Generating Low-cost Plans From Proofs [*]

Michael Benedikt
Oxford University

Balder ten Cate
LogicBlox Inc. & UC Santa Cruz

Efthymia Tsamoura
Oxford University

ABSTRACT

We look at generating plans that answer queries over restricted interfaces, making use of information about source integrity constraints, access restrictions, and access costs. Our method can exploit the integrity constraints to find low-cost access plans even when there is no direct access to relations appearing in the query. The key idea of our method is to move from a search for a plan to a search for a proof that a query is answerable, and then *generate a plan from a proof*. Discovery of one proof allows us to find a single plan that answers the query; exploration of several alternative proofs allows us to find low-cost plans. We start by mapping out a correspondence between proofs and restricted-interface plans in the context of arbitrary first-order constraints, based on interpolation. The correspondence clarifies the connection between preservation and interpolation theorems in predicate logic and reformulation problems, and generalizes it in several dimensions. We then provide algorithms for schemas based on tuple-generating dependencies. These algorithms perform interpolation, but generate plans directly. Finally, we show how the direct plan-generation approach can be adapted to take into account the cost of plans.

1. INTRODUCTION

This work concerns answering queries in the presence of integrity constraints, where the datasources may vary in terms of their access restrictions and access costs. Examples of restricted interfaces include web forms, web services, and legacy databases. Examples of access cost include the monetary cost of accessing certain services, and the cost in time of accessing data through web service calls, by iteratively filling out web forms, or using particular indices. Our goal is getting plans which (a) give *complete answers to queries* while (b) *minimizing the access cost*. The first condition means that, for a query asking for the office number of all Professors with last name "Smith", we want a plan that returns all tuples in the answer, even if access to the Professor relation is limited. The second condition means that our goal is not merely to get a complete answer, but also to take into account the cost of making accesses to the sources, which we assume will dominate the cost of local query processing. An obvious question arises: if access to a source is restricted, how can one hope to get complete answers, or any answers at all? *Relationships between sources* can help us find plans that can answer a query.

[*]Benedikt and Tsamoura are supported by EPSRC grant EP/G004021/1. Ten Cate is supported by NSF grant IIS-1217869.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
PODS'14, June 22–27, 2014, Snowbird, UT, USA.
Copyright 2014 ACM 978-1-4503-2375-8/14/06 ...$15.00.
http://dx.doi.org/10.1145/2594538.2594550.

EXAMPLE 1. *Consider a* Profinfo *table containing information about faculty, including their last names, office number, and employeeid, but with a restricted interface that requires giving an employeeid as an input. The query Q asking for ids of faculty named smith cannot be completely answered over this schema.*

If another source has a Udirect *table containing the employeeid and last name of every university employee, with unrestricted access, then Q* is *completely answerable: one plan pulls all tuples from the* Udirect *table and check them with the* Profinfo *table.*

The above example shows that complete answers may or may not be obtainable, and that information on source overlap, which we can formalize using *referential integrity constraints*, can play an important role in determining whether and how queries can be completely answered. The simple example above involves two kinds of inferences to see that the query is answerable: reasoning about the logical relations between relations (referential constraints, mappings, view definitions), and also reasoning concerning what kinds of access we have to the data. The latter must account for the fact that some relations, such as materialized views, are freely accessible, while others (base relations hidden behind views, virtual relations within a data integration schema) may not be directly accessible at all. Or as above, it may capture finer notions of access – e.g. that a relation can be accessed only via certain indexes. In Example 1, the reasoning was straightforward, but in the presence of more complex schemas we may have to chain several inferences:

EXAMPLE 2. *We consider two telephone directory datasources with overlapping information. One source exposes information from* $\mathsf{Direct1}(\mathsf{uname},\mathsf{addr},\mathsf{uid})$ *via an access requiring a* uname *and* uid. *There is also a table* $\mathsf{Ids}(\mathsf{uid})$ *with no access restriction, that makes available the set of* uid*s (hence a referential constraint from* Direct1 *into* Ids *on* uid*). The other source exposes* $\mathsf{Direct2}(\mathsf{uname},\mathsf{addr},\mathsf{phone})$*, requiring a* uname *and* addr*, and also a table* $\mathsf{Names}(\mathsf{uname})$ *with no access restriction that reveals all* uname*s in* Direct2 *(that is, a referential constraint from* Direct2 *to* Names*). There is also a referential constraint from* Direct1 *to* Direct2 *on* uname *and* addr*. Consider a query asking for all phone numbers in the second directory:*

$Q = \{\mathsf{phone} \mid \exists\ \mathsf{uname}\ \mathsf{addr}\ \mathsf{Direct2}(\mathsf{uname},\mathsf{addr},\mathsf{phone})\}.$

There is a plan that answers this query: it gets all the uid*s from* Ids *and* uname*s from* Names *first, puts them into the access on* Direct1*, then uses the* uname *and* addr *of the resulting tuples to get the phone numbers in* Direct2.

We will begin with the case of arbitrary first-order logic constraints. We show that existence of a plan for a query is equivalent to an entailment between formulas, with the entailment holding relative to a set of rules that encode both integrity constraints and access/availability restrictions. We show that by "tweaking" the axioms, we can characterize the existence of a plan that uses only positive relational algebra operators, and a plan that uses positive relational algebra operators and the difference operator restricted to atomic relations. Our equivalence theorems are closely-related to interpolation and preservation theorems in first-order logic. On the one hand, we generalize results of Nash, Segoufin, and Vianu [15] to show that several of the main preservation theorems in first-order model theory correspond to results that characterize which queries can be reformulated over restricted interfaces. Futhermore, we show that these results have effective content, allowing us to compute a reformulation from a proof of a certain entailment. Finally, our results on first-order constraints show that the connection between interpolation and reformulation applies not only to the setting of view definitions, but to the more refined notion of restricted interface given by access patterns, and in the presence of arbitrary first-order integrity constraints. The latter extension is obtained via a new interpolation theorem, which is of independent interest as it generalizes existing interpolation theorems in the literature and provides the first constructive proof of a prior theorem.

We go on to show that for a wide class of constraints used in databases, tuple-generating dependencies, plan-generation from a proof can be done via a particularly simple algorithm, which produces plans that are efficient in terms of number of accesses made. We provide algorithms for relational algebra plans, along with an algorithm geared towards plans that use conjunctive queries. This latter algorithm generalizes the decidability of conjunctive reformulation over views (a seminal result of Levy et al. [**?**]) and is the basis for the work on low-cost plans in the remainder of the paper.

Our initial plan-generation algorithm would generate the "obvious plans" for either of the examples above, thus satisfying requirement (a) in the first paragraph above. But what about (b)? In the setting of overlapping datasources, there can be many plans with very distinct costs. Consider a variant of Example 1 in which there are two tables $\mathsf{Udirect}_1$ and $\mathsf{Udirect}_2$ that contain the necessary information. In this case we would have at least three plans: one that first accesses Udirect1 as above and then checks the results in Profinfo, another that first accesses $\mathsf{Udirect}_2$, and a third that accesses both $\mathsf{Udirect}_1$ and $\mathsf{Udirect}_2$ and intersects the results in middleware before doing the check in Profinfo. Which of these is best will depend on how costly access is to each of the directory tables, and what percentage of the tuples in the two directory tables match a result in Profinfo. Notice that these plans are not variants of one another, and one cannot be obtained from the other by applying algebraic transformations.

A salient fact about our algorithms is that they can directly produce physical plans whose structure mirrors the structure of proofs. We can thus search for a good physical plan while searching through the space of proofs, rather than having separate reformulation and optimization phases. We discuss how to use this approach to find the lowest-cost plan, thus addressing requirement (b), in the setting where we may have complex constraints, and a "generic" cost function on access plans. The main idea is to *explore the full space of proofs, but guiding the search by cost as well as proof structure.* The notion of searching simultaneously in "proof space" and "plan space" is a key contribution – we believe that it is applicable in a wide variety of query reformulation settings, unifying cases with very distinct integrity constraints and optimisation requirements.

Related Work. The goal here of plans that get *complete answers* to queries contrasts with much work in data integration and knowledge representation, which deals with the more general problem of getting the maximal number of answers possible (the *certain answers*) or the best plan in a certain language (the *maximal contained rewriting*). For example, in the setting of the semantic web it is generally assumed that sources are inherently incomplete, and thus one cannot hope to get the complete answers. Although we believe our techniques can apply to this broader problem, by restricting to the "completely answerable" setting we circumvent many complications: for example, the resulting plans will never require recursion [3]. This restriction will allow us to focus on the basic ideas in our approach to query reformulation via proof exploration.

We are interested not just in getting the complete answers, but in getting them with the lowest possible cost. The impact of access restrictions on cost-based optimization has been considered before. Florescu et al. [9] look at integrating access restrictions into a cost-based optimizer, following up on earlier cost-agnostic work on querying with access patterns by Li and Chang [12]. In the absence of integrity constraints, querying with access patterns amounts to a limitation on the search space, restricting the ordering of atoms within a query plan. In contrast, we allow schemas that can simultaneously restrict the search space (via access restrictions) and extend it (via integrity constraints, which allow relations outside of the query to become relevant) in comparison to the search space of traditional query optimizers.

Our approach starts by connecting plan generation with interpolation and preservation properties, building on the work of Nash, Segoufin, and Vianu [15]. [15] introduces the idea of going from a preservation property (in their case, determinacy of a query by views) to a plan (in their case, a rewriting of the query over the set of views). We give theorems for plan-generation over arbitrary first-order constraints, which work via new preservation and interpolation theorems. The results connecting preservation properties and existence of a plan can be seen as generalizations of [15] to the setting with access patterns and constraints, as explained later on. We also show how this connection can be "pushed down" to smaller classes of constraints, generating plans directly, rather than going via queries, and how this approach can be combined with cost considerations.

Several works provide algorithms for querying in the presence of both access patterns and integrity constraints, usually in the context of computing maximal answers rather than complete answers. Duschka et al. [8] include access patterns in their Datalog-based approach to data integration. They observe, following [11], that the accessible data can be "axiomatized" using recursive rules. We will make use of this axiomatization (see the "accessibility axioms" defined later on) but establish a tighter relationship between proofs that use these axioms and query plans.

Much closer to our work is the chase and backchase (C&B) method elaborated in work of Popa [17], Tannen, and Deutsch (e.g., [7, 6]) for reformulating queries on a physical schema while exploiting constraints. The main idea is to produce a "universal plan" (the chase) and then simplify it (back-chasing). Our work connects the approach via preservation of Nash, Segoufin and Vianu with the C&B method, while pointing out new applications. The proof-to-plan approach here applies to logics beyond dependencies (where the chase is not applicable), and to dependencies where the chase does not converge. We demonstrate this flexibility by discussing a cost-aware plan-generation algorithm for *guarded dependencies*. This class of constraints is orthogonal to those in [17, 7], which use an OO model that includes both TGDs and EGDs (e.g. keys), but relying on chase termination. In the special case of TGDs where the chase terminates, our algorithm can be seen as a way of combining the cost-based C&B described in Chapter 6 of Popa's thesis [17] with the access-method extension of C&B given in the work of Deutsch, Ludäscher, and Nash [5]. We note that unlike [17, 6] the cost-based algorithm outlined in this paper is not targeted towards generating physical plans on a single source (and thus we do not assume, as in these works, a fine model of how physical and logical schemas interact) – instead we are interested in optimizing the performance of expensive queries on top of sources, using a DBMS in the middleware.

Chapter 5 of the book of Toman and Weddell [18] (see also [**?**]) outlines an approach to reformulating queries with respect to a physical schema that is based on proofs. They discuss proofs using the chase algorithm, as well as an extended proof system connected to Craig Interpolation, remarking as we do that the latter can synthesize plans that are not conjunctive. Our results give a finer look at how plan-generation is impacted by the expressiveness of integrity constraints, axioms for capturing access restrictions, and the chosen proof system.

The recent work [3] studies the complexity of "complete answerability" for constraints in guarded logics. We will use several tools from [3], but our focus is exploring the relationship of proofs to plans: in the general first-order setting as well as for restricted constraint classes; finding optimal plans in addition to finding some plan; checking if a plan exists and finding plans efficiently.

Summary of Contributions. In summary, this work outlines a new perspective on implementing queries over restricted interfaces, by generating plans from proofs.

