can we get all the data in a given relation? Can we get all the data in a given position of a relation? Here we are interested in finding out whether one can obtain all the values that occur in a particular position of a relation R through repeated and nested invocation of the access methods at hand. This is a trivial notion without integrity constraints – the only way we can extract all the data from a position is if we have an input-free access on that relation. But in the presence of integrity constraints the notion is non-trivial: if we have an inclusion constraint from R to S, then we may be able to get data from R by getting it from S.

We will consider the ability to get a superset of the values, resulting in the notion of super-extractability, which we now explain in detail. Given a schema S, a position i of relation R and a finite structure I we say $R[i]$ is *super-extractable* on I if every value v of $\pi_i(\mathsf{I}(R))$ is an accessible value of I; that is there is a valid access path p where v occurs in one of its outputs. $R[i]$ is super-extractable (over S) if it is super-extractable on every structure I satisfying S.

Given a value v in a structure I satisfying S, the *extraction depth* of v is the shortest valid access path that returns v (and is infinite otherwise). For a super-extractable position $R[i]$, we say the extraction depth of the position is the supremum over all finite structures I of S and all values v in $R[i]$ of the extraction depth of v.

If the extraction depth of $R[i]$ is some $k < \infty$, then it is easy to see that performing the concatenation of all access

paths of length k will yield a substructure where all values of $R[i]$ are present. Also note that if $R[i]$ has a static extraction plan, then the extraction depth is finite, and is bounded by the length of the plan.

If $R[i]$ is super-extractable, we can obtain (a superset of) all the values in the i^{th} projection, but we may not be able to determine whether a value is in the projection or not. For example, consider a schema with binary relation R and unary relation T. Suppose we have an inclusion dependency $R[1] \longrightarrow T[1]$ and T has an input-free access. Then $R[1]$ is super-extractable, since we get all of its values by querying T. However we will not know whether these values are actually in R or not. Later in the paper we will consider a variant of super-extractability, *extractability*, that tells exactly which values are present.

Query answering I: semantic answering. We now look at boolean conjunctive queries $\mathcal{Q}$, and discuss what it means to answer Q in the presence of constraints and access patterns. Unlike prior work ([2]) we start with a semantic notion of answerability. Informally, a query is answerable with respect to S if on every structure conforming to S we can extract enough information to determine the satisfaction of $\mathcal{Q}$.

$\mathcal{Q}$ is *access-determined* on I (with respect to S) if there is a valid path P, such that $\mathcal{Q}$ is either inconsistent or certain after P: in this case we say that "P witnesses that $\mathcal{Q}$ is access-determined on I". We say $\mathcal{Q}$ is access-determined on S if it is access-determined on every finite structure I satisfying S.

We can restate being access-determined using the following terminology: the *accessible substructure* of a structure I is the structure I' for the vocabulary of I such that for any relation R in the vocabulary, a tuple is in $\mathsf{I}'(R)$ iff it is returned by an access to R in a valid access path in I. The domain of the structure is the union of all values occurring in such tuples. We also say that I' contains exactly the *accessible facts* of R, and refer to it as the "accessible part of I", writing $\mathsf{AccessiblePart}(\mathsf{I})$. For a number k, the *k-accessible substructure* is similarly defined as the structure containing all facts that are extractable using a path of size at most k.

Note that if S contains at least one access method for every relation, then for any structure I, the accessible substructure of I is the induced substructure obtained by restricting I to the *accessible values*, where these are the values that occur in the output of some valid access path.

The following lemma rephrases being access-determined in this terminology:

LEMMA 2.1. *$\mathcal{Q}$ is access-determined iff for any two finite structures $\mathsf{I}_1, \mathsf{I}_2$ for the schema S, if they have the same accessible substructure then either both of them satisfy $\mathcal{Q}$ or neither of them do.*

PROOF. Suppose $\mathcal{Q}$ satisfies the condition above, and let I be a structure where $\mathcal{Q}$ holds. Let P be a path that includes all valid accesses on I (such a path clearly exists). Then $\mathcal{Q}$ is certain after P, since any structures agreeing on P will have the same accessible substructure as I. If $\mathcal{Q}$ does not hold in I the same argument shows that Q is inconsistent after P. Thus $\mathcal{Q}$ is access-determined.

Conversely, suppose $\mathcal{Q}$ is access-determined, and let $\mathsf{I}_1, \mathsf{I}_2$ be two structures of the schema sharing the same accessible substructure. Let P be a path such that $\mathcal{Q}$ is either inconsistent or certain after P. We can take P to contain all valid accesses on I_1. The hypotheses imply that performing the same accesses as in P on I_2 must give the same result. Thus if $\mathcal{Q}$ is certain after P on I_1, the same must be true on I_2, and similarly if $\mathcal{Q}$ is inconsistent. □

An a priori weaker condition considers only substructures on which the query is true: on every valid structure I on which $\mathcal{Q}$ holds, there is a valid path P on I such that $\mathcal{Q}$ is certain after P. We say "a priori weaker", since it is easy to see that these are equivalent (e.g. using the lemma above). We will thus use this definition interchangeably with the others.

Query answering II: answering a query with bounded information. In our definition of answering above, we required only that the answer to the query can be determined by some path, but the length of this path may be unbounded. It is easy to see that, as with super-extraction, we may need paths of unbounded length, since we may need to put in each output of some initial access. We will thus look for situations in which the answer to a query must be known after *all* access paths of a given length are explored.

We say $\mathcal{Q}$ has *answering depth* k if for any finite structure I for the schema if we let p be the concatenation of all valid paths of length at most k, then $\mathcal{Q}$ is either certain or inconsistent after p. We also say that $\mathcal{Q}$ is k-access-determined in this case. Again it is sufficient to check that when $\mathcal{Q}$ holds in the structure, it is certain after all paths of length k are tested.

Query answering III: rewritings. The notion of access-determinacy is semantic, in that the plan only gets enough information to answer the query – it does not tell how to effectively obtain the answer itself. If $\mathcal{Q}$ is access-determined, then to answer it we can perform all the valid accesses on a structure I until a fixed point is reached, at which point we know that the query must either be certain or inconsistent after that resulting path p. Checking whether $\mathcal{Q}$ is certain or inconsistent on a structure is a matter of "query answering with respect to integrity constraints": the constraints being the integrity constraints in the schema and the constraints expressing that the target structure must be consistent with p. Here query answering with constraints means checking whether a query is implied by the constraints and the facts holding in a particular structure. It is known that for guarded constraints this problem is 2EXPTIME-complete [7]. We will present a superior method for answering an answer-determined query, based on rewritings.

Prior work [9, 3] has considered syntactic notions of answerability, restricting to the setting where there is a single access per relation. There the focus has been on unions of conjunctive queries that are *executable* – those that are the union of executable CQs, where executability is as defined previously.

We will consider a more general version of rewritability, allowing querying over the accessible substructure. For a logic L whose vocabulary consists of predicates in the original schema S, a query $\mathcal{Q}$ is L *access-rewritable* for schema S if there is a query $\mathcal{Q}'$ in L such that for all finite structures I satisfying S, $\mathcal{Q}$ is true in I iff $\mathcal{Q}'$ is true in the accessible substructure of I. For a number k, a query $\mathcal{Q}$ is L, k-access-rewritable for schema S if there is a query $\mathcal{Q}'$ in L such that for all finite structures I satisfying S, $\mathcal{Q}$ is true in I iff $\mathcal{Q}'$ is true in the k-accessible substructure of I. Since this is the major kind of rewriting we consider in the paper, we

will generally drop the prefix (access-), and just refer to L-rewritable, L, k-rewritable, etc.

