Chapter 14

Interpolation and Query Rewriting

Michael Benedikt
University of Oxford, UK

Abstract

We overview applications of Craig interpolation and Beth definability to simplifying logical expressions or database queries. From the perspective of the theory of interpolation and definability the results give a number of new angles. First, they give a different take on what it means to make definability or interpolation results effective, looking at algorithms that take a proof as input and return an interpolant or explicit definition as output. Secondly, they relate interpolation and definability to preservation theorems in model theory: interpolation and definability theorems are the basis for many 'semantics-to-syntax", relating a semantic property of a formula to its equivalence with a certain syntactic form. Thirdly, they motivate new forms of interpolation and definability, focusing on syntactic forms that are of interest in databases.

Contents

In Balder ten Cate, Jean Christoph Jung, Patrick Koopmann, Christoph Wernhard, and Frank Wolter, editors. *Theory and Applications of Craig Interpolation*. Ubiquity Press, 2026. To appear.
https://www.doi.org/10.5334/bdg.n

1 Overview

There are a number of applications of Craig interpolation in data management, but this chapter will focus on one type of application, to *rewriting queries*. The idea is that we want to translate a *declarative source query* into a declarative query or procedural plan in a *target language* that abides by certain *interface restrictions*. By a declarative source query we will usually mean a request for information from a database, where the request is specified using a formula of first-order logic, or its equivalent in the database query language SQL. We often focus on source queries given in the language of *conjunctive queries*, which corresponds to a very simple subset of SQL. In logical terms these correspond to formulas built up using only conjunction and existential quantification. By a plan in a target language we may mean either another declarative query, or something more operational, such as a relational algebra expression, possibly using some restricted set of physical operators.

Translating from a declarative source query to a target in this broad sense captures a huge number of activities that take place within data management. One of the basic features of modern database systems is to decouple the vocabulary employed by users to ask questions about a dataset from the data structures that are used to implement data access: the "logical model" and the "physical model". In relational databases, the logical model might be a set of tables given an SQL schema: users interact with the data by sending SQL queries that mention these tables. The database manager performs a translation of the queries written over this high-level vocabulary into a program that interacts with the data using some set of data access functions.

In this "classical query evaluation" scenario, the relationship of the source vocabulary to the target vocabulary is very simple: every source relation corresponds to one or more "physical datasource" relations or functions. A more complicated example comes from data integration systems. The goal there is to mask a diverse set of datasources by providing a single unified schema. Suppose we have a multitude of "local databases" with information about different hotel chains. A data integration system would present to users a "global schema" — an easier-to-understand vocabulary that can model information in all of the sources. The system accepts queries written over the global schema and translates them into queries over the various local sources. The "global database" is virtual, implicitly defined by its relationship with local data.

Having talked generally about what we mean by sources and targets, we mention several flavors of languages that represent targets for a rewriting algorithm. The most basic kind of restriction we look at is a *vocabulary-based restriction*. There the source is a query represented as a logical formula, and the target is also a logical formula, but we limit the target's vocabulary. We begin with a logic-based query Q written using some relations $R_1 \dots R_j$, and want to convert it to another logic-based query $Q_{\vec{V}}$ making use of a different set of relations $V_1 \dots V_k$. If the queries Q and $Q_{\vec{V}}$ mention different relations, we cannot expect Q to give the same answers as $Q_{\vec{V}}$ on arbitrary instances. But our schema will come with *integrity* *constraints* — usually given as a set of first-order sentences — which restrict the possible instances of interest. We will thus be considering equivalence only on instances satisfying the constraints.

An example of vocabulary-based restriction comes from *reformulating queries over views.* We have a collection of view relations $V_1 \dots V_k$, and each V_i is associated with a query Q_i defined using some other set of relations $R_1 \dots R_j$. Given a query Q defined over $R_1 \dots R_j$, the goal is to find an equivalent query $Q_{\vec{V}}$ that mentions only $V_1 \dots V_k$. Generally, additional restrictions will be put on $Q_{\vec{V}}$. For example, it should be a conjunctive query, or a union of conjunctive queries, or a relational algebra query. The view-based query reformulation problem can thus be seen as a special case of vocabulary-based restriction, where the integrity constraints are of the form:

$$\forall x_1 \dots \forall x_n \; V_i(x_1 \dots x_n) \leftrightarrow Q_i(x_1 \dots x_n)$$

for $1 \leq i \leq k$. Here the constraint with $\leftrightarrow$ could be rewritten to be a conjunction of two constraints with $\rightarrow$, one in each direction, and we will use this version below.

▶ **Example 1.** A university database has a table Professor containing ids and last names of professors, along with the name of the professor's department. It also has a table Student listing the id and last name of each student, along with their advisor's id.

The database does not allow users to access the Professor and Student tables directly, but instead exposes a view Professor′ where the id attribute is dropped, and a table Student′ where the advisor's id is replaced with the advisor's last name.

That is, Professor′ is a view defined by the formula:

$$\{\, \mathsf{lname},\ \mathsf{dname} \ \mid \exists \ \mathsf{profid}\ \mathsf{Professor}(\mathsf{profid},\ \mathsf{lname},\ \mathsf{dname})\}$$

or equivalently by the constraints:

$$\forall \mathsf{profid}\ \forall \mathsf{lname}\ \forall \mathsf{dname}\ \mathsf{Professor}(\mathsf{profid}, \mathsf{lname}, \mathsf{dname}) \rightarrow \mathsf{Professor}'(\mathsf{lname}, \mathsf{dname})$$
$$\forall \mathsf{lname}\ \forall \mathsf{dname}\ \mathsf{Professor}'(\mathsf{lname}, \mathsf{dname}) \rightarrow \exists \mathsf{profid}\ \mathsf{Professor}(\mathsf{profid}, \mathsf{lname}, \mathsf{dname})$$

Student′ is a view defined by the formula:

$$\{\mathsf{studid}, \mathsf{lname}, \mathsf{profname} \mid$$
$$\exists \mathsf{profid}\ \exists \mathsf{dname}\ \mathsf{Student}(\mathsf{studid}, \mathsf{lname}, \mathsf{profid}) \wedge \mathsf{Professor}(\mathsf{profid}, \mathsf{profname}, \mathsf{dname})\}$$

or equivalently by constraints:

$$\forall \mathsf{studid}\ \forall \mathsf{lname}\ \forall \mathsf{dname}\ \forall \mathsf{profid}\ \forall \mathsf{profname}\ [\mathsf{Professor}(\mathsf{profid}, \mathsf{profname}, \mathsf{dname}) \wedge$$
$$\mathsf{Student}(\mathsf{studid}, \mathsf{lname}, \mathsf{profid}) \rightarrow \mathsf{Student}'(\mathsf{studid}, \mathsf{lname}, \mathsf{profname})]$$

$$\forall \mathsf{studid}\ \forall \mathsf{lname}\ \forall \mathsf{profname}\ [\mathsf{Student}'(\mathsf{studid}, \mathsf{lname}, \mathsf{profname}) \rightarrow$$
$$\exists \mathsf{profid}\ \exists \mathsf{dname}\ \mathsf{Professor}(\mathsf{profid}, \mathsf{profname}, \mathsf{dname}) \wedge \mathsf{Student}(\mathsf{studid}, \mathsf{lname}, \mathsf{profid})]$$

Consider the query asking for the names of the advisors of a given student. We can answer this by simply using the $\mathsf{Student}'$ view, returning the profname attribute of a tuple in the view. A formula over relational expression that uses $\mathsf{Student}'$ to answer the query is called a *reformulation of the query over the views*. More generally it is an example of a *reformulation of the query over the subvocabulary* $\mathsf{Student}'$*, relatively to the background theory consisting of constraints relating* $\mathsf{Student}'$ *to the* Student.

But a query asking for the last names of all students that have an advisor in the history department can *not* be answered using the information in the views: knowing the advisor's name is not enough to identify the department.

Integrity constraints need not be restricted to expressing view definitions. A natural use of constraints is to represent *relationships between sources*, such as overlap in the data. This overlap can be exploited to take a query specified over a source that *a priori* does not have sufficient data, and reformulate it over a source that provides the necessary data. ⌟

Access Methods. A finer notion of interface than vocabulary-based restrictions is based on *access methods*, which state that a relation can only be accessed by lookups where certain arguments must be provided. One example of a restricted interface based on access methods comes from web forms. Thinking of the form as exposing a virtual table, the mandatory fields must be filled in by the user, while submitting the form returns all tuples that match the entered values. Other examples include web services and legacy databases.

▶ **Example 2.** Consider a Profinfo table available via a web form, containing information about faculty, including their last names, office number, and employee id, but with a restricted interface that requires giving an employee id as an input. The interface could be implemented as a web form that requires entering an employee's id and then pressing a submit button to get the matching records. The query Q asking for ids of faculty named "Smith" cannot be "completely answered" over this schema: that is, there is no function over the available data in this schema which is equivalent to Q.

But suppose another web form provides access to a Udirectory table containing the employee id and last name of every university employee, with an interface that allows one to access the entire contents of the table. Then we can reason that Q is answerable using the information in this schema: a plan would pull tuples from the Udirectory form and check them within the Profinfo form to see if they correspond to a faculty member. ⌟

In Example 2, reasoning about access considerations was straightforward, but in the presence of more complex schemas, we may have to chain several inferences, resulting in a plan that makes use of several auxiliary accesses.

▶ **Example 3.** Suppose we have two directory data sources with overlapping information. One source exposes information from $\mathsf{Udirectory}_1(\mathsf{uname}, \mathsf{addr}, \mathsf{uid})$ via an access method requiring a uname and uid. There is also a "web table" Ids(uid) with no access restriction, that makes available the set of uids (hence we have a "referential integrity constraint" saying that every uid in $\mathsf{Udirectory}_1$ matches a uid in Ids). The other source exposes $\mathsf{Udirectory}_2(\mathsf{uname}, \mathsf{addr}, \mathsf{phone})$, requiring a uname and addr, and also a web table $\mathsf{Names}(\mathsf{uname})$ with no access restriction that reveals all unames in $\mathsf{Udirectory}_2$ (that is, a constraint that each uname in $\mathsf{Udirectory}_2$ appears in Names). There is also a constraint saying that each uname and addr in $\mathsf{Udirectory}_2$ appears in $\mathsf{Udirectory}_1$.

A query asking for all phone numbers in the second directory could be written:

$$Q = \{\mathsf{phone} \mid \exists \mathsf{uname}\ \exists \mathsf{addr}\ \mathsf{Udirectory}_2(\mathsf{uname}, \mathsf{addr}, \mathsf{phone})\}.$$

There is a plan that implements this query: it gets all the uids from Ids and unames from Names first, puts them into the access on $\mathsf{Udirectory}_1$, then uses the uname and addr of the resulting tuples to get the phone numbers in $\mathsf{Udirectory}_2$. ⌟

We emphasize that the notion of rewriting we consider here is getting target queries which give the same answers as the original query. This means that if we have a query asking for the office number of all professors with last name “Smith”, the plan produced should return all tuples in the answer, even if access to the Professor relation is limited. This contrasts with a line of work in data integration that considers the broader question of how to answer any query “as much as possible given the available data”: how to get the *certain answers* or how to compute the *maximally contained query* [24].

Rewriting via Interpolation: the Meta-Algorithm. We will now explain the connection of rewriting to interpolation and Beth definability. There is a general template for solving a rewriting problem:

1. Isolate a *semantic property* that any input query Q must have with respect to the target $\mathcal{T}$ and constraints Σ in order to have a reformulation of the desired type.
2. Express this property as a *proof goal* (in the language we use later on, an *entailment*): a statement that a certain formula follows from another formula.
3. Search for a proof of the entailment within a given proof system. For example, one could use tableau proofs or resolution proofs, both well-known proof systems within computational logic.
4. From the proof, extract an *interpolant* using an interpolation algorithm. We will review some standard interpolation algorithms and also present new ones.
5. Convert the interpolant to a reformulation.