- We prove theorems characterizing the existence of a plan for a query. These results can be read as semantics-to-syntax results

– if a query has a certain preservation property then it can be rewritten in a certain form. They have an alternative effective reading, providing a recipe for deriving a plan from a proof.

- We show that these results generalize a number of theorems concerning views, and give a parallel between reformulation results for queries in databases and preservation and definability theorems in classical logic.
- We provide a new constructive interpolation theorem for predicate logic that generalizes many prior results, and provides a powerful new tool in the study of query reformulations.
- For TGDs, we give algorithms that produce plans directly from chase proofs. Informally, these algorithms perform interpolation directly over a plan language. We show that the proof-generated plans are as efficient as arbitrary plans based on conjunctive queries.
- We show how the direct proof-to-plan approach outlined for TGDs can be extended to find the lowest cost plan, even in the case of TGDs where the chase does not converge.

2. DEFINITIONS

Our starting point will be a *schema* which describes a querying scenario, consisting of:

- A collection of relations, each of a given arity. A *position* of a relation R is a number $\leq arity(R)$.
- A finite collection C of *schema constants* (“smith”, 3, ...). Informally, these represent a fixed set of values that a querier might use as test values in accesses. For example, if the user is performing a query involving the string “smith”, we would assume that “smith” was a schema constant – but not arbitrary unrelated strings.
- For each relation R, a collection (possibly empty) of *access methods*. Each method mt is associated with a collection of positions of R – the *input positions* of mt.
- A collection of *integrity constraints*, which we will always assume are given by sentences of first-order logic (interpreted under the active domain semantics [1]), using only relations and constants from the schema.

  We will give particular attention to constraints given by *tuple-generating dependencies* (TGDs), given syntactically as

  $$\forall \mathbf{x} \varphi(\mathbf{x}) \rightarrow \exists \mathbf{y} \rho(\mathbf{x}, \mathbf{y})$$

  where φ and ρ are conjunctions of relational atoms, possibly including constants.

  A special subclass consists of *Guarded TGDs*, in which φ is of the form $R(\mathbf{x}) \wedge \varphi'$ where $R(\mathbf{x})$ contains all variables of φ'. These subsume *inclusion dependencies* (IDs): where φ is of the form $R(\mathbf{x})$ in which no variables are repeated and there are no constants, while ρ is also a single atom with no repeated variables or constants. IDs are also called “referential constraints”.

Informally, the access methods give restrictions on how relations can be accessed. A standard example of relations with access methods comes from Web forms, where the input positions represent mandatory fields of the form.

We will use standard terminology for describing queries in first-order logic, including the notion of free variable, quantifiers, connectives, etc. [1]. A database instance (or just database) I for schema S assigns to every relation R in S a collection of tuples $I(R)$ of the right arity, in such a way that any integrity constraints of S are satisfied. An association of a database relation R with a tuple $\mathbf{c}$ of the proper arity will be referred to as a *fact*. A database instance can equivalently be seen as a collection of facts.

We consider *conjunctive queries* (CQs) $Q(\mathbf{x}) = \exists \mathbf{y}(A_1 \wedge \cdots \wedge A_n)$, where A_i is an atom using a relation of the schema and variables from $\mathbf{x}$ and $\mathbf{y}$ and/or constants from the schema as arguments. These are equivalent to *ESPJ* queries in relational algebra (defined below), and we will freely move back-and-forth between logic-based notation and relational algebra notation, and also between positional and attribute-based notation for components of a tuple. Given a conjunctive query Q and instance I, $Q(I)$ is the result of evaluating Q on I.

Access plans and costs. We look at plans for answering a query respecting the access methods.

An *access command* over a schema S with access methods consists of an access method mt from S on some relation R, a relational algebra expression E over some set of relations not in S (“temporary relations” henceforward), a bijective mapping b_{in} taking output attributes of E to the input positions of mt, an output temporary relation T and a binary relation b_{out} relating positions of R to positions of T, such that every position of T is in the range of b_{out}.

We will generally omit the mappings in the notation (and it will be clear from context in the examples), and we write an access command as $T \Leftarrow \mathsf{mt} \Leftarrow E$, denoting that mt is invoked with inputs produced by relational algebra expression E with the result going into T. For example, a plan for the query in Example 2 might begin with a command $T_1 \Leftarrow \mathsf{mt}_{\mathsf{Ids}} \Leftarrow \emptyset$, where $\mathsf{mt}_{\mathsf{Ids}}$ is the input-free access to table Ids, T_1 is a temporary table with a single column, and $\emptyset$ (by convention) represents no input.

A *middleware query command* is of the form $T := E$, where T is a temporary relation and E is a relational algebra expression. The input relations of E are temporary relations filled by other access commands, while the relation T may be used as input in further commands.

An *RA-plan* consists of a sequence of access and middleware query commands, along with a distinguished final output relation T_{fin}. We can similarly talk about *SPJ-plans*, where the expressions in access and middleware query commands are built up from relational algebra operators SELECT, PROJECT, and JOIN, *USPJ-plans* that allow UNION in addition to SPJ operators, and $USPJ^{\neg}$*-plans* that allow the difference operator $E - E'$ to be applied only in taking the output tuples of E and subtracting out tuples that are in some relation R, where R has at least one method. Hence $USPJ^{\neg}$-plans implement queries that use the difference operator only when the second argument is a relation. Similarly we can talk about plans for other relational algebra fragments. We allow for the use of inequalities in selections and join conditions, and we denote by *ESPJ* the fragment of SPJ where only equality conditions are allowed. *EUSPJ* and $EUSPJ^{\neg}$ are defined analogously.

The semantics of plans is straightforward. Given a database instance I that interprets a relation R with access method mt having input positions $j_1 \dots j_m$, and also interpreting the relations mentioned in expression E, an access command $T \Leftarrow \mathsf{mt} \Leftarrow E$ is executed by performing the query E on I and “accessing mt on every result tuple”. That is, each output tuple of E is mapped to a tuple $t_{j_1} \dots t_{j_m}$ using the input mapping. For each tuple $\mathbf{t} = t_1 \dots t_n \in R$ that “matches” (i.e. that extends) $t_{j_1} \dots t_{j_m}$, $\mathbf{t}$ is transformed to a tuple $\mathbf{t}'$ added to T using the output mapping b_{out}. If b_{out} maps one position of R to multiple positions in T, the values t_i are duplicated in $\mathbf{t}'$. If b_{out} maps positions p, p' of R to the same position in T, then a tuple $\mathbf{t}'$ is only added to T if $t_p = t_{p'}$. A middleware query command $T := E$ executes query E on the current contents of the temporary tables mentioned in E, and then places the result in temporary table T. A plan is evaluated on an instance of S by evaluating each command in sequence, starting with all temporary relations initially empty. The *output* of the plan on an input database is the content of the table T_{fin}.

Given a schema S with access methods and constraints, a plan *answers* a query Q *(over all instances)* if for every instance I satisfying the constraints of S, the output of the plan on I is the same as the output of Q. We say that the plan *answers Q over finite instances* if this holds for every finite instance I satisfying the constraints.

For general constraints, answering over all instances is not always the same as answering over finite instances, and in this work *we will always deal, by default, with the unrestricted notion of answering*. We will discuss the modification for the finite case later in the paper, and in particular we will show that for the main class of constraints we work with in our implementation (Guarded TGDs) the two notions of answerability coincide.

The requirement that the plan generate *all* outputs of Q on *every* instance formalizes our notion of “complete answerability” mentioned in the introduction.

Cost. We will be interested in finding plans that minimize a *plan cost function*, which associates every plan with a real-valued cost. Our framework can work with a “black box” cost function on plans that is monotone as additional access commands are concatenated to the plan.

If no information about the underlying sources is available, a default cost metric would associate each access method mt with a positive rational cost c_{mt}, and then the total cost of a plan whose access commands are calls to $\mathsf{mt}_1 \dots \mathsf{mt}_n$ (with possibly the same method repeated with different arguments) would be defined as $\Sigma_{i \leq n} c_{\mathsf{mt}_i}$. We refer to these as *simple cost functions* in the remainder, and we will provide refinements of the algorithms for this case.

3. FROM FO PROOFS TO PLANS

We describe how to generate plans that correspond to proofs that a query can be answered, in the setting of arbitrary first-order integrity constraints. Two kinds of reasoning are needed to know whether a query can be answered. One concerns the semantic relationships between tables, captured by integrity constraints. Another concerns what sort of access we have to relations. We formalize this second type of reasoning by revisiting and extending a prior technique for *axiomatizing properties of accesses in rules* (see [8, 5]).

Given schema S_0, the *Accessible Schema for* S_0, denoted $\mathsf{AcSch}(S_0)$, is the schema without any access restrictions, such that:

- The constants are those of S_0.
- The relations are those of S_0, a copy of each relation R denoted $\mathsf{Accessed}R$ (the "accessible version of R"), a unary relation $\mathsf{accessible}(x)$ ("x is an accessible value") plus another copy of each relation R of S_0 called $\mathsf{InferredAcc}R$ – the "inferred accessible version of R". We refer to the relations of the form $\mathsf{Accessed}R$ and the relation $\mathsf{accessible}$ as the "accessible copy of S_0" .
- The constraints are those of S_0 (referred to as "original constraints" below) along with the following constraints (dropping universal quantifiers on the outside for brevity)
  - A "defining axiom" for the relation $\mathsf{accessible}$:

$$\mathsf{Accessed}R(\mathbf{x}) \rightarrow \mathsf{accessible}(x_i)$$

    for every R and x_i in $\mathbf{x}$. Informally, this says that $\mathsf{accessible}$ is the active domain of all accessible facts.
  - *accessibility axioms*: for each access method mt on relation R of arity n with input positions $j_1 \dots j_m$ we have a rule:

$$\mathsf{accessible}(x_{j_1}) \wedge \dots \wedge \mathsf{accessible}(x_{j_m}) \wedge R(x_1 \dots x_n) \rightarrow \mathsf{Accessed}R(x_1 \dots x_n)$$

    In addition, we have $\mathsf{accessible}(c)$ for each constant c of S_0.
  - *inferred accessible fact rules*, which are of two forms. First we have rules:

$$\mathsf{Accessed}R(\mathbf{x}) \rightarrow \mathsf{InferredAcc}R(\mathbf{x})$$

    Secondly, we have a copy of each of the original integrity constraints, with each relation R replaced by $\mathsf{InferredAcc}R$.

Informally, the accessible versions of relations represent the facts that can be explicitly retrieved from the access methods, while $\mathsf{accessible}(c)$ indicates that the value c can be returned by some access. The inferred accessible relations represent facts that can be derived from the accessible facts using reasoning. Thus the accessible schema represents the rules that allow one to move from a "hidden fact" (e.g. $R(c_1 \dots c_n)$) to an accessible fact ($\mathsf{Accessed}R(c_1 \dots c_n)$), and from there – using the constraints – to an inferred fact (e.g. $\exists y\ \mathsf{InferredAcc}S(c_1 \dots c_n, y)$). From the structure of the rules it is easy to see that "inferred accessible rules" can fire based upon facts generated by other kinds of rules, but not vice versa.

Given a query Q, its *inferred accessible version* $\mathsf{InferredAcc}Q$ is obtained by replacing each relation R by $\mathsf{InferredAcc}R$ and adding the atom $\mathsf{accessible}(x)$ for every free variable. $\mathsf{InferredAcc}Q$ represents the fact that the existence of a witness to Q can be obtained through making accesses and reasoning.

We say that Q *entails* Q' with respect to a set of integrity constraints if in any instance that satisfies the constraints $\forall \mathbf{x}(Q(\mathbf{x}) \rightarrow Q'(\mathbf{x}))$ holds. As with other notions, by default, we deal here with arbitrary instances, not necessarily finite.

In particular, if Q entails $\mathsf{InferredAcc}Q$, this means that we can infer from Q holding in a hidden database that Q's truth can be learned by a user via accesses and reasoning with constraints.

