Journal of Universal Computer Science, vol. 15, no. 1 (2009), 33-62
submitted: 23/1/08, accepted: 15/8/08, appeared: 1/1/09 © J.UCS

Dynamic Query Optimization under Access Limitations and Dependencies

Andrea Calì
(University of Oxford[1], United Kingdom
ac@andreacali.com)

Diego Calvanese
(Free University of Bozen-Bolzano, Italy
calvanese@inf.unibz.it)

Davide Martinenghi
(Politecnico di Milano, Italy
martinen@elet.polimi.it)

Abstract: Unlike relational tables in a database, data sources on the Web typically can only be accessed in limited ways. In particular, some of the source fields may be required as input and thus need to be mandatorily filled in order to access the source. Answering queries over sources with access limitations is a complex task that requires a possibly recursive evaluation even when the query is non-recursive. After reviewing the main techniques for query answering in this context, in this article we consider the impact of functional and inclusion dependencies on dynamic query optimization under access limitations. In particular, we address the implication problem for functional dependencies and simple full-width inclusion dependencies, and prove that it can be decided in polynomial time. Then we provide necessary and sufficient conditions, based on the dependencies together with the data retrieved at a certain step of the query answering process, that allow avoiding unnecessary accesses to the sources.

Key Words: Access Limitations, Functional Dependencies, Inclusion Dependencies, Query Optimization
Category: H.2

1 Introduction

In the context of query answering over the Web, queries can be conceived as in the traditional relational setting, but with the extra requirement that certain fields be mandatorily filled in by the user in order to obtain a result. This is the case of data sources that are accessible through web forms, and, more generally, of web services, which typically distinguish between input and output parameters. Consider for example the DBLP site[2], where, in order to obtain a list of publications extracted from an underlying database, either an author or a title

[1] Computing Laboratory and Oxford-Man Institute of Quantitative Finance
[2] http://dblp.uni-trier.de/

has to be specified, while it is not possible to ask directly, e.g., for all publications of a certain conference[3].

Query processing under such access limitations on the sources is significantly more complex than in the traditional case. In fact, for some queries it may happen that the access limitations make it impossible to obtain all the tuples answering the query, because there is at least one source whose mandatory input fields cannot be populated in any way by using join paths with the other sources used in the query. However, even in such a case, it may still be possible to obtain a subset of the answers to the query by using sources that are not necessarily mentioned in the query, but that are part of the schema; in particular, such off-query sources may be used to provide useful bindings for the mandatory fields of the sources in the query.

Example 1. Suppose we have two sources, say A and B. Source A stores information about cars: given a person (required field), A provides model, plate number, and color of the cars owned by the person. Source B provides a list of persons with their address (no field is required in B). Suppose we are searching for all the plate numbers of Ferrari's. Accessing only source A, where the information of interest is stored, is impossible, because we do not know any name of a car owner. But we can retrieve owner names from B, use them to query A, and select from the obtained tuples those in which the car model is Ferrari. ∎

Several works [Rajaraman et al., 1995; Li and Chang, 2000; Li and Chang, 2001b; Florescu et al., 1999; Duschka and Levy, 1997; Calì and Martinenghi, 2008b] have studied the problem of finding the maximal set of answers obtainable in this way, called the maximally contained answer. As discussed in the mentioned works, this possibly lengthier process in general requires the evaluation of a *recursive* query plan, which can be suitably expressed in Datalog.

Accesses to Web data sources are inherently more costly than in a centralized setting, e.g., due to possible delays caused by network links. It is therefore crucial to avoid all unnecessary accesses to the sources while still being able to retrieve the maximally contained answer. *Static* optimizations techniques for the generation of a query plan are discussed in [Li and Chang, 2000; Li and Chang, 2001b; Calì and Martinenghi, 2008b].

In this article we deal with an important optimization phase that has not been addressed before, namely whether one can optimize query-plans at *run-time*, possibly exploiting additional information available about the sources. Indeed, by exploiting knowledge about integrity constraints present on the sources, one may detect during query evaluation that a certain access to a source is useless, in the sense that it may provide only answers that are already known from previous accesses. Relevant classes of integrity constraints that turn out to be

[3] We do not consider here advanced search, which is also supported at DBLP.

of importance for this setting are functional dependencies and simple full-width inclusion dependencies (see, e.g., [Abiteboul et al., 1995], Chapters 8 and 9).

We revisit the solutions proposed for the context of data integration systems in [Calì and Calvanese, 2002], and focuses on conjunctive queries over relational sources with access limitations. In particular, the following results are obtained.

1. We show that the implication problem for functional dependencies and simple full-width inclusion dependencies is decidable in polynomial time.
2. We present a necessary and sufficient condition to determine, given the dependencies, whether during query evaluation a given source has to be accessed or not in order to obtain the maximally contained answer to a query.

As soon as a query plan is available, by result (1) we can derive all possible simple full-width inclusion dependencies and functional dependencies that hold for a set of sources, and by result (2) we can exploit such dependencies for optimizing the query plan at run time.

The rest of the article is organized as follows. In Section 2, we present the technical preliminaries. In Section 3, we quickly survey existing techniques for generating efficient query plans for queries over sources with access limitations. In Section 4, we discuss implication of functional and simple full-width inclusion dependencies. In Section 5, we present the condition for minimizing run-time source accesses and prove its correctness and completeness. In Section 6, we discuss related work, and finally, in Section 7, we conclude the article.

2 Accessing sources with access limitations

We present the formal framework in which we address query optimization. We consider conjunctive queries over sources with access limitations. Let $\mathcal{S}$ be a source schema, i.e., a set of relational symbols, each with an associated arity. A *conjunctive query* (CQ) q of arity n over $\mathcal{S}$ is written in the form

$$q(X_1, \ldots, X_n) \leftarrow conj(X_1, \ldots, X_n, Y_1, \ldots, Y_m)$$

where $conj(X_1, \ldots, X_n, Y_1, \ldots, Y_m)$ is a conjunction of atoms involving the variables $X_1, \ldots, X_n, Y_1, \ldots, Y_m$ and constants, and the predicate symbols of the atoms are in $\mathcal{S}$. We assume that the query is *safe*, i.e., that each variable X_j appears in at least one atom in $conj$.

Given a database DB for $\mathcal{S}$, consisting of one relation of appropriate arity for each symbol in $\mathcal{S}$, the answer $q(DB)$ to q over DB is the set of tuples $(c_1, \ldots, c_n)$ of constants in DB such that there are constants $d_1, \ldots, d_m$ in DB, such that each atom in $conj(c_1, \ldots, c_n, d_1, \ldots, d_m)$ holds in DB.

To each attribute in a source we associate a domain, which specifies the legal values for that attribute. Instead of using concrete domains, such as `Integer`

or `String`, we deal with *abstract domains*, which have an underlying concrete domain, but represent information at a higher level of abstraction. This makes it possible to distinguish, e.g., strings representing person names from strings representing plate numbers.

Formally, we have:

- a *source schema* $\mathcal{S}$, which is a set of *relational symbols*, each with an associated *arity*, a tuple of abstract domains, and a *binding pattern*. In the following, we will call the symbols in $\mathcal{S}$ simply *sources*. However, when providing intuitions, we will use the term "source" also to refer to the actual data sources with access limitations corresponding to the binding pattern of the associated source symbol. The meaning will be clear from the context. We call *bound* the attributes that must mandatorily be provided with a constant in order to query the source, and *free* the remaining ones. The binding pattern specifies which subset of the attributes of the source are bound and which ones are free. In the examples, we specify sources with their abstract domains, and underline the abstract domains in the positions of bound attributes.
- a *user query*, which is a CQ over $\mathcal{S}$.

In order to answer a user query, one has to compute a *query plan* specifying how to access the sources. In the presence of access limitations on the sources, traditional query answering is in general not sufficient to extract all obtainable answers from the sources, as shown by the following example.

Example 2. Consider the following sources with access limitations:
$s_1(\mathit{Title}, \underline{\mathit{Year}}, \mathit{Artist})$, which stores data about songs, and $s_2(\underline{\mathit{Artist}}, \mathit{Nation})$, which stores artists with their nationality. Consider the following user query

$$q(A) \leftarrow s_2(A, \mathit{italian})$$

asking for names of Italian artists. In this case, q cannot be immediately evaluated over the sources, since s_2 requires the first attribute to be bound to a constant. Therefore, with traditional query answering techniques we cannot extract any answer to q, no matter what the extension of the source relations. However, if we knew some constants for the abstract domain "year", we could access s_1, extract artist names from it, and access s_2 with these, potentially extracting tuples that answer the query. ∎

The example shows how an apparently useless, off-query source may provide constants with which useful information can be retrieved.