As we will see below, the properties of an interpolant will play a role in the ability to convert to a reformulation, and different notions of interpolation, which control how similar an interpolant is to the statements we interpolate over, will be needed for different variations of reformulation. This approach is very general and can be applied to a variety of proof systems and restrictions on the target, with different target languages corresponding to different entailments. We prove theorems saying that the method is complete: there is a reformulation exactly when there is a proof of the semantic property. These completeness theorems give as a consequence a *definability or preservation theorem*: a query Q has a certain kind of reformulation if and only if it has a certain semantic property. Such theorems are well-known in model theory, and indeed our theorems can be seen as "database versions" of the preservation theorems that are known from classical model theory textbooks, e.g. [14].

The fact that interpolation theorems can be used to prove preservation theorems is certainly not new. As mentioned earlier in this volume, William Craig used interpolation to prove Beth's definability theorem, and all of our results can be seen as generalizations of Craig's technique: see the bibliographic remarks at the end of this chapter for further details.

What we emphasize in this chapter, as in [6] is that the connection between reformulation, interpolation, and preservation can be made effective: *interpolation algorithms yield reformulation algorithms.* Actually, the interpolation technique yields many of the algorithmic results about reformulation that had appeared in the database setting previously.

2 Preliminaries

We start with some preliminaries about relational databases and logic. A good reference for this is [2].

Structures, Instances, and Range of Quantification. A *schema* is a finite set of relation symbols, each associated to a number (the arity), along with a set of constant symbols. A database *instance* I for a schema Sch assigns to every relation R in Sch of arity n a collection of n-tuples $[\![R]\!](\mathsf{I})$.

We call $[\![R]\!](I)$ the *interpretation* of R in I. An association of a database relation R with a tuple $\vec{c}$ of the proper arity will be referred to as a *fact*. A database instance can equivalently be seen as a collection of facts. An instance of a schema that has only a single relation R is a *relation instance*. The *active domain* of an instance I, denoted $\mathsf{adom}(\mathsf{I})$, is the union of the one-dimensional projections of all interpretations of relations: that is, all the elements that participate in some fact of I.

We can extend the notion of a schema to allow constant symbols: now an instance also maps each constant symbol to a value. We can also extend the notion of a schema to allow *integrity constraints*: in general these are an arbitrary subset of the instances. An instance of a schema with constraints is one where all integrity constraints of the schema are satisfied.

▶ **Example 4.** Suppose our schema consists of only one unary relation $\mathsf{UEmployee}$, containing the ids of university employees. One possible instance I interprets $\mathsf{UEmployee}$ by the singleton set $\{e_0\}$. We can alternatively define I by the set of facts $\{\mathsf{UEmployee}(e_0)\}$. In this case we have $\mathsf{adom}(\mathsf{I}) = \{e_0\}$. ⌟

A *query* (over a given vocabulary) and output arity m is a function from instances of the schema to a set of m-tuples. A *Boolean query* is the case where $m = 0$: thus the output is a set of 0-tuples. There are only two 0-tuples: the empty set, which we identify with False, and the singleton 0-tuple, which we identify with True. If $m > 0$ we talk of a *non-Boolean query.*

Classical logic considers *structures* rather than *instances.* A structure consists of a set, the *domain* of the structure, interpretations for each relation as sets of tuples with values in the domain, and an interpretation for each constant as a single element of the domain.

▶ **Example 5.** Returning to the schema in Example 4, one structure M that conforms to it consists of a two-element domain $\{e_0, f_0\}$, with $\mathsf{UEmployee}$ interpreted as $\{e_0\}$. ⌟

In the *classical semantics* of first-order logic, quantifiers range over the domain of the structure. In databases, quantifiers are usually given the *active domain* semantics, in which the quantified variable ranges over the union of the values in the interpretations of relations. The active domain semantics can be used to give a meaning to a sentence in an instance, since the meaning only depends on the instance, not some domain in which the instance sits.

A *relational atom* is an expression $R(u_1 \dots u_n)$ where R is an n-ary relation symbol and the u_i are either constants or variables. We will use *first-order logic with relativized quantifiers*, RQFO. RQFO is built up from equality and relational atoms via the Boolean operators and the quantifications:

$$\exists \vec{x}\ R(\vec{s}, \vec{x}) \wedge \varphi(\vec{s}, \vec{x}, \vec{t})$$

and

$$\forall \vec{x}\ R(\vec{s}, \vec{x}) \rightarrow \varphi(\vec{s}, \vec{x}, \vec{t})$$

for R a relational symbol and φ an RQFO formula. Those familiar with active domain semantics will be able to see that it is subsumed by RQFO, in the sense that an active domain semantics formula can be converted to an equivalent RQFO formula under classical semantics: the active domain interpretation of the quantification $\exists x\ \varphi(x, \vec{y})$ is a disjunction $\bigvee_i \exists \vec{u}_i\ R_i(\vec{u}_i) \wedge \varphi(x, \vec{y})$, where x is one of the variables in each $\vec{u}_i$. Similarly a quantification $\forall x\ \varphi$ under active domain semantics translates into a conjunction of relativized universal quantifications.

We can similarly talk about equality-free RQFO formulas, by disallowing equality atoms. By convention, we allow the prefix of existential quantifiers $\exists x_1\ \dots\ \exists x_n$ to be empty, and similarly for universal quantifiers. In this way we can capture an atom $R(x_1 \dots x_n)$ and negated atoms $\neg R(x_1 \dots x_n)$ as specialized RQFO formulas $R(\vec{x}) \wedge \mathsf{True}$ and $R(\vec{x}) \rightarrow \mathsf{False}$, respectively. Thus for equality-free formulas we can in fact assume that the only base formulas are True and False.

The truth value of an RQFO sentence without constants is well-defined over an instance, since from the instance we can determine the range of each relativized quantifier. Similarly the truth value for an RQFO formula without constants is well-defined given an instance and a mapping of the free variables to some values (a *variable binding* or just *binding* for short).

An RQFO formula is in *Negation Normal Form* (NNF) if negation is only applied to relational atoms. By simple rules, one can convert any RQFO formula into NNF.

Relational Algebra. RQFO gives a natural way of defining Boolean queries over finite instances. For non-Boolean queries we review another standard formalism. Given a relational vocabulary, we can build up *relational algebra* a language for defining queries. A relational algebra expression is associated with an output type, which is a finite set of attribute names. The atomic expressions are the relation symbols for names in the vocabulary, and for a relation symbol R of arity n, the output type has names $\{1 \ldots n\}$. The language is closed under the operations:

- difference, intersection, and union: if E and F are expressions having the same output T, $E - F$, $E \cup F$ and $E \cap F$ are new expressions also having type T
- projection: if E is an expression with output type $a_1 \ldots a_n$ and S is a subset of $a_1 \ldots a_n$ then $\pi_S(E)$ is an expression with output type S
- renaming: if E is an expression with output type S and F is a bijection from S to S' then $\rho_F(E)$ is an expression with output type S'
- product: if E and F are expressions with disjoint output types S and S', then $E \times F$ is an expression with output type $S \cup S'$
- selection: if E is an expression with output type $\{a_1 \ldots a_k\}$ then $\sigma_{a_i=a_j} E$ is an expression with the same output type.

The semantic function for relational algebra maps each term E with output type S to a query whose input is an instance over the input vocabulary and whose output is an instance of the signature consisting of an S-ary relation symbol, along with a mapping of positions to elements of S. For atomic terms it is the identity. The inductive definition of each operator is also straightforward.

We note a superficial difference between relational algebra and logic-based formalisms like RQFO. Relational algebra uses *named notation*: it deals with instances where in each tuple of a relation, the different components are referred to via attribute names. As opposed to logical formalisms, where we have *positional notation*: each relation has some arity m and different components are referred to by a position $1 \leq i \leq m$. It is easy to map between named and positional notation. Below we will abuse notation slightly and talk about a formalism in one notation being equivalent in expressiveness to a formalism in the other: this is up to some canonical way of mapping between names and positions. One can find the formalization of such mappings in [2].

A term with empty output type returns either the empty instance or the instance consisting of a single empty tuple: it can thus be considered as a Boolean query.

▶ **Proposition 6.** *A relational algebra expression of Boolean type can be mapped to an equivalent* RQFO *expression, and vice versa.*

▶ **Proposition 7.** *The output of a relational algebra expression on a finite instance is a finite relation.*

A query is *safe* if on every finite instance, there are only finitely many variable bindings that satisfy it.

There is a converse to Proposition 7: for any RQFO formula, if the query it defines is safe, then there is a relational algebra expression that represents the same query.

A relational algebra expression that does not use the difference operator is said to be a $USPJ^{\neq}$ expression (for union-selection-projection-join). If further it does not use union, it is an $SPJ^{\neq}$ expression.

3 First Example: Query Reformulation over a Subvocabulary

We now drill down into the first example of reformulation-via-interpolation: vocabulary-based reformulation.

3.1 Reformulating Queries over Restricted Vocabularies: Definitions

Let $\mathcal{S}$ be a collection of relations, Σ a set of integrity constraints, and $\mathcal{T}$ a subset of $\mathcal{S}$. Given a query Q specified by a logical formula with n free variables over $\mathcal{S}$, our first goal will be to get a *relativized-quantifier first-order reformulation of* Q *over* $\mathcal{T}$ *with respect to* Σ. This means a query $Q_{\mathcal{T}}$ given by a relativized-quantifier first-order formula using only the relations in $\mathcal{T}$ such that for every instance I satisfying Σ,

$$\forall x_1 \dots x_n \; Q(x_1 \dots x_n) \leftrightarrow Q_{\mathcal{T}}(x_1 \dots x_n)$$

holds in I. We use the same notation when Q is a relational algebra query, identifying its output attributes with free variables. Thus a reformulation is another query $Q_{\mathcal{T}}$ that is *equivalent to* Q *w.r.t.* Σ or *answers* Q *w.r.t.* Σ, meaning that they have the same output on instances satisfying Σ.

3.2 From a Semantic Property to a First-Order Reformulation

Let us return to the "meta-algorithm" for reformulating a query Q with respect to a target language and a set of constraints, mentioned in the introduction. Recall that step (1) is to identify a property that Q must have in order to admit the desired reformulation. Clearly, for Q to have any reformulation over $\mathcal{T}$, its output should depend only on the interpretations of the relations in $\mathcal{T}$. This notion of "implicit definability" has been formalized by Segoufin and Vianu in [28], as the notion of *determinacy* for database views and queries. We extend the definition here to the setting where there are constraints given by RQFO sentences.

If Σ is a collection of RQFO constraints, we say that an RQFO query Q over $\mathcal{S}$ is *determined over* $\mathcal{T}$ *relative to* Σ if:

For any two instances I and I' that satisfy Σ and have the same interpretation of all relations in $\mathcal{T}$ (that is, they have the same T-facts for each $T \in \mathcal{T}$) $[\![Q]\!](I) = [\![Q]\!](I')$.

▶ **Example 8.** We consider the case where the vocabulary consists of the Department and Employee relations, the constraints Σ are inclusion dependencies from Employee to Department (on deptid) and from Department to Employee (on mgrid and deptid), while the target signature $\mathcal{T}$ consists of only Department.

Suppose our query Q asks for the ids of all employees. We claim that Q is *not* determined over $\mathcal{T}$ relative to Σ. To see this, consider two instances I and I', such that I consists of facts:

$$\{\mathsf{Employee}(123, \text{“}\mathsf{Jones}\text{”}, 1112), \mathsf{Employee}(134, \text{“}\mathsf{Smith}\text{”}, 1112), \\ \mathsf{Department}(1112, \mathsf{mathematics}, 134)\}$$

and I' consists of facts

$$\{\mathsf{Employee}(134, \text{“}\mathsf{Smith}\text{”}, 1112), \mathsf{Department}(1112, \mathsf{mathematics}, 134)\}$$

I and I' both satisfy the schema constraints, and they have the same restriction to Department, but the output of Q on I includes employee id 123, while Q evaluated on I' does not.