EXAMPLE 3. *The query Q from Example 1 can be expressed as* $\exists \mathsf{onum}\ \mathsf{Profinfo}(\mathsf{eid}, \mathsf{onum}, \text{"smith"})$. *Therefore* $\mathsf{InferredAcc}Q$ *is* $\mathsf{accessible}(\mathsf{eid}) \wedge \exists \mathsf{onum}\ \mathsf{InferredAccProfinfo}(\mathsf{eid}, \mathsf{onum}, \text{"smith"})$. *The accessible schema includes rules:*

- $\mathsf{Profinfo}(\mathsf{eid}, \mathsf{onum}, \mathsf{lname}) \rightarrow \mathsf{Udirect}(\mathsf{eid}, \mathsf{lname})$
- $\mathsf{Udirect}(\mathsf{eid}, \mathsf{lname}) \rightarrow \mathsf{AccessedUdirect}(\mathsf{eid}, \mathsf{lname})$
- $\mathsf{AccessedUdirect}(\mathsf{eid}, \mathsf{lname}) \rightarrow \mathsf{accessible}(\mathsf{eid}) \wedge \mathsf{accessible}(\mathsf{lname})$
- $\mathsf{Profinfo}(\mathsf{eid}, \mathsf{onum}, \mathsf{lname}) \wedge \mathsf{accessible}(\mathsf{eid}) \rightarrow \mathsf{AccessedProfinfo}(\mathsf{eid}, \mathsf{onum}, \mathsf{lname})$
- $\mathsf{AccessedProfinfo}(\mathsf{eid}, \mathsf{onum}, \mathsf{lname}) \rightarrow \mathsf{InferredAccProfinfo}(\mathsf{eid}, \mathsf{onum}, \mathsf{lname})$

One can see that Q entails $\mathsf{InferredAcc}Q$ *with respect to these rules.*

The relationship between entailment using the accessible schema and plans is encoded in the following theorem:

THEOREM 1. *For any conjunctive query Q and schema S_0 containing constraints specified in first-order logic and access restrictions, there is a complete USPJ-plan for Q (over databases in S_0) if and only if Q entails* $\mathsf{InferredAcc}Q$ *with respect to* $\mathsf{AcSch}(S_0)$.

Furthermore, if the constraints of S_0 are specified by equality-free first-order formulas (e.g., TGDs), then we can replace USPJ-plan with EUSPJ-plan in the above statement.

Theorem 1 gives a correspondence between Q entailing $\mathsf{InferredAcc}Q$ and the existence of complete plans based on unions of conjunctive queries. Its proof, discussed further down, yields an algorithm for generating plans, given a proof witnessing that Q entails $\mathsf{InferredAcc}Q$. But for first-order constraints such proofs cannot be found effectively. We will look at plan-generation algorithms for more restricted constraints in the next section.

Extending to larger classes of plans. The proof/plan correspondence is not limited to USPJ plans. Let $\mathsf{AcSch}^{\leftrightarrow}(S_0)$ extend the axioms of $\mathsf{AcSch}(S_0)$ with the axioms $\forall \mathbf{x}\ \mathsf{Accessed}R(\mathbf{x}) \rightarrow R(\mathbf{x})$ and the following axioms (universal quantifiers omitted):

$$\bigwedge_{i \leq m} \mathsf{accessible}(x_{j_i}) \wedge \mathsf{InferredAcc}R(x_1 \dots x_n) \rightarrow \mathsf{Accessed}R(x_1 \dots x_n)$$

Above, R is a relation of S_0 having an access method with input positions $j_1 \dots j_m$. Notice that these rules are obtained from those of $\mathsf{AcSch}(S_0)$ by switching the roles of $\mathsf{InferredAcc}R$ and R, resulting in a rule set where the original schema and the $\mathsf{InferredAcc}$ copy are treated symmetrically. The following result shows that provability with these "bi-directional axioms" corresponds to existence of a relational algebra plan:

THEOREM 2. *For any relational query Q and schema S_0 containing access restrictions and constraints specified in first-order logic, there is an RA-plan answering Q (over databases in S_0) if and only if Q entails* $\mathsf{InferredAcc}Q$ *with respect to the rules in* $\mathsf{AcSch}^{\leftrightarrow}(S_0)$.

In between the RA and USPJ versions, we have a version for queries that allow atomic negation: allowing the difference operator $E - R$, but where R must be a relation. Let $\mathsf{AcSch}^{\neg}(S_0)$ extend $\mathsf{AcSch}(S_0)$ with the following "negative accessibility axioms", for all relations R that have some access method:

$$\bigwedge_{i \leq n} \mathsf{accessible}(x_i) \wedge \neg R(x_1 \dots x_n) \rightarrow \neg \mathsf{InferredAcc}R(x_1 \dots x_n)$$

Intuitively, the rule says that if the hidden database does *not* include a hidden fact, and all the values in the fact are known to a user, then the user can infer that the fact does not hold using accesses. The result below shows that this axiom system characterizes $USPJ^{\neg}$ plans:

THEOREM 3. *For any relational algebra query Q and schema S_0 containing access restrictions and constraints specified in first order logic, there is a complete $USPJ^{\neg}$-plan for Q (over databases in S_0) if and only if Q entails* $\mathsf{InferredAcc}Q$ *with respect to* $\mathsf{AcSch}^{\neg}(S_0)$.

Furthermore, if the constraints of S_0 are equality-free then we can replace $USPJ^{\neg}$-plan with $EUSPJ^{\neg}$-plan.

Proofs of the main first-order theorems. We now begin the proofs of these three theorems, beginning with two main tools, executable queries and an interpolation theorem that can produce executable queries.

Executable queries and plans. In creating our plans that witness the proof-to-plan direction of the theorems, we will sometimes find it convenient to produce not a plan, but an *executable FO rewriting*. For us, an *executable FO query* (relative to a schema with access methods) is a first-order formula built up from equalities and the formula True using arbitrary boolean operations and the quantifiers:

$$\forall \mathbf{y}\ [(R(\mathbf{x},\mathbf{y}) \rightarrow \varphi(\mathbf{x},\mathbf{y},\mathbf{z})] \quad \text{and} \quad \exists \mathbf{y}\ R(\mathbf{x},\mathbf{y}) \wedge \varphi(\mathbf{x},\mathbf{y},\mathbf{z})$$

where R has an access method mt such that, in $R(\mathbf{x},\mathbf{y})$ above, all of the input positions of mt are occupied by some x_i. An executable FO rewriting of Q (relative to a schema S_0 with access methods and constraints) is an executable FO query for S_0 that is equivalent to Q over instances that satisfy the constraints of S_0. To improve readability, we will drop the schema S_0 from the terminology, and just talk about executable FO queries and rewritings henceforward.

The intuition is that these queries are such that given an input tuple matching the free variables, we can use the access methods to verify whether it satisfies the query. In particular, executable boolean RA queries can be easily converted directly to RA-plans via induction.

PROPOSITION 1. *There is a linear time procedure converting an executable boolean FO query into an RA-plan. Furthermore, if the FO query is existential (resp. existential without inequalities) the result is a $USPJ^{\neg}$ (resp. $EUSPJ^{\neg}$)-plan, while if the query is positive existential, the result is a USPJ (resp. EUSPJ)-plan*

Interpolation. The next key element of our proofs will be *interpolation theorems*. The Craig interpolation theorem states that if φ_1 and φ_2 are first-order formulas such that φ_1 entails φ_2, then there exists a first-order formula φ such that φ_1 entails φ and φ entails φ_2, and φ uses only relation symbols occurring in both φ_1 and φ_2.

Variants of the Craig interpolation theorem exist that allow one to make further conclusions about the interpolant. For example, the Lyndon interpolation theorem [14] restricts the relations that occur positively (resp. negatively) in the interpolant to those occurring positively and negatively on both the left and right. More recently Otto [16] has proven a more powerful *relativized interpolation theorem* that not only controls the polarity of relations, but the pattern of quantification that occurs within them. Inspired by Otto's result, we prove a version of Craig interpolation that allows us to relate the "binding patterns" used in the interpolant φ with those used in φ_1 or φ_2. When we apply this theorem to the entailment of $\mathsf{InferredAcc}Q$ by Q, we will be able to conclude that the interpolant is an executable query.

We associate to first-order formulas the set of binding patterns used in quantification, where a binding pattern is a relation and a subset of the positions. This is done by induction on the formula:

$$\begin{array}{lcl}
\mathsf{BindPatt}(\top) = \mathsf{BindPatt}(x=y) & = & \emptyset \\
\mathsf{BindPatt}(R(t_1,\ldots,t_n)) & = & \{(R,\{1,\ldots,n\})\} \\
\mathsf{BindPatt}(\neg\varphi) & = & \mathsf{BindPatt}(\varphi) \\
\mathsf{BindPatt}(\varphi\wedge\psi) & = & \mathsf{BindPatt}(\varphi)\cup\mathsf{BindPatt}(\psi) \\
\mathsf{BindPatt}(\varphi\vee\psi) & = & \mathsf{BindPatt}(\varphi)\cup\mathsf{BindPatt}(\psi) \\
\mathsf{BindPatt}(\exists\mathbf{x}(R(t_1,\ldots,t_n)\wedge\varphi)) & = & \mathsf{BindPatt}(\varphi)\cup\{(R,\{i \mid t_i\not\in\mathbf{x}\})\} \\
\mathsf{BindPatt}(\forall\mathbf{x}(R(t_1,\ldots,t_n)\rightarrow\varphi)) & = & \mathsf{BindPatt}(\varphi)\cup\{(R,\{i \mid t_i\not\in\mathbf{x}\})\}
\end{array}$$

Intuitively, $\mathsf{BindPatt}(\varphi)$ describes the kind of access that is used if φ is evaluated in an instance using a straightforward inductive evaluation procedure. For example,

$$\mathsf{BindPatt}(\exists xy(Rxy \wedge \forall z(Sxyz \rightarrow Uxyz))) = \{(R,\emptyset),(S,\{1,2\}),(U,\{1,2,3\})\}$$

In particular, if each pattern in $\mathsf{BindPatt}(\varphi)$ is represented by a method in a schema S_0, then φ is an executable FO query for S_0.

Note that, for formulas φ containing unrestricted quantifiers, such as $\exists x\ \neg P(x)$, $\mathsf{BindPatt}(\varphi)$ is undefined. However, every conjunctive query can be viewed (modulo minor syntactic transformations) as having no unrestricted quantifiers, and, furthermore, under the active domain semantics, every formula is equivalent to one without unrestricted quantifiers.

We say that a relation symbol R *occurs positively* (*negatively*) in a formula φ if some occurrence of R in φ is in the scope of an even (odd) number of negations. For the purpose of this definition, we view the implication symbol as a shorthand: $\psi \rightarrow \chi$ stands for $\neg\psi \vee \chi$. Thus, for example, in the formula $\forall x(P(x) \rightarrow \exists y R(x,y))$, the relation symbol P occurs negatively and R occurs positively.

THEOREM 4 (ACCESS INTERPOLATION). *Let φ_1 and φ_2 be first-order sentences such that φ_1 entails φ_2. Then there exists a first-order sentence φ such that*
1. *φ_1 entails φ and φ entails φ_2,*
2. *A relation symbol occurs positively (negatively) in φ only if it occurs positively (negatively) in both φ_1 and φ_2.*
3. *A constant symbol occurs in φ only if it occurs both in φ_1 and φ_2*
4. *If* $\mathsf{BindPatt}(\varphi_1)$ *and* $\mathsf{BindPatt}(\varphi_2)$ *are both defined, then* $\mathsf{BindPatt}(\varphi) \subseteq \mathsf{BindPatt}(\varphi_1) \cup \mathsf{BindPatt}(\varphi_2)$.
5. *If φ_1 and φ_2 are both equality-free, then φ is equality-free.*

Furthermore, ψ can be computed in polynomial time from a proof (in a suitable proof system) of the entailment $\varphi_1 \rightarrow \varphi_2$.

The proof of the Access interpolation theorem, deferred to the full paper, is constructive (in fact, polynomial time), producing an interpolant inductively from a proof that φ_1 entails φ_2 in a particular proof system. Both the proof system used (tableaux) and the technique used to extract the interpolant follow along the lines of a standard technique for interpolation, annotating the proof elements with a "bias" and extracting an interpolant bottom-up on the proof tree (e.g. [2, 4]). The new component is an analysis of the relationship between binding patterns in the output formula and those in the input formula.

$\mathsf{AcSch}^{\leftrightarrow}$ **and RA-plans.** We now begin with the proofs, starting with Theorem 2, because it is the simplest of the results, and will establish a template used in the proofs of the other two results.