In order to formally characterize the notion of maximally contained answer, we need to introduce the notion of access to a source with access limitations.

Intuitively, an access is the smallest legal operation that can be performed on sources with access limitations.

Definition 1 (Access). An *access* to a source s is a CQ of the form

$$q(X_1, \ldots, X_m) \leftarrow s(Z_1, \ldots, Z_n)$$

(i.e., whose body is a single atom) such that each of the variables $X_1, \ldots, X_m$ occurs exactly once in q's body, and such that, for $1 \leq i \leq n$,

- if the i-th position in s is bound, then Z_i is a constant of the corresponding abstract domain;
- if the i-th position in s is free, then Z_i is one of $X_1, \ldots, X_m$.

The constants used in the body are called *input values*. A tuple consisting of the input values of an access is called a *binding*. The answer to an access in a database DB is called the *extraction* made by the access in DB. ∎

A source can be accessed if all its bound arguments have been instantiated with constants. Then, as soon as new constants are extracted with an access, these can be used to make more accesses. Note that, consistently with what is commonly done in the literature, it is assumed that the strategy of enumerating all possible elements of a given domain to access a source is not feasible and that, rather, the values for the bound positions of a source are obtained either from constants in the query or from tuples retrieved from other sources. For instance, in the evaluation of the query of Example 2, one cannot expect to enumerate all possible artist names without retrieving this information from the sources. The sequences of accesses that are needed to retrieve data in order to answer a query are captured by the notion of access plan.

Definition 2 (Access plan). An *access plan*, given a set I of constants, is a sequence of source accesses such that the input values used in an access consist of constants either in I or coming from tuples in the extraction of previous accesses in the sequence.

For each database instance, each query should then be associated with an access plan that extracts the query answer. This is precisely what a query plan does.

Definition 3 (Query plan). A *query plan* Π for an n-ary query q over a sources schema $\mathcal{S}$, given a set I of constants including the constants in q, is a function that maps a database instance DB into a pair $\Pi(DB) = (A, T)$, where

- A is an access plan given I, and

– T is a set of n-ary tuples, called the *obtained answer*, consisting of constants in the extractions of A or in I.

Π is *sound* whenever, for each database DB, if $\Pi(DB) = (A, T)$ then $T \subseteq q(DB)$.

Π is *non-prescient* if, for every pair of databases DB_1 and DB_2 such that $\Pi(DB_1) = (A_1, T_1)$ and $\Pi(DB_2) = (A_2, T_2)$ hold, for every $i \geq 0$, if A_1 and A_2 coincide on the first i accesses and the corresponding extractions (cf. Definition 1) in T_1 and T_2, then A_1 and A_2 also coincide on the $(i+1)$-st access. ∎

Note that a non-prescient query plan essentially is a query plan that is deterministic at the level of the single accesses to the sources. This follows immediately from observing that A_1 and A_2 necessarily coincide before any access (i.e., when $i = 0$); then each subsequent access and the corresponding extraction are uniquely determined.

Definition 4 (Maximally contained answer). Given a source schema $\mathcal{S}$, a query q, a set I of constants including the constants in q, and a database DB, a tuple in $q(DB)$ is $(I, \mathcal{S})$-*obtainable* if it is a tuple in the answer obtained for DB by some sound and non-prescient query plan for q over $\mathcal{S}$ given I. The set of $(I, \mathcal{S})$-obtainable answer tuples in $q(DB)$ is denoted $\text{ans}(q, \mathcal{S}, DB, I)$ and is called *maximally contained answer*. ∎

A similar definition of maximally contained answer is given in [Millstein et al., 2000; Millstein et al., 2003], where the authors call it *reachable certain answer* in their particular setting; there, the authors restrict query plans to be Datalog programs, and therefore do not need to impose non-prescience, which is implicit in the semantics of Datalog, once an evaluation strategy for the Datalog program has been fixed (e.g., by considering the rules in the order in which they are written, the atoms in the rules from left to right, and by assuming an order on the database facts according to which they are accessed). The definition proposed here is different from theirs, in that it allows query plans to be implemented in any programming language, including Turing-complete ones, which are more expressive than Datalog. However, non-prescience is required to correctly characterize the notion of query plan, since a prescient query plan could, in principle, employ *ad hoc* access plans for each database, thus retrieving an answer tuple without actually checking that it is in the answer, but only guessing it. This is clarified in the next example.

Example 3. Consider a CQ $q(X) \leftarrow r_1(X), r_2(X, Y)$ over the source schema $\mathcal{S} = \{r_1(A), r_2(\underline{A}, \underline{B})\}$, given an empty set of initial constants. Suppose the database is $DB = \{r_1(a), r_2(a, b)\}$. Clearly, a prescient plan could access r_1,

thus extracting constant a, and then, in principle, claim that $\langle a \rangle \in q(DB)$, even without accessing r_2. Conversely, after making the same access to a database $DB' = \{r_1(a)\}$, it could guess that $\langle a \rangle \notin q(DB')$. However, DB and DB' are indistinguishable as for the accesses made by the plan, because r_2 is inaccessible according to the limitations. Indeed, any non-prescient plan reporting that $\langle a \rangle \in q(DB)$ would necessarily be unsound, since it should then also conclude that $\langle a \rangle \in q(DB')$, which does not hold. Therefore, $\langle a \rangle$ is not obtainable with respect to the access limitations according to Definition 4. ∎

In general, $\text{ans}(q, \mathcal{S}, DB, I) \subseteq q(DB)$, and therefore, in the presence of access limitations on the sources, queries cannot be evaluated as in the traditional case. The next section will present techniques, all based on Datalog, to extract the maximally contained answer to a conjunctive query.

3 Existing techniques for static optimization of query plans

In the previous section, we have presented the notion of *query plan*, which is intended to retrieve exactly all the tuples in the maximally contained answer to a conjunctive query expressed on a relational schema with access limitations.

An algorithm to construct a naive, recursive query plan that is able to compute the maximally contained answers $\text{ans}(q, \mathcal{S}, DB, I)$ for given q, $\mathcal{S}$, DB, and I, is presented in [Li and Chang, 2000]. The program is expressed in Datalog, and the key idea in it is that some of the clauses suitably encode the access limitations that are to be considered during the evaluation; this ensures that every answer returned by the plan is indeed obtainable through the extraction process that we have presented in Section 2. Notice that the fact that one considers Datalog plans does not imply that in general one has to restrict the attention to plans expressed in Datalog; indeed, in the results presented in the rest of the paper, we shall make no assumption on the language that is used to express the plan; in general, the plan can be encoded as a Turing machine.

We describe now the construction of a recursive plan according to [Li and Chang, 2000].

Given a set I of constants and a CQ of the form $q(\mathbf{X}) \leftarrow s_1(\mathbf{Z}_1), \ldots, s_k(\mathbf{Z}_k)$ over a source schema $\mathcal{S}$, a new predicate dom_A, called *domain predicate*, is introduced for every abstract domain A used in $\mathcal{S}$ ($\mathbf{X}$ is a sequence of variables and each $\mathbf{Z}_i$ is a sequence of constants and variables). Such a predicate will be used to represent constants belonging to A. Its extension, at every step of the execution of the plan, will consist of all constants of A that have been extracted so far (plus those already in I, if any).

The program consists of the following rules:

– A fact $dom_A(a)$ for every constant $a \in I$ with abstract domain A.

– A rule, called *cache rule*, of the form

$$\hat{s}(\mathbf{X}) \leftarrow s(\mathbf{X}), dom_{i_1}(X_{i_1}), \ldots, dom_{i_n}(X_{i_n})$$

for every source $s \in \mathcal{S}$, where $\hat{s}$ is a fresh predicate, called s's cache, $\mathbf{X}$ is a sequence of distinct variables, $i_1, \ldots, i_n$ are all the bound positions in s, $dom_{i_1}, \ldots, dom_{i_n}$ are the domain predicates of the corresponding abstract domains, and $X_{i_1}, \ldots, X_{i_n}$ are the variables occurring in those positions in $\mathbf{X}$. Cache rules are crucial in the construction, since they ensure that sources with limitations are accessed *only* by having a selection on the bound positions that uses constants already extracted (or present in I).

– A rule, called *domain rule*, of the form

$$dom_A(X_i) \leftarrow \hat{s}(X_1, \ldots, X_{i-1}, X_i, X_{i+1}, \ldots, X_n)$$

for every output argument with abstract domain A in every source $s \in \mathcal{S}$, where i is its position in s, and $X_1, \ldots, X_n$ are distinct variables. Domain rules serve to populate the domain predicates with all the extracted constants.