EXAMPLE 2.2. *Consider Example 1.3. The query is* FO*-access-rewritable, and is in fact* FO, 1*-rewritable, since we can obtain the answerable part with paths of length* 1*. Indeed, it is* GF, 1*-access-rewritable, since the rewriting is in the guarded fragment.*

Li and Chang [9] initiated the study of rewritability. They prove that in the absence of constraints answerability is equivalent to rewritability. This was extended by Nash and Ludäscher [3] to unions of conjunctive queries. Nash and Ludäscher use the terminology "executable" as we do, and use "feasible" for what we term rewritable. We will show (Proposition 4.18) that $\exists^+\mathsf{FO}, k$-access-rewritability corresponds to feasibility in the sense of [3], where $\exists^+\mathsf{FO}$ is positive existential first-order logic.

3. SUPER-EXTRACTABILITY

We will study which relation positions are super-extractable, beginning with schemas in which constraints are guarded sentences, our largest class. We will show that we can decide super-extractability for this class, and that for super-extractable positions we can perform the extraction in a particularly simple way.

We then turn to more restricted classes starting with inclusion dependencies and then expanding to more general classes. In those cases we show that we can decide super-extractability more efficiently.

Our definition of super-extractability allows us to over-estimate the data in a given position. We discuss getting "the best" extraction plan – one that gives the most precise over-estimate.

Super-extractability for guarded constraints. Our first observation is that super-extractability is decidable for guarded constraints. We will make use of the following fundamental results of Grädel:

THEOREM 3.1. *[10] One can decide whether a guarded sentence has a model in* 2EXPTIME*, and the problem is complete for* 2EXPTIME*. When the arity is bounded, the complexity becomes* EXPTIME*-complete. Furthermore, any guarded sentence that has a model has a finite model.*

We now recall (e.g. from [2]) the usual "axioms" for a set of access methods. Given a set of access methods AM over relations S, we consider a conjunction ϕ_{AM} over the schema S^+ consisting of S and an additional relation AccVal. For each access method $a \in AM$ with relation R and for each position j of R, ϕ_{AM} has the conjunct:

$$\phi_{a,j} = \forall \vec{x}\ \big(R(\vec{x}) \wedge \bigwedge_{i \in Input(a)} \mathsf{AccVal}(x_i)\big) \rightarrow \mathsf{AccVal}(x_j)\,.$$

These are guarded dependencies that can also be directly rewritten in GF. In addition, ϕ_{AM} has conjuncts stating that constants are accessible. Note that all of these conjuncts are guarded. Let $con(\mathsf{S})$ be the constraints in schema S.

We now show:

CLAIM 3.2. *Position i of relation R is super-extractable iff $\psi = con(\mathsf{S}) \wedge \phi_{AM} \wedge \exists \vec{z} R(\vec{z}) \wedge \neg \mathsf{AccVal}(z_i)$ is unsatisfiable.*

PROOF. Suppose $R[i]$ is not super-extractable. Then there is a structure I and a tuple $\vec{c} \in I(R)$ such that c_i is not returned in any valid access path. Expand I by letting $\mathsf{AccVal}(x)$ hold of any value that is returned by a valid access path on I; then the resulting structure witnesses the satisfiability of the sentence ψ above.

In the other direction, suppose ψ is satisfiable by some structure I. By the finite model property of the Guarded Fragment I can be taken to be finite. Let A_1 be $I(\mathsf{AccVal})$ and A_0 be a set of tuples for the relation AccVal obtained by using the sentences $\phi_{a,j}$ above as Datalog rules, and taking the usual least fixpoint semantics. Then A_0 is contained in A_1, since A_1 is a fixpoint. If we replace A_1 by A_0 in I, we do not change the truth value of ψ, since $con(\mathsf{S})$ does not mention the predicate AccVal and the other conjuncts are monotone as AccVal decreases (since each conjunct in ϕ_{AM} quantifies only universally over AccVal and the other conjunct quantifies existentially over $\neg\mathsf{AccVal}$). The resulting structure thus serves as a counterexample to super-extractability. □

We thus have:

THEOREM 3.3. *For any schema with guarded constraints, one can decide whether a position is super-extractable in* 2EXPTIME*. Conversely, the problem is* 2EXPTIME*-hard.*

The hardness follows from prior results on satisfiability, because in the absence of accesses, a position is super-extractable (vacuously) exactly when the constraints are inconsistent.

We can also conclude that the extraction depth is finite:

THEOREM 3.4. *For any schema consisting of guarded constraints, for any position i in relation R, if $R[i]$ is super-extractable then it has finite extraction depth.*

PROOF. Suppose $R[i]$ has extraction depth above k for every number k. There is a sentence ϕ_k of first-order logic over the relational schema that asserts this, hence by the compactness theorem there is a model M (not necessarily finite) for the relations in the schema, and an element v in $\pi_i[R]$ in M such that no valid (finite) access path extracts v. Let M' expand M by letting AccVal be the set of elements obtainable via a finite access path.

Then M' is a model of the extraction axiom ψ of Claim 3.2, hence by that claim $R[i]$ is not super-extractable. □

Super-extractability for dependencies. We now look at cases where we can get a better extraction algorithm.

We review the well-known *chase* algorithm in the general context of "tuple generating dependencies", that is, constraints of the form:

$$\forall \vec{x}\ \phi(\vec{x}) \rightarrow \exists \vec{y}\ \phi'(\vec{x}, \vec{y})$$

where ϕ and ϕ' are conjunctions of atoms from a relational signature.

Given a structure I and schema S consisting of tgds only, the *chase sequence* is a sequence of structures $\langle C_n \rangle$ in the vocabulary of S expanded with the predicate AccVal, defined as follows:

- $C_0 = \mathsf{I}$
- C_{n+1} is formed from C_n by applying a "chase step" with respect to the dependencies and accessibility axioms of ϕ_{AM} from Section 3 to every structure $\mathsf{I} \in C_n$. A chase

step is defined exactly analogously in the classical case (see e.g. [12], for inclusion dependencies, and [13] for extensions to (positive) dependencies): In a single chase step, applied to structure I, for every $\vec{c} \in \mathsf{I}$ that satisfies $\phi(\vec{c})$ but where $\mathsf{I}, \vec{c}$ does not satisfy $\exists \vec{y}\, \phi'(\vec{x}, \vec{y})$, we create a new tuple $\vec{d}$ and add facts so that $\mathsf{I}, \vec{c}, \vec{d} \models \phi'(\vec{x}, \vec{y})$.

We denote the union of the C_n by $\mathsf{Univ}(\mathsf{Sch}, \mathsf{I})$, referring to it also as the *chase structure* or *universal structure*. It is easy to see that it is uniquely defined up to isomorphism.

We will consider how the chase simplifies the analysis of super-extractability.