We have shown via witnesses I and I' that Q is not determined over $\mathcal{T}$ *w.r.t.* Σ. ⌟

A special case of interest is where the relations in the restricted vocabulary $\mathcal{T}$ are “virtual tables” defined by *view definitions*. For each $V \in \mathcal{T}$ of arity n there is a corresponding query $Q_V(x_1 \dots x_n)$, while the constraints Σ consist only of the statements that a view contains exactly those tuples that satisfy its definition: $\forall \vec{x}\ V(\vec{x}) \leftrightarrow Q_V(\vec{x})$. In this case, determinacy of another query Q with respect to $\mathcal{T}$ and Σ can be rephrased as:

> For any two instances I and I', if I and I' yield the same results for each Q_V, then they yield the same results for Q.

Segoufin and Vianu use the terminology “Q is determined by the views $\{Q_V : V \in \mathcal{T}\}$”. We will apply our methodology to the semantic property of determinacy, turning it into a proof goal, and showing that reformulations can be read off from these proofs witnessing the proof goal.

Translating the Semantic Property to an Entailment. Returning to Step (2) of our “meta-algorithm” from the introduction, we write out the semantic property (in this case, determinacy) as a proof goal. We now formalize what this means.

Recall that an *entailment* is a statement of the form $\lambda(\vec{x}) \models \rho(\vec{x})$, which means that for every instance I and any binding vbind of $\vec{x}$, if instance I with binding vbind satisfies λ then it satisfies ρ. By convention, we write $\models \rho$ to mean $\mathsf{True} \models \rho$: that is, ρ holds in every structure. Let us extend our original signature for the constraints Σ and the query Q by making a copy R' of every relation $R \in \mathcal{S}$. Let Q' be the copy of Q on the new relations, and Σ' be the copy of the constraints Σ on the new relations. Our assumption of determinacy of Q can be restated as an entailment:

$$\models \forall \vec{x}\ [\Sigma \wedge \Sigma' \wedge (\bigwedge_{T \in \mathcal{T}} \forall \vec{y}\ T(\vec{y}) \leftrightarrow T'(\vec{y})) \wedge Q(\vec{x}) \rightarrow Q'(\vec{x})]$$

Or, rewriting,

$$\Sigma \wedge Q(\vec{x}) \models [(\bigwedge_{T \in \mathcal{T}} \forall \vec{y}\, T(\vec{y}) \leftrightarrow T'(\vec{y})\ ) \wedge \Sigma'] \rightarrow Q'(\vec{x})$$

This is the *entailment associated with determinacy.*

How Interpolation for the Entailment Gives a Reformulation. Going back to our general plan:

1. We have isolated a *semantic property* that an input query Q must have with respect to the target $\mathcal{T}$ and constraints Σ in order to have a relativized-quantifier first-order reformulation: it must be determined over $\mathcal{T}$ w.r.t.Σ.
2. We have expressed this property as an entailment:

$$\Sigma \wedge Q(\vec{x}) \models [(\bigwedge_{T \in \mathcal{T}} \forall \vec{y}\, T(\vec{y}) \leftrightarrow T'(\vec{y})) \wedge \Sigma'] \rightarrow Q'(\vec{x})$$

We now relate this to *interpolants.*

If $\lambda \models \rho$ is an entailment, an *interpolant* is a formula χ, such that:

- $\lambda \models \chi$ and $\chi \models \rho$
- Every relation in χ occurs in both λ and ρ.

Interpolation theorems state that any entailment for formulas in a logic L has an interpolant in logic L, possibly satisfying additional conditions.

To instantiate the meta-algorithm for this setting, we need to:

- show that an interpolant for this entailment gives the desired reformulation
- develop an algorithm to extract these interpolants

We start with the first item, the argument that an interpolant for the entailment above represents the first-order reformulation we want:

▶ **Proposition 9** (Variation of [15]). *Suppose Q is a first-order logic formula and Σ is a first-order sentence. Let χ be any interpolant for the entailment*

$$\Sigma \wedge Q \models [(\bigwedge_{T \in \mathcal{T}} \forall \vec{y}\, T(\vec{y}) \leftrightarrow T'(\vec{y})) \wedge \Sigma'] \rightarrow Q'$$

Then χ uses only the relations in $\mathcal{T}$, and is equivalent to Q for structures that satisfy Σ.

In particular, if χ is an interpolant in relative-quantifier first-order logic, we have found an RQFO reformulation of Q relative to Σ.

The proof of the proposition is quite straightforward: by the definition of an interpolant, χ uses only the relations common to the left and right, which are those in $\mathcal{T}$. And it is easy to see that Σ entails that χ is equivalent to Q. For one direction we use that $\Sigma \wedge Q \models \chi$, by the definition of an interpolant. For the other part of the equivalence, using the fact that χ entails the right side, and setting the primed and unprimed relations equal we see that $\Sigma \wedge \chi \models Q$.

We have thus completed all the steps of the meta-algorithm, provided that we have a way to get RQFO interpolants from an entailment between RQFO formulas.

3.3 The Interpolation Theorem We Need: Relativized-Quantifier Interpolation

We discuss the modification of Craig interpolation needed to take a proof witnessing an entailment for relativized-quantifier first-order formulas and produce an RQFO interpolant. This will allow us to instantiate the meta-algorithm for RQFO constraints:

The interpolation result we need is a relativized-quantifier version of Craig interpolation, a variant of a result proven originally by Martin Otto.

▶ **Theorem 10** ([27]). *[Relativized-Quantifier Craig Interpolation Theorem] If λ and ρ are* RQFO *formulas such that $\lambda \models \rho$, then there is an interpolant χ in* RQFO*. Furthermore, if λ and ρ do not use equality, neither does χ.*

In [6], an effective version of this was proven, where we assume a proof of the entailment in a certain proof system. In [6] we use analytic tableaux as a proof system: this proof system and the interpolation algorithm for it are discussed in [12]. But in the results we will just refer to a *suitable proof system*, leaving the details to the references.

▶ **Theorem 11** (Effective Relativized-Quantifier Craig Interpolation Theorem). *If λ and ρ are* RQFO *formulas* and *we have a suitable proof of $\lambda \models \rho$, then* in polynomial time in the proof *we can find an interpolant χ in* RQFO*. Furthermore, if λ and ρ do not use equality, neither does χ.*

Putting It All Together. We are ready to give the instantiation of the general methodology. Instantiating it to this setting, we see that to find an RQFO reformulation of Q with respect to subvocabulary $\mathcal{T}$ and constraints Σ we should:

- Search for a proof that the semantic property determinacy holds. That is, find a proof witnessing

$$\Sigma \wedge Q(\vec{x}) \models [(\bigwedge_{T \in \mathcal{T}} \forall \vec{y}\, T(\vec{y}) \leftrightarrow T'(\vec{y})) \wedge \Sigma'] \rightarrow Q'(\vec{x})$$

- Use the relativized-quantifier tableau-based interpolation algorithm to produce an interpolant $Q_{\mathcal{T}}(\vec{x})$.
- As shown in Proposition 9, such an interpolant will give the reformulation we want.

We also know that to have an RQFO reformulation, Q must be determined over $\mathcal{T}$ *w.r.t.* Σ, and thus there must be a proof of the entailment for determinacy. This means we have proven the following *effective reformulation theorem*:

▶ **Theorem 12.** *To find a reformulation of an* RQFO *query Q with respect to subvocabulary $\mathcal{T}$ and* RQFO *constraints Σ, it is sufficient to find a proof (in a suitable proof system) witnessing the entailment:*

$$\Sigma \wedge Q \models [(\bigwedge_{T \in \mathcal{T}} \forall \vec{y}\; T(\vec{y}) \leftrightarrow T'(\vec{y})) \wedge \Sigma'] \rightarrow Q'$$

where Σ' is formed from Σ by replacing each relation R with R', and Q' is formed similarly from Q.

From any such proof, we can effectively produce a reformulation.

Theorem 12 reduces searching for a reformulation to searching for a proof. Since the existence of a reformulation implies that the entailment above holds, and the entailment above captures determinacy:

▶ **Corollary 13** (Equivalence of Reformulations, Entailments, and Semantic Properties)**.** *An* RQFO *query Q has an* RQFO *reformulation with respect to subvocabulary $\mathcal{T}$ and constraints Σ if and only if Q is determined over subvocabulary $\mathcal{T}$ with respect to Σ if and only if the entailment in Theorem 12 holds.*

4 Vocabulary-Based Reformulation with Positive Existential Queries

Expanding on our general program outlined in the introduction, we show what happens when we restrict the target language for a rewriting. Recall that a *positive existential formula with inequalities* ($\exists^{+,\neq}$ formula) is a formula built up using only $\exists, \wedge, \vee$ from relational atoms and inequalities. We also consider the formula False to be positive existential with inequalities.

Given an RQFO formula Q, restricted vocabulary $\mathcal{T}$, and constraints Σ given by RQFO sentences, we are interested in getting a $\exists^{+,\neq}$ reformulation of Q over $\mathcal{T}$ with respect to Σ. This means we want a $\exists^{+,\neq}$ formula over $\mathcal{T}$ that agrees with Q for instances satisfying the constraints.

The Semantic Property for $\exists^{+,\neq}$ Reformulation. Following the "meta-algorithm" from the introduction, we start by finding the appropriate semantic property that an input Q must have to admit a $\exists^{+,\neq}$ reformulation. [26] isolated such a property, which we call monotonic determinacy[1]. We say that a query Q is *monotonically-determined over* $\mathcal{T}$ relative to Σ if:

for any two instances $\mathsf{I}_1, \mathsf{I}_2$ that satisfy Σ and such that for all relations $T \in \mathcal{T}$, $[\![T]\!](\mathsf{I}_1) \subseteq [\![T]\!](\mathsf{I}_2)$, then $[\![Q]\!](\mathsf{I}_1) \subseteq [\![Q]\!](\mathsf{I}_2)$.

[1] [26] used the term "monotone"

The Entailment Corresponding to the Semantic Property. Proceeding to the second step of the meta-algorithm, we will express this semantic property as an entailment. Let Σ' be a copy of the constraints Σ where each occurrence of a relation R in $\mathcal{S}$ has been replaced by a copy R'.

Monotonic determinacy of a first-order query Q over $\mathcal{T}$ relative to Σ can be restated as saying that the following sentence holds on all instances:

$$\forall \vec{x}\ [\Sigma \wedge \Sigma' \wedge (\bigwedge_{T \in \mathcal{T}} \forall \vec{y}\ T(\vec{y}) \rightarrow T'(\vec{y})) \wedge Q(\vec{x})] \rightarrow Q'(\vec{x})$$

Above, Q' and Σ' are again the result of changing unprimed relations R to their primed counterparts R' within Q and within Σ respectively. Observe that if a $\exists^{+,\neq}$ formula over $\mathcal{T}$ is true on an instance I, then it is true on any instance I' which only adds tuples to the relations in $\mathcal{T}$. It is thus easy to see that a sufficient condition for monotonic determinacy of Q over $\mathcal{T}$ w.r.t.Σ is that Σ implies the sentence $\forall \vec{x}\ Q(\vec{x}) \leftrightarrow \varphi(\vec{x})$, where φ is a $\exists^{+,\neq}$ formula mentioning only relations in $\mathcal{T}$.

We highlight the difference from the entailment for determinacy: here we only have implication in the "forward" direction, from unprimed to primed, while for determinacy we have implications in both directions.

Generating $\exists^{+,\neq}$ Reformulations from Proofs of the Entailment. We will show that by applying interpolation to proofs of the entailment for monotonic determinacy, we get $\exists^{+,\neq}$ reformulations. In doing so, we will prove the following analog of Theorem 10:

▶ **Theorem 14.** *If the constraints Σ are in* RQFO *and* RQFO *query Q is monotonically-determined over $\mathcal{T}$, then there is a $\exists^{+,\neq}$ formula $\varphi(\vec{x})$ using only relations in $\mathcal{T}$ such that $\Sigma \models \forall \vec{x}\ [Q(\vec{x}) \leftrightarrow \varphi(\vec{x})]$. Furthermore, if the constraints and the query Q do not make use of equality, then χ can be taken to be positive existential (without inequalities).*

We refer to this as the *Projective Monotone Preservation Theorem.* The adjective "Projective" emphasizes that we are dealing with a subset of the signature, as opposed to many preservation theorems one encounters in logic textbooks and papers, which deal with syntactically characterizing a semantic property involving the entire signature.