– A rule of the form $q(\mathbf{X}) \leftarrow \hat{s}_1(\mathbf{Z}_1), \ldots, \hat{s}_k(\mathbf{Z}_k)$, which is the rewrite of the query over the cache predicates.

The evaluation of q over this program will start from the set of initial values, stored in the facts, to populate the domain predicates; the program will then access all the sources it can, according to their access limitations, via the cache rules, to populate the caches. With the new facts obtained (if any), new values will be added to the domain predicates via the domain rules. With these, it will repeat the process and access the sources again, until there is no way of making new accesses, i.e., a fixpoint has been reached. The returned answer consists of all facts obtainable while respecting the access limitations, and is therefore the same as $\text{ans}(q, \mathcal{S}, DB, I)$.

The construction of the Datalog program for the query of Example 2 is shown in the next example.

Example 4. Consider again Example 2, with sources s_1 and s_2. Assuming that the constant 1998 is known for the abstract domain "year", the Datalog program generated by the algorithm of [Li and Chang, 2000] for the query $q(A) \leftarrow s_2(A, italian)$ is shown in Figure 1.

The query is rewritten over the caches (rule ρ_1), which are defined in the cache rules ρ_2 and ρ_3; these also ensure that the facts that are stored in the caches are retrieved from the sources according to the access limitations. Rules $\rho_4 - \rho_6$ are the domain rules. Finally, ρ_7 and ρ_8 are facts assigning the right abstract domain to the initial constants. ∎

$$
\begin{array}{lrcl}
\rho_1: & q(A) & \leftarrow & \hat{s}_2(A, italian) \\
\rho_2: & \hat{s}_1(T, Y, A) & \leftarrow & s_1(T, Y, A), dom_Y(Y) \\
\rho_3: & \hat{s}_2(A, N) & \leftarrow & s_2(A, N), dom_A(A) \\
\rho_4: & dom_T(T) & \leftarrow & \hat{s}_1(T, Y, A) \\
\rho_5: & dom_A(A) & \leftarrow & \hat{s}_1(T, Y, A) \\
\rho_6: & dom_N(N) & \leftarrow & \hat{s}_2(A, N) \\
\rho_7: & dom_N(italian) & \leftarrow & \\
\rho_8: & dom_Y(1998) & \leftarrow &
\end{array}
$$

Figure 1: Datalog program for Example 4

The problem of such recursive query plans is that their evaluation usually requires a large number of accesses to data sources; moreover, such sources may be on the web, so that accessing them can be slow. It is therefore important to optimize the query plans so as to reduce as much as possible the number of accesses. A first kind of optimization is the *static* optimization, which is done at the intensional level, by modifying the query plan so that it will avoid certain accesses. An example of static optimization is presented in [Li and Chang, 2000; Li and Chang, 2001b; Li and Chang, 2001a]. All of these works consider a class of queries called *connection queries*, which are a strict subclass of union of conjunctive queries and incomparable with conjunctive queries. Roughly speaking, connection queries capture selection, projection and a limited form of equi-join, but not Cartesian products: the attributes with the same name must be all in an equi-join[4]. This makes it impossible for connection queries to express some simple and common conjunctive queries, e.g., asking for the grandparent-grandchild pairs that can be extracted from a binary relation, $parent(Par, Child)$, because in the conjunctive query $q(X, Y) \leftarrow parent(X, Z), parent(Z, Y)$ the first arguments of the two occurrences of *parent* are not equi-joined, and neither are the second ones. The optimization for connection queries is based on a hypergraph encoding of the query and of the correspondence between the abstract domains of the different sources: with a technique based on a notion similar to reachability (called backward-closure), the sources that are relevant with respect to a query are determined, and the others discarded.

A technique to determine relevant sources for the class of conjunctive queries is described in [Calì and Martinenghi, 2008b], a work describing its application in a system capable of providing integrated access to a set of sources with limitations. The technique is based on a graph representation of the query and the sources that are not in the query, together with the correspondence among the abstract domains of the attributes of the various sources. Two mutually-recursive

[4] There, the notion of abstract domain is implicit: abstract domains are represented by attribute names, so the same attribute name means the same abstract domain.

functions are able to suitably prune the graph, discarding non-relevant sources. Also, [Calì and Martinenghi, 2008b] describes a technique for constructing, starting from a pruned graph, a query plan that *minimizes* the number of accesses, discarding "at compile time" (we remind that this optimization is static) all accesses that are useless *for every instance* of the sources.

The above static optimizations are based on the same theoretical framework, which is the same that we described in Section 2. However, this framework is unfit to model the case where the same source is accessible with different input/output patterns; this is a common case in data accessible through web forms, where usually at least one of the fields is to be filled in, but there is no fixed input pattern. Indeed, it is possible to represent this real-world case by introducing a separate source symbol, for every input/output pattern admitted for a certain data source (notice that in general the number of such source symbols can be exponential in the number of attributes of the original data source). For example, if a data source $r(A, B, C)$ requires at least one attribute to be selected, we represent it by means of the following source symbols with binding patterns: $r(\underline{A}, B, C)$, $r(A, \underline{B}, C)$, $r(A, B, \underline{C})$, $r(\underline{A}, \underline{B}, C)$, $r(A, \underline{B}, \underline{C})$, $r(\underline{A}, B, \underline{C})$, $r(\underline{A}, \underline{B}, \underline{C})$. However, while this correctly represents the initial source, there is some additional information that we know, and that is due to the fact that all these source symbols are actually mapped to the same data source: all obtained sources with access limitations have the same extension, in every database instance. Therefore, since the above techniques assume arbitrary extensions of the sources, they cease to be optimal because they do not make use of this additional information.

In the following, we shall introduce *relational constraints* that are able to represent this additional knowledge (and more); we will then devise techniques to employ such information *during the execution of the plan* (from which the denomination as *dynamic* optimization) in order to avoid unnecessary accesses.

4 Implication of Functional and Simple Full-Width Inclusion Dependencies

In the previous section we have surveyed techniques for optimization of the query plan that allow one to retrieve all obtainable tuples from a set of sources, given a query over the source schema. Notice that such an optimization is performed at compile time, and it is obviously relevant for every source database.

However, further optimization techniques, to be applied during the query evaluation process, can be conceived. Such techniques take into account tuples already extracted from the sources at a certain step of the evaluation of the Datalog program associated to a query. They exploit simple full-width inclusion dependencies and functional dependencies on the source relations so as to determine in advance, at a certain step of the evaluation of the Datalog program,

whether an access is potentially useful for the answer, i.e., whether it could return tuples with new values.

In order to fully characterize our dynamic optimization, which will be described in the next section, we now formally introduce the kind of integrity constraints that we use for source access optimization, namely simple full-width inclusion dependencies and functional dependencies. In the following, we denote sets of attributes (i.e., positions) with boldface letters, and we use $\mathcal{A}(s)$ to denote the set of attributes of a source s. Given a database DB, we denote the extension of s in DB with s^{DB}. Given a set of attributes $\mathbf{A} \subseteq \mathcal{A}(s)$, and a tuple $t \in s^{DB}$, we denote with $t[\mathbf{A}]$ the projection of t over $\mathbf{A}$.

Definition 5. Let s_1 and s_2 be two sources having the same arity and whose attributes have pairwise the same abstract domains. A *simple full-width inclusion dependency* (SFWID) between s_1 and s_2 has the form

$$s_1 \subseteq s_2$$

Such an inclusion dependency is *satisfied* in a database DB if $s_1^{DB} \subseteq s_2^{DB}$. ∎

Simple full-width inclusion dependencies are a limited form of inclusion dependencies (IDs), which allow one to state that an entire relation is included in another one having the same arity and the same abstract domains. SFWIDs turn out to be essential for modeling the common case of real data sources that can be accessed through different binding patterns, e.g., a database relation that can be accessed from a Web site using different forms. Consider, e.g., bibliographic data at DBLP accessible either by author or by title: such a data source can be modeled by means of two source symbols with different binding patterns, which are declared equivalent by means of two SFWIDs. In general, we can represent in our model a real data source with several binding patterns as a set $s_1, \ldots, s_n$ of distinct source symbols, one for each different way of accessing the data source, with a binding pattern that reflects the access modality. The fact that the source symbols $s_1, \ldots, s_n$ represent the same data source is expressed by means of the two SFWIDs $s_i \subseteq s_j$ and $s_j \subseteq s_i$ between each pair of sources s_i and s_j, with $1 \leq i < j \leq n$. Observe that this is a very common and thus particularly relevant case which arises each time the same underlying data is accessible, e.g., on the web, via two different forms. Moreover, by means of a SFWID we can also capture the case of a Web site in which a form gives access to a subset of the data contained in the site.

For the sake of completeness, we include the definition of *inclusion dependencies* (IDs).