Consider the structure consisting of just fact $R(c_1, \ldots, c_n)$, where $\vec{c}$ is a tuple of distinct constants, and the schema S^+ above – i.e. the original dependencies plus the "axioms" for the accessibility predicate. Then we claim :

CLAIM 3.5. *If all dependencies are guarded, then $R[i]$ is super-extractable iff the value c_i is an accessible value in the chase structure iff $\mathsf{AccVal}(c_i)$ holds in the chase structure. Furthermore, the extraction depth of c_i in the chase structure bounds the extraction depth for $R[i]$.*

PROOF. We prove only the first part. If $R[i]$ is super-extractable, then finite controllability of the chase for guarded dependencies [14, 7] yields that the axioms unioned with $R(\vec{c})$ imply $\mathsf{AccVal}(c_i)$, and thus in particular $\mathsf{AccVal}(c_i)$ holds in the chase structure. It is easy to see that this implies that c_i is an accessible value in the chase structure. On the other hand, suppose c_i is an accessible value in the chase structure; clearly $\mathsf{AccVal}(c_i)$ will then hold in the chase structure. If we have an arbitrary finite model I of the theory above (including $R(\vec{c})$), there is a homomorphism of the chase structure into I, and hence $\mathsf{AccVal}(c_i)$ holds in I, and $R[i]$ is super-extractable. □

Thus we have seen that it suffices to analyze the chase structure.

We now restrict further to constraints where both left- and right- hand sides are guarded; note that the "accessibility axioms" can be taken to be of this form. Let GS be the set of guarded facts over a collection of distinguished constants $d_1 \ldots d_n$, where n is the maximal arity of relations in S: here guarded means that the set includes a fact R containing all d_i. For $G_1, G_2 \in \mathsf{GS}$, let $G_1 \rightarrow_{\mathsf{CH}} G_2$ denote the fact that starting with a structure where the $\vec{d}$ satisfy G_1 and applying some sequence of chase steps, we get to a structure satisfying G_2.

Let $\mathsf{EV} = \{(G_1, G_2) \in \mathsf{GS} \times \mathsf{GS}, G_1 \rightarrow_{\mathsf{CH}} G_2\}$. Consider the set EV_0 containing all pairs of the forms: 1. $(G, G \cup \mathsf{AccVal}(d_j))$, where G contains $R(\vec{d}), \mathsf{AccVal}(d_{m_1}) \ldots \mathsf{AccVal}(d_{m_n})$ and there is an access on R with input positions $m_1 \ldots m_n$, 2. Pairs (G, G') with $G' \subseteq G$.

We claim that EV is formed from the set EV_0 by closing under the following rules:

- Suppose $G_1(\vec{d}) \in \mathsf{GS}$, and $G_1'(\vec{e})$ is the result of applying a chase step to G_1, where $\vec{e}$ contains all values appearing in generated facts that are either from $\vec{d}$ or created in the chase step; note that our requirement on the dependencies implies that the number of such values is at most n, the maximal arity of a relation in the schema. Suppose G_1'' results from applying to G_1' a renaming f of $\vec{e}$ to $d_1 \ldots d_n$, $(G_1'', G_2'') \in \mathsf{EV}$, and G_2 is the result of applying f^{-1} to G_2''. Then we can conclude that $(G_1, G_1 \cup G_2 \restriction \vec{d}) \in \mathsf{EV}$.
- If $(G_1, G_2) \in \mathsf{EV}$, $(G_2, G_3) \in \mathsf{EV}$, then $(G_1, G_3) \in \mathsf{EV}$.

Above $G_2 \restriction \vec{d}$ refers to the facts in G_2 which mention only values in $\vec{d}$.

To see this, consider a pushdown automaton whose stack contents are elements of GS, with the $\vec{d}$ annotated with a partial function to a copy $\vec{d'}$, indicating their correspondence with variables in the previous stack. The transitions of the PDA correspond to applications of the chase. We have a push rule that corresponds to applying a dependency to any guarded set of facts, producing new facts over the set of constants but with a correspondence to the old constants. In the special case of the rules corresponding to access patterns, we have a swap rule that takes any guarded set of facts and adds on the facts derived by the access methods, leaving the mapping the same. We have a pop-and-swap move that pops the top of the stack while moving down all new facts that hold on the shared variables between the top two stacks. It is easy to verify that $G_1 \rightarrow_{\mathsf{CH}} G_2$ iff G_1 with the empty mapping reaches G_2 with the empty mapping in the PDA described above. Indeed, more generally there is a correspondence between paths of chase rules and reachable stacks of the PDA, where the correspondence can be proven by induction.

Reachability in such a PDA can be tested using the classical saturation procedure, starting with the reachable pairs given by swap moves and then closing under transitivity and matches of pushes with with pop-and-swaps. This yields exactly the rules above.

Now the set GS can be doubly-exponential in general. However, in the case where the right-hand side of all constraints consists only of a guard predicate and unary predicates, we need not consider all guarded sets, but just ones having only a single guard predicate. Note that the sentences that involve the accessibility predicate are of this form. This modified GS has exponential size. Since the closure process above is monotone, we have the following:

THEOREM 3.6. *For a schema in which constraints consist only of inclusion dependencies, we can compute the set* EV *in* EXPTIME*, and hence can determine super-extractability in* EXPTIME.

With each pair $(R(\vec{x}) \wedge \bigwedge_{i \in I} \mathsf{AccVal}(x_i), R(\vec{x}) \wedge \bigwedge_{i \in J} \mathsf{AccVal}(x_i))$ we can associate a *relative extraction plan*. This is an executable plan, having two distinguished sets of free variables, inputs $x_i : i \in I$ and outputs $x_i : i \in J - I$. It has the property that on any structure, if we replace the input variables by all tuples of values in the projection of R onto $x_i : i \in I$, the corresponding conjunctive query (where the other free variables are existentially quantified) will return all values $x_i : i \in J - I$ that are in the projection of R onto $J - I$. Each closure rule for EV corresponds to a rule for concatenating such plans. E.g. for the second closure rule (transitivity), we simply concatenate the two relative extraction plans. In the closure process the size of this plan may double with each iteration of a rule, leading to a doubly-exponential bound on the size of an extraction plan.

The argument above gives a refinement of Theorem 3.4:

COROLLARY 3.7. *For a schema consisting of only inclusion dependencies alone, if an attribute can be completely extracted, then there is a static plan that witnesses this of at worst doubly-exponential size.*

Extensions to more general dependencies. We now consider linear-guarded dependencies. Recall that a LinGDep is a sentence of the form:

$$\forall \vec{x}\ R(\vec{x}) \rightarrow \exists \vec{y} \bigwedge_{j<m} T_j(\vec{x}, \vec{y}) \quad (1)$$

We now show that the results for IDs extend to this case:

THEOREM 3.8. *There is an* EXPTIME *algorithm that takes as input a schema with* LinGDep*'s and a position $R[j]$ and determines whether it is super-extractable. In addition, a super-extractable position has a static extraction plan.*

PROOF. We omit the full proof, noting only that the simpler case where no repeated variables can be reduced to the case of inclusion dependencies. Given a schema S consisting of LinGDep we create a schema S' with only ID in a larger vocabulary. For each dependency dep as in (1), we introduce a relation R_{dep} with no access methods and with arity equal to the number of variables in dep, and replace dep with ID:

$$\forall \vec{x}\ R(\vec{x}) \rightarrow \exists \vec{y} R_{\mathsf{dep}}(\vec{x}, \vec{y})$$

and for each $j < m$

$$\forall \vec{x}\ \vec{y}\ R_{\mathsf{dep}}(\vec{x}, \vec{y}) \rightarrow T_j(\vec{x}, \vec{y})$$

We claim that a position j of R is super-extractable in S iff it is super-extractable in S'.