Recall that given a schema S_0 with constraints and access restrictions, the set of constraints $\mathsf{AcSch}^{\leftrightarrow}(S_0)$ includes all constraints of $\mathsf{AcSch}(S_0)$, as well as the additional rules:

$$\forall x_1 \ldots x_n\ \mathsf{Accessed}R(x_1 \ldots x_n) \rightarrow R(x_1 \ldots x_n)$$

and also the rules:

$$\begin{array}{l}\forall x_1 \ldots x_n\ \mathsf{InferredAcc}R(x_1 \ldots x_n) \wedge \\ \quad \mathsf{accessible}(x_{j_1}) \ldots \mathsf{accessible}(x_{j_m}) \rightarrow \mathsf{Accessed}R(x_1 \ldots x_n)\end{array}$$

for every relation R with an access method mt on positions $j_1 \ldots j_m$.

For simplicity, in this proof, as well as others in the section, we deal with the case where Q is boolean and there are no constants in the schema.

We recall several notions from [3]. Given an instance I for schema S_0 the *accessible part of I*, denote $\mathsf{AccPart}(I)$ consists of all the facts over I that can be obtained by starting with empty relations and iteratively entering values into the access methods. Formally, it is a database containing a set of facts $\mathsf{Accessed}R(v_1 \ldots v_n)$, where R is a relation and $v_1 \ldots v_n$ are values in the domain of I such that $R(v_1 \ldots v_n)$ holds in I, obtained by starting with relations $\mathsf{Accessed}R_0$ and $\mathsf{accessible}_0$ empty [1], and then iterating the following process until a fixpoint is reached:

- $\mathsf{accessible}_{i+1} = \mathsf{accessible}_i \cup \bigcup_{\substack{R \text{ a relation} \\ j<arity(R)}} \pi_j(\mathsf{Accessed}_i(R))$

[1]In the presence of schema constants, we would start with $\mathsf{accessible}_0$ consisting of the schema constants

- $\mathsf{Accessed}_{i+1}(R) = \mathsf{Accessed}_i(R) \cup$
$$\bigcup_{\substack{(R,\{j_1,\ldots,j_m\}) \\ \text{a method of } S_0}} \{(v_1 \ldots v_n) \in I(R) \mid v_{j_i} \in \mathsf{accessible}_i \text{ for all } i \leq m\}$$

where $\pi_j(\mathsf{Accessed}_i(R))$ is the j-th projection of $\mathsf{Accessed}_i(R)$.

Above we consider $\mathsf{AccPart}(I)$ as a database instance for the schema with relations $\mathsf{accessible}$ and $\mathsf{Accessed}R$. Below we will sometimes refer to the values in the relation $\mathsf{accessible}$ as the *accessible values of I*. An immediate observation is that if we expand the database I with the interpretations of $\mathsf{accessible}$ and the $\mathsf{Accessed}R$ from $\mathsf{AccPart}(I)$, the result will satisfy all the axioms of $\mathsf{AcSch}^{\leftrightarrow}(S_0)$ that relate the original relations R to $\mathsf{accessible}$ and the relations $\mathsf{Accessed}R$.

The following proposition states that an instance and its accessible part agree on executable FO queries.

PROPOSITION 2. *For any instance I of S_0, let I' be the accessible part of I, seen as a structure for the relations of S_0: that is, the relations R is interpreted in I' by $\mathsf{Accessed}R$ in the accessible part of I. Then for executable FO formula ρ and any binding $\mathbf{b}$ of the variables in ρ to elements in $\mathsf{AccPart}(I)$, ρ is true on $I, \mathbf{b}$ iff ρ is true on $I', \mathbf{b}$. In particular, I and I' agree on all executable boolean FO queries.*

The proposition can be proven straightforwardly via induction on ρ, or via the appropriate back-and-forth game.

Q is said to be *access-determined* over S_0 if for all instances I and I' satisfying the constraints of S_0 with $\mathsf{AccPart}(I) = \mathsf{AccPart}(I')$ we have $Q(I) = Q(I')$. If a query is *not* access-determined, it is obvious that it cannot be answered through any plan, since it is easy to see that any plan can only read accessible tuples. The following claim restates our entailment hypothesis in terms of this preservation property.

CLAIM 1. *The following are equivalent (for any schema consisting of first-order constraints and access restrictions):*
1. *Q entails $\mathsf{InferredAcc}Q$ with respect to the rules in $\mathsf{AcSch}^{\leftrightarrow}(S_0)$*
2. *Q is access-determined over S_0*

PROOF. We prove that the first item implies the second. Fix I and I' satisfying the schema with the same accessible part, and assume I satisfies Q. Consider the instance I'' for the accessible schema formed by interpreting the relations R as in I, each relation $\mathsf{Accessed}R$ by the accessible tuples that R has in I (that is, the relation $\mathsf{Accessed}R$ of the accessible part of I, defined via the fixpoint process described above), the relation $\mathsf{accessible}$ by the accessible values of I, and each $\mathsf{InferredAcc}R$ by the interpretation of R in I'. Then one can easily verify that I'' satisfies the constraints of $\mathsf{AcSch}^{\leftrightarrow}(S_0)$. Since I (and hence I'') satisfies Q, and we are assuming that Q entails $\mathsf{InferredAcc}Q$ with respect to $\mathsf{AcSch}^{\leftrightarrow}(S_0)$ we can conclude that I'' must satisfy $\mathsf{InferredAcc}Q$. Thus Q holds in I' as required.

We now argue from the second item to the first, which will complete the proof of the claim. Suppose Q does not imply $\mathsf{InferredAcc}Q$ with respect to the rules in $\mathsf{AcSch}^{\leftrightarrow}(S_0)$. Hence there is an instance $I^{\mathsf{AcSch}^{\leftrightarrow}}$ satisfying the rules of $\mathsf{AcSch}^{\leftrightarrow}(S_0)$ and also satisfying $Q \wedge \neg\mathsf{InferredAcc}Q$. Let I_1 consist of the restriction of $I^{\mathsf{AcSch}^{\leftrightarrow}}$ to the original schema relations. Let I_2 consist of the inferred accessible relations from $I^{\mathsf{AcSch}^{\leftrightarrow}}$, renamed to the original schema. We first claim that a fact $F = R(e_1 \ldots e_n)$ of the accessible part of I_1 is in the accessible part of I_2. We prove this by induction on the iteration of the fixpoint where F appears. F must be generated by an access using elements $e_{j_1} \ldots e_{j_m}$ which in turn satisfy accessible facts generated earlier in the fixpoint iteration. Thus by induction these earlier facts are in the accessible part of I_2, and in particular $e_{j_1} \ldots e_{j_m}$ are accessible values of I_2. Using the axioms we have that $\mathsf{InferredAcc}R(e_1 \ldots e_n)$ holds, and thus $R(e_1 \ldots e_n)$ holds in I_2. Using the definition of accessible part, we conclude that F is in the accessible part of I_2 as required. Arguing symmetrically, we have that I_1 and I_2 have the same accessible part, and hence they contradict access-determinacy. $\square$

From this claim, we easily see the "plan-to-proof" direction of Theorem 2. Suppose Q does not imply $\mathsf{InferredAcc}Q$ with respect to the rules in $\mathsf{AcSch}^{\leftrightarrow}(S_0)$. By Claim 1, Q is not access-determined, and thus no plan can answer Q.

Note that Claim 1 allows us to restate Theorem 2 as: *A query Q has an RA-plan iff Q is access-determined.* Thus, we can think of the theorem as a kind of preservation theorem (see the comparison with earlier results below).

We now prove the "proof-to-plan" direction of Theorem 2: assuming Q entails $\mathsf{InferredAcc}Q$, we construct an RA-plan that answers Q.

We can remove the relation $\mathsf{accessible}$ from the schema, modifying the rules of $\mathsf{AcSch}^{\leftrightarrow}(S_0)$ by breaking up the relation $\mathsf{accessible}$ as a disjunction of relations $\mathsf{Accessed}S$. That is, we could replace every accessibility axiom

$$\bigwedge_{i \leq m} \mathsf{accessible}(x_{j_i}) \wedge R(\mathbf{x}) \rightarrow \mathsf{Accessed}R(\mathbf{x})$$

by all axioms of the form

$$\alpha_1(x_{j_1}) \wedge \cdots \wedge \alpha_n(x_{j_n}) \wedge R(\mathbf{x}) \rightarrow \mathsf{Accessed}R(\mathbf{x}))$$

where $\alpha_i(x_{j_i})$ is of the form $\mathsf{Accessed}S(\mathbf{y}, x_{j_i}, \mathbf{z})$ for some S and fresh $\mathbf{y}, \mathbf{z}$.

Thus from now on we will assume that $\mathsf{accessible}$ does not appear.

We can rephrase the assumption as:

$$Q \wedge \Sigma_1 \text{ entails } (\Sigma_2 \rightarrow \mathsf{InferredAcc}Q)$$

where Σ_1 is the set of constraints in $\mathsf{AcSch}^{\leftrightarrow}(S_0)$ mentioning only the original relations and the relations $\mathsf{Accessed}R$, while Σ_2 contains all constraints mentioning relations of the form $\mathsf{InferredAcc}R$. As observed above, we can assume that Σ (and hence Σ_1, Σ_2) do not use unrestricted quantification.

By the Access interpolation theorem there is a first-order sentence φ using only relations of the form $\mathsf{Accessed}R$ such that:
1. $Q \wedge \Sigma_1$ entails φ
2. φ entails $\Sigma_2 \rightarrow \mathsf{InferredAcc}Q$
3. φ only uses binding patterns occurring in $Q \wedge \Sigma_1$ or in $\Sigma_2 \rightarrow \mathsf{InferredAcc}Q$

For a formula ρ using only the relations $\mathsf{Accessed}R$, let $\mathsf{deacc}(\rho)$ be obtained be replacing each relation $\mathsf{Accessed}R$ of ρ with R. Note that since the binding patterns of the accessibility axioms are all compatible with some method of the schema (inputs of the pattern are contained in the input positions of the method) $\mathsf{deacc}(\varphi)$ is an executable FO query. Further, Proposition 2 implies that $\mathsf{deacc}(\varphi)$ holds on I iff φ holds on $\mathsf{AccPart}(I)$.

We claim that $\mathsf{deacc}(\varphi)$ is an executable FO rewriting of Q (and hence can be converted to an RA-plan using Proposition 1). We prove this by looking at the case where Q holds on an instance I of S_0 and the case where Q does not hold on I. If Q is true on I, then letting I' be the extension of I with relations $\mathsf{Accessed}R$ interpreted as in $\mathsf{AccPart}(I)$, we have that $I + \mathsf{AccPart}(I)$ satisfies $Q \wedge \Sigma_1$. So by the first condition above we will have φ is true on $\mathsf{AccPart}(I)$, and thus $\mathsf{deacc}(\varphi)$ is true on I. On the other hand, if $\mathsf{deacc}(\varphi)$ is true on I then φ is true on $\mathsf{AccPart}(I)$. Consider the database instance I'' interpreting each $\mathsf{Accessed}R$ as in $\mathsf{AccPart}(I)$ and each $\mathsf{InferredAcc}R$ by R in I. I'' satisfies Σ_2. Thus applying the second condition on φ to I'', we get that $\mathsf{InferredAcc}Q$ holds in I'', and hence Q holds in I.

This completes the proof of Theorem 2.

AcSch and USPJ-plans. We now turn to the proof of Theorem 1. As before, we will assume for simplicity that Q is a boolean query and there are no schema constants present.

We will again translate entailment in our axiom schema into a preservation property of models.

We say Q is *subinstance-access-determined* over S_0 if for all instances I and I' satisfying the constraints of S_0 with every fact of $\mathsf{AccPart}(I)$ contained in $\mathsf{AccPart}(I')$ (that is, $\mathsf{AccPart}(I)$ is a subinstance of $\mathsf{AccPart}(I')$), if I satisfies Q, then I' satisfies Q.

That is, we have weakened the hypothesis of access-determinacy to require only containment of facts, not equality.

The following claim now relates these notions to our axioms, analogously to Claim 1. Its proof follows along the lines of Claim 1, and is given in the full paper.

CLAIM 2. *The following are equivalent (for any schema consisting of first-order constraints and access restrictions):*
1. Q entails $\mathsf{InferredAcc}Q$ *with respect to the rules in* $\mathsf{AcSch}(S_0)$
2. Q is subinstance-access-determined w.r.t. S_0

Assuming this claim, Theorem 1 can be restated as saying that a query is subinstance-access-determined iff it has a USPJ-plan. The proof of Theorem 1 follows along the lines of the proof of Theorem 2, with the direction from plan to proof using Claim 2, and the proof to plan applying the Access interpolation theorem to the entailment.