Definition 6. Let s_1 and s_2 be two sources. An *inclusion dependency* (ID) between s_1 and s_2 has the form

$$r_1[\mathbf{A}] \subseteq r_2[\mathbf{B}]$$

where the sequences $\mathbf{A}$ and $\mathbf{B}$ have the same number of elements and their attributes have pairwise the same abstract domains. Such a dependency is satisfied in a database DB if for every tuple $t_1 \in s_1^{DB}$ there exists a tuple $t_2 \in s_2^{DB}$ such that $t_1[\mathbf{A}] = t_2[\mathbf{B}]$.

Definition 7. A *functional dependency* (FD) over a source s has the form

$$s : \mathbf{A} \rightarrow \mathbf{B}$$

with $\mathbf{A}, \mathbf{B} \subseteq \mathcal{A}(s)$. Such a dependency is *satisfied* in a database DB if for every pair of tuples $t_1, t_2 \in s^{DB}$ we have that $t_1[\mathbf{A}] = t_2[\mathbf{A}]$ implies $t_1[\mathbf{B}] = t_2[\mathbf{B}]$. ∎

Functional dependencies allow one to state that a certain set A of attributes in a source functionally determines another set B of attributes, in the sense that two tuples in the extension of the source that coincide on A also have to coincide on B. They are an important type of integrity constraint, which generalize key constraints and thus are actually present in most relational data sources.

We write $DB \models \gamma$ to indicate that a dependency γ is satisfied in a database DB; similarly, we write $DB \models \Gamma$ to indicate that all the dependencies in the set of dependencies Γ are satisfied in DB. A dependency γ is *implied* by a set of dependencies Γ, written $\Gamma \models \gamma$, if, for every database DB such that $DB \models \Gamma$, we also have $DB \models \gamma$.

The described implication is also called *unrestricted implication*, since it does not exclude databases that have infinite size. Differently, the notion of *finite implication* deals with finite databases only. In particular, we say that Σ *finitely implies* σ, written $\Sigma \models_f \sigma$, if, for every *finite* database DB, $DB \models \Sigma$ implies $DB \models \sigma$.

We discuss now the implication problem for FDs and SFWIDs, which is an important tool for our dynamic optimization technique.

An *inference rule* ρ for IDs and FDs is an expression of the form

$$\text{if } \alpha, \text{ then } \gamma$$

where α (the *premise*) is a conjunction of satisfaction conditions for a set of IDs and FDs, and γ (the *conclusion*) is an ID or a FD. Given a set Γ of IDs and FDs, γ is said to be *directly deduced* from Γ via ρ if α holds for Γ. An ID or FD γ is said to be *deduced* from Γ via a set of rules $\mathbf{R}$ if there is finite sequence $\gamma_1, \ldots, \gamma_n$ of dependencies such that $\gamma_n = \gamma$ and, for all $1 \leq i \leq n$, either $\gamma_i \in \Gamma$ or γ_i is directly deduced from $\Gamma \cup \{\gamma_1, \ldots, \gamma_{i-1}\}$ via some inference rule in $\mathbf{R}$.

We say that a set $\mathbf{R}$ of inference rules for IDs and FDs is *sound* if, for every set Γ of IDs and FDs, all dependencies that are deduced from Γ via $\mathbf{R}$ are also implied by Γ; $\mathbf{R}$ is *complete* if all dependencies that are implied by Γ are also deduced from Γ via $\mathbf{R}$.

1. For every source s and all sets of attributes $\mathbf{A}, \mathbf{B} \subseteq \mathcal{A}(s)$,
   if $\mathbf{A} \subseteq \mathbf{B}$, then $s : \mathbf{B} \rightarrow \mathbf{A}$.
2. For every source s and all sets of attributes $\mathbf{A}, \mathbf{B}, \mathbf{C} \subseteq \mathcal{A}(s)$,
   if $s : \mathbf{A} \rightarrow \mathbf{B}$, then $s : \mathbf{AC} \rightarrow \mathbf{BC}$.
3. For every source s and all sets of attributes $\mathbf{A}, \mathbf{B}, \mathbf{C} \subseteq \mathcal{A}(s)$,
   if $s : \mathbf{A} \rightarrow \mathbf{B}$ and $s : \mathbf{B} \rightarrow \mathbf{C}$, then $s : \mathbf{A} \rightarrow \mathbf{C}$.
4. For every source s,
   $s \subseteq s$.
5. For all sources s_1, s_2, s_3,
   if $s_1 \subseteq s_2$ and $s_2 \subseteq s_3$, then $s_1 \subseteq s_3$.
6. For all sources s_1, s_2 with $\mathcal{A}(s_1) = \mathcal{A}(s_2)$ and sets of attributes $\mathbf{A}, \mathbf{B} \subseteq \mathcal{A}(s_1)$,
   if $s_1 \subseteq s_2$ and $s_2 : \mathbf{A} \rightarrow \mathbf{B}$, then $s_1 : \mathbf{A} \rightarrow \mathbf{B}$.

Figure 2: Inference rules for SFWIDs and FDs

The inference rules shown in Figure 2 are a specialization of the more general sound (but not complete) inference rules for arbitrary inclusion dependencies and functional dependencies [Cosmadakis and Kanellakis, 1986] to the case where all inclusion dependencies are SFWIDs. We will show that for such a case these inference rules are not only sound but also complete.

We note that the only rule that makes SFWIDs and FDs interact is Rule 6. The following theorem shows that FDs do not influence the implication of a SFWID. Hence, a SFWID can be derived *only* from the set of available SFWIDs.

Lemma 8. *Given a source schema $\mathcal{S}$, two sources $s_1, s_2 \in \mathcal{S}$, a set Γ_i of SFWIDs, and a set Γ_f of FDs, we have that $\Gamma_i \models s_1 \subseteq s_2$ if and only if $(\Gamma_i \cup \Gamma_f) \models s_1 \subseteq s_2$.*

Proof.

"$\Rightarrow$" Trivial.

"$\Leftarrow$" We prove the claim by showing its contrapositive. In particular, we show that (a) implies (b), where

(a) $\Gamma_i \not\models s_1 \subseteq s_2$, i.e., there exists a database DB such that $DB \models \Gamma_i$, but $DB \not\models s_1 \subseteq s_2$.

(b) $(\Gamma_i \cup \Gamma_f) \not\models s_1 \subseteq s_2$, i.e., there exists a database DB' such that $DB' \models \Gamma_i$ and $DB' \models \Gamma_f$, but $DB' \not\models s_1 \subseteq s_2$.

Assuming (a) to be true, we construct a database DB' that makes (b) true as follows. Let $\mathcal{S}'$ be the subset of $\mathcal{S}$ consisting of all sources $s' \in \mathcal{S}$ such that there exists a sequence

$$s_1 = s'_1 \subseteq \cdots \subseteq s'_k = s'$$

of SFWIDs in Γ_i, where $\{s'_1, \ldots, s'_k\} \subseteq \mathcal{S}$, for some $k \geq 0$. Since (a) holds, there is a tuple t in DB such that $t \in s_1^{DB}$ and $t \notin s_2^{DB}$. We set $s^{DB'} = \{t\}$ for each source $s \in \mathcal{S}'$, and $s^{DB'} = \emptyset$ for each source $s \notin \mathcal{S}'$. It is immediate to see that $DB' \models \Gamma_i$ by construction. Also, $DB' \models \Gamma_f$ because each source contains at most one tuple. Moreover, $s_1 \in \mathcal{S}'$ and hence $s_1^{DB'} = \{t\}$. To show that (b) holds, it remains to show that $s_2^{DB'} = \emptyset$, i.e., that $s_2 \notin \mathcal{S}'$. Assume by contradiction that $s_2 \in \mathcal{S}'$. Then, we would have in Γ_i a chain of inclusion dependencies

$$s_1 = s''_1 \subseteq \cdots \subseteq s''_h = s_2$$

By $t \in s_1^{DB}$ and $DB \models \Gamma_i$, we would have that $t \in s_2^{DB}$, contradicting the fact that $t \notin s_2^{DB}$. Thus (b) holds. □

Next, we show that FDs and SFWIDs interact only in a limited form. In particular, Lemma 9 below states that implication of a FD on a source s can be decided by disregarding all sources s_i for which $s \subseteq s_i$ is not derived.

Lemma 9. *Let $\mathcal{S}$ be a source schema, where all sources in $\mathcal{S}$ have the same arity and the same abstract domains, let Γ be a set of FDs on sources in $\mathcal{S}$ and of SFWIDs of the form $s_1 \subseteq s_2$ with $s_1, s_2 \in \mathcal{S}$, and let $s \in \mathcal{S}$. Consider the set of sources $\mathcal{S}' = \{s' \mid \Gamma \models s \subseteq s'\}$ and the set Γ' containing all and only the dependencies in Γ that are on sources in $\mathcal{S}'$. Then $\Gamma \models s : \mathbf{A} \to \mathbf{B}$ if and only if $\Gamma' \models s : \mathbf{A} \to \mathbf{B}$.*

Proof.