The direction from right to left is clear, since every structure for S can be expanded into a structure satisfying S'. In the other direction, suppose that we have a structure I' for S' and a value v for $R[j]$ that is not super-extractable on I'. We take a superstructure I of I' by replacing R_{dep} by $\{\vec{x}, \vec{y} | \bigwedge_j T_j(\vec{x}, \vec{y})\}$. The schema S is now satisfied, and we have not added any accessible tuples. Thus the set of access paths is unaffected, and v is thus not superextractable on I, allowing us to conclude that $R[j]$ is not super-extractable in S.

Furthermore, any static extraction plan for S' must be an extraction plan for S, since the new predicates cannot occur. From this and Corollary 3.7 the theorem follows. □

For guarded dependencies, we get a similar result, but with a worse bound, using the same technique; details are in the full paper.

THEOREM 3.9. *There is a* 2EXPTIME *algorithm that takes as input a schema with* GTGD*'s and a position $R[j]$ and determines whether it is super-extractable. A super-extractable position has a static extraction plan.*

Exact Extraction. We have seen that if a relation is super-extractable, then we have a simple static plan that gets a superset of its values: a UCQ in the case of guarded constraints, and a CQ for inclusion dependencies. But what about getting the exact set of values?

We say that a position i in a relation R with arity n is *extractable* if it is super-extractable and furthermore for every structure I, for every value c the query

$$\exists x_1 \dots x_{i-1} x_{i+1} \dots x_n R(x_1 \dots x_{i-1}, c, x_{i+1} \dots x_n)$$

is access-determined. That is, we can determine exactly the values in that position using the access methods. Upper bounds on extractability will follow from later results on query answering:

THEOREM 3.10. *For any set of guarded constraints, extractability is decidable in* 2EXPTIME*. For inclusion dependencies it is decidable in* EXPTIME

The result will follow from Claim 4.15, proved later on, that gives a reduction from extractability to answer-determinacy, along with a 2EXPTIME bound on answer-determinacy for guarded constraints (proven in Theorem 4.1) and an EXPTIME bound on answer-determinacy for inclusion dependencies (proven in Proposition 4.13).

Lower bounds for (super-)extractability. Recall that for general guarded constraints, deciding super-extractability is 2EXPTIME-complete (Theorem 3.3). We now establish matching lower bounds for extractability and super-extractability for weaker dependencies. First note the following:

PROPOSITION 3.11. *For any constraint language containing inclusion dependencies, there is polynomial time reduction from super-extractability to extractability. Thus hardness results for super-extractability carry over immediately to extractability.*

PROOF. Given position i of R to be checked for super-extractability with respect to constraints S, we let S' be the extension of S with unary predicate P, with an inclusion dependency from $R[i]$ to $P[1]$ and a boolean access on P. It is easy to see that $R[i]$ is super-extractable iff $P[1]$ is extractable. □

We can thus get bounds on extractability via super-extractability. We now give a lower bound on both for inclusion dependencies:

THEOREM 3.12. *The problem of determining, given a schema* S *whose constraints consist only of inclusion dependencies and a position $R[i]$ whether or not $R[i]$ is super-extractable, is* EXPTIME*-hard. The same holds (via Proposition 3.11) for extractability.*

The result is a bit surprising, in that the lower bound for inclusion dependency implication is PSPACE. The proof is by a rather involved reduction taking an alternating PSPACE machine M and input word w, producing in polynomial time a schema with distinguished relation R and position t such that M accepts w exactly when $R[t]$ is super-extractable. It is given in the full version of the paper.

For guarded dependencies, we do not get a better bound then for general guarded constraints:

THEOREM 3.13. *The problem of determining, given a schema* S *whose constraints consist only of guarded constraints, and a position $R[i]$, whether or not $R[i]$ is super-extractable (resp. extractable), is* 2EXPTIME*-hard.*

The proof is by a reduction to hardness results for determining certain answers with guarded dependencies [15].

4. QUERY ANSWERING

We now study deciding whether a query is access-determined. As in the case of super-extractability, we start with the general case of guarded constraints, showing that the notion is decidable, and that when it happens we get a "static plan" for doing the answering. In this case, the static plan is a

rewriting. We then turn to more restricted classes, moving this time first to guarded dependencies, and then to linear-guarded dependencies.

Access-determined queries over Guarded constraints. We start our investigation of access-determinacy with a general result:

THEOREM 4.1. *For a schema* S *comprising guarded constraints and a conjunctive query* $\mathcal{Q}$ *it is decidable in* 2EXPTIME *whether* $\mathcal{Q}$ *is access-determined on* S *(and indeed, this problem is* 2EXPTIME*-complete).*

For simplicity, we will prove these and other results in the section in the restricted case where every relation has at least one access, and hence the accessible substructure is the substructure induced by the accessible values. We consider a vocabulary extending S with unary predicates $M_1, M_2, \mathsf{AccVal}$, along with a set C of $|\mathcal{Q}|$ many constants.

For a formula ϕ and unary predicate A, let ϕ_A be its restriction to A: subformulas of the form $\exists x \psi$ are replaced by $\exists x A(x) \wedge \psi$. Further let $h(\mathcal{Q})$ be the image of $\mathcal{Q}$ under a homomorphism h, where the variables are mapped onto constants in C.

For a homomorphism h consider the sentence T_h that is a conjunction of sentences asserting:

- the image of h is in M_2
- $\neg \mathcal{Q}_{M_1}$, where $\mathcal{Q}_{M_1}$ is the relativization of $\mathcal{Q}$ to M_1
- $\forall x \ (\mathsf{AccVal}(x) \rightarrow M_1(x) \wedge M_2(x))$
- $con(\mathsf{S})_{M_1} \wedge con(\mathsf{S})_{M_2}$, where these denote the relativization of the constraints to the predicates M_1 and M_2, respectively.
- $\bigwedge_R$ has an access with inputs $\vec{x}$ $\forall \vec{x}, \vec{y} \left(R(\vec{x}, \vec{y}) \wedge \bigwedge_i \mathsf{AccVal}(x_i) \ \rightarrow \ \bigwedge_j \mathsf{AccVal}(y_j) \right)$
- The analogous axioms to the one above for free accesses, and axioms stating that all constants are in AccVal.

We now relate these axioms to the notion of access-determinacy:

LEMMA 4.2. *Some* T_h *is satisfied by a finite model iff* $\mathcal{Q}$ *is not access-determined on* S *iff* T_{id} *has a finite model, where id is the identity homomorphism onto a set of constants of the same size as* $\mathcal{Q}$*.*

PROOF. We prove only the first equivalence, the second follows since if T_h has a finite model for some h, then T_{id} has a finite model.

Suppose T_h has a finite model, which must be of the form $M = (D, M_1, M_2, \mathsf{AccVal}, \ldots)$, where $\ldots$ indicates the relations of the original schema S. Let AV be the accessible values of the reduct of M to S. It is clear from the axioms that the set AccVal in M must contain AV, although it may strictly contain it. We claim that if we replace AccVal by AV, T_h is still true: this is because the predicate AccVal is not mentioned in the rules of items 1,2 and 4 above; it is only universally quantified in those of item 3, while those of items 5 and 6 hold of AV. Thus we can assume AccVal is exactly the accessible values AV. Now consider the structure M_1' for S formed by restricting all S predicates to M_1. $\mathcal{Q}$ does not hold on this structure, but after making all possible accesses, $\mathcal{Q}$ is still consistent – since there is a structure M_2' satisfying the constraints with the same accessible substructure as M_1' on which $\mathcal{Q}$ holds, namely the one formed by restricting all S predicates to M_2. Thus $\mathcal{Q}$ is not access-determined.