The steps in proving Theorem 14 will follow the meta-algorithm. We will prove a modification of the Craig Interpolation Theorem, the Relativized-Quantifier Lyndon Interpolation Theorem, and our proof will provide an algorithm for generating interpolants of a special form from an entailment. We then show that the interpolants produced by this algorithm, when applied to a re-arrangement of the entailment corresponding to the semantic property of monotonic determinacy, will give us the $\exists^{+,\neq}$ reformulation of our query Q with respect to the constraints.

The Interpolants Required for $\exists^{+,\neq}$ Reformulation. Before giving the argument, we provide some motivation. We can restate the entailment for monotonic determinacy as:

$$\Sigma \wedge (\bigwedge_{T \in \mathcal{T}} \forall \vec{y}\, T(\vec{y}) \rightarrow T'(\vec{y})) \wedge Q(\vec{x}) \models (\Sigma' \rightarrow Q'(\vec{x}))$$

As before, Σ' is a copy of the constraints on primed versions of the relations, and Q' is a copy of the query Q on primed versions of the relations. The common vocabulary on the left and right consists of exactly the relations T' for $T \in \mathcal{T}$.

Further, we note that these common relations only occur on the right of an implication on the left-hand side. Writing out the implication $A \rightarrow B$ as $\neg A \vee B$, we see that the common relations would not occur within a negation on the left side. Informally, we can say that these relations "occur positively" on the left hand side in the original formula. Thus we want to show that the interpolant will also contain these common relations positively. To do this, we need a formal definition of "occurring positively" that applies to arbitrary first-order formulas, and a version of interpolation that connects the relations occurring positively in the interpolant with those that occur positively in both sides of the entailment.

Formally, we say that a relation *occurs positively* in a first-order formula if it occurs in the scope of an even number of negations, when we rewrite the formula to use only the quantifiers and the connectives $\wedge, \vee, \neg$. A relation occurs negatively in a formula if it occurs in the scope of an odd number of negations. Intuitively if a relation occurs only positively, then the set of solutions to the formula can only increase or stay the same as tuples are added to that relation (this is easy to check if the relation does not appear under any negations at all).

The interpolation theorem that tracks which occurrences are positive is the following result, which is a strengthening of the Relativized-quantifier Craig Interpolation Theorem, Theorem 10:

▶ **Theorem 15** (Relativized-quantifier Lyndon Interpolation Theorem). *Suppose λ and ρ are in* RQFO *and $\lambda \models \rho$. Then there is an* RQFO *interpolant χ for the entailment such that a relation occurs positively in χ only if it occurs positively in both λ and ρ, and a relation occurs negatively in χ only if it occurs negatively in both λ and ρ. Furthermore if equality does not occur in λ or ρ, then it does not occur in χ, so in particular χ can not contain inequalities.*

Again, there is a well-known version of this for the classical semantics of first-order logic, Lyndon's interpolation theorem [25].

The construction that witnesses the Relativized-quantifier Lyndon Interpolation Theorem, Theorem 15, is a variant of the one for the Relativized-quantifier CIT, which is discussed [12].

Getting $\exists^{+,\neq}$ and Positive Existential Reformulations via Interpolants. Theorem 15 shows that we can extract a certain kind of interpolant from a tableau proof witnessing an entailment corresponding to monotonic determinacy. We are now ready to instantiate the last step of our meta-algorithm, extracting a reformulation from an interpolant. This will complete the proof of Theorem 14.

Apply Theorem 15 to the entailment

$$\Sigma \wedge (\bigwedge_{T \in \mathcal{T}} \forall \vec{y}\ T(\vec{y}) \rightarrow T'(\vec{y})) \wedge Q(\vec{x}) \models (\Sigma' \rightarrow Q'(\vec{x}))$$

We can conclude that there is an RQFO interpolant γ mentioning only relations in the primed copy of $\mathcal{T}$, where these relations only occur positively. We can assume γ is built up from True and False via connectives and relativized quantifiers $\forall \vec{x}\ R(\vec{x}) \rightarrow \varphi$ and $\exists \vec{x}\ R(\vec{x}) \wedge \varphi$. We claim that any RQFO formula in which all relations occur positively must be equivalent to a positive existential formula. To see this, convert an RQFO formula to NNF. If the resulting formula γ had any relativized universal quantifier, then consider an outermost quantification of the form $\forall \vec{x}\ (R(\vec{x}) \rightarrow \varphi)$. R must occur negatively in this formula. But then it must occur negatively within γ, since existential quantification and the positive Boolean operators preserve the polarity of a subformula, and this contradicts the assumption on γ. Hence the Negation Normal Form of γ can not contain any relativized universal quantifier, and thus must be $\exists^{+,\neq}$.

We say $\exists^{+,\neq}$ above, rather than positive existential, because the argument applies only to relations R of the schema, not equality. If the equality symbol does not appear in the entailment we know that it is not generated in the interpolant. Therefore when equality does not occur in the constraints or the query, we can strengthen the conclusion to be that the interpolant is positive existential.

This completes the proof of the Projective Monotone Preservation Theorem, Theorem 14.

Application to View-Based Query Reformulation. Let us look at the setting where there are no constraints other than those that come from views defined by conjunctive queries. As a corollary of Theorem 14 we have:

▶ **Corollary 16.** *Suppose Q is a CQ and $V_1 \dots V_n$ are views defined by arbitrary equality-free* RQFO *formulas. Then, Q is monotonically-determined in $V_1 \dots V_n$ if and only if there is a positive existential reformulation of Q in terms of $V_1 \dots V_n$.*

5 Vocabulary-Based Existential Reformulation

At this point, we have proven a statement about first-order reformulations and one about positive existential reformulations. What about queries that can be reformulated using *existential formulas*? That is, formulas that are built up from atoms and negated atoms by positive Boolean operators and existential quantification. There are conjunctive queries that are equivalent to existential formulas but not to positive existential ones. For example, in the absence of any constraints $\exists x\ S(x) \wedge \neg R(x)$ is not equivalent to a positive existential formula. Can we use a similar technique to detect which formulas are equivalent to an existential formula with respect to a set of constraints, and if so find such a reformulation? We give a positive answer to this below, restricting for simplicity to equality-free RQFO.

The Semantic Property for Existential Reformulation. As before, let Σ be a set of integrity constraints in equality-free RQFO, and $\mathcal{T}$ a subset of the relations of Sch. We start by isolating a property that Q must have in order to possess an existential reformulation.

We say that a query Q is *induced-subinstance-monotonically-determined over* $\mathcal{T}$ relative to Σ if:

Whenever we have two instances $\mathsf{I}_1, \mathsf{I}_2$ that satisfy Σ and such that I_1 is an induced subinstance of I_2 then $[\![Q]\!](\mathsf{I}_1) \subseteq [\![Q]\!](\mathsf{I}_2)$.

An instance I_1 is an induced subinstance of I_2 means that I_2 contains all facts of I_1 and I_2 does not add any facts over the active domain of I_1.

Note that if an existential formula over $\mathcal{T}$ is true on an instance I, then it is true on any instance I' which only adds tuples to the relations in $\mathcal{T}$ and never "destroys a negated assertion about a relation of $\mathcal{T}$ holding in I". From this we see that if a formula is equivalent to an existential formula under a set of constraints Σ, then the formula is induced-subinstance-monotonically-determined over $\mathcal{T}$ *w.r.t.* Σ.

The Entailment Corresponding to the Semantic Property. As in the previous cases, we instantiate our meta-algorithm by writing out the semantic property as an entailment.

Let $\mathsf{InDomain}_{\mathcal{T}}(x)$ abbreviate the formula:

$$\bigvee_{T\in\mathcal{T}} \bigvee_{j} \exists w_1 \ldots \exists w_{j-1}\ \exists w_{j+1} \ldots w_{\mathsf{arity}(T)}\ T(w_1, \ldots w_{j-1}, x, w_{j+1}, \ldots, w_{\mathsf{arity}(T)})$$

So $\mathsf{InDomain}_{\mathcal{T}}$ states that x is in the domain of a relation in $\mathcal{T}$. The entailment we need is:

$$Q(\vec{x}) \wedge \Sigma \wedge \Sigma' \wedge \bigwedge_{T\in\mathcal{T}} (\forall \vec{y}\ T(\vec{y}) \rightarrow T'(\vec{y})) \wedge \bigwedge_{T\in\mathcal{T}} (\forall \vec{y}\ \bigwedge_{i} \mathsf{InDomain}_{\mathcal{T}}(y_i) \wedge T'(\vec{y}) \rightarrow T(\vec{y})\ ) \models\ Q'(\vec{x})$$

Comparing with the two previous entailments, we have the forward implication as before, and a restriction of the backward implication. It is clear that induced-subinstance-monotonicity is equivalent to this entailment holding.

Extracting Interpolants from a Proof of the Entailment: Statement of Result. We will show later that if we have a "suitable" interpolant for the entailment corresponding to induced-subinstance-monotonicity, then we can extract an existential reformulation from it, completing another instantiation of the meta-algorithm.

This will give us a proof of another analog of Theorem 10:

▶ **Theorem 17.** *If the constraints Σ are in equality-free* RQFO*, and* RQFO *query Q is induced-subinstance-monotonically-determined over $\mathcal{T}$, then there is an existential first-order formula $\varphi(\vec{x})$ using only relations in $\mathcal{T}$ such that $\Sigma \models \forall \vec{x}\ Q(\vec{x}) \leftrightarrow \varphi(\vec{x})$.*

A similar theorem will hold if the constraints are in RQFO with equality, with the conclusion being that φ is an existential formula with inequalities. There is also an effective variant of the theorem above, an analog of Theorem 12. It states that the existential φ can be generated efficiently from a suitable proof of the entailment for induced-subinstance-monotonicity.

We defer the proof of Theorem 17 for the moment. It will follow from a more general theorem, Theorem 27, proved later. The reformulation we need will come from applying an interpolation procedure to the entailment above. But we need a new interpolation result to guarantee that the interpolant will be existential, and this extended interpolation theorem is "Access Interpolation", described later.

Łos-Tarski. The theorem above is closely related to another result in logic, the Łos-Tarski preservation theorem. This states that a formula is preserved under extensions exactly when it is equivalent to an existential formula. Roughly speaking, the Łos-Tarski theorem is a special case of Theorem 17, where Σ is empty and $\mathcal{T}$ contains all relations in the schema. The "roughly speaking" disclaimer is because the Łos-Tarski theorem deals with classical first-order logic formulas rather than for RQFO, and the notion of induced subinstance must be replaced by the analogous notion for structures.

6 Rewriting with Access Patterns

Thus far the target of rewriting was specified through vocabulary restrictions. We wanted a query that used a fixed set of target relations, perhaps restricted to be positive existential or existential. We now consider a finer notion of reformulation, where the target has to satisfy *access restrictions*.

We begin with the definitions of access methods along with a programming language, the RA-plans, that combines data access by means of a fixed set of access methods with data manipulation using relational algebra queries. Recall from Section 2 that relational algebra represents an algebraic programming language that is equivalent in expressiveness to first-order logic formulas that are *safe*, in that for every finite instance, the number of satisfying assignments is finite. Implementation of logic-based languages proceeds by translating a logical formula into relational algebra. RA-plans can be thought of as a variation of relational algebra in which we want to abide by a fixed set of data interfaces given by access methods. They will thus be the new target of reformulation.

We return to our program of going from semantic properties to rewritings, following the methodology outlined in the introduction. We present semantic properties that must hold for a query to be implemented using the interface given by a set of access methods, and show that these properties can be captured by entailments. In the remainder of the section we explain how proofs of these entailments can be converted into plans within our plan languages. The conversion from proof to plan will again proceed via interpolation. But in this case we need a new interpolation theorem tailored to the setting of access methods.