"$\Leftarrow$" Trivial.

"$\Rightarrow$" Assume by contradiction that $\Gamma \models s : \mathbf{A} \to \mathbf{B}$ but $\Gamma' \not\models s : \mathbf{A} \to \mathbf{B}$. The latter means that there is a database DB such that $DB \models \Gamma'$ and $DB \not\models s : \mathbf{A} \to \mathbf{B}$. Let now DB' be the same as DB, but with all sources in $\mathcal{S} \setminus \mathcal{S}'$ having an empty extension. We first show that $DB' \models \Gamma$: (*i*) For the sources in $\mathcal{S} \setminus \mathcal{S}'$, the extensions in DB' are empty, and hence all the FDs and SFWIDs involving such sources are satisfied in DB'. (*ii*) For the sources in $\mathcal{S}'$, DB and DB' coincide, and since $DB \models \Gamma'$, all the FDs and SFWIDs involving such sources are satisfied in DB'. (*iii*) For two sources $s_1 \in \mathcal{S}'$ and $s_2 \in \mathcal{S} \setminus \mathcal{S}'$, the SFWID $s_2 \subseteq s_1$ holds, since $s_2^{DB'} = \emptyset$. The only remaining case would be that of a SFWID $s_1 \subseteq s_2$, with $s_1 \in \mathcal{S}'$ and $s_2 \in \mathcal{S} \setminus \mathcal{S}'$. However, such a SFWID cannot occur in Γ. Indeed, if $s_1 \in \mathcal{S}'$, by construction of $\mathcal{S}'$ we have that $\Gamma \models s \subseteq s_1$; if in Γ we had also $s_1 \subseteq s_2$, by inference rule 5 in Figure 2 it would follow that $\Gamma \models s \subseteq s_2$, and hence we would also have $s_2 \in \mathcal{S}'$. Therefore $DB' \models \Gamma$ and,

since $\Gamma \models s : \mathbf{A} \rightarrow \mathbf{B}$, then $DB' \models s : \mathbf{A} \rightarrow \mathbf{B}$ too. However, since DB and DB' necessarily coincide on s (since $s \in \mathcal{S}'$, being $s \subseteq s$), then it must also hold that $DB \models s : \mathbf{A} \rightarrow \mathbf{B}$, against the assumptions. Contradiction. □

Once all uninteresting sources are disregarded, Lemma 10 below states that, to decide the implication of a FD on s, we can assert on s all FDs holding on the other sources and decide the implication using only the FDs on s and disregard all the rest.

Lemma 10. *Let $\mathcal{S}$ be a source schema all of whose sources have the same arity and abstract domains. Let s be a source in $\mathcal{S}$ and $\Gamma = \Gamma_f \cup \Gamma_i$, where $\Gamma_i = \{s \subseteq s' \mid s' \in \mathcal{S}\}$[5], and Γ_f is a set of FDs on sources in $\mathcal{S}$. Consider the set*

$$\Gamma' = \{s : \mathbf{A}' \rightarrow \mathbf{B}' \mid \textit{there is } s' \in \mathcal{S} \textit{ such that } s' : \mathbf{A}' \rightarrow \mathbf{B}' \in \Gamma_f\}.$$

Then $\Gamma \models s : \mathbf{A} \rightarrow \mathbf{B}$ if and only if $\Gamma' \models s : \mathbf{A} \rightarrow \mathbf{B}$.

Proof.

"$\Leftarrow$" It suffices to show that, for every dependency of the form $s' : \mathbf{A}' \rightarrow \mathbf{B}'$ in Γ_f where $s' \in \mathcal{S}$, we have $\Gamma \models s : \mathbf{A}' \rightarrow \mathbf{B}'$. This follows directly from inference rule 6 in Figure 2.

"$\Rightarrow$" Suppose by contradiction that $\Gamma \models s : \mathbf{A} \rightarrow \mathbf{B}$ and $\Gamma' \not\models s : \mathbf{A} \rightarrow \mathbf{B}$. Then there exists a database DB that is a witness of the latter non-implication, i.e., such that $DB \models \Gamma'$ and $DB \not\models s : \mathbf{A} \rightarrow \mathbf{B}$. Consequently there are two tuples t_1, t_2 in s^{DB} such that

$$t_1[\mathbf{A}] = t_2[\mathbf{A}] \text{ and } t_1[\mathbf{B}] \neq t_2[\mathbf{B}]. \tag{1}$$

Let DB' be a database in which the extension of all sources in $\mathcal{S}$ consists exactly of the tuples t_1 and t_2. By (1), $DB' \not\models s : \mathbf{A} \rightarrow \mathbf{B}$, and, trivially, $DB' \models \Gamma'$, since all FDs in Γ' are on s and were already satisfied in DB, and we have $s^{DB'} \subseteq s^{DB}$. Now, since all sources have the same extension in DB', then $DB' \models \Gamma_i$. Moreover, every FD in Γ_f on some $s' \in \mathcal{S}$ involves the same attributes as the corresponding FD on s in Γ'; therefore, we have $DB' \models \Gamma_f$. By compositionality of $\models$, it follows that $DB' \models \Gamma$; however $DB' \not\models s : \mathbf{A} \rightarrow \mathbf{B}$, which is a contradiction to $\Gamma \models s : \mathbf{A} \rightarrow \mathbf{B}$. □

Finally, to deal with functional dependencies within one relation we can apply the following theorem.

Theorem 11 [Armstrong, 1974]. *The inference rules 1, 2, and 3 in Figure 2 are sound and complete for implication of functional dependencies within one relation.*

[5] Note that Γ_i consists of one SFWID between s and each source in $\mathcal{S}$.

Before we proceed, we prove the following result.

Lemma 12. *The reflexivity and transitivity rules from Figure 2 (numbers 4 and 5) for SFWIDs are sound and complete.*

Proof (sketch). Soundness is trivial. Completeness descends from [Casanova et al., 1984], where the following rules are proved to be correct and complete for inclusion dependencies:

1. *(Reflexivity)* For all r and for all $\mathbf{X} \subseteq \mathcal{A}(r)$,
   $r[\mathbf{X}] \subseteq r[\mathbf{X}]$
2. *(Projection and permutation)* If $r_1[A_1, \ldots, A_m] \subseteq r_2[B_1, \ldots, B_m]$, then for every sequence $i_1, \ldots, i_k$ of distinct integers with $\{i_1, \ldots, i_k\} \subseteq \{1, \ldots, m\}$ it holds
   $r_1[A_{i_1}, \ldots, A_{i_k}] \subseteq r_2[B_{i_1}, \ldots, B_{i_k}]$
3. *(Transitivity)* If $r_1[\mathbf{A}] \subseteq r_2[\mathbf{B}]$ and $r_2[\mathbf{B}] \subseteq r_3[\mathbf{C}]$, then
   $r_1[\mathbf{A}] \subseteq r_3[\mathbf{C}]$

It is easy to see that we can transform every set of SFWIDs Σ, expressed over a relational schema $\mathcal{R}$, into a set of unary IDs Σ_u expressed over a relational schema $\mathcal{R}_u$ having only unary relational predicates, such that taken a SFWID and its corresponding unary ID σ_u, obtained by means of the above transformation, we have $\Sigma \models \sigma$ iff $\Sigma_u \models \sigma_u$. The transformation is done as follows: for every n-ary relation symbol $r \in \mathcal{R}$, the corresponding symbol in $\mathcal{R}_u$ has a single attribute whose abstract (resp. concrete) domain is the cartesian product of the abstract (resp. concrete) domains of r, in their order; the single attribute of r_u serves to represent the values of all attributes of r. Now, for unary IDs over unary relational predicates in $\mathcal{R}_u$, it is easy to see that the above rules taken from [Casanova et al., 1984] become as follows.

1. *(Reflexivity)* For all r,
   $r[1] \subseteq r[1]$
2. *(Transitivity)* If $r_1[1] \subseteq r_2[1]$ and $r_2[1] \subseteq r_3[1]$, then
   $r_1[1] \subseteq r_3[1]$

Notice that Rule 2 among those from [Casanova et al., 1984] has disappeared, since there is no way of doing projections and permutations on unary IDs. The set of unary IDs we have obtained are also SFWIDs, and they are the same as the reflexivity and transitivity rules from Figure 2 (number 4 and 5) for SFWIDs, as we can see by rewriting them into the SFWID notation. This proves the lemma. □

This last result allows us to prove the following theorem.