Conversely, suppose $\mathcal{Q}$ is not access-determined, and consider a witness to this, consisting of structures I_1 and I_2 sharing the same accessible substructure, where $\mathcal{Q}$ does not hold in I_1 but does hold in I_2. We create a structure for the vocabulary used in each T_h by letting: M_1 be the domain of I_1, AccVal be the common accessible values of I_1 and I_2, M_2 be the domain of I_2, and the S relations be as in $\mathsf{I}_1 \cup \mathsf{I}_2$. We let h be the homomorphism that witnesses that $\mathcal{Q}$ does hold in I_2, and interpret the constants in C accordingly. Then the resulting structure is a finite model of T_h. □

We return to the proof of Theorem 4.1. Write the sentence T_h as $\neg \mathcal{Q}_{M_1} \wedge G$, where G is the conjunction of all conjuncts of T_h except the second one (referring to items in the ordering given before).

Observe that G is in the guarded fragment. For the rules in item 1 and 3 this is clear. For item 4 it follows from the fact that the constraints are guarded. For the last two items it follows since the accessed atom R serves as a guard.

$\neg Q_{M_1}$ is the negation of a conjunctive query. Hence to check finite satisfiability of T_h we need to see if a GF sentence implies a conjunctive query over finite models (for short, we say that a theory *finitely implies* a query when this happens).

Thus, given the lemma, Theorem 4.1 follows from the following variant of the main result of Bárány, Gottlob, and Otto [7]:

THEOREM 4.3. *One can decide in* 2EXPTIME *whether a* GF *sentence finitely implies a conjunctive query* $\mathcal{Q}$*. Furthermore, this holds iff the sentence implies* $\mathcal{Q}$*.*

This result is stated in [7], with constants excluded. The extension to constants can be seen by examination of the proof, or by applying the more general result of [5] discussed further on, and removing constants by replacing them with existentially quantified variables (staying in the 2EXPTIME-decidable language of [5]).

The corresponding hardness follows by noting that a query over inaccessible relations is access-determined exactly when the constraints are unsatisfiable, which is known to be 2EXPTIME-hard [10].

As with extractability, our axiomatization implies the existence of a bound on the number of accesses needed:

THEOREM 4.4. *For any schema based on guarded constraints and conjunctive query* $\mathcal{Q}$*, if* $\mathcal{Q}$ *is access-determined then it is* k*-access-determined for some* k*.*

PROOF. If $\mathcal{Q}$ is not k-access-determined for any k, then by the Compactness Theorem of classical model theory [8] we see that there is a (possibly infinite) model M for the vocabulary extending the schema S with unary predicates M_1, M_2 where the submodel generated by each M_i satisfies $con(\mathsf{S})$, M_1 does not satisfy $\mathcal{Q}$ but M_2 does, and the accessible substructures of M_1 and M_2 are the same. Extending by the predicate AccVal, we get a model of T_h for some homomorphism h, and hence applying the finite controllability result of [7] we get a finite model. Now Lemma 4.2 implies that $\mathcal{Q}$ is not access-determined. □

Finally, we relate this to rewritability.

THEOREM 4.5. *For any schema* S *having only guarded constraints and UCQ* $\mathcal{Q}$ *access-determined on* S*, there is an integer* k *such that* $\mathcal{Q}$ *is FO,*k*-rewritable over* S*.*

This result has the following consequence for the "data complexity" of access-determined queries:

COROLLARY 4.6. *For any fixed UCQ $\mathcal{Q}$ that is access-determined on* S *there is a polynomial time algorithm that finds the answer to $\mathcal{Q}$ on any structure for* S.

In contrast, note that if $\mathcal{Q}$ is not access-determined, the data complexity of finding the certain answers to $\mathcal{Q}$ can be CoNP-hard, even for guarded constraints [7, Theorem 19(5)].

In proving Theorem 4.5 we make use of the *guarded negation fragment of first-order logic*, denoted GNF, which we will discuss further below. GNF is built up from atomic formulas (including equality) via the rules: $\phi_1, \phi_2 \in \mathsf{GNF} \Rightarrow \phi_1 \wedge \phi_2, \phi_1 \vee \phi_2 \in \mathsf{GNF}$; $\phi \in \mathsf{GNF} \Rightarrow \exists x_i\ \phi \in \mathsf{GNF}$; and $\phi(\vec{x}) \in \mathsf{GNF} \Rightarrow R(\vec{x}) \wedge \neg\phi \in \mathsf{GNF}$, where the atom $R(\vec{x})$, which may be a predicate or an equality, must contain all free variables of ϕ. For a relational signature σ, we write $\mathsf{GNF}(\sigma)$ for the set of formulas in the guarded negation fragment that make use of only predicates in σ. The importance of GNF for us is that, as far as sentences are considered, it contains GF, contains all conjunctive queries, is closed under boolean combinations of sentences as well as guarded quantification, and that GNF has the finite model property. See Bárány, ten Cate, and Segoufin [5] for more information about GNF.

PROOF OF THEOREM 4.5. The proof uses exactly the line of argument in Marx's [6]. By Theorem 4.4 we know that for some k, $\mathcal{Q}$ is determined by the k-accessible substructure over finite models, relative to the constraints $con(\mathsf{S})$. Note that the k-accessible substructure can be described as a set of acyclic conjunctive queries, which implies that it can be described by formulas in the guarded fragment. Also note that the free variables in the defining formulas are guarded. Hence $\mathcal{Q} \vee \neg con(\mathsf{S})$ is a query in the guarded negation fragment that is determined over finite structures by a set of guarded negation views with free variables guarded, namely: $\{con(\mathsf{S})\} \cup \{k - acc(R) : R \in \mathsf{S}\}$, where $k-acc(R)$ is the restriction of relation R to the k-accessible substructure. Since all queries involved are in the guarded negation fragment, and all free variables are guarded, the statement that they are determined can be expressed as a sentence in the guarded negation fragment over a signature with two copies of the view predicates:
$\big(\forall\vec{x} \bigwedge_i \phi_i(\vec{x}) \leftrightarrow \phi_i'(\vec{x})\big) \rightarrow \big(\forall\vec{x}\rho(\vec{x}) \leftrightarrow \rho'(\vec{x})\big)$ where ϕ_i are the view definitions and ρ is $\mathcal{Q} \vee \neg con(\mathsf{S})$.

We can then apply the finite model property for GNF to infer that $\mathcal{Q} \vee \neg con(\mathsf{S})$ is determined by the views over all models. Thus by Theorem 3.1 of Segoufin and Vianu's paper [16], $\mathcal{Q} \vee \neg con(\mathsf{S})$ is rewritable in first-order logic using these views over all models, and hence $\mathcal{Q}$ is rewritable as a first-order query using $\{k - acc(R) : R \in \mathsf{S}\}$. In addition, the rewritten query is domain-independent: access-determinacy implies that changing the structures outside of the active-domains of $\{k - acc(R) : R \in \mathsf{S}\}$ does not impact the result of $\mathcal{Q} \vee \neg con(\mathsf{S})$. A domain-independent query over a schema can be rewritten as a query using only quantification restricted to the active domain of structures in the schema (see, e.g. Lemma 5.3.8 in [17]), which in this case is exactly the k-accessible substructure. □

Theorem 4.5 shows that we have a rewriting, but how complex can it be? Clearly, the rewriting can be a UCQ (e.g. if everything is accessible). It can also be an arbitrary guarded formula: the constraints may say that some inaccessible predicate $R(x)$ is defined by a guarded formula $\phi(x)$ over accessible predicates, in which case the query $\exists x R(x)$ is access-rewritable as $\exists x\phi(x)$. It is easy to contrive examples that mix and match the two cases above, requiring a rewriting that is a combination of a conjunctive query and guarded formulas. It is thus natural to conjecture that whenever we have an access-determined query over a schema with guarded constraints, the rewriting can be found as a GNF formula. It turns out this is indeed the case.