6.1 Basics of Target Restrictions Based on Access Methods

We now look at a notion of interface that is closer to the traditional notion in programming languages: a set of functions that access the data. A specification of this interface will be an extended set of metadata describing both the format of the data (e.g. the vocabulary that would be used in queries and constraints) and the access methods (functions that interact with the stored data).

An *access schema* consists of:

- A collection of relations, each of a given arity.
- A finite collection C of schema constants ("Smith", 3, ...). Schema constants represent a fixed set of values that will be known to a user prior to interacting with the data. Values that can be used in queries and constraints should be schema constants, as before. In addition, any fixed values that might be used in plans that implement queries should come from the set of schema constants. For example, a plan to answer a query about the mathematics department might involve first putting the string "mathematics" into a university directory service.
- For each relation R, a collection (possibly empty) of *access methods*[2]. Recall that a *position* of a relation R is a number between 1 and $\mathsf{arity}(R)$. Each access method mt is associated with a collection (possibly empty) of positions of R — the *input positions* of mt.
- Integrity constraints, which are sentences in relativized-quantifier first-order logic as before.

An *access* (relative to a schema as above) consists of an access method of the schema and a *method binding* — a function assigning values to every input position of the method. If mt is an access method on relation R with arity n, I is an instance for a schema that includes R, and $\mathsf{AccBind}$ is a method binding on mt, then the *output* or *result* of the access $(\mathsf{mt}, \mathsf{AccBind})$ on I is the set of n-tuples $\vec{t} \in [\![R]\!](I)$ such that $\vec{t}$ restricted to the input positions of mt is equal to $\mathsf{AccBind}$.

An access method may have an empty collection of input positions. In this case, the only access that can be performed using the method is with the empty method binding. When a method has no input positions, we say that the access method is "input-free". In Example 2 of Section 1, the $\mathsf{Udirectory}$ table was assumed to have such an input-free access method.

The goal is to reformulate source queries in a target language that represents the kind of restricted computation done over an interface given by an access schema. We first formalize this operationally, as a language of *plans*. Plans are straight-line programs that can perform accesses and manipulate the results of accesses using relational algebra expressions. This language could model, at a high-level, the plans used internally in a database management

[2] Our definition of "access methods" is a variant of the terminology "access patterns" or "binding patterns" found in the database literature.

system. It could also describe the computation done within a data integration system, which might access remote data via a web form or web service and then combine data from different sources using SQL within its own database management system.

All of our plan languages have as a primitive an *access command*. Over a schema Sch with access methods, an access command is of the form:

$$T \Leftarrow_{\mathsf{OutMap}} \mathsf{mt} \Leftarrow_{\mathsf{InMap}} E$$

where:

- E is a relational algebra expression, the *input expression*, over some set of relations not in Sch (henceforward "temporary tables");
- mt is a method from Sch on some relation R;
- InMap, the *input mapping* of the command, is a function from the output attributes of E onto the input positions of mt;
- T, the *output table* of the command, is a temporary table;
- OutMap, the *output mapping* of the command, is a bijection from positions of R to attributes of T.

Note that an access command using an input-free method must take the empty relation algebra expression $\emptyset$ as input.

The manipulation of data retrieved by an access is modeled with the other primitive of our languages, a *middleware query command*. These are of the form $T := Q$, where Q is a relational algebra expression over temporary tables and T is a temporary table. We use the qualifier "middleware" to emphasize that the queries are performed on temporary relations created by other commands, rather than on relations of the input schema.

A *relational algebra-plan* (or simply, *RA-plan*) consists of a sequence of access and middleware query commands, ending with at most one *return command* of the form $\mathsf{Return}\ E$, where E is an RA expression.

▶ **Example 18.** We return to Example 2 of Section 1, where we had two sources of information. One was $\mathsf{Profinfo}$, which was available through an access method $\mathsf{mt}_{\mathsf{Profinfo}}$ requiring input on the first position. The second was $\mathsf{Udirectory}$, which had an access method $\mathsf{mt}_{\mathsf{Udirectory}}$ requiring no input. Our query Q asked for ids of faculty named "Smith". One plan that is equivalent to Q would be represented as follows

$$T_1 \Leftarrow \mathsf{mt}_{\mathsf{Udirectory}} \Leftarrow \emptyset$$

$$T_2 := \pi_{\mathsf{eid}}(\sigma_{\mathsf{lname}=\text{``}\mathsf{Smith}\text{''}} T_1)$$

$$T_3 \Leftarrow \mathsf{mt}_{\mathsf{Profinfo}} \Leftarrow T_2$$

$$\mathsf{Return}\ \pi_{\mathsf{eid}}(T_3)$$

Above we have omitted the mappings in writing access commands, since they can be inferred from the context. We will often do this in plans for brevity. ⌟

Semantics of Plans. A temporary table is *assigned* in a plan if it occurs on the left side of a command, and otherwise is said to be *free*. The semantics of plans is defined as a function that takes as input an instance I for Sch and interpretations of the free tables. If the plan has no Return statement, the output consists of interpretations for each assigned temporary table. If the plan contains a statement Return E, the output is an interpretation of a relation with attributes for each output attribute of E. In the latter case, we refer to this as the *output* of the plan.

An access command $T \Leftarrow_{\mathsf{OutMap}} \mathsf{mt} \Leftarrow_{\mathsf{InMap}} E$ is executed by evaluating the expression 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 InMap. For each tuple $\vec{t} = t_1 \dots t_n \in R$ that "matches" (i.e.that extends) $t_{j_1} \dots t_{j_m}$, $\vec{t}$ is transformed to a tuple $\vec{t'}$ using the output mapping OutMap. The interpretation of T is then the union of all such tuples $\vec{t'}$. A middleware query command $T := E$ executes query E on the contents of the temporary tables mentioned in E, and then assigns the result to temporary table T.

A plan is executed by evaluating its commands in sequence, with each command operating on the instance formed from the input instance, adding the interpretations of assigned tables produced by earlier commands. For a plan having as its final command Return E, the output of the plan is the evaluation of E on the instance formed as above.

We usually assume (without loss of generality) that each table is only assigned once within a plan. Given plan PL, temporary table T that occurs in PL, and an instance I for the schema Sch, we let $[\![T|\ \mathsf{PL}]\!]_I$ be the content of T when PL is run on I. For plan PL including a Return statement, we let $[\![\mathsf{PL}]\!](I)$ be the output of PL on I. Similarly, given a relational algebra expression E over temporary tables $T_1 \dots T_n$ of PL and instance I, $[\![E|\ \mathsf{PL}]\!]_I$ represents the result of E when run on $[\![T_1|\ \mathsf{PL}]\!]_I \dots [\![T_n|\ \mathsf{PL}]\!]_I$.

Fragments of the Plan Language. We now define fragments of our plan language, analogs of the fragments of relational algebra and first-order logic. In RA-plans, we allowed arbitrary relational algebra expressions in both the inputs to access commands and the middleware query commands. We can similarly talk about *$SPJ^{\neq}$-plans*, where the expressions in access and middleware query commands are built up from the $SPJ^{\neq}$ relational algebra operators and *$USPJ^{\neq}$-plans* that allow the union operator of relational algebra in addition to $SPJ^{\neq}$ operators. We define *$USPJAD^{\neq}$-plans* as RA-plans in which relational algebra's difference operator only occurs in a *non-membership check*, which tests whether the tuples in a projection of a temporary table are not in a given relation R. Formally, a non-membership check is a sequence of two commands:

$$T' \Leftarrow_{\mathsf{OutMap}} \mathsf{mt} \Leftarrow_{\mathsf{InMap}} \pi_{a_{j_1} \dots a_{j_m}}(T)$$
$$T'' := T - (T \bowtie T')$$

where in the first command:

- mt is an access method on some relation R with input positions $j_1 \dots j_m$;

- the input mapping InMap maps attribute a_{j_i} to position j_i;
- the attributes of the output table T' are a subset of the attributes of T containing each a_{j_i};
- the output mapping OutMap maps j_i back to a_{j_i}.

In the second command, the join condition identifies attributes that have the same name. SPJ-plans, $USPJ$-plans, and $USPJAD$-plans are defined analogously to the classes above, but not allowing inequality conditions in selections or joins.

Plans that Answer Queries. We now define what it means for a plan to correctly implement a query. Given an access schema Sch, a plan *answers* a query Q *(over all instances)* if for every instance I satisfying the constraints of Sch, the output of the plan on I is the same as the output of Q. We often omit the schema from our notation, since it is usually clear from context, saying that a plan PL answers Q.

From Plans to Specialized Formulas. We have stated our target language for reformulation as a procedural language. But to make use of interpolation-based methods, it will be useful to have a logic, consisting of formulas. We will be able to state an equivalence of plans in the case of sentences — that is, Boolean queries. In this case, we can show that a plan using a given set of access patterns is equivalent to a logical sentence where the quantification structure is compatible with the access patterns. This approach can be bootstrapped to general logical formulas, but we defer the discussion here, see [6].

An RQFO sentence is *executable* (relative to an access schema Sch) if it is built up from equalities and the formula True using arbitrary Boolean operations and the quantifiers:

$$\forall \vec{y}\ [R(\vec{x},\vec{y}) \rightarrow \varphi(\vec{x},\vec{y},\vec{z})]$$
$$\exists \vec{y}\ R(\vec{x},\vec{y}) \wedge \varphi(\vec{x},\vec{y},\vec{z})$$

and for any such quantification above, if R is a Sch relation, then R has an access method mt such that all of the input positions of mt are occupied by some x_i (that is: by a free variable or constant).

This definition captures the informal idea that the sentence is compatible with the access methods: we should be quantifying only over the output positions, while the values input positions should be provided.

The next proposition says that executable RQFO sentences we can find a plan that filters an input table down to the subset satisfying the formula.

▶ **Proposition 19.** *There is a linear time procedure that takes as input an executable* RQFO *sentence φ and produces an equivalent Boolean RA-plan* Plan_φ.

Furthermore, if the RQFO *sentence is existential with inequalities (resp. existential) the result is a $USPJAD^{\neq}$-plan (resp. $USPJAD$-plan). If the sentence is positive existential with inequalities (resp. positive existential) the result is a $USPJ^{\neq}$-plan (resp. $USPJ$-plan).*

In the other direction, we can convert every Boolean plan (of the given form) into the corresponding logic.

This result is not difficult, and the inductive arguments are given in [6]. It will allow us to deal with logical formulas having these "access-restricted quantifiers" from now on.

The Accessible Part. In order to begin our instantiation of the meta-algorithm for plans, we need a semantic property corresponding to existence of a plan. The property should say that formula "only depends on the accessible data", and thus we need to formalize what accessible data means.

▶ **Definition 20.** *Given an instance* I *for schema* Sch *the* accessible part of I*, denoted* $\mathsf{AccPart}(\mathsf{I})$*, and the* accessible values of I*, denoted* $\mathsf{accessible}(\mathsf{I})$*. Informally the accessible part consists of all the facts over* I *that can be obtained by starting with empty relations and iteratively entering values into the access methods. If* Sch *contains no schema constants, this will be an instance containing a set of facts* $\mathsf{Accessed}R(v_1 \dots v_n)$*, where R is a relation and $v_1 \dots v_n$ are values in the domain of* I *such that $R(v_1 \dots v_n)$ holds in* I*, obtained by starting with relations* $\mathsf{Accessed}R_0$ *and* $\mathsf{accessible}_0$ *empty and then iterating the following process until a fixpoint is reached:*

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

and

$$\mathsf{Accessed}R_{i+1} = \mathsf{Accessed}R_i \ \cup \bigcup_{\substack{(R,\{j_1,\dots,j_m\}) \\ \text{there is a method of } \mathsf{Sch} \text{ on } R \text{ with inputs } j_1,\dots,j_m}} \{(v_1 \dots v_n) \in [\![R]\!](I) \mid v_{j_1} \dots v_{j_m} \in \mathsf{accessible}_i\}$$

Above $\pi_j(\mathsf{Accessed}R_i)$ *denotes projection of* $\mathsf{Accessed}R_i$ *on the j^{th} position. For a finite instance, this induction will reach a fixpoint after* $|\mathsf{I}|$ *iterations, where* $|\mathsf{I}|$ *denotes the number of facts in* I*. For an arbitrary instance the union of these instances over all i will be a fixpoint. Assuming* Sch *does include schema constants, we modify the definition by starting with an* $\mathsf{accessible}_0$ *consisting of the schema constants, rather than being empty.*

Above we consider $\mathsf{AccPart}(I)$ as a database instance for the schema with relations $\mathsf{accessible}$ and $\mathsf{Accessed}R$. The accessible values are the union of the $\mathsf{accessible}_i$.