Theorem 13. *The inference rules of Figure 2 are sound and complete for implication of functional dependencies and simple full-width inclusion dependencies.*

Proof (sketch). By Lemma 8, we conclude that implication of SFWIDs is not affected by FDs and therefore can be reduced to implication of SFWIDs alone. For this problem, the reflexivity and transitivity rules of Figure 2 (number 4 and 5) are sound and complete by Lemma 12. From Lemmas 9 and 10, it follows that implication of a FD on a source s can be decided by (*i*) disregarding all sources that do not contain s by means of SFWIDs, (*ii*) asserting on s all FDs holding on the remaining sources (via inference rule 6 in Figure 2), and (*iii*) deciding the implication using only the FDs on s. For this latter point, by Theorem 11, inference rules 1, 2, and 3 in Figure 2 are sound and complete. □

From the previous results we can prove the following theorem, which, to the best of our knowledge, is not covered by the known results about implication of dependencies [Cosmadakis and Kanellakis, 1986; Cosmadakis et al., 1990].

Theorem 14. *Implication of simple full-width inclusion dependencies and functional dependencies can be decided in polynomial time.*

Proof. By Lemma 8, implication of a SFWID amounts to reachability on the inclusion dependency graph (having the n sources as nodes and an arc (s_1, s_2) if $s_1 \subseteq s_2$ is in the set of SFWIDs), which can be decided in NLogSpace by a nondeterministic algorithm that guesses a path of at most n nodes in the graph, and storing in space $O(\log n)$ only the (label of the) current node plus a counter from 0 to n. The reachability problem is known to be NLogSpace-complete [Jones et al., 1976]. Computing *all* the dependencies in the reflexive and transitive closure of a set of n SFWIDs can be done in time $O(n^3)$ by computing the transitive closure of the corresponding inclusion dependency graph [Floyd, 1962].

To decide implication of functional dependencies we proceed as suggested by the proof of Theorem 13: (*i*) compute the transitive closure of SFWIDs using inference rule 5, (*ii*) propagate all FDs across SFWIDs using rule 6, and (*iii*) compute implication of FDs within a single source. Step (*i*) requires cubic time, step (*ii*) is clearly polynomial, and by [Beeri and Bernstein, 1979; Maier, 1980], step (*iii*) can be carried out in polynomial time. □

Finally, we state the equivalence of finite and unrestricted implication for SFWIDs and FDs.

Theorem 15. *Finite and unrestricted implication are equivalent for SFWIDs and FDs.*

Proof (sketch). Given a set Σ of SFWIDs and FDs and a FD of SFWID σ, we prove $\Sigma \models \sigma$ iff $\Sigma \models_f \sigma$.

"⇒" Trivial.

"⇐" We prove that $\Sigma \not\models \sigma$ implies $\Sigma \not\models_f \sigma$; in particular, we show that if there exists a database C such that $C \models \Sigma$ and $C \not\models \sigma$, then there exists a *finite* database C_f such that $C \models \Sigma$ and $C \not\models \sigma$. The case where C is finite is trivial (we take $C_f = C$); assume C is infinite. We have two cases.

(a) σ is a SFWID, say of the form $s_1 \subseteq s_2$. By hypothesis $C \not\models \sigma$, therefore there exists a tuple t_0 such that $t_0 \in s_1^{\mathcal{C}} \setminus s_2^{C}$. We construct a database as follows; we start from t_0 in s_1, and whenever, in this new database, there is a tuple t such that t is in s_a but not in s_b for some s_a and s_b, and Σ contains the SFWID $s_a \subseteq s_b$, we add t to s_b. The obtained database, denoted C_0, is such that $C_0 \models \Sigma$: the SFWIDs in Σ are satisfied by construction, and it is easy to see the FDs are satisfied because otherwise they would also be violated in C. At the same time, it is immedately seen that $C \not\models \sigma$, because we add t_0 to s_2 by construction only if $\Sigma \models \sigma$. Therefore, being finite by construction, C_0 is the desired counterexample to the implication.

(b) σ is a FD, say of the form $s : \mathbf{A} \rightarrow \mathbf{B}$. By hypothesis $C \not\models \sigma$, therefore there exist two tuples $t_1, t_2 \in s^C$ such that $t_1[\mathbf{A}] = t_2[\mathbf{A}]$ and $t_1[\mathbf{B}] \neq t_2[\mathbf{B}]$. Similarly to the previous case, we construct a database as follows; we start from t_1 and t_2 in s, and whenever, in this new database, there is a tuple t such that t is in s_a but not in s_b for some s_a and s_b, and Σ contains the SFWID $s_a \subseteq s_b$, we add t to s_b. The obtained database, denoted C_0, is such that $C_0 \models \Sigma$: the SFWIDs in Σ are satisfied by construction, and the FDs are satisfied because otherwise they would also be violated in C. Moreover, $C_0 \not\models \sigma$ because of the presence of t_1 and t_2. Since C_0 is finite by construction, it is the desired counterexample to the implication.

□

5 Dynamic Optimization

In Section 3, we have surveyed techniques for generating query plans for retrieving the maximally contained answer to a query over sources with access limitations. Some of these techniques also try to minimize source accesses by identifying, at query plan generation time, those sources that are not relevant for the query, i.e., such that they can never provide useful bindings for retrieving tuples in the maximally contained answer. Accesses to sources that are not relevant can thus be completely avoided.

Instead, here we introduce further optimization techniques, which can be applied during the query evaluation process. Such techniques take into account tuples already extracted from the sources at a certain step of the evaluation of

the query plan associated to a query. They exploit simple full-width inclusion dependencies and functional dependencies on the source relations to know in advance whether an access to be made by the query plan is potentially useful for the answer, i.e., whether it could return tuples with new values that have not been already extracted by previous accesses.

Example 5. Consider a source $s(\underline{A}_1, \underline{A}_2, A_3)$ (supposing for simplicity to have a distinct abstract domain for each attribute) with the functional dependency

$$s : A_1 \rightarrow A_2, A_3$$

Let DB be a database and t be a tuple previously extracted from s^{DB}, with $t[A_1] = a_1$. Suppose that we have some value a_2 with which to bind attribute A_2. If we try to access s^{DB} using the binding (a_1, a_2), the access cannot provide any new tuple: because of the functional dependency that has to be satisfied by DB, it can either provide t alone (in case $a_2 = t[A_2]$), or no tuple (in case $a_2 \neq t[A_2]$). ∎

We remark that the functional dependency of Example 5 is actually a key constraint, and the key is a subset of the bound attributes of the source. We now introduce the notion of dynamic relevance, which characterizes those accesses that may extract new tuples.

In the following, we denote the bound attributes of a source s with $\mathcal{B}(s)$. Furthermore, we say that a database DB for a source schema $\mathcal{S} = \{s_1, \ldots, s_n\}$ *captures* the sets of tuples $\mathbf{T}_{s_1}, \ldots, \mathbf{T}_{s_n}$ for $s_1, \ldots, s_n$ if there exists a set of accesses such that the tuples extracted by it from s_i^{DB} is exactly $\mathbf{T}_{s_i}$, for $1 \leq i \leq n$. In other words, the sets $\mathbf{T}_{s_1}, \ldots, \mathbf{T}_{s_n}$ may be interpreted as the sets of tuples that have been already extracted at a certain point of the query answering process.

Definition 16. Let $\mathcal{S} = \{s_1, \ldots, s_n\}$ be a source schema, Γ a set of dependencies for $\mathcal{S}$, and $\mathbf{T}_{s_1}, \ldots, \mathbf{T}_{s_n}$ sets of tuples associated to $s_1, \ldots, s_n$, respectively. An access to a source $s_k \in \mathcal{S}$ using binding b is *dynamically relevant* with respect to Γ and $\mathbf{T}_{s_1}, \ldots, \mathbf{T}_{s_n}$ if, for at least one database DB that captures $\mathbf{T}_{s_1}, \ldots, \mathbf{T}_{s_n}$ for $s_1, \ldots, s_n$ and such that $DB \models \Gamma$, the extraction made by the access in DB contains at least one tuple that is not in $\bigcup_{1 \leq i \leq n} \mathbf{T}_{s_i}$. ∎

Notice that, a necessary condition for an access to be dynamically relevant with respect to Γ and $\mathbf{T}_{s_1}, \ldots, \mathbf{T}_{s_n}$, is that for each i, the set of tuples $\mathbf{T}_{s_i}$, when considered as (part of) the extension of a source s_i, does not violate any FD on s_i in Γ. With the notion of dynamic relevance at hand, we can generalize the observation made in Example 5, with the following result.