THEOREM 4.7. *For every schema* S *having only guarded constraints and every* UCQ $\mathcal{Q}$ *that is access-determined on* S*, there is a natural k such that $\mathcal{Q}$ is* GNF*,k-rewritable over* S.

The result follows from a recently-proved result that GNF has the Craig Interpolation property. For convenience, we will state the property only for sentences below, although it normally talks about open formulas.

THEOREM 4.8. *[18] If $\phi \rightarrow \phi'$ is a validity, with ϕ a* GNF*-sentence over signature τ and ϕ' a* GNF*-sentence over signature σ, then there is a* GNF*-sentence ρ such that all predicates in the signature are in $\sigma \cap \tau$ and all constants are in $\sigma \cup \tau$ such that both $\phi \rightarrow \rho$ and $\rho \rightarrow \phi'$ are valid.*

Given the above theorem, the proof follows along the same lines as Theorem 4.5. It is well-known (and is easy to show) that Craig Interpolation implies the *Projective Beth Definability property* for the logic (see [6]). From this we can in turn show that if a boolean query is determined by a set of views in the logic, then the query is rewritable in the logic. Applying this as in the proof of Theorem 4.5 we see that $\mathcal{Q} \vee \neg con(\mathsf{S})$ is rewritable as a GNF-sentence using predicates $\{k - acc(R) : R \in \mathsf{S}\}$.

As in the proof of Theorem 4.5 the rewriting produced by the argument above quantifies over the whole structure (including the inaccessible values), rather than only the accessible ones. However, access-determinacy implies that the query is "domain-independent" (depending only on the extensions of the accessible predicates). By results of Bárány, ten Cate, and Otto [19], it can be converted into "Guarded-Negation Relational Algebra", and from there into an active domain formula in the Guarded Negation Fragment.

We note an important consequence of Theorem 4.7: for access-determined queries, *the rewriting can be found effectively.* The algorithm is a brute-force check of every GNF sentence: we can check whether each is a rewriting, using the decidability of GNF, and (since the query is access-determined), eventually the check will return true. Note also that this argument can be applied not just to access-determinacy of UCQs (nowhere did we use that the query was positive) but to access-determinacy of GNF queries; it can also be applied to constraints given in GNF rather than the guarded fragment.

Answerability for Dependencies. We now examine the relationship of access-determinacy to rewritability in restricted cases. We begin with the case of guarded dependencies. Recall that these are of the form:

$$\forall\vec{x}\ R(\vec{x}) \wedge \phi(\vec{x}) \rightarrow \exists\vec{y}\phi'(\vec{x},\vec{y})$$

where R is an atom and ϕ, ϕ' are conjunctions of atoms.

We let $\mathsf{Chase}(\mathsf{Sch}, \mathcal{Q})$ be the structure formed by starting with the canonical structure of $\mathcal{Q}$ (that is, the structure whose domain elements are the variables, with the structure given by the query [17]) as the initial structure, and then applying the chase construction using the constraints of schema Sch.

Note that the resulting structure can be expanded to the vocabulary for T_h, by letting M_2 be the entire structure, AccVal the accessible elements, and M_1 the chase closure of AccVal, and we will sometimes abuse notation by viewing it that way subsequently. We can now relate the theory T_h presented before to this structure. The following is a variant of the standard universality property of the chase:

LEMMA 4.9. *Given a homomorphism h from $\mathcal{Q}$ to a set of constants C, and a structure M containing C that satisfies T_h, there is a homomorphism from* $\mathsf{Chase}(\mathsf{Sch}, \mathcal{Q})$ *onto a substructure M' of M satisfying T_h.*

We can now show that checking whether a query is access-determined can also be done by looking at structures formed via the chase.

PROPOSITION 4.10. *For schemas where all constraints are guarded dependencies, $\mathcal{Q}$ is access-determined iff the universal structure* $\mathsf{Chase}(\mathsf{Sch}, \mathcal{Q})$ *satisfies the relativization of $\mathcal{Q}$ to the chase closure of* AccVal.

PROOF. In one direction, suppose $\mathsf{Chase}(\mathsf{Sch}, \mathcal{Q})$ has the property above. If $\mathcal{Q}$ is not access-determined, onsider a finite structure I for schema Sch where $\mathcal{Q}$ holds, and another structure I′ sharing the same accessible facts with I where $\mathcal{Q}$ does not hold. We can convert I and I′ into a finite model M of T_h in the obvious way. By Lemma 4.9 we have a homomorphism from $M' = \mathsf{Chase}(\mathsf{Sch}, \mathcal{Q})$ onto a submodel of M. Further, we can easily see that since the relativization of $\mathcal{Q}$ to the chase closure of AccVal is satisfied in M', its homomorphic image will witness satisfaction of $\mathcal{Q}$ in I′, contradicting our assumption.

In the other direction, if $\mathcal{Q}$ is access-determined, then by Lemma 4.2 T_{id} does not have a finite model. Hence by Theorem 4.3 T_{id} does not have any model, and thus no expansion of $M' = \mathsf{Chase}(\mathsf{Sch}, \mathcal{Q})$ is a model of T_{id}. But if there were no witness to $\mathcal{Q}$ in the chase closure of AccVal within M', consider the "obvious interpretation" of the chase closure as a structure for the vocabulary of T_{id} – that is, expanding to let M_1' the chase closure of the image of $\mathcal{Q}$ under id, and M_2' is the chase closure of AccVal. This structure would satisfy T_{id}, a contradiction. □

Consequences. We first refine Proposition 4.10 to get a bound on the size of the chase closures that we need to consider.

Given any structure I of a schema with linear-guarded dependencies, and a set $S \subset \mathsf{I}$, a k-step chase closure of S is any subset S_k of I that can be formed from $S_0 = S$ the following inductive process: let S_{i+1} is formed from S_i by adding witnesses $\vec{y}$ for every dependency, $\forall \vec{x} \phi \rightarrow \exists \vec{y} \psi$ where $\vec{x}$ is instantiated for every tuple in S_i satisfying ϕ.

LEMMA 4.11. *Given a schema* S *whose constraints consist of guarded dependencies, a conjunctive query $\mathcal{Q}$ is access-determined iff for every structure* I *of the schema, there is a witness to $\mathcal{Q}$ in the k-step chase-closure of the k-accessible substructure of* I*, where k is doubly exponential in the size of $\mathcal{Q}$ and* S.