The schema $\mathsf{AcSch}^{\neg}$ and $USPJ^{\neg}$-plans. Finally, we discuss the proof of Theorem 3. We need a characterization corresponding to Claim 1 for $\mathsf{AcSch}^{\neg}$.

Recall that given an instance I, an *accessible value* is an element in the domain of I that occurs in some fact $\mathsf{Accessed}R(\mathbf{c})$ of $\mathsf{AccPart}(I)$. Equivalently, it is an element that satisfies the relation $\mathsf{accessible}$ in $\mathsf{AccPart}(I)$.

Given instances I and I', we say $\mathsf{AccPart}(I)$ is an *induced subinstance* of $\mathsf{AccPart}(I')$ if (*i*) every fact $\mathsf{Accessed}R(\mathbf{c})$ of $\mathsf{AccPart}(I)$ is in $\mathsf{AccPart}(I')$ and (*ii*) for every $\mathbf{c}$ with each c_i an accessible value of I, if $R(\mathbf{c})$ is a fact of $\mathsf{AccPart}(I')$ then $\mathsf{Accessed}R(\mathbf{c})$ is in $\mathsf{AccPart}(I)$.

We say that Q is *induced-subinstance-access-determined* with respect to S_0 if *whenever we have two instances I, I' satisfying the constraints of the schema S_0, I satisfies Q, and* $\mathsf{AccPart}(I)$ *is an induced subinstance of* $\mathsf{AccPart}(I')$*, then I' satisfies Q.*

The following claim, whose proof follows along the lines of the earlier ones, relates this new preservation property to provability in the axiom schema $\mathsf{AcSch}^{\neg}$.

CLAIM 3. *The following are equivalent (for any schema consisting of first-order constraints and access restrictions):*
1. Q entails $\mathsf{InferredAcc}Q$ *with respect to the rules in* $\mathsf{AcSch}^{\neg}(S_0)$
2. Q is induced-subinstance-access-determined w.r.t. S_0

The proof of Theorem 3 uses the claim above in one direction and Access interpolation in the other.

Consequences in the case of views, and comparison with preservation theorems.. We will now look at what the prior theorems say in the simpler case of *view-based access restrictions*. By this we mean that there is a subcollection of relations in the schema that are designated as fully accessible, and the integrity constraints just state that these relations are equivalent to first-order queries defining them. We will see that our results in this case are closely-related to preservation theorems in classical model theory.

Let us start with looking at Theorem 2, concerning RA-plans, in the setting of view-based access restrictions. In this simpler setting the notion of access-determinacy degenerates to the notion of *query-view determinacy* of Nash, Segoufin, and Vianu [15]. Theorem 2 then says that if a query is determined (over all instances) it is first-order rewritable over the views – a basic observation of [15], and one which is almost a restatement of the Projective Beth Definability Theorem of first-order logic. The proof of our Theorem 2 follows the proof of the Projective Beth Theorem via interpolation due to Craig [**?**]. Note that, unlike [15], the significance of this result for us is not as a "completeness theorem" (first-order logic suffices to express all rewritings), but as a way to obtain rewritings: we can begin to explore proofs, and from the proofs we can efficiently read off the rewritings.

The generalization for access patterns and arbitrary first-order constraints given in Theorem 2 has not been stated before, but it is very similar to results already proven in [3]: Theorem 4.5 of [3] starts with a preservation property of a query and concludes that the query has an "FO-k-rewriting", which is syntactically different from having an RA-plan.

Let us now look at Theorem 1, which concerns USPJ-plans, in the special case of view-based access restrictions. In this case the notion of subinstance-access-determinacy is again one considered by [15], the notion of *monotonicity*. Given a boolean relational algebra query Q and a set of relational algebra queries $Q_1 \dots Q_n$, we say that Q is *monotone in $Q_1 \dots Q_n$*, with respect to a set of constraints, if for any two instances I, I' satisfying the constraints, if I satisfies Q and for each $i \leq n$, the tuples returned by Q_i on I are a subset of those returned by Q_i on I', then Q is true on I. In the setting of finite instances, where there are no constraints, the proof of Theorem 5.6 of [15] shows that CQ Q is monotone in CQs $Q_1 \dots Q_n$ iff there is a rewriting of Q as a CQ in terms of the $Q_1 \dots Q_n$.

A corollary of the proof of Theorem 1 is the following variant for arbitrary RA queries in the presence of first-order constraints (where monotonicity is required over all instances):

COROLLARY 1. *Q is monotone in $Q_1 \dots Q_n$ iff there is a USPJ-rewriting of Q in terms of $Q_1 \dots Q_n$.*

That is, queries monotone in a set of views are USPJ-rewritable. For RA queries and arbitrary FO constraints, this requirement of monotonicity over all instances is needed. But when Q and the Q_i are CQs and the constraints come only from view definitions, we will see later that it can be weakened. Again, we want to emphasize the effective aspect of this: from a proof witnessing monotonicity, we can derive a rewriting.

As with the results on views, our theorem and also the special case above for views are closely related to a theorem in classical model theory, the Lyndon Preservation Theorem, which states that formulas preserved under surjective homomorphism are equivalent to positive formulas. Roughly speaking, the special case above is a version of Lyndon's Theorem that deals with the active domain semantics (which is what allows us to move from "positive" to "positive existential") and which is "projective" – allowing Q to be preserved under mappings preserving only a subset of the relations, and concluding that it has a nice rewriting using only the subset.

We now turn to Theorem 3. To our knowledge, the notion of preservation/determinacy considered in this theorem does not have any analog in the earlier literature on views. But as with Theorem 1, the proof of Theorem 3 can be applied to characterize queries that are $USPJ^{\neg}$-rewritable in terms of a distinguished set of view relations in the presence of constraints. We modify the definition of monotonicity to require that, for each $i \leq n$, $Q_i(I')$ is obtained from $Q_i(I)$ by adding facts, but never adding facts all of whose elements lie in some $Q_j(I)$ for $j \leq n$. If we call such queries *induced-subinstance-monotone* we can then conclude:

COROLLARY 2. *A conjunctive query is induced-subinstance-monotone over a set of queries $\{Q_1 \dots Q_n\}$ exactly when it is $USPJ^{\neg}$ rewritable over relations V_i for each Q_i.*

Again, there is an analogous result to Theorem 3 in classical model theory, namely the Łoś-Tarski Theorem, which characterizes existential formulas as those that are preserved under the notion of "model extension" used in classical model theory. Compared to this classical theorem, the corollary of Theorem 3 for views given just above differs in being "projective", and being used in the context of active-domain semantics.

Alternative proofs of the main first-order theorems. We mention here that all of these theorems can be proven directly from Otto's relativized interpolation theorem [16] and the compactness theorem of first-order logic. Applying Otto's theorem to the entailment of $\mathsf{InferredAcc}Q$ by Q, what we obtain is that Q can be answered by a first-order sentence φ of the appropriate form (e.g. existential for Theorem 3) that is to be evaluated over the accessible part of the instance. Computing the accessible part requires a recursive query, but the compactness theorem can be applied to show that only k "levels" of the accessible part are necessary (see the notion of "k-accessible part" in [3]). There is an $EUSPJ$-plan P_k that will produce this truncation of the accessible part: P simply performs k rounds of making every possible access with values produced by the previous round. Since RA-plans are allowed to run arbitrary RA queries in the middleware, the composition of a first-order query φ with P_k is also given by an RA-plan. Similar reasoning shows that an existential query composed with P_k is implementable by a USPJ plan, and so forth.

In fact, if we apply this alternative approach to the most basic result, Theorem 2, we see that for this theorem we do not need Otto's theorem but only the determinacy-implies-rewriting theorem of Nash, Segoufin, and Vianu (Theorem 3.1 of [15]), whose proof in turn makes use of Otto's theorem.

One drawback of this alternative approach is that it is non-constructive, since the proof of Otto's result in [16] is non-constructive, and also because it appeals to the compactness theorem. But even with a constructive proof of Otto's result and a bound on k, it has limitations: even when one can find short proofs effectively, the plans resulting from this technique will begin by doing every possible access up to k iterations, which is certainly not feasible. We thus find the approach via Access interpolation more promising for implementation, and also closer to the direct algorithms used for TGDs. In addition, we believe that the Access interpolation theorem is of independent interest.

Finite instances and restricted constraints. The results above related existence of a plan for a query Q that works over all instances that satisfy the constraints of schema S_0 with entailment of $\mathsf{InferredAcc}Q$ from Q relative to a schema derived from S_0. But suppose we want a method that will check existence of a plan that answers Q only over all *finite* instances? One can construct example schemas, even using TGD constraints, where there is an RA-plan that works over finite instances, but no plan that works over all instances.

Using standard counterexamples in finite model theory, we can show that if we restrict to finite instances Theorem 1, Theorem 2, and Theorem 3 all fail. Indeed, there can not even be any effective *semi-decision* procedure that will check given a schema with first-order constraints and a CQ Q, whether Q has a plan (RA, USPJ, etc.) over finite instances. Thus the problem here is intrinsic to the hardness of reformulating queries in the presence of first-order logic constraints over finite structures, not specific to our approach via proofs. This follows easily from the fact that the valid first-order sentences can not be computably enumerated.

If we restrict our constraints to "finitely controllable" fragments (e.g. Guarded TGDs, Guarded Fragment), then we can regain completeness in the finite. For example, if the constraints of a schema are inclusion dependencies and our CQs are boolean, then Theorem 1–3 all hold when only finite instances are considered, and the method of plans-from-proofs will always generate a plan for any query that has a plan over finite instances. The key observation here is that when the constraints of S_0 are in these finitely-controllable fragments, then $\mathsf{AcSch}(S_0)$ is as well.

Decidability and complexity. The correspondences in Theorem 1, Theorem 2, and Theorem 3 deal with arbitrary first-order constraints, where both existence of a proof and the existence of a plan are undecidable. But since access interpolation can be done effectively, it follows that for any subclass S of first-order logic, we can decide if a plan exists whenever containment of CQs is decidable in the corresponding class of accessible schemas. Using this we can get decidability for "tame" integrity constraint classes studied in the literature. For example, for Guarded TGDs, which we will deal with later on in the paper, the derived schema $\mathsf{AcSch}^{\leftrightarrow}$ also consists only of Guarded TGDs, and a 2EXPTIME upper bound follows from bounds on querying with respect to Guarded TGDs. On the other hand, a reduction from query answering can be used to show that when constraints are Guarded TGDs deciding if a plan exists is 2EXPTIME-hard.

4. PLANS FROM TGD PROOFS

The previous results give a correspondence between entailment of $\mathsf{InferredAcc}Q$ from Q and the existence of a query that abides by the access restrictions. We will now focus on making this transformation from proofs more concrete, and also making the corresponding proof search more practical, restricting our attention now to constraints in the form of TGDs. For TGDs we can make use of a "forward-chaining" proof system known in the database literature as *the chase*. A proof can be rephrased as a sequence of database instances, beginning with the *canonical database* of query Q: the database whose elements are the constants of Q plus copies c_1 of each variable x_1 in Q and which has a fact $R(c_1 \dots c_n)$ for each atom $R(x_1 \dots x_n)$ of Q. These databases evolve by *firing rules*. Given a set of facts I and a TGD $\delta = \forall x_1 \dots x_j \varphi(\mathbf{x}) \rightarrow \exists y_1 \dots y_k \rho(\mathbf{x}, \mathbf{y})$ a *candidate match for* δ is a $\mathbf{e}$ such that $\varphi(\mathbf{e})$ holds but there is no $\mathbf{f}$ such that $\rho(\mathbf{e}, \mathbf{f})$ holds in I. A *rule firing* for this candidate match adds facts to I that make $\rho(\mathbf{e}, \mathbf{f})$ true, where $f_1 \dots f_k$ are new constants ("chase constants").

A *chase sequence* following a set of dependencies Σ consists of a sequence of instances $F_i : 1 \leq i \leq n$, where F_{i+1} is obtained from F_i by some rule firing of a dependency in Σ. Thus each $1 \leq i \leq n$ is associated to a set of facts F_i (which we sometimes refer to as a *chase configuration*), to a rule firing, and to a set of *newly-generated facts* – the ones produced by the last rule firing. A homomorphism of a query Q' into the configuration of a chase sequence is called a *match* for Q' in the configuration.