Proposition 17. *Given a source s, let γ be the FD*

$$s: \mathbf{K} \rightarrow \mathcal{A}(s)$$

with $\mathbf{K} \subseteq \mathcal{B}(s)$. Let b be a binding for s and $\mathbf{T}_s$ a set of tuples that satisfies γ, when considered as the extension of s. Then accessing s using b is dynamically relevant with respect to γ and $\mathbf{T}_s$ if and only if there exists no tuple $t \in \mathbf{T}_s$ such that $b[\mathbf{K}] = t[\mathbf{K}]$.

Proof. Let $\mathbf{B} = \mathcal{B}(s)$.

"$\Leftarrow$" Consider a database DB such that $s^{DB} = \{t_0\} \cup \mathbf{T_s}$, for some tuple t_0 with $b[\mathbf{B}] = t_0[\mathbf{B}]$. Since (i) $\mathbf{T_s}$, when considered as the extension of s, satisfies γ, (ii) $\mathbf{T_s}$ does not contain a tuple t such that $b[\mathbf{K}] = t[\mathbf{K}]$, and (iii) $\mathbf{K} \subseteq \mathbf{B}$, we have that $DB \models \gamma$ and also that $t_0 \notin \mathbf{T}_s$. Therefore, accessing s^{DB} using b returns a tuple not in $\mathbf{T}_s$, namely t_0.

"$\Rightarrow$" Suppose accessing s using b as binding is dynamically relevant with respect to γ and $\mathbf{T}_s$. This means that there is at least one database DB capturing $\mathbf{T_s}$ for s and satisfying γ in which the access extracts at least one tuple t_0 that was not in $\mathbf{T}_s$. All tuples that may be extracted from s^{DB} using b have the same values over $\mathbf{K}$, due to the fact that $\mathbf{K} \subseteq \mathbf{B}$. Suppose, by contradiction, that there is a tuple $t \in \mathbf{T}_s$ such that $t[\mathbf{K}] = b[\mathbf{K}]$. Then, because of the functional dependency γ, every tuple extracted from s^{DB} using b, including t_0, has the same values as t over $\mathcal{A}(s)$, i.e., it coincides with t and is therefore already in $\mathbf{T}_s$. Contradiction to $t_0 \notin \mathbf{T}_s$. □

Clearly, no binding that, together with the set of already extracted tuples, violates a FD can be used to extract new tuples. This circumstance is expressed in the following proposition.

Proposition 18. *Given a source s, let γ be the FD*

$$s: \mathbf{A} \rightarrow \mathbf{B}$$

with $\mathbf{A} \subseteq \mathcal{B}(s)$. Let b be a binding for s and $\mathbf{T}_s$ a set of tuples. Then accessing s using b is dynamically relevant with respect to γ and $\mathbf{T}_s$ if and only if

1. *there exists no tuple $t \in \mathbf{T}_s$ such that $t[\mathcal{B}(s)] = b[\mathcal{B}(s)]$, and*
2. *there exists no tuple $t \in \mathbf{T}_s$ such that $t[\mathbf{A}] = b[\mathbf{A}]$ and $t[\mathbf{B} \cap \mathcal{B}(s)] \neq b[\mathbf{B} \cap \mathcal{B}(s)]$.*

Proof. Clearly, if Condition 1 holds, then the access made by b has already been done to obtain t, and thus it is not dynamically relevant. If Condition 2 holds, then every tuple extracted with binding b would necessarily violate γ; therefore, since we only consider databases capturing $\mathbf{T_s}$ and satisfying γ, no tuple can

be extracted with binding b. If neither Condition 1 nor Condition 2 holds, it suffices to consider a database DB and a tuple t' such that $s^{DB} = \{t'\} \cup \mathbf{T}_s$ and $t'[\mathcal{B}(s)] = b[\mathcal{B}(s)]$: the access is new, it extracts t', and γ is satisfied, therefore it is dynamically relevant. □

Now we illustrate another optimization technique, which also exploits simple full-width inclusion dependencies.

Example 6. Suppose we have the following sources:

$$s_1(\underline{Code}, \underline{Surname}, City)$$
$$s_2(Code, Surname, \underline{City})$$

where s_1 stores data about employees and s_2 stores data about persons, and the SFWID $s_1 \subseteq s_2$. The attribute with abstract domain *City* represents the city where the corresponding person (or employee) lives. We also have the following FD on s_1:

$$s_1 : Code \rightarrow Surname, City$$

Consider a database DB in which both s_1^{DB} and s_2^{DB} are as follows:

Code	*Surname*	*City*
2	*brown*	*sidney*
5	*williams*	*london*
7	*yamakawa*	*kyoto*
1	*wakita*	*kyoto*
9	*marietti*	*rome*

If our set of initial values is *rome* and *kyoto*, we may use them as bindings to access s_2^{DB} and get the following tuples:

Code	*Surname*	*City*
7	*yamakawa*	*kyoto*
1	*wakita*	*kyoto*
9	*marietti*	*rome*

Now we have six new values: the three codes 1, 7, and 9 and the three surnames *yamakawa*, *wakita*, and *marietti*. With these values we could access s_1^{DB} to try and get other values (we may get only cities, as the other attributes are bound). But we can easily observe that, because of the functional dependency on s_1, if we bind the attribute with domain *Code* with one of the known values, we get a tuple we had already obtained from s_2^{DB}. Therefore the access to s_1^{DB} is useless, once we have accessed s_2^{DB}. Instead, if we obtained a code, say 2, and a surname, say *brown*, from some other source, we could then access s_1^{DB} using binding 2 and *brown*, and get new tuples. ■

The following proposition provides a necessary and sufficient condition for dynamic relevance for sources characterized by FDs and SFWIDs as in Example 6.

Proposition 19. *Let s_1 and s_2 be two sources, and Γ the following set of dependencies:*

$$s_1 \subseteq s_2$$
$$s_1 : \mathbf{C} \rightarrow \mathbf{D}$$

with $\mathbf{C} \subseteq \mathcal{B}(s_1)$ and $\mathbf{D} \supseteq \mathcal{B}(s_2)$. Let b be a binding for s_1 and $\mathbf{T_{s_2}}$ a set of tuples. Then, accessing s_1 with b is dynamically relevant with respect to Γ and $\emptyset, \mathbf{T_{s_2}}$ if and only if there exists no tuple $t \in \mathbf{T_{s_2}}$ such that $t[\mathbf{C}] = b[\mathbf{C}]$.

Proof. Let $\mathbf{B}_1 = \mathcal{B}(s_1)$ and $\mathbf{B}_2 = \mathcal{B}(s_2)$

"$\Leftarrow$" By hypothesis, there is no tuple in $\mathbf{T_{s_2}}$ agreeing with b on $\mathbf{C}$. Suppose, by contradiction, that accessing s_1 with b is not dynamically relevant. Then, for every database DB capturing $\emptyset, \mathbf{T_{s_2}}$ for s_1, s_2 and satisfying Γ, the extraction made by such an access contains only tuples that are in $\mathbf{T_{s_2}}$. There is clearly at least one such database DB for which the access to s_1^{DB} using b has a non-empty extraction. For every tuple t in such an extraction, since $\mathbf{C} \subseteq \mathbf{B}_1$, we have $t[\mathbf{C}] = b[\mathbf{C}]$. Then, by dynamic irrelevance of the access, t must necessarily be in $\mathbf{T_{s_2}}$, against the hypothesis.

"$\Rightarrow$" By hypothesis, there is a database DB capturing $\emptyset, \mathbf{T_{s_2}}$ for s_1, s_2 and satisfying Γ in which the extraction made by an access to s_1^{DB} with b contains at least one tuple that is not in $\mathbf{T_{s_2}}$. Suppose by contradiction that there exists a tuple t in $\mathbf{T_{s_2}}$ such that $t[\mathbf{C}] = b[\mathbf{C}]$; then, since $\mathbf{C} \subseteq \mathbf{B}_1$ and because of the dependency $s_1 : \mathbf{C} \rightarrow \mathbf{D}$, we have that every tuple t' extracted from s_1 using b is such that $t'[\mathbf{D}] = t[\mathbf{D}]$. Now, since $\mathbf{B}_2 \subseteq \mathbf{D}$, we obviously have that $t'[\mathbf{B}_2] = t[\mathbf{B}_2]$. Observe that, as t was extracted from s_2^{DB}, all tuples of s_2^{DB} coinciding with t on $\mathbf{B}_2$ have been extracted as well. Due to the inclusion dependency $s_1 \subseteq s_2$, we can conclude that every tuple t' that we may get from s_1^{DB} using b as binding was already extracted from s_2^{DB}. Contradiction. □

Using the previous results, we can show that verifying dynamic relevance of accesses can be done in polynomial time.