PROOF. Note that we can consider $\mathsf{Chase}(\mathsf{Sch}, \mathcal{Q})$ as an ω-tree where the nodes have labels that correspond to facts in the vocabulary of Sch unioned with AccVal, where the free variables are bounded by the number of variables in the right-hand side of any rule. When a new fact is generated in a chase step, it is added as a child of the node containing the guard fact. This is then an ω-regular tree, and the rules in the schema along with the axioms for AccVal can be converted into the rules of a top-down non-deterministic Büchi tree automaton A_1 that accepts the tree representation of $\mathsf{Chase}(\mathsf{Sch}, \mathcal{Q})$; at every node, a guess is made of all the facts that will hold of the variables in that node, with the state also tracking facts that have already been verified in an ancestor of the node. In each transition, a guess is made as to how the remaining facts to be verified are passed on to the children, and if the children do not include the necessary variables, the run fails. Note that this may require at most a doubly-exponential number of states, corresponding to every guarded collection.

By Lemma 4.10 $\mathcal{Q}$ is not access-determined iff A_1 accepts a tree which fails to contain a witness to $\mathcal{Q}$ consisting of accessible elements. One can create an exponential-sized non-deterministic automaton A_2 that accepts exactly trees that contain such a witness to $\mathcal{Q}$; the complement of A_2 can be obtained as doubly-exponential size non-deterministic automata. Thus $\mathcal{Q}$ is not access-determined iff $A_1 \cap A_2^c$ accepts a tree. This is true exactly when A_2^c does not accept on the unique tree accepted by A_1 truncated to depth the number of states of $A_1 \cap A_2^c$.

This shows that in the canonical instance, it suffices to look for a witness to $\mathcal{Q}$ in the chase closure of AccVal within a doubly-exponential truncation of $\mathsf{Chase}(\mathsf{S}, \mathcal{Q})$. The result now follows using the universality of the chase and of the canonical instance. □

We can get a refinement of Theorem 4.5, guaranteeing that queries can be rewritten in a special form.

THEOREM 4.12. *Given a schema* S *whose constraints consist of guarded dependencies, a conjunctive query $\mathcal{Q}$ that is access-determined is $\exists^+\mathsf{FO}$, k-rewritable for some k.*

PROOF. By Lemma 4.11 we know that there is some k such that $\mathcal{Q}$ is true iff the query holds in the k-step chase closure of the k-accessible substructure. We now use the following fact:

Given a set of TGDs, a number k, and a conjunctive query $\mathcal{Q}$, there is a UCQ $\mathcal{Q}'$ such that for every instance I, $\mathcal{Q}$ holds in the k-step closure of I iff $\mathcal{Q}'$ holds in I itself.

This result is well-known (see e.g. [11]); by induction, it suffices to prove it for $k = 1$, and for $k = 1$ it is proven by unioning the queries formed by 1. matching subqueries $\mathcal{Q}'(\vec{x}, \vec{y})$ with the heads $\phi(\vec{x}, \vec{y})$ of TGDs $\rho(\vec{x}) \rightarrow \exists \vec{y} \phi(\vec{x}, \vec{y})$ where the variables $\vec{y}$ cannot appear in $\mathcal{Q}$ outside of $\mathcal{Q}'$ 2. replacing $\mathcal{Q}'$ in $\mathcal{Q}$ with $\rho(\vec{x})$ □

We can use the chase-based approach to give a more precise bound on verifying that a query is access-determined.

PROPOSITION 4.13. *In the case of schemas having only linear-guarded dependencies, we can determine if $\mathcal{Q}$ is access-determined in* EXPTIME.

PROOF. (sketch) Notice that in these cases we know that $\mathcal{Q}$ is access-determined iff the chase structure contains a homomorphic image of $\mathcal{Q}$ in the chase closure of its accessible part. An exponential-sized PDA can explore the chase closure, guessing a homomorphism from the query to elements within guarded sets on the stack, and recording in the control

state atoms that have already occurred in the pre-image of the homomorphism. Since reachability in a PDA is PTIME, we have an EXPTIME upper bound. □

Lower bounds. As with extractability, we have matching lower-bounds for checking if a query is access-determined.

PROPOSITION 4.14. *Even for schemas having only inclusion dependencies, the problem of checking if $\mathcal{Q}$ is access-determined is* EXPTIME*-hard.*

PROOF. We make use of the following claim, which is of independent interest:

CLAIM 4.15. *There is a polynomial time reduction from extractability to access-determinacy, for each constraint class (inclusion dependencies, guarded constraints, ...).*

The proposition follows immediately from the claim, since extractability was shown to be EXPTIME-hard in Theorem 3.12.

Given a position $R[i]$ in a schema S with inclusion dependencies, our reduction generates the query $\mathcal{Q} = \exists \vec{x} P(x_i) \wedge R(\vec{x})$, over a schema S' obtained from adding unary predicate P to S, where P has only a "boolean access" – one requiring the value of the free variable as input.

We check that this is a reduction. It is easy to see that if $R[i]$ is extractable, $\mathcal{Q}$ is access-determined – indeed, it has an obvious query plan. On the other hand, if $R[i]$ is not extractable, we have a value v and two instances I and I' with the same accessible substructure, where $v \in R[i]$ in I but $v \notin R[i]$ in I'. By expanding I and I' to have $P(v)$ true, we get that $\mathcal{Q}$ is not access-determined. This completes the proof of the claim and (hence) the proposition. □

Comparison with non-answerable queries. We compare the results we have obtained for access-determined query to the situation in the case of a non-access-determined query $\mathcal{Q}$.

For a logic L, say that a conjunctive query $\mathcal{Q}$ is L *access certain-answer rewritable* over schema S if there is $\mathcal{Q}'$ such that: for any structure I satisfying the schema constraints, $\mathcal{Q}'$ returns true on the accessible substructure of I iff $\mathcal{Q}$ is certain after the path returning the accessible substructure. More formally, $\mathsf{AccessiblePart}(\mathsf{I}) \models \mathcal{Q}'$ iff for all $\mathsf{I}' \models \mathit{con}(\mathsf{Sch})$ with $\mathsf{AccessiblePart}(\mathsf{I}) = \mathsf{AccessiblePart}(\mathsf{I}')$ it holds that $\mathsf{I}' \models \mathcal{Q}$.

We say $\mathcal{Q}$ is L,k *access certain-answer rewritable* iff the above is true but with the accessible substructure replaced by the k-accessible substructure.

If $\mathcal{Q}$ is access-determined, then Theorem 4.5 shows that it is FO, k access certain-answer rewritable.

We show that we may not have FO, k access certain-answer rewritability for any k, even in the absence of constraints:

PROPOSITION 4.16. *There are conjunctive queries that are not FO, k access certain-answer rewritable for any k, even in the absence of constraints.*

PROOF. We simply take a schema with no integrity constraints, containing a binary relation R which has an access method on the first position along with a unary relation U that is completely accessible. Thus the accessible substructure of a structure I is the restriction to the transitive closure under R of the interpretation of U. The query $\exists x R(x,x)$ is certain iff the accessible substructure contains a tuple of the form $R(x,x)$: clearly this requires us to look at the full accessible substructure. □

It is well-known that conjunctive queries are not FO-rewritable over general guarded constraints. The following shows that they are not even FO access certain-answer rewritable:

PROPOSITION 4.17. *There are conjunctive queries that are not* FO *access certain-answer rewritable over guarded constraints.*

PROOF. We use the fact that query answering under guarded tgds (hence under guarded constraints) is not FO-rewritable in the usual sense of open world query answering [11]. Let $\mathcal{Q}'$ and S be a query and schema such that $\mathcal{Q}$ is not FO-rewritable w.r.t. the constraints in S in the usual open-world query-answering sense. Consider S' with an additional copy R' of every predicate R in S. All predicates R' are fully accessible, while all the original predicates R are completely inaccessible. The constraints in the new schema include all the constraints in S, and in addition inclusion dependencies stating that R' is included in R. Thus the accessible substructure of a structure I is simply an arbitrary substructure of a structure satisfying the schema constraints. It is easy to see that our notion of access certain-answer rewritable degenerates to the standard notion of rewritability in this setting, and hence $\mathcal{Q}$ is not FO access certain-answer rewritable. □

Rewritability and Executability. Recall that Theorem 4.5 showed that for schemas with guarded constraints, any access-determined query is FO, k-rewritable and that Theorem 4.12 shows that for linear-guarded constraints each access-determined query is $\exists^+\mathsf{FO}, k$ rewritable.