We now have the following well-known result: for any conjunctive queries Q and Q' (with the same free variables), and any TGD constraints Σ, Q entails Q' w.r.t. Σ iff there is a chase sequence following Σ beginning with the canonical database of Q, leading to a configuration that has a match for Q', mapping the free variables of Q' to the same constants corresponding to the free variables of Q. In particular Q entails $\mathsf{InferredAcc}Q$ exactly when there is a chase sequence beginning with the canonical database of Q leading to an element that has a match for $\mathsf{InferredAcc}Q$.

We know from the previous section that for schemas S_0 with arbitrary first-order constraints, applying interpolation to a proof of $\mathsf{InferredAcc}Q$ from Q using $\mathsf{AcSch}(S_0)$ gives a query that can be converted to a USPJ-plan for Q. We will show that when S_0 has only TGDs, and we use forward-chaining proofs as our proof system, we can generate SPJ-plans directly from a proof using $\mathsf{AcSch}(S_0)$.

Given a chase sequence, let C_∞ be the set of chase constants generated by firings of original constraints of S_0 within this sequence. Our plans will make use of temporary tables T_j whose attributes correspond to a subset C_j of C_∞; informally, rows of these tables will store possible homomorphisms that map the chase constants into the instance being queried. The C_j will be monotonic in j under inclusion as j increases.

We will construct the commands in the plan by induction on the number of rule firings of an accessibility axiom in the chase sequence.

We will maintain as an invariant that the set of attributes C_j are exactly the set of constants $c \in C_\infty$ such that $\mathsf{accessible}(c)$ holds in the configuration of the last element of the sequence. We will also restrict to *eager proofs*: those which do not have a firing at some step i of an accessibility axiom, and then at a later step a rule firing involving the initial integrity constraints or their copies on the relations $\mathsf{InferredAcc}R$ that was already applicable at step i. Informally, in eager proofs, we always perform "cost-free rules" before we perform a rule firing that corresponds to an access. It is clear that any proof can be turned into an eager proof by re-arranging the proof steps.

In the induction step, we consider an eager chase sequence ending with the firing of a rule:

$$\mathsf{accessible}(c_{j_1}) \wedge \dots \mathsf{accessible}(c_{j_m}) \wedge R(c_1 \dots c_n) \\ \rightarrow \mathsf{Accessed}R(c_1 \dots c_n)$$

associated with method mt on relation R having input positions $j_1 \cdots j_m$. Let v_{j-1} be the chase configuration prior to the firing of this rule. Note that by the inductive invariant, each c_{j_i} must be an attribute of table T_{j-1} associated to the sequence prior to the firing. We define the *commands that correspond to this rule firing*, denoted $\mathsf{Comms}(v_{j-1}, R(c_1 \dots c_n), \mathsf{mt})$. We will focus on the case where no c_{j_i} are schema constants, no constant is repeated in $R(c_1 \dots c_n)$, and $R(c_1 \dots c_n)$ is the unique R-fact of v_{j-1} that has c_{j_i} at position j_i for each $i \leq m$; we defer the additional cases to the full paper. We first generate an access command whose input expression is the projection of T_{j-1} onto $c_{j_1} \dots c_{j_m}$, with the input mapping b_{in} taking column c_{j_i} of T_{j-1} to input position j_i of mt. The command's output relation will be a table T_j with attributes $C_j = C_{j-1} \cup \{c_1 \dots c_n\}$. We follow the access command by a middleware query command that sets T_j to the join of itself with T_{j-1}, again using the mapping associating the i^{th} position in an output tuple with the attribute c_i in T_j.

Let C_{ret} be the set of chase constants c corresponding to the free variables of Q. If the configuration of the element v has a match for $\mathsf{InferredAcc}Q$ in its configuration, we will add a query that will set a final table T_{fin} to the projection of T_j on C_{ret}. In the special case that Q is boolean, the final query amounts to checking that the table T_j is non-empty.

EXAMPLE 4. *Consider the same schema as in Example 1. Let* $Q = \exists \mathsf{eid\ onum\ lname\ Profinfo}(\mathsf{eid}, \mathsf{onum}, \mathsf{lname})$. *Using the chase, we get the following* **proof***:*

1. *Create the canonical database, which in this case contains the single fact* $\mathsf{Profinfo}(\mathsf{eid}_0, \mathsf{onum}_0, \mathsf{lname}_0)$
2. *One of the initial integrity constraints matches* $\mathsf{Profinfo}(\mathsf{eid}_0, \mathsf{onum}_0, \mathsf{lname}_0)$*, and by firing the rule, we derive* $\mathsf{Udirect}(\mathsf{eid}_0, \mathsf{lname}_0)$.
3. $\mathsf{Udirect}(\mathsf{eid}_0, \mathsf{lname}_0)$ *matches an accessibility axiom, and the rule firing generates* $\mathsf{AccessedUdirect}(\mathsf{eid}_0, \mathsf{lname}_0)$*, which in turn generates* $\mathsf{InferredAccUdirect}(\mathsf{eid}_0, \mathsf{lname}_0)$ *and* $\mathsf{accessible}(eid_0)$.
4. *An accessibility axiom matches* $\mathsf{Profinfo}(\mathsf{eid}_0, \mathsf{onum}_0, \mathsf{lname}_0) \wedge \mathsf{accessible}(\mathsf{eid}_0)$*, creating the fact* $\mathsf{AccessedProfinfo}(\mathsf{eid}_0, \mathsf{onum}_0, \mathsf{lname}_0)$*, which in turn generates* $\mathsf{InferredAccProfinfo}(\mathsf{eid}_0, \mathsf{onum}_0, \mathsf{lname}_0)$.
5. *We now have a match for* $\mathsf{InferredAcc}Q$*, so we have a successful sequence.*

Here is the generated **plan***:*

1. *The firing of the accessibility axiom on the third line above generates access command* $T_1 \Leftarrow \mathsf{mt}_{\mathsf{Udirect}} \Leftarrow \emptyset$*, where* T_1 *is a table with attributes for* eid_0 *and* lname_0.
2. *The accessibility axiom on the fourth line generates commands* $T_2 \Leftarrow \mathsf{mt}_{\mathsf{Profinfo}} \Leftarrow T_1$ *and* $T_2 := T_2 \bowtie T_1$.
3. *The match at the end generates the command output* $\pi_\emptyset(T_2)$*, which returns non-empty if* T_2 *is non-empty.*

That is, we do an input-free access on $\mathsf{Udirect}$ *and put all the results into* $\mathsf{Profinfo}$.

The following theorem shows the soundness of this approach to generating plans from proofs:

THEOREM 5. *For every chase sequence proving* $\mathsf{InferredAcc}Q$ *from conjunctive query* Q *using the rules above, the corresponding SPJ-plan produced by the translation above answers* Q.

Thus every query that is completely answerable can be answered by a proof-based plan. We want to emphasize that this approach does not depend on any acyclicity condition on the constraints – thus the set of possible chase sequences can be infinite.

The proof, deferred to the full paper, proceeds by showing an invariant on the intermediate chase configurations F_j and associated partial plans PL_j produced by the algorithm, each of which outputs a temporary table T_j. Focusing on the case where Q is boolean, let $\mathsf{Accessed}(F_j)$ be the conjunctive query formed by taking the conjunction of all facts of the form $\mathsf{InferredAcc}R(\mathbf{c})$ in F_j and turning them into an existentially quantified conjunction of facts $R(\mathbf{w})$, changing the chase constants c that satisfy $\mathsf{accessible}(c)$ to free variables and the other chase constants to existentially quantified variables. Note that if F_j has a match for $\mathsf{InferredAcc}Q$, then $\mathsf{Accessed}F_j$ entails Q. The attributes C_j of T_j will be all chase constants such that the relation $\mathsf{accessible}$ holds in F_j – hence these match the free variables of $\mathsf{Accessed}(F_j)$. Let $T_j(I)$ be the instance of table T_j produced by PL_j when run on an instance I of schema S_0. The key invariant is that the following holds for any instance I of the schema:

1. If Q returns a non-empty result on I, then $T_j(I)$ is non-empty.
2. $T_j(I)$ is a subset of the tuples in $\mathsf{Accessed}(F_j)(I)$.

This implies the theorem, since on the final configuration, $\mathsf{Accessed}(F_j)$ entails Q, as noted above. The two assertions above can be thought of as saying that PL_j interpolates between Q and $\mathsf{Accessed}(F_j)$, in the sense of the Craig Interpolation Theorem.

Decidability, the case of views, and finite instances. For general TGDs, one cannot decide the existence of a proof or a plan, just as for general FO constraints: thus there is no advantage in "worst-case effectiveness" by restricting to TGDs. But whenever the class of accessible schemas is tame enough, we get decidability. For many restricted classes the chase on the accessible schema *terminates* (see, [?] for a survey) – there is a point after which no rules can add new facts. One important case is where the access restrictions in schema S_0 are given by a set of view relations which are fully accessible, with the constraints merely relating each view relation V_i to a conjunctive query Q_i that defines it. In this case the TGDs in the generated schema $\mathsf{AcSch}(S_0)$ will terminate after polynomially many steps.

Thus we can search for a chase-based proof to decide if Q can be conjunctively reformulated over the views. One can also show that whenever the chase terminates in the schema $\mathsf{AcSch}(S_0)$, our technique for determining a plan is complete for *finite instances*. Thus in particular, we have the following corollary, which implies the seminal result of Levy et. al. [?] on finding conjunctive reformulations over a set of views:

THEOREM 6. *Let schema* S_0 *have TGD constraints stating that each view relations* V_i *is equivalent to the result of a conjunctive query* Q_i *over some base signature* B*, for* $i \leq k$*. Then for any conjunctive query* Q *over* B*, we can determine whether or not* Q *can be rewritten as a conjunctive query over* $V_i : i \leq n$ *(over finite instances, equivalently over all instances) by performing the chase on* Q *using* $\mathsf{AcSch}(S_0)$ *until it terminates, and then checking* $\mathsf{InferredAcc}(Q)$ *on the result.*

RA-plans for schemas with TGDs. The proof-to-plan algorithm above focused on generating SPJ-plans. It is known [15] that there are conjunctive queries that have rewritings over a set of views defined by conjunctive queries, but the rewritings require the relational difference operator. From this it follows that there are schemas S_0 consisting of access restrictions and TGDs and conjunctive queries Q that have RA-plans but no SPJ-plans. From Theorem 2, we know that Q has an RA-plan with respect to S_0 iff Q entails $\mathsf{InferredAcc}Q$ with respect to $\mathsf{AcSch}^{\leftrightarrow}(S_0)$. We now give a slight extension of the prior algorithm to read off a RA-plan from a chase proof using the rules of $\mathsf{AcSch}^{\leftrightarrow}(S_0)$. For convenience we assume that our queries are boolean, our constraints contain no schema constants, and that our queries and constraints contain no repeated variables in atoms – thus the chase proofs will not produce any configurations that contain such facts.

Algorithm Description. The algorithm proceeds by backward induction on the size of the proof. We group proofs into

- the firing of integrity constraints from the schema or their copies on the $\mathsf{InferredAcc}R$ relations are fired.
- the firing of rules $R(\mathbf{x}) \wedge \bigwedge_i \mathsf{accessible}(x_{j_i}) \rightarrow \mathsf{Accessed}R(\mathbf{x})$, where there is at least one method mt with input $j_1 \dots j_m$ on relation R. We assume these are immediately followed by the corresponding firing of the rule $\mathsf{Accessed}R(\mathbf{x}) \rightarrow \mathsf{InferredAcc}R(\mathbf{x})$, and consider this to be a single step. If such a rule is applied to $R(\mathbf{c})$ we refer to this as "a positive accessibility axiom firing exposing fact $R(\mathbf{c})$".
- the firing of rules $\mathsf{InferredAcc}R(\mathbf{x}) \wedge \bigwedge_i \mathsf{accessible}(x_{j_i}) \rightarrow \mathsf{Accessed}R(\mathbf{x})$, which we assume are immediately followed by the corresponding firing of the rule $\mathsf{Accessed}R(\mathbf{x}) \rightarrow R(\mathbf{x})$. If such a rule is applied to $R(\mathbf{c})$ we refer to this as "a negative accessibility axiom firing and exposing fact $\mathsf{InferredAcc}R(\mathbf{c})$".