In the case of vocabulary-based access-restrictions, the accessible part of an instance just represents the restriction of the instance to the relations in the subsignature $\mathcal{T}$. Thus access determinacy of a query Q in the case of vocabulary-based restrictions is the same as determinacy of Q with respect to the subsignature.

The Semantic Property and Entailment for RA-Plans. We now give the analogous property and entailment for RA-plans. Q is said to be *access-determined* over Sch if for all instances I and I′ satisfying the constraints of Sch 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 tuples in the accessible part.

▶ **Example 21.** We return to the setting of Example 2, where we have a Profinfo table available via a web form, containing information about faculty, including their last names, office number, and employee id, but with only an access method $\mathsf{mt}_{\mathsf{Profinfo}}$ that requires giving an employee id as an input. We consider a query Q asking for ids of faculty named "Smith", where "Smith" is a schema constant.

We show that Q is not access-determined. For this, take I to be any instance that contains exactly one tuple, with lastname "Smith", but with no schema constant as its employee id. Let I' be the empty instance. The accessible parts of I and I' are empty, since in both cases when we enter all the constants we know about in $\mathsf{mt}_{\mathsf{Profinfo}}$, we get the empty response. But Q has an output on I but no output on I'.

I and I' witness that Q is not access-determined. From this we see that Q can not be implemented by any plan using $\mathsf{mt}_{\mathsf{Profinfo}}$. ⌟

We now turn to the corresponding entailment for access-determinacy. The accessible part itself is not a first-order definable object — it requires an infinite disjunction or recursion. But it will turn out that "agreeing on the accessible part" can be expressed in a first-order way.

We start with a schema Sch consisting of relation symbols and RQFO sentences — integrity constraints, as well as access restrictions. The *bidirectional accessible schema for* Sch, denoted $\mathsf{AcSch}^{\leftrightarrow}(\mathsf{Sch})$, is a vocabulary without access restrictions, and also a set of integrity constraints in the form of RQFO sentences. It is defined as follows:

- The constants are those of Sch.
- The relations are those of Sch, a unary relation $\mathsf{accessible}(x)$ ("x is an accessible value") plus a copy of each relation R of Sch called $\mathsf{InfAcc}R$ (the "inferred accessible version of R").
- The constraints are those of Sch (referred to as "Sch constraints" below) for each access method mt on relation R of arity n with input positions $j_1 \dots j_m$ along with the following constraints (dropping universal quantifiers on the outside for brevity)
  - *forward accessibility axioms*: we have a rule:

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

  - *backward accessibility axioms*

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

In addition, we have $\mathsf{accessible}(c)$ for each constant c of Sch.

- A copy of each of the original integrity constraints, with each relation R replaced by $\mathsf{InfAcc}R$, denoted "InfAccCopy constraints" below.

Given a query Q, its *inferred accessible version* $\mathsf{InfAcc}Q$ is obtained by replacing each relation R by $\mathsf{InfAcc}R$. Informally, $\mathsf{InfAcc}Q$ represents the fact that the existence of a witness to Q can be obtained through making accesses and reasoning.

We overload $\mathsf{AcSch}(\mathsf{Sch})$ to also refer to the conjunction of axioms in this schema $\mathsf{AcSch}(\mathsf{Sch})$.

A way of motivating the axioms is to think informally of the relations R $\mathsf{InfAcc}R$ as representing the instances considered in access determinacy. By having two copies of the constraints, we are stating that both instances satisfy the constraints of the schema. The forward and backward accessibility axioms will ensure that these two instances have the same accessible part. The entailment thus captures that if two instances have the same accessible part, and both satisfy the constrains, and one satisfies query Q, then so does the other.

Again, we show that this entailment captures the proposed preservation property, access-determinacy.

▷ Claim 22. The following are equivalent (for any Boolean RQFO query Q and access schema consisting of RQFO constraints):

1. Q entails $\mathsf{InfAcc}Q$ with respect to the rules in $\mathsf{AcSch}^{\leftrightarrow}(\mathsf{Sch})$
2. Q is access-determined over Sch

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 $\mathsf{AcSch}^{\leftrightarrow}(\mathsf{Sch})$ formed by interpreting the relations R as in I, the relation $\mathsf{accessible}$ by the accessible values of I, and each $\mathsf{InfAcc}R$ by the interpretation of R in I'. Then one can verify that I'' satisfies the constraints of $\mathsf{AcSch}^{\leftrightarrow}(\mathsf{Sch})$. Since I (and hence I'') satisfies Q, and we are assuming that Q entails $\mathsf{InfAcc}Q$ with respect to $\mathsf{AcSch}^{\leftrightarrow}(\mathsf{Sch})$ we can conclude that I'' must satisfy $\mathsf{InfAcc}Q$. So Q holds in I' as required.

We complete the proof of the claim by arguing from the second item to the first. Suppose Q is not contained in $\mathsf{InfAcc}Q$ with respect to the rules in $\mathsf{AcSch}^{\leftrightarrow}(\mathsf{Sch})$. Hence there is an instance $I^{\mathsf{AcSch}^{\leftrightarrow}}$ satisfying the rules of $\mathsf{AcSch}^{\leftrightarrow}(\mathsf{Sch})$ and also satisfying $Q \wedge \neg\mathsf{InfAcc}Q$. 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 claim that a fact $R(e_1 \ldots e_n)$ in the accessible part of I_1 is in the accessible part of I_2. This is proven by induction on the point in which the fact goes into the accessible part, using the forward accessibility axioms. Arguing symmetrically (now using the backward axioms), we find that I_1 and I_2 have the same accessible part, and hence they contradict access-determinacy. ◀

We are now ready to state our main results on the relationship between semantic properties, entailments, and plans.

▶ **Theorem 23.** *For any Boolean* RQFO *query Q and access schema* Sch *with constraints specified in* RQFO*, there is an RA-plan answering Q (over instances satisfying* Sch*) if and only if $Q \wedge \mathsf{AcSch}^{\leftrightarrow}(\mathsf{Sch}) \models \mathsf{InfAcc}Q$. If the query and constraints do not include equality, then the RA-plan will not make use of equality in any of its RA expressions.*

Further, from any tableau proof witnessing $Q \wedge \mathsf{AcSch}^{\leftrightarrow}(\mathsf{Sch}) \models \mathsf{InfAcc}Q$ we can extract (in linear time) an RA-plan for Q over Sch*.*

Using Claim 22, we can restate Theorem 23:

> For any Boolean RQFO query Q and access schema Sch with constraints specified in RQFO, there is an RA-plan answering Q (over instances of Sch) if and only if Q entails $\mathsf{InfAcc}Q$ with respect to the rules in $\mathsf{AcSch}^{\leftrightarrow}(\mathsf{Sch})$ if and only if Q is access-determined.

In the direction from right to left we are again going from a preservation property to a syntactic restriction. Thus Theorem 23 can be considered as an "access-restricted variant" of the definability theorems given in the vocabulary-based case.

6.2 The Interpolation Result Behind the Rewriting Results: Access Interpolation

We have instantiated the first steps of our meta-algorithm for reformulation via relational algebra access plans. For RA-plans, we have stated a semantic property of a query that is required for it to have an equivalent target RA plan. And we have shown that the semantic property is equivalent to an entailment. The next ingredient in proving the main results relating plans to proofs of these entailments is an interpolation theorem that tracks the "access methods used in the interpolant".

Interpolation and Access Methods. Recall Proposition 19. It states that Boolean RA-plans have the same expressiveness as RQFO sentences that are executable for membership checks. For simplicity, we call these "executable FO Boolean queries" below.

By this corollary, we can prove our main results by finding reformulations that are executable FO Boolean queries, rather than RA-plans. To find these reformulations, we require a version of Craig interpolation that allows us to relate the "binding patterns" — i.e. the subset of variables that are quantified in each relativized-quantifier quantification — used in the interpolant χ of the entailment $\lambda \models \rho$ with those used in λ or ρ. When we apply this theorem to the entailment of $\mathsf{InfAcc}Q$ by Q, we can conclude that the interpolant is an executable FO Boolean query.

We associate to RQFO 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 structural induction on the formula, assuming that the formula is first put into negation normal form.

$$\begin{array}{lcl}
\mathsf{BindPatt}(\top) \;=\; \mathsf{BindPatt}(x=y) & = & \emptyset \\
\mathsf{BindPatt}(R(t_1,\dots,t_n)) & = & \{(R,\{1,\dots,n\})\} \\
\mathsf{BindPatt}(\neg\varphi) & = & \mathsf{BindPatt}(\varphi) \\
\mathsf{BindPatt}(\varphi\wedge\psi) & = & \mathsf{BindPatt}(\varphi)\cup\mathsf{BindPatt}(\psi) \\
\mathsf{BindPatt}(\varphi\vee\psi) & = & \mathsf{BindPatt}(\varphi)\cup\mathsf{BindPatt}(\psi) \\
\mathsf{BindPatt}(\exists\vec{x}\ (R(t_1,\dots,t_n)\wedge\varphi)) & = & \mathsf{BindPatt}(\varphi)\cup\{(R,\{i\mid t_i\notin\vec{x}\})\} \\
\mathsf{BindPatt}(\forall\vec{x}\ (R(t_1,\dots,t_n)\rightarrow\varphi)) & = & \mathsf{BindPatt}(\varphi)\cup\{(R,\{i\mid t_i\notin\vec{x}\})\}
\end{array}$$

For example,

$$\mathsf{BindPatt}(\exists x\,\exists y\ (R(x,y)\wedge\forall z\ (S(x,y,z)\rightarrow U(x,y,z)))) = \{(R,\emptyset),(S,\{1,2\}),(U,\{1,2,3\})\}$$

We can similarly talk about the *universal binding patterns* of a formula — those that arise from a subformula $\forall\vec{x}\ (R(t_1,\dots,t_n)\rightarrow\varphi)$ — and the *existential binding patterns* — those that arise from a subformula $\exists\vec{x}\ (R(t_1,\dots,t_n)\wedge\varphi)$. These can be defined formally by modifying the inductive definition above. Note that every binding pattern of a formula is either a universal pattern or an existential pattern, since a non-quantified relation can be considered as a vacuous case of existential quantification (if occurring positively) or universal quantification (if occurring negatively).

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 a sentence φ, if for each pattern $(R\{s_1\dots s_j\})$ in $\mathsf{BindPatt}(\varphi)$ Sch contains an access method on R whose input positions are contained in $\{s_1\dots s_j\}$, then φ is an executable FO Boolean query. We say that a binding pattern $(R_1,\{s_1\dots s_j\})$ is *covered by* another binding pattern $(R_2,\{t_1\dots t_k\})$ if $R_1=R_2$ and $\{s_1\dots s_j\}$ is a superset of $\{t_1\dots t_k\}$. To show that a sentence φ_1 is an executable FO Boolean query, it suffices to get another executable FO Boolean query φ_2 whose binding patterns cover those of φ_2.

Recall that a relation 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 $\neg\psi\vee\chi$. For example, in the formula $\forall x\ (P(x)\rightarrow\exists y\ R(x,y))$, the relation P occurs negatively and the relation R occurs positively.