We will now show that the latter queries have executable rewritings in the sense of [2]. Recall that an *executable* CQ is one where the order of the atoms is consistent with the access methods (variables occurring in input positions always occur at an earlier output position), and a UCQ is executable if it is a union of executable CQs. The following result is easy to show using the definitions:

PROPOSITION 4.18. *A query $\mathcal{Q}$ is equivalent to an executable* UCQ *iff it is $\exists^+\mathsf{FO}, k$ rewritable for some k.*

5. RELATED WORK

Our work has a close connection to two lines of research. The most obvious connection is to the study of answering queries with limited access constraints. This notion has been studied from the theoretical point of view in the 90's [20, 21]. There were also systems created which optimize queries with respect to access restrictions, e.g. Florescu et. al [22]. Chang and Li [1, 9] began the study of which queries were answerable, with particular emphasis on the application to integration from web-based data sources. The study of answerability was continued by Nash, working with Ludäscher, and later with Deutsch [3, 4, 2].

Our work is directly inspired by Deutsch, Ludäscher, and Nash's [2], where the interaction of integrity constraints and access patterns is considered. The main contribution of [2] is an algorithm for determining whether a query is "feasible" – rewritable as an executable query – the algorithm applies to tuple-generating dependencies with negation, and requires the chase procedure to terminate; thus whether the algorithm terminates is undecidable in general. Query-independent properties are not considered in [2]. Our work on query-answering differs from [2] with regard to the class

of constraints considered and to the notion of "answerability" considered. In terms of constraints consider guarded constraints, which can have arbitrary quantifier alternation, and thus are not contained in dependencies of any sort. On the other hand, [2] deals with dependencies that are not guarded. Although the TGDs of [2] subsume inclusion dependencies, their algorithm does not terminate on cyclic inclusion dependencies. Our results give the first complete analysis for querying answering in the presence of access constraints and IDs. While [2] define answerability in terms of an executable rewriting, we consider semantic notions of answerability and more general rewritings, which can involve quantifier alternation. We show that our more general rewritings are necessary for guarded constraints, and we related the semantic notion of answerability to the syntactic one. Even for the more restricted notion of rewritability of [2], our results yield new algorithms for detecting rewritability. We make use of the "repeated chase technique" introduced by [2], coupling it with model-theoretical methods.

The second line of work that we build on concerns the relation between semantic implication and rewritability for views and queries. Nash, Segoufin, and Vianu initiated the study of the relationship between the notions "Views V determine query Q" and "Q is rewritable over V in language L" [16]. Among other contributions, [16] gives conditions on Q and L that imply that determined queries are rewritable, and present cases where no such rewriting exists. Afrati [23], and Pasailă [24] study the question for restricted classes.

Particularly relevant to this paper is Marx's [6], which shows that determinacy is decidable for views and queries in the "Packed Fragment" (which subsumes guarded logic), and argues that for determined queries, the rewriting can be taken to be in the Packed Fragment as well. The proof of the latter result in [6] is flawed, as discussed in [18]. Indeed, ten Cate and Marx have shown that determinacy-implies-rewriting fails for the fragments considered in [6]. Nevertheless, our result uses similar techniques and a proof development inspired by Marx's work in [6].

Acknowledgement Benedikt and Bourhis are supported by the Engineering and Physical Sciences Research Council project "Query-Driven Data Acquisition from Web-based Datasources" (EPSRC EP/H017690/1). We are very grateful to Balder ten Cate for many helpful comments and corrections on the draft.

6. REFERENCES

[1] C. Li, "Computing complete answers to queries in the presence of limited access patterns," *VLDB Journal*, vol. 12, no. 3, pp. 211–227, 2003.

[2] A. Deutsch, B. Ludäscher, and A. Nash, "Rewriting queries using views with access patterns under integrity constraints," in *ICDT*, 2005.

[3] A. Nash and B. Ludäscher, "Processing union of conjunctive queries with negation under limited access patterns," in *EDBT*, 2004.

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

[5] V. Bárány, B. ten Cate, and L. Segoufin, "Guarded negation," in *ICALP*, 2011.

[6] M. Marx, "Queries determined by views: pack your views," in *PODS*, 2007.

[7] V. Bárány, G. Gottlob, and M. Otto, "Querying the guarded fragment," in *LICS*, 2010.

[8] C. Chang and H. J. Keisler, *Model Theory*. Elsevier, 1990.

[9] C. Li and E. Chang, "Answering queries with useful bindings," *TODS*, vol. 26, no. 3, pp. 313–343, 2001.

[10] E. Grädel, "On the restraining power of guards," *J. Symb. Logic*, vol. 64, pp. 1719–1742, 1999.

[11] A. Calì, G. Gottlob, and T. Lukasiewicz, "A general datalog-based framework for tractable query answering over ontologies," in *PODS*, 2009.

[12] D. S. Johnson and A. C. Klug, "Testing Containment of Conjunctive Queries under Functional and Inclusion Dependencies," *JCSS*, vol. 28, no. 1, pp. 167–189, 1984.

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

[14] R. Rosati, "On the decidability and finite controllability of query processing in databases with incomplete information," in *PODS*, 2006.

[15] A. Calì, G. Gottlob, and M. Kifer, "Taming the infinite chase: Query answering under expressive relational constraints," 2012. full version of KR 2008 paper, to appear.

[16] A. Nash, L. Segoufin, and V. Vianu, "Views and queries: Determinacy and rewriting," *ACM Trans. Database Syst.*, vol. 35, no. 3, 2010.

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

[18] V. Bárány, M. Benedikt, and B. ten Cate, "Rewriting guarded negation queries," 2012. Submitted, available from the authors.

[19] V. Bárány, B. ten Cate, and M. Otto, "Queries with guarded negation," in *VLDB*, vol. 5, pp. 1328–1339, 2012.

[20] O. M. Duschka and A. Y. Levy, "Recursive plans for information gathering," in *IJCAI*, 1997.

[21] A. Rajaraman, Y. Sagiv, and J. D. Ullman, "Answering queries using templates with binding patterns," in *PODS*, 1995.

[22] D. Florescu, A. Y. Levy, I. Manolescu, and D. Suciu, "Query optimization in the presence of limited access patterns," in *SIGMOD*, 1999.

[23] F. N. Afrati, "Determinacy and query rewriting for conjunctive queries and views," *Theor. Comput. Sci.*, vol. 412, no. 11, pp. 1005–1021, 2011.

[24] D. Pasailă, "Conjunctive queries determinacy and rewriting," in *ICDT*, 2011.