The algorithm takes as input a proof beginning with some configuration C_i and produces an executable FO query $P_i(\mathbf{x})$, where $\mathbf{x}$ are variables indexed by the accessible constants in C_i. If the proof is trivial (only one configuration), the algorithm returns a plan that always returns true. Otherwise the algorithm analyzes the first rule firing in the proof.

- No commands are generated by rules of the first type above, so the algorithm just proceeds to the remaining rules.
- We consider a proof $C_i \dots$ where the transition from C_i to C_{i+1} is formed via a positive accessibility axiom firing exposing fact $R(\mathbf{c})$. We generate the executable query that does an access to R using the projection of $\mathbf{x}$ to the chase constants $c_{j_1} \dots c_{j_m}$, then returns true only if for some tuple $\mathbf{w}$ in the result, $\mathbf{w}$ joins with $\mathbf{x}$ to give $\mathbf{u}$ and $P_{i+1}(\mathbf{u})$ returns true.

- We now consider a proof $C_i \ldots$ where the transition from C_i to C_{i+1} is formed via a negative accessibility axiom firing exposing fact $\mathsf{InferredAcc}R(\mathbf{c})$. We generate an executable rewriting that does an access to R with the projection of $\mathbf{x}$ to the chase constants $c_{j_1} \ldots c_{j_m}$, and returns true only if, for *every* tuple $\mathbf{w}$ in the result of the access that joins with $\mathbf{x}$ giving joined tuple $\mathbf{u}$, $P_{i+1}(\mathbf{u})$ returns true.

The reader should note that this algorithm is extremely close to the one given in the view case by Nash, Segoufin, and Vianu (page 21:29 of [15]). But the proof of correctness introduces several new subtleties: one needs a much more complex invariant than that used in the proof of Theorem 5.

In the case of views given by conjunctive queries, the chase using the constraints in $\mathsf{AcSch}^{\leftrightarrow}(S_0)$ does not necessarily converge, since facts propagate in both directions. Indeed, the question of deciding whether a query can be reformulated using a relational algebra query over a set of views is open.

Also note that the algorithm will produce a *USPJ*$^{\neg}$ plan in the case where the proof used only accessibility axioms in the restricted schema $\mathsf{AcSch}^{\neg}(S_0)$. Hence we have:

THEOREM 7. *For any schema S_0 using TGDs and CQ Q, for every chase proof using* $\mathsf{AcSch}^{\leftrightarrow}(S_0)$*, the algorithm above produces an RA-plan that completely answers Q. If the proof uses only rules in* $\mathsf{AcSch}^{\neg}(S_0)$*, the result is a USPJ$^{\neg}$-plan.*

5. LOW-COST PLANS VIA PROOFS

We now look at finding efficient plans, focusing for the remainder of the paper on generating SPJ-plans with respect to schemas consisting of TGDs, and letting the function Plan be the one described using AcSch in the beginning of the previous section. We first note that the proof-based plans that were generated by the SPJ algorithm are as access-efficient as arbitrary plans, and thus we can focus on these.

A plan PL *makes fewer accesses* than plan PL' if for every pair consisting of a method mt and method input $\mathbf{t}$ that is executed in running PL on instance I of the schema, the same pair is also accessed in running PL' on I. Thus "fewer" means "no more than". The following result captures the claim proof-based plans are no more costly than general plans; it is proven by first finding a proof-based plan PL' that mimics the given plan PL on the chase, then using universality properties of the chase to argue that PL' behaves as well as PL on all instances.

THEOREM 8. *For conjunctive query Q and schema with TGD constraints Σ and access restrictions, for every SPJ-plan* Plan *that answers Q, there is a chase sequence v proving* $\mathsf{InferredAcc}Q$ *from Q, such that* $\mathsf{Plan}(v)$ *makes fewer access than* Plan.

Note that this theorem does not imply anything about the cost of proof-based plans versus arbitrary plans according to particular cost functions, since cost functions look at plans statically, and are thus not necessarily monotone in the set of (method, input) pairs produced at runtime. For example, what we call simple cost functions are based on the set of access commands (that is, bulk accesses) that are performed.

The proof works by taking a plan PL and constructing a chase proof that mimics its behavior, in terms of accesses that are made and facts exposed, when applied to the canonical database for the input query Q. This plan is constructed inductively, firing one accessibility axiom at a time until all the facts exposed by PL are present in the chase proof. We then argue, using the universality of the chase, that the plan generated from this proof will make fewer access than PL on arbitrary inputs.

Adding cost to plan search. Theorem 8 implies that the plans produced from proof-to-plan algorithms are optimal in a certain sense. Moreover the SPJ algorithm of Section 4 generates physical plans directly, rather than going via queries, with the structure of the plans directly reflecting the structure of the firing of accessibility axioms. We can thus apply a plan cost function to partial plans while searching for a proof, thus merging the proof search with the search for a low cost plan. This is the last main idea of the paper: *we can find low-cost plans by exploring the space of proofs*.

Our search will maintain a *partial proof tree* – a tree consisting of chase sequences, ordered by extension. We refer to the configuration (set of facts) of the final element in the chase sequence associated with a node v as $\mathsf{config}(v)$. The plan associated with v is the one generated by the proof-to-plan algorithm given previously, while by the cost of v we mean the cost of the associated plan. We now give an algorithm for extending the tree to find new proofs.

For node v if there is a fact $R(c_1 \ldots c_m)$ in $\mathsf{config}(v)$ with $\mathsf{Accessed}R(c_1 \ldots c_m)$ not yet in $\mathsf{config}(v)$ and there is an access method mt on R with input positions $j_1 \ldots j_m$ such that $\mathsf{accessible}(c_{j_1}) \ldots \mathsf{accessible}(c_{j_m})$ all hold in $\mathsf{config}(v)$, then we call $R(c_1 \ldots c_m)$ a *candidate for exposure* at v, and mt an *exposing method for* $R(c_1 \ldots c_m)$. Note that if a fact is a candidate for exposure, then firing an accessibility axiom will add that fact to the associated chase sequence.

When we explore the impact of making an access, we want to include all relevant consequences that do not involve further accesses, thus producing an eager proof (as defined in Section 4). This corresponds to the following requirements on the configurations in a partial proof tree:

- (Original Schema Reasoning First) The configuration of the root node (henceforward "initial configuration") corresponds to the canonical instance of Q plus the result of firing integrity constraints of the original schema S_0 until a termination condition is reached – the termination condition will be explained further below.
- (Fire Inferred Accessible Rules Immediately) For a non-root node v, there is a candidate fact for exposure $R(c_1 \ldots c_m)$ in its parent with exposing method mt such that $\mathsf{config}(v)$ is obtained from the parent by
  - adding the *facts induced by firing* mt *with* $c_{j_1} \ldots c_{j_m}$ – that is, all facts $\mathsf{Accessed}R(d_1 \ldots d_m)$ such that $R(d_1 \ldots d_m)$ is in the parent configuration and $\mathbf{d}$ agrees with $\mathbf{c}$ on the input positions of mt. Note that there may be several such facts, but they will include $R(c_1 \ldots c_m)$.
  - firing inferred accessible axioms on the result until some termination condition is reached.

Thus the successive configurations are connected by firing a rule associated with an accessibility axiom, firing additional accessibility rules corresponding to the other facts exposed by the same access and exploring the cost-free consequences. Thus we can also characterize a node v by the sequence of rule firings of accessibility axioms leading to it.

We also label a node as *successful* if $\mathsf{InferredAcc}Q$ holds in the corresponding configuration (preserving free variables in the non-boolean case).

The idea is that we have labelled each node with a configuration of the proof, and whenever we choose an access to fire, after firing we immediately fire all the relevant rules that do not generate accesses.

We explore downward from a node v of a partial proof tree by choosing a *candidate fact for exposure* at $\mathsf{config}(v)$ along with the methods that expose the fact. A node is *terminal* if it is either successful or has no candidate facts. Note that non-terminal nodes do not have to be leaves of the tree.

The basic search structure is outlined in Algorithm 1. At each iteration of the while loop at line 5 we have a partial proof/plan tree satisfying the properties above. We look for a node v corresponding to a partial proof that is not yet successful, has not yet exhausted the maximum number of accesses we allow, and for which the firings of accessibility axioms can add new facts. We non-deterministically choose such a path and such a rule (lines 6-7), and calculate both the new configuration that comes from firing the rule, the commands that will be added to the corresponding plan, and the cost via a call to the "blackbox" cost function, denoted $\mathsf{AtomicCost}$ (lines 8-9). We update the candidate list (line 11) and determine whether the new path is successful, recording whether this gives the new lowest cost plan (lines 12-15).

Search order and termination conditions. The non-deterministic algorithm above leaves open a number of issues. The first is how the non-terminal node is chosen. Our policy is to do this *depth-first*: always pick the leaf of the leftmost branch (where left is defined using some ordering on facts) as long as it does not go past a

Algorithm 1: generic search

```
Input:  query $Q$, schema $S$, depth $d$
Output: plan BestPlan
1  ProofTree := an initial node $v_0$ labelled with the configuration
   obtained by firing original integrity constraint rules up to
   termination condition.
2  Set Candidates($v_0$) = all pairs $(R(c_1 \dots c_n), \mathsf{mt})$, $R(c_1 \dots c_n)$ a
   fact in the original configuration, mt a method on $R$.
3  BestPlan := $\perp$
4  BestCost := $\infty$
5  while there is a non-terminal node v at depth at most d in
   ProofTree do
6      Choose such a node $v$.
7      Choose a candidate fact and method
       $(R(c_1 \dots c_n), \mathsf{mt}) \in$ Candidates($v$) with
       accessible($c_{j_1}$)...accessible($c_{j_m}$) $\in$ config($v$) and mt
       having inputs $j_1 \dots j_m$.
8      Add a new node $v'$ as a child of $v$ with configuration formed
       by adding all the accessible facts induced by exposing
       $R(c_1 \dots c_n)$ with mt and then closing under sufficiently many
       firings of the "inferred accessible rules".
9      Set Cost(Plan($v'$)) using call to AtomicCost.
10     Remove $(R(d_1 \dots d_n), \mathsf{mt})$ with $\mathbf{d}$ extending $c_{j_1} \dots c_{j_m}$ from
       Candidates($v$), marking $v$ as terminal if it has no more
       candidates.
11     Determine if $v'$ is successful by checking if InferredAcc$Q$
       holds, and if so also mark it as terminal.
12     if $v'$ is successful and Cost(Plan($v'$)) < BestCost then
13         BestPlan := Plan($v'$)
14         BestCost := Cost(Plan($v'$))
15 return BestPlan;
```

threshold d on access commands that is assumed to be provided as an input. In this way we explore the paths with the most accesses, which maximizes our chances of finding a match. The second question is which candidate fact to choose when there is more than one at a node. One policy chooses a candidate node of minimal derivation depth, where the derivation depth of a fact represents the number of rule firings needed to generate in it within this sequence– that is, its depth in the dependency graph associated with the chase sequence. Finally, we must determine the order with which we choose the exposure method mt for relations where there is more than one method. Here we assume some fixed priority for the methods – e.g. based on some notion of expected cost.

Algorithm 1 describes firing "all" rules that involve only reasoning with constraints – but such rules can fire a large number of times, even infinitely often for cyclic collections of referential constraints. For Guarded TGDs, we do not require any chase termination condition on our constraints, but instead rely on a "local blocking condition" for safely terminating such rules, a variant on the technique used within theorem-proving for guarded sublogics of first-order logic (see, e.g. [10, 13]). We organize every configuration into a tree of "guarded bags" – sets of facts B such that there is an atom $R(\mathbf{c})$ containing every chase constant appearing in B. We consider only rule firings that match within a single bag B. Any fact generated from this firing that contains a fresh constant will be added into a new child B$'$ of B, while facts $F(\mathbf{c})$ containing only constants of B will be both added to B$'$ and propagated back up the tree, added recursively to any other elements that contain all elements of $\mathbf{c}$. We abort the rule firing if the generated bag B$'$ is "blocked" by a previously existing B$''$ – that is, there is a homomorphism h of B$'$ into B$''$ such that for every query Q' based on quantifying a subset of the conjuncts of InferredAcc$Q$, if Q' is satisfied in the configuration by constants $\mathbf{c}$ of B$'$, then it is satisfied by $h(\mathbf{c})$ in the configuration. We refresh the set of rule firings that need to be considered whenever the state (facts and subqueries of InferredAcc$Q$) of the parent bag B changes.