▶ **Theorem 24** (Access interpolation). *Let λ and ρ be* RQFO *sentences such that $\lambda\models\rho$. Then there exists an* RQFO *sentence χ such that*

1. *$\lambda\models\chi$ and $\chi\models\rho$.*
2. *A relation occurs positively (negatively) in χ only if it occurs positively (negatively) in both λ and ρ.*
3. *A constant occurs in χ only if it occurs both in λ and ρ.*

4. *Every existential binding pattern of χ is covered by an existential binding pattern of ρ, and the relation it binds occurs positively in λ. Every universal binding pattern of χ is covered by a universal binding pattern of λ, and the relation it binds occurs negatively in ρ.*
5. *If λ and ρ are both equality-free, then χ is equality-free.*

Furthermore, χ can be computed in polynomial time from a proof (in a suitable proof system) of the entailment $\lambda \models \rho$.

We show how Theorem 24 suffices to prove Theorem 23. Recall the theorem:

> For any Boolean RQFO query Q and access schema Sch containing constraints expressible in RQFO, there is an RA-plan answering Q (over instances in Sch) if and only if Q entails InfAcc$Q$ with respect to the rules in AcSch$^{\leftrightarrow}$(Sch). If the query and constraints do not include equality, then the RA-plan will not make use of equality in any of its relational algebra expressions.

From Claim 22, we obtain the "plan-to-proof" direction of Theorem 23. Suppose Q does not imply InfAcc$Q$ with respect to the rules in AcSch$^{\leftrightarrow}$(Sch). By Claim 22, Q is not access-determined, and it follows that no plan can answer Q.

For the "proof-to-plan" direction of Theorem 23, we assume Q entails InfAcc$Q$ and construct an RA-plan that answers Q. We will use a slight modification of the axiom schema AcSch$^{\leftrightarrow}$, denoted AltAcSch$^{\leftrightarrow}$ in which the relation accessible does not appear. In every forward accessibility axiom an atom accessible(x) on the left is replaced by a relation InfAcc$R(\vec{z})$, where $\vec{z}$ contains x in at least one position (and the other variables are universally quantified). Occurrences of accessible on the right are dropped. For example, the axiom accessible$(x) \wedge R(x, y) \rightarrow$ InfAcc$R(x, y) \wedge$ accessible(y) would be replaced by many axioms, including InfAcc$S(x, w, z) \wedge R(x, y) \rightarrow$ InfAcc$R(x, y)$. We also allow all variants of these axioms in which free variables corresponding to input positions on the left are substituted by schema constants. Similarly, in every backward accessibility axiom we replace accessible(x) on the left by an atom in the original schema containing x, again dropping occurrences of accessible on the right.

▶ **Proposition 25.** *Q proves* InfAcc$Q$ *using the axioms of* AcSch$^{\leftrightarrow}$ *if and only if Q proves* InfAcc(Q) *in the modified schema* AltAcSch$^{\leftrightarrow}$*.*

Proof. In one direction, suppose that Q does not prove InfAcc$Q$ using the axioms of AcSch$^{\leftrightarrow}$. Then by Claim 22, Q is not access-determined. Fix I and I$'$ instances for the original schema satisfying Σ with the same accessible part, but disagreeing on the output of Q. Without loss of generality, we can assume that there is a tuple $\vec{d}$ in the output of Q on I but not in the output of Q on I$'$. By applying an isomorphism to non-accessible values of I$'$ and I, we can assume that every non-accessible value of I$'$ is not in I and vice versa. Let I* be the instance in which relations R are interpreted as in I and relations InfAcc$R$ are interpreted, as in I$'$, by a new relation symbol.

We argue that I^* satisfies $\mathsf{AltAcSch}^{\leftrightarrow}$. Clearly both the original relations and the relations $\mathsf{InfAcc}R$ satisfy Σ. Consider a modified forward accessibility axiom (universal quantifiers omitted, and with no schema constants for simplicity):

$$\mathsf{InfAcc}R_{j_1}(\ldots x_{j_1}\ldots) \wedge \ldots \mathsf{InfAcc}R_{j_m}(\ldots x_{j_m}\ldots) \wedge R(\vec{x}) \rightarrow \mathsf{InfAcc}R(\vec{x})$$

where R has an access method on positions $j_1 \ldots j_m$. Suppose we have a tuple $c_1 \ldots c_m$ in I^* satisfying the left-hand side of this implication. Then $c_{j_1} \ldots c_{j_m}$ must be accessible values of I and I'. Since the fact $R(\vec{c})$ holds in I, it must be in the accessible part of I, and hence in the accessible part of I'. So $\mathsf{InfAcc}R(\vec{c})$ holds in I^* as required. The backward accessibility axioms are argued symmetrically.

The tuple $\vec{d}$ is in $[\![Q]\!](\mathsf{I}^*)$, since it is in $[\![Q]\!](\mathsf{I})$ and Q is a formula using the relations in the original schema. But $\vec{d}$ is not in $[\![Q]\!](\mathsf{I}')$. Therefore $\vec{d}$ can not be returned by $\mathsf{InfAcc}Q$ on I^*. So I^* witnesses that Q does not prove $\mathsf{InfAcc}Q$ in $\mathsf{AltAcSch}^{\leftrightarrow}$.

In the other direction, suppose we have I^* witnessing that Q does not prove $\mathsf{InfAcc}(Q)$ in $\mathsf{AltAcSch}^{\leftrightarrow}$. Expand I^* to an instance I^+ for the signature extended with $\mathsf{accessible}$, by interpreting $\mathsf{accessible}$ by all values of schema constants unioned with all values that lie in the domain of some $\mathsf{InfAcc}R$ relation and in the domain of some relation of the original schema. We show that the resulting instance satisfies the constraints of $\mathsf{AcSch}^{\leftrightarrow}$. The accessibility axioms follow directly from the corresponding axioms of $\mathsf{AltAcSch}^{\leftrightarrow}$. Similarly, the output of Q in I^+ is the same as the output of Q in I, while the output of $\mathsf{InfAcc}Q$ on I^+ is the same as the output of $\mathsf{InfAcc}Q$ on I^*. Therefore Q does not prove $\mathsf{InfAcc}Q$ in $\mathsf{AcSch}^{\leftrightarrow}$. ◀

We can rephrase the modified assumption on Q as:

$$Q \wedge \Sigma_1 \models \Sigma_2 \rightarrow \mathsf{InfAcc}Q$$

where Σ_1 contains the Sch constraints along with the "backwards" accessibility axioms going from $\mathsf{InfAcc}R$ relations to R relations within $\mathsf{AltAcSch}^{\leftrightarrow}$, while Σ_2 contains the forward accessibility axioms and the $\mathsf{InfAccCopy}$ constraints.

By the access interpolation theorem there is an RQFO formula χ in negation normal form such that:

1. $Q \wedge \Sigma_1 \models \chi$
2. $\chi \models \Sigma_2 \rightarrow \mathsf{InfAcc}Q$
3. a relation occurs positively (respectively negatively) in χ if and only if it occurs positively (resp. negatively) on both sides of the entailment.
4. for a binding pattern p on a relation U in χ if p is existential, then it is covered by an existential pattern in $\Sigma_2 \rightarrow \mathsf{InfAcc}Q$, and U occurs positively in $Q \wedge \Sigma_1$. If p is a universal pattern, then it is covered by a universal pattern of $Q \wedge \Sigma_1$ and U occurs negatively in $\Sigma_2 \rightarrow \mathsf{InfAcc}Q$.

Because we are dealing with the modified axioms $\mathsf{AltAcSch}^{\leftrightarrow}$, the only relations in the entailment are the original schema relations and their InfAcc copies. Occurrences of the original schema relations R on the right-hand side of the entailment are all in the forward accessibility axioms, and they correspond exactly to access methods of the schema. It follows that every existential pattern over the relations R in χ is of the proper form. Furthermore the relations R occur only positively on the right — since $\Sigma_2 \to \mathsf{InfAcc}Q$ in negation normal form is $(\sim \Sigma_2) \vee \mathsf{InfAcc}Q$, and $\sim$ applied to a forward accessibility axiom is of the form $\exists \vec{x}\ R(\vec{x}) \wedge \ldots \wedge \neg \mathsf{InfAcc}R(\vec{x})$. So relations R can occur only positively in χ, and hence can occur only in existential patterns of χ. We conclude that all the patterns in χ involving the relations of the original schema must be existential and of the proper form.

Now let us examine occurrences of relations of the form $\mathsf{InfAcc}R$ within χ. The last property of χ above implies that any existential pattern on relations of the form $\mathsf{InfAcc}R$ would need to correspond to a positive occurrence of $\mathsf{InfAcc}R$ in $Q \wedge \Sigma_1$. But there are no such occurrences. Any universal pattern on relations of the form $\mathsf{InfAcc}R$ would need to correspond to a universal pattern in $Q \wedge \Sigma_1$, and hence must be covered by the use of $\mathsf{InfAcc}R$ in a backward axiom. Hence all such occurrences are covered by an access method of the schema.

We conclude that χ is an executable FO Boolean query. Let $\mathsf{DeAcc}(\chi)$ be the result of changing all relations of the form $\mathsf{InfAcc}R$ in χ to R. We claim that $\mathsf{DeAcc}(\chi)$ is an executable rewriting of Q (and hence can be converted to an RA-plan using Proposition 19).

We justify this by proving containments between Q and $\mathsf{DeAcc}(\chi)$ in both directions. Suppose tuple $\vec{t}$ is returned by Q on an instance I of Sch. Let I' be the instance for the augmented schema in which both $\mathsf{InfAcc}R$ and R relations are interpreted as in I. We can see that I' satisfies the constraints in Σ_1 and Σ_2.

Applying the first condition on an interpolant above, we infer that χ holds of $\vec{t}$ in I', which implies that χ holds of $\vec{t}$ in I. But $\mathsf{DeAcc}(\chi)$ evaluated on I yields the same set of tuples as evaluating χ on I'. So $\vec{t}$ satisfies $\mathsf{DeAcc}(\chi)$ in I.

In the other direction, suppose tuple $\vec{t}$ satisfies $\mathsf{DeAcc}(\chi)$ in I. Then letting I' be as above, we have that $\vec{t}$ satisfies χ in I'. By the second property of an interpolant we have that $\Sigma_2 \to \mathsf{InfAcc}Q$ holds in I'. Since I' satisfies Σ_2, $\vec{t}$ is returned by $\mathsf{InfAcc}Q$ in I'. This tells us that $\vec{t}$ is returned by Q as required.

This completes the proof of the first assertion in Theorem 23. The assertion about equality-free queries and constraints follows since the Access interpolation theorem produces an equality-free interpolant when the entailment involves equality-free formulas, and the other transformations (e.g., from nested plans to RA-plans) do not introduce equality.

6.3 Variants of the Results for $USPJ^{\neq}$-Plans

We now state an analogous result for negation-free plans, which will be an access-related variant of the Projective Monotone Preservation Theorem, Theorem 14.

Here we use the schema $\mathsf{AcSch}(\mathsf{Sch})$ obtained from $\mathsf{AcSch}^{\leftrightarrow}$ by throwing away all the backward accessibility axioms, and keeping only the forward ones.

▶ **Theorem 26.** *For any Boolean* RQFO *query Q and access schema* Sch *containing constraints specified in* RQFO*, the following are equivalent:*

- *there is a $USPJ^{\neq}$-plan answering Q (over instances in* Sch*)*
- *Q entails $\mathsf{InfAcc}Q$ with respect to $\mathsf{AcSch}(\mathsf{Sch})$ ($Q \wedge \mathsf{AcSch}(\mathsf{Sch}) \models \mathsf{InfAcc}Q$).*

Furthermore, for every tableau proof witnessing $Q \wedge \mathsf{AcSch}(\mathsf{Sch}) \models \mathsf{InfAcc}Q$, we can extract a $USPJ^{\neq}$-plan. If the query and the constraints of Sch *are specified by* RQFO *formulas without any equalities (e.g., TGDs), then we can replace $USPJ^{\neq}$-plan with $USPJ$-plan in the above statement. So we do not need inequalities in the reformulation unless we have equalities in the constraints.*

The Schema $\mathsf{AcSch}^{\neg}$ and $USPJAD^{\neq}$-Plans. We now state an extension of the "Projective Łos-Tarski theorem", Theorem 17, to the setting of access methods, again focusing on the case of Boolean queries.

▶ **Theorem 27.** *For any Boolean* RQFO *query Q and access schema* Sch *containing constraints specified in* RQFO*, the following are equivalent:*

- *there is a $USPJAD^{\neq}$-plan answering Q (over instances for* Sch*)*
- *$Q \wedge \mathsf{AcSch}^{\neg}(\mathsf{Sch}) \models \mathsf{InfAcc}Q$. That is, Q entails $\mathsf{InfAcc}Q$ with respect to $\mathsf{AcSch}^{\neg}(\mathsf{Sch})$.*

Furthermore, from every tableau proof witnessing $Q \wedge \mathsf{AcSch}^{\neg}(\mathsf{Sch}) \models \mathsf{InfAcc}Q$, we can effectively extract a $USPJAD^{\neq}$-plan. If the constraints of Sch *and the query Q are specified by equality-free* RQFO *sentences (in particular, TGDs), then we can replace $USPJAD^{\neq}$-plan by $USPJAD$-plan.*

In the vocabulary-based setting, the entailment $Q \wedge \mathsf{AcSch}^{\neg}(\mathsf{Sch}) \models \mathsf{InfAcc}Q$ reduces to the entailment given in Section 5, which was shown there to be equivalent to the query being induced-subinstance-monotonically-determined. And clearly, in the vocabulary-based setting, a $USPJAD^{\neq}$-plan can be expressed as an $\exists^{\neq}$ formula. Thus Theorem 27 generalizes the "Projective Łos-Tarski Theorem", Theorem 17, characterizing queries that can be reformulated using $\exists^{\neq}$ formulas.

Both of these results are proven in the same way as the result for RA plans. By Proposition 19, instead of dealing with plans as the target language, we deal with RQFO formulas with certain binding patterns. We apply the Access Interpolation Theorem, and then analyze the binding patterns in the resulting formulas, and show that we satisfy the requirements to convert to the associated plan.

7 State of Play and Future Directions

Here we have given a flavor of the use of interpolation in database rewriting problems, focusing on the cases of vocabulary-based restriction and access patterns.

Several other directions have been explored in the literature:

Richer Data Models. The examples we have gone through here both concern relational data - i.e. traditional set-based semantics. In databases there are tools and query languages for other collection types, like bags, nested sets, nested bags, sequences, and nested sequences. The main work we know of beyond relational data is about nested sets, summarized in [9]. The model starts with an infinite scalar type, and builds up objects via iterating tupling and powersets: thus one can have sets of sets of pairs of scalars. Pairs of sets of sets of scalars, etc. There is a standard query language in the literature for this data model, nested relational calculus (NRC). In [9] a notion of implicit definability/determinacy is defined for first-order logic formulas in a simple set-theoretic language. And it shown that any query over nested sets that is determined in this sense can be converted to NRC, effectively from a certain kind of proof. Although at first glance this appears to be a version of Beth Definability for a fragment of set theory, in fact the only set theoretic axiom used is extensionality: sets with equal members are equal. Thus implicit definability in the context of nested data is really a variation of standard implicit definability where the uniqueness is only "up to extensional equivalence". Extensional equivalence is definable in the language of set theory. Thus the question is whether there is a version of Beth's theorem that considers uniqueness up definable equivalence. One of the key insights in [9] is that there is a variation of Beth that applies to this setting, originating in work of Gaifman [20] under the name of "rigid categoricity". While the results on rigid categoricity were proven model theoretically in [20] (see [1]), in [9] a proof-theoretic argument is developed for the special case of extensional-equivalence.

A natural question is what about the case of *bags*: finite multi-sets. While there is no standard semantics for first-order logic over bags, there is a semantics for conjunctive queries, and thus one can talk about determinacy of CQ views over bag semantics. Note that even for Boolean queries, it is not clear whether determinacy is decidable, or whether it implies rewritability in some natural query language for bags. The main results on this topic are in [23], where the case of Boolean views is demystified.

Richer Query Languages over Traditional Data Models. The work we overview deals with fragments of first-order logic. The only other work we are aware of in this space concerns fragments of the recursive query language Datalog, a fragment of fixpoint logic. The first work in this space was [19], which investigates the case of views in a fragment of Datalog, the "regular path queries": it is shown that determinacy implies rewritability, an analog of implicit definability. The case of general Datalog queries and views was investigated in [4, 3]. The only implicit definability results are for fragments of Datalog where containment is decidable — which means, very limited fragments. The main technique used is a variation of uniform interpolation, which hold for limited fixpoint languages [16]. See [21] for a discussion of uniform interpolation algorithms.

Other Proof Systems and Implementations. Toman and Weddell were the first to develop interpolation-methods for query rewriting. Their initial approach, based on a tableaux calculus, is presented in [29], with follow-up work in [30, 18]. A rewriting approach for access patterns was developed in [8, 7], but only in a restricted context where interpolation is not necessary. [5] developed an approach for interpolation-based rewriting where the underlying proof system is resolution. This made use of the standard algorithmic approach to resolution-based interpolation due to Huang [22], which is discussed in [31].

8 Bibliographic Remarks

The precursors of the interpolation-based approach to rewriting are in preservation or "semantics to syntax" theorems in model theory, such as the Łos-Tarski theorem, which states that preservation under extensions is equivalent to being rewritable as an existential formula: see for example [1, 14]. After the work of Lyndon [25], these results would be stated and proved model-theoretically, and effective aspects would be ignored. In addition, the presentations did not include the "projective versions", in which one looks for a restricted formula in a particular vocabulary: these variants are important for the applications in databases.

The idea that Craig interpolation is relevant to query reformulation in databases, from the point of view of theory, dates from the work of Segoufin and Vianu [28], later explored jointly with Nash [26]. Their investigation focused on the case of views — a special case of vocabulary-based reformulation. They phrase their results as showing that relational algebra was sufficient in seeking a reformulation over views.

Toman and Weddell [29] were the first to explore the use of effective interpolation to actually generate rewritings, in the context of query reformulation with constraints. Toman and Weddell also note connections with other query reformulation methods that have an interpolation-like feel, like the 'chase and backchase method", which is used to reformulate queries in the presence of a certain class of background theories, so-called Tuple-Generating Dependencies (TGDs): for an overview of the chase and backchase, see [17].

Much of the material in this chapter here is taken from [6], which extends earlier work [10, 11] looking systematically at interpolation for both vocabulary-based reformulation and rewriting with access methods. In this chapter we have given some samples of vocabulary-based reformulation for "first-order" (or relational algebra) queries, and on access-method based reformulation for the same queries, but the book covers several other cases. The book and the related papers explicitly connect these reformulation methods with preservation theorems and other semantics-to-syntax results in the earlier model theory literature.

The case of so-called "nested relations", which can be considered as a fragment of set theory, is taken from [9]. We have overviewed these results briefly in Section 7.

The case of Datalog (fixpoint) queries and views is considered in [19, 4, 3]. Here the results are related to uniform interpolation, and they work via automata-theoretic methods, as in [16].

References

1 Wilfrid Hodges. *Model Theory.* Cambridge University Press, 1993.

2 Serbe Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases.* Addison-Wesley, 1995.

3 Michael Benedikt, Stanislav Kikot, Johannes Marti, and Piotr Ostropolski-Nalewaja. Monotone rewritability and the analysis of queries, views, and rules. In *KR*, 2024.

4 Michael Benedikt, Stanislav Kikot, Piotr Ostropolski-Nalewaja, and Miguel Romero. On monotonic determinacy and rewritability for recursive queries and views. *TOCL*, 24(2):16:1–16:62, 2023.

5 Michael Benedikt, Egor V. Kostylev, Fabio Mogavero, and Efthymia Tsamoura. Reformulating queries: Theory and practice. In *IJCAI*, 2017.

6 Michael Benedikt, Julien Leblay, Balder ten Cate, and Efthymia Tsamoura. *Generating Plans from Proofs: the Interpolation-based approach to Query Reformulation.* Morgan Claypool, 2015.

7 Michael Benedikt, Julien Leblay, and Efi Tsamoura. PDQ: Proof-driven query answering over web-based data. In *VLDB*, 2014.

8 Michael Benedikt, Julien Leblay, and Efthymia Tsamoura. Querying with access patterns and integrity constraints. *PVLDB*, 8(6):690–701, 2015.

9 Michael Benedikt, Cécilia Pradic, and Christoph Wernhard. Synthesizing nested relational queries from implicit specifications: via model theory and via proof theory. *Log. Methods Comput. Sci.*, 20(3), 2024.

10 Michael Benedikt, Balder ten Cate, and Efi Tsamoura. Generating low-cost plans from proofs. In *PODS*, 2014.

11 Michael Benedikt, Balder Ten Cate, and Efthymia Tsamoura. Generating plans from proofs. *TODS*, 40(4), February 2016.

12 Balder ten Cate and Jesse Comer. Interpolation in first-order logic. In ten Cate et al. [13], chapter 2. Preprint: `https://arxiv.org/abs/2510.03822`. `doi:10.5334/bdg.b`.

13 Balder ten Cate, Jean Christoph Jung, Patrick Koopmann, Christoph Wernhard, and Frank Wolter, editors. *Theory and Applications of Craig Interpolation.* Ubiquity Press, 2026. To appear; preprints accessible from `https://cibd.bitbucket.io/taci/`. `doi:10.5334/bdg`.

14 C. C. Chang and H. J. Keisler. *Model Theory.* North-Holland, 1990.

15 William Craig. Three uses of the Herbrand-Gentzen theorem in relating model theory and proof theory. *Journal of Symbolic Logic*, 22(3):269–285, 1957.

16 Giovanna D'Agostino and Marco Hollenberg. Logical questions concerning the μ-calculus: Interpolation, Lyndon and Łos-Tarski. *J. Symb. Log.*, 65(1):310–332, 2000.

17 Alin Deutsch, Lucian Popa, and Val Tannen. Query reformulation with constraints. *SIGMOD Record*, 35(1):65–73, 2006.

18 Eva Feng, David Toman, and Grant E. Weddell. Magic sets in interpolation-based rule driven query optimization. In *RuleML+RR*, 2022.

19 Nadime Francis, Luc Segoufin, and Cristina Sirangelo. Datalog rewritings of regular path queries using views. *Log. Methods Comput. Sci.*, 11(4), 2015.

20 Haim Gaifman. Operations on relational structures, functors and classes I. In *Proc. of the Tarski Symposium*, volume 25, page 20–40, 1974.

21 Sam van Gool. Uniform interpolation. In ten Cate et al. [13], chapter 9. Preprint: `https://arxiv.org/abs/2512.15391`. `doi:10.5334/bdg.i`.

22 Guoxiang Huang. Constructing Craig interpolation formulas. In *COCOON*, 1995.

23 Jaroslaw Kwiecien, Jerzy Marcinkowski, and Piotr Ostropolski-Nalewaja. Determinacy of real conjunctive queries. the boolean case. In *PODS*, 2022.

24 Maurizio Lenzerini. Data integration: A theoretical perspective. In *PODS*, 2002.

25 Roger C. Lyndon. An interpolation theorem in the predicate calculus. *Pacific Journal of Mathemantics*, 9:129–142, 1959.

26 Alan Nash, Luc Segoufin, and Victor Vianu. Views and queries: Determinacy and rewriting. *TODS*, 35(3), 2010.

27 Martin Otto. An interpolation theorem. *Bulletin of Symbolic Logic*, 6(4):447–462, 2000.

28 Luc Segoufin and Victor Vianu. Views and queries: determinacy and rewriting. In *PODS*, 2005.

29 David Toman and Grant Weddell. *Fundamentals of Physical Design and Query Compilation.* Morgan Claypool, 2011.

30 David Toman and Grant E. Weddell. FO Rewritability for OMQ using Beth Definability and Interpolation. In *DL*, 2021.

31 Christoph Wernhard. Interpolation with automated first-order reasoning. In ten Cate et al. [13], chapter 12. Preprint: `https://arxiv.org/abs/2507.01577`. `doi:10.5334/bdg.l`.