The blocking condition guarantees that any rule firing that occurs in B$'$ would have also occurred in B$''$, and that these firings will lead to a match for some $\mathbf{c}$ in B$'$ iff they lead to a match for $h(\mathbf{c})$ in B$''$. The approach is very naive compared to the optimized blocking strategies available in the description logic community (which study logics incomparable in expressiveness with Guarded TGDs). But even this simple version suffices to guarantee termination, since a bound on the number of guarded bags implies a bound on the depth of a path with no blocked nodes.

The algorithm given before can be applied to any cost function. But we will need assumptions on the cost to prove that it obtains the optimal plan. In this work, we state an optimality result only for simple cost functions:

THEOREM 9. *For all schemas S consisting of access restrictions and Guarded TGDs, for all simple cost functions, for all conjunctive queries Q, and for all numbers d, Algorithm 1 will always return a plan with the lowest cost, among all those SPJ plans that completely answer Q w.r.t. S and which make at most d access commands.*

Note that one important class of constraints, those generated from view definitions, are generally not expressible as Guarded TGDs. But for the constraints generated via view definitions over CQs, the chase on AcSch(S_0) will terminate quickly (see comments before Theorem 6). Thus we can avoid using blocking in this case.

Optimizations. The prior algorithm performs exhaustive search of proofs up to some level. We defer a discussion of a more realistic implementation to a later paper, but make a few observations about pruning the search space.

Notions of reducing the search space must consider at *proof structure* and *cost analysis*, both individually and in combination. Looking at proof structure in isolation, we should prune paths that cannot lead to a valid proof while preferring ones that are more likely to lead to a proof. As an example of an optimization related only to cost, we always assume monotonicity of cost functions, and exploit it by aborting exploration of a node if the corresponding partial plan has cost that is already worse than the cost of a known successful plan.

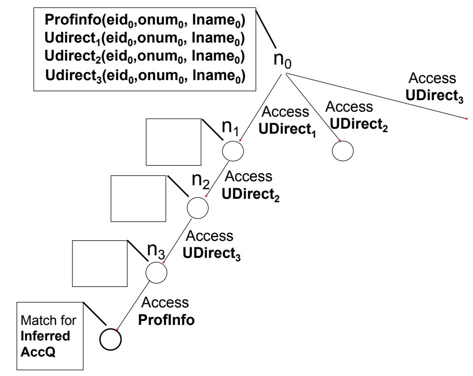
As an example of the interaction of proof structure and cost analysis, we will wish to abort the search below a node if it is "worse than" another node in the search tree. Consider the case where we are at a node v in the search space, and have a candidate fact c at v and method mt for exposing c, such that when we generate a new node v' from c we find that there is a node v'' already in ProofTree such that config(v'') has "at least as many useful facts" as config(v') and we know that Plan(v'') is "at least as efficient" than Plan(v'). Then there is no need to generate v', since if a sequence of further accesses added on to the actions of v' generates a complete plan, the same sequence will generate a complete plan with no higher cost when added on to the actions of v''. The notion "at least as many useful facts" can be formalized via the existence of a mapping from chase constants of v' to those of v'' that preserves facts over relations of the original schema and those of the form InferredAcc$R$. For simple cost functions, the notion of efficiency is captured by the notion of having lower cost. For general cost functions the notion of "worse plan" must be more complex, since v' might produce some temporary relations that are smaller than those of v'', and the size of these relations may diminish the cost of later accesses. In our follow-up work, we investigate heuristic notions of comparison for more general cost-functions, and their relationship with notions used in traditional query optimization.

EXAMPLE 5. *Let us return to the setting of Example 1, assuming we have 3 directory sources* Udirect$_1$, Udirect$_2$, Udirect$_3$. *The integrity constraints contain:*

$$\mathsf{Profinfo}(\mathsf{eid}, \mathsf{onum}, \mathsf{lname}) \rightarrow \mathsf{Udirect}_i(\mathsf{eid}, \mathsf{lname})$$

for $i = 1, 2, 3$, *with* Profinfo *having an access that requires all arguments to be given and each* Udirect$_i$ *having unrestricted access. Consider the query* $Q = \exists$eid onum lname Profinfo(eid, onum, lname).

Figure 1 illustrates the exploration. The canonical database of Q consists of the fact Profinfo(eid$_0$, onum$_0$, lname$_0$). *The proof configuration of the initial node* n_0 *will then add* Udirect$_i$(eid$_0$, lname$_0$) *for*



Profinfo(eid$_0$,onum$_0$, lname$_0$)
Udirect$_1$(eid$_0$,onum$_0$, lname$_0$)
Udirect$_2$(eid$_0$,onum$_0$, lname$_0$)
Udirect$_3$(eid$_0$,onum$_0$, lname$_0$)
n$_0$
Access UDirect$_3$
Access UDirect$_1$
Access UDirect$_2$
n$_1$
n$_2$
Access UDirect$_2$
n$_3$
Access UDirect$_3$
Access ProfInfo
Match for Inferred AccQ

Figure 1: Exploration in the running example

$i = 1,2,3$. *There are thus three candidates facts to expose, corresponding to firing accessibility axioms that expose* $\mathsf{Udirect}_i(\mathsf{eid}_0, \mathsf{lname}_0)$: $i = 1,2,3$ *in the initial node.*

Assuming that we have a heuristic that tells us to prefer to explore access to $\mathsf{Udirect}_i$ *before access to* $\mathsf{Udirect}_j$ *for* $i < j$, *we will choose fact* $\mathsf{Udirect}_1(\mathsf{eid}_0, \mathsf{lname}_0)$ *to expose first. This creates a child proof configuration* n_1. *The transition from parent to child is associated with a plan command performing input-free access to* $\mathsf{Udirect}_1$, *putting the output into a table* T_1 *with attributes* $\{\mathsf{eid}_0, \mathsf{lname}_0\}$. *The associated proof configuration for* n_1 *(shown as a box to the upper left) adds the exposed fact* $\mathsf{AccessedUdirect}_1(\mathsf{eid}_0, \mathsf{lname}_0)$, *and then immediately the fact* $\mathsf{InferredAccUdirect}_1(\mathsf{eid}_0, \mathsf{lname}_0)$.

In n_1 *there are three candidate facts to expose via accessibility axioms:* $\mathsf{Udirect}_2(\mathsf{eid}_0, \mathsf{lname}_0), \mathsf{Udirect}_3(\mathsf{eid}_0, \mathsf{lname}_0)$ *and now also* $\mathsf{Profinfo}(\mathsf{eid}_0, \mathsf{onum}_0, \mathsf{lname}_0)$, *since there is an accessibility axiom that would expose this last fact now. The highest priority one is* $\mathsf{Udirect}_2(\mathsf{eid}_0, \mathsf{lname}_0)$. *Thus a child* n_2 *will be generated (again including the exposed fact* $\mathsf{AccessedUdirect}_2(c_0)$ *and one inferred fact). The transition to* n_2 *will be associated with an input-free access command on* $\mathsf{Udirect}_2$ *and a command joining the results with the previous table.*

The node n_2 *will have the facts* $\mathsf{Udirect}_3(\mathsf{eid}_0, \mathsf{lname}_0)$ *and* $\mathsf{Profinfo}(\mathsf{eid}_0, \mathsf{onum}_0, \mathsf{lname}_0)$ *as candidates for exposure, of which* $\mathsf{Udirect}_3(\mathsf{eid}_0, \mathsf{lname}_0)$ *has highest priority. We will thus generate a child* n_3, *whose configuration adds the exposed fact and the corresponding inferred accessible fact.*

The node n_3 *will have only one candidate fact, corresponding to* $\mathsf{Profinfo}(\mathsf{eid}_0, \mathsf{onum}_0, \mathsf{lname}_0)$, *so a child* n_4 *will be generated. The access associated with the edge from* n_3 *to* n_4 *will be of the form* $T_4 \Leftarrow \mathsf{mt}_{\mathsf{Profinfo}} \Leftarrow T_3$, *where* T_3 *will be a table with attributes* $\mathsf{eid}_0, \mathsf{lname}_0$ *containing the intersection of the outputs of the 3 prior accesses. The query* $\mathsf{InferredAcc}Q$ *matches the configuration of* n_4, *so it is designated a success node, and hence is terminal.*

Now the search can go back up the tree to a node with more candidates to explore – e.g. following the "depth-first on nodes" approach, it will move to n_3, *and pick the highest priority child of* n_3 *to explore. Note that at some point in the process, the tree extension process will consider creating a node* n''' *corresponding to firing the first two axioms in the reverse order than in the path above – exposing first fact* $\mathsf{Udirect}_2(\mathsf{eid}_0, \mathsf{lname}_0)$ *then* $\mathsf{Udirect}_1(\mathsf{eid}_0, \mathsf{lname}_0)$. *This chase node would have the same configuration as the node* n_2 *above; assuming it has no larger cost,* n''' *will be determined to be "no better than"* n_2, *and hence would not be generated.*

6. CONCLUSIONS AND FUTURE WORK

The main goal of this work is to introduce a means for generating query plans from "proofs of answerability of a query". By exploring many proofs, one can guide the search for good query plans by the structure of proofs. The technique is particularly useful in the presence of rich integrity constraints, which cannot be exploited by traditional query planners. The further presence of access restrictions can make the use of constraints essential for finding any plan, and crucial for finding a good plan.

We have stressed the continuity between the general technique based on interpolation and the direct algorithms for TGDs based on the chase. We have implemented a proof-to-plan generator in the setting of forward-chaining proofs. While we gave a flavor of the system here, an overview of the prototype and a full description of the system are in preparation. But note that proof systems exist that are very different from forward-chaining ones, such as those used in proving the Access interpolation theorem – e.g. tableaux,backward-chaining systems, saturation-based procedures that compute all provable sequences of a given form. We wish to investigate how the correspondence between proofs and plans works for each of these proof schemes.

7. REFERENCES

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

[2] C. Areces, P. Blackburn, and M. Marx. Hybrid logic: Characterization, interpolation and complexity. *J.Symb. Log.*, 66(3):977–1010, 2001.

[3] V. Bárány, M. Benedikt, and P. Bourhis. Access restrictions and integrity constraints revisited. In *ICDT*, 2013.

[4] P. Blackburn and M. Marx. Constructive interpolation in hybrid logic. *J. Symb. Log.*, 68(2):463–480, 2003.

[5] A. Deutsch, B. Ludäscher, and A. Nash. Rewriting queries using views with access patterns under integrity constraints. *TCS*, 371(3):200–226, 2007.

[6] A. Deutsch, L. Popa, and V. Tannen. Physical data independence, constraints, and optimization with universal plans. In *VLDB*, 1999.

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

[8] O. Duschka, M. Genesereth, and A. Levy. Recursive query plans for data integration. *The Journal of Logic Programming*, 43(1):49 – 73, 2000.

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

[10] C. Hirsch and S. Tobies. A tableau algorithm for the clique guarded fragment. In *Adv. Modal Logic*, 2000.

[11] C. Li. Computing complete answers to queries in the presence of limited access patterns. *VLDB Journal*, 12(3):211–227, 2003.

[12] C. Li and E. Chang. Answering queries with useful bindings. *TODS*, 26(3):313–343, 2001.

[13] T. Lukasiewicz, A. Cali, and G. Gottlob. A general datalog-based framework for tractable query answering over ontologies. *J. Web Sem.*, 14(0), 2012.

[14] Roger C. Lyndon. An interpolation theorem in the predicate calculus. *Pacific J. Math.*, 9:129–142, 1959.

[15] A. Nash, L. Segoufin, and V. Vianu. Views and queries: Determinacy and rewriting. *TODS*, 35(3), 2010.

[16] M. Otto. An interpolation theorem. *B. Symb. Log.*, 6(4):447–462, 2000.

[17] L. Popa. *Object/Relational Query Optimization with Chase and Backchase*. PhD thesis, U. Penn., 2000.

[18] D. Toman and G. Weddell. *Fundamentals of Physical Design and Query Compilation*. Morgan Claypool, 2011.