Theorem 20. *Let $\mathcal{S}$ be a source schema for which the maximum arity of all sources is fixed, and let Γ be a set of FDs and SFWIDs on sources in $\mathcal{S}$. Then one can check in polynomial time in the number of dependencies in Γ, the number of attributes in $\mathcal{S}$, and the number of tuples already extracted from sources in $\mathcal{S}$, whether accessing a source in $\mathcal{S}$ with a certain binding is dynamically relevant with respect to Γ.*

Proof (sketch). By Theorem 14, implication of FDs and SFWIDs can be checked in polynomial time, which allows us to test whether the dependencies required for the application of Propositions 17, 18, and 19 hold.

The conditions of applicability of Proposition 17 need to be checked only for a polynomial number of key dependencies, since the maximum arity is fixed for each source in the schema.

A similar argument can be used to show that there is only a polynomial number of dependencies to be tested for the application of Proposition 19.

As for the applicability of Proposition 18, one only needs to consider the set Γ' of implied FDs obtained from Γ by first computing the transitive closure of SFWIDs using inference rule 5 and then by propagating all FDs across SFWIDs using rule 6. All other implied FDs obtained through rules 1–3 need not be considered, because they do not add anything to the FDs in Γ' with respect to the conditions of applicability of Proposition 18. In particular, in rule 1, if b and an extracted tuple t agree on $\mathbf{B}$, with $\mathbf{B} \supseteq \mathbf{A}$, then they necessarily also agree on $\mathbf{A}$; in rule 2, if b and an extracted tuple t agree on $\mathbf{A}$ and on $\mathbf{B}$, then if they agree on $\mathbf{AC}$, they necessarily also agree on $\mathbf{BC}$; in rule 3, if b and an extracted tuple t agree on $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$, then they trivially also agree on $\mathbf{A}$ and $\mathbf{C}$. Clearly, the size of Γ' is polynomial in the size of Γ.

Finally, for any access to be made with a binding b, in order to apply the claims of the mentioned propositions, one simply needs to check whether any of the known tuples agrees with b on some given attributes, which can also be done in polynomial time in the number of extracted tuples. □

The result of Theorem 20 can be complemented with the practically relevant observation that minimal keys are the only key constraints that need to be considered when testing dynamic relevance of an access. We recall that, for a set of dependencies Γ and a source s, a constraint γ_k of the form $s : \mathbf{K} \to \mathcal{A}(s)$ is a *minimal key* implied by Γ if $\Gamma \models \gamma_k$ and there exists no $\mathbf{K}' \subset \mathbf{K}$ such that $\Gamma \models s : \mathbf{K}' \to \mathcal{A}(s)$.

Proposition 21. *Let Γ be a set of FDs for a source s, b a binding for s, and $\mathbf{T}_s$ a set of tuples; let further Γ' be the set of constraints of the form $s : \mathbf{K} \to \mathcal{A}(s)$ implied by Γ. Then b is dynamically relevant with respect to Γ' and $\mathbf{T}_s$ if and only if, for each minimal key γ_k implied by Γ, b is dynamically relevant with respect to γ_k and $\mathbf{T}_s$.*

Proof (sketch). Let $s : \mathbf{K} \to \mathcal{A}(s)$ be a minimal key implied by Γ and let $s : \mathbf{K}' \to \mathcal{A}(s)$ be a FD implied by Γ, with $\mathbf{K}' \supset \mathbf{K}$. It follows straightforwardly by Definition 16 that if b is dynamically relevant with respect to $s : \mathbf{K} \to \mathcal{A}(s)$ and $\mathbf{T}_s$ then it is also dynamically relevant with respect to $s : \mathbf{K}' \to \mathcal{A}(s)$ and $\mathbf{T}_s$. □

Similarly, the only relevant FDs for the application of Proposition 19 are of the form $s_1 : \mathbf{C} \rightarrow \mathcal{B}(s_2)$, with $\mathbf{C} \subseteq \mathcal{B}(s_1)$ and $\mathbf{C}$ being a minimal set of attributes for which an FD of this form is implied, since those of the form $s : \mathbf{C}' \rightarrow \mathbf{D}$, with $\mathbf{D} \supseteq \mathcal{B}(s_2)$ and $\mathbf{C}' \supseteq \mathbf{C}$ do not add any requirement to dynamic relevance.

We conclude the section by distinguishing between two different kinds of dynamically (ir)relevant accesses. When an access to a source s is recognized as dynamically irrelevant, we are guaranteed that the tuples it extracts, if any, do not contain any constant that is not already known. However, in some cases we may already know that the access is doomed to extract no tuples at all or to only extract tuples that are already known to be in s; in such cases, we say that the access is *tuple-irrelevant*. The conditions of Propositions 17 and 18 sanction an access as tuple-(ir)relevant. It is never useful to make tuple-irrelevant accesses, since no new piece of information can be gained from it, provided that the results from previous accesses are remembered. In some other cases we simply know that the access will not extract any new tuple of constants, but we may not already know whether such tuple is in s or not before making the access (in particular, we may know the tuple from other sources than s); in such cases, we say that the access is *binding-irrelevant*. The conditions of Proposition 19 determine whether an access is binding-(ir)relevant (if $s_1 \neq s_2$) or tuple-(ir)relevant (if $s_1 = s_2$). The previous observations suggest that, during query answering under access limitations, tuple-irrelevant accesses should always be avoided, whereas accesses that are binding-irrelevant but not tuple-irrelevant should be avoided only when accessing off-query sources, i.e., those sources that are only accessed in order to retrieve potentially useful bindings to compute the maximally contained answer, but that are not themselves part of the query.

6 Related work

Processing queries under access limitations is a problem that, with the advent of the Web, has gained increased importance in database theory and information systems. Several works in the literature have investigated different aspects of this problem, directly or indirectly related to the issue of query optimization. We briefly list and discuss the most relevant ones in the following.

A seminal work is [Rajaraman et al., 1995], which deals with the problem in an information integration setting, providing techniques for determining whether a CQ can be answered in the presence of access limitations, and showing that the problem can be decided in polynomial time. [Yerneni et al., 1999] deals with mediators that compute answers to queries, and considers for such mediators a more refined notion of capability than the one of binding patterns used in the present paper. It proposes techniques to compute the set of queries that are supported by specific mediators, given sources with access limitations.

[Florescu et al., 1999] considers non-recursive plans and proposes algorithms to optimize the search space for such plans, while [Duschka and Levy, 1997] presents algorithms for query answering using views [Halevy, 2001] under access limitations. This kind of problem is taken up again in [Deutsch et al., 2005] by considering the extension with various forms of integrity constraints. The paper shows that access limitations can be encoded into integrity constraints of a suitable form, which can then be processed together with the other constraints. Optimization is, however, not dealt with in the paper.

Recursive query plans for query answering under access limitations are introduced in [Li and Chang, 2000]. Instead, [Millstein et al., 2003] studies the problem of containment of queries *relative* to a set of views, and considers also sources with access limitations. For this case, decidability of containment of a Datalog query in a non-recursive Datalog query is established.

We have already discussed in Section 3 the works addressing the problem of static optimizations, i.e., those that can be made at the time of the generation of the query plan [Li and Chang, 2000; Li and Chang, 2001b; Li and Chang, 2001a; Calì and Martinenghi, 2008b].

Several other problems have been addressed in the context of processing queries in the presence of access limitation. A basic problem is *feasibility*, i.e., the existence of a query that is equivalent to a given one and that is executable as is from left to right, while respecting the access limitations. The problem of checking the *feasibility* of a query has been addressed in [Nash and Ludäscher, 2004; Ludäscher and Nash, 2004] for various classes of queries up to first-order queries, showing that it is as hard as query containment. [Yang et al., 2006] studies, again for various classes of queries, whether there is an ordering of subgoals in a query plan that enables answering the query, and, if multiple such orderings are possible, how to pick the best one.

Another issue in this framework is *stability*, i.e., determining whether the *complete* answer to a query (the one that would be obtained with no access limitations) can always be computed despite the access limitations; this is addressed in [Li, 2003].

[Calì and Martinenghi, 2008b] introduces a graph model and presents an algorithm for the generation of minimal query plans for CQs. The paper presents also a system that implements suitable heuristics to query sources in parallel as much as possible, thus reducing the overall query answering time.

Access limitations have also been studied in the context of logic programs with (input or output) *modes*; in particular, the notion of *well-modedness*, due to [Dembinski and Maluszynski, 1985], corresponds to the notion of executability of a query from left to right of the previously mentioned works.

We remark that, with the exception of [Deutsch et al., 2005], which may use integrity to enforce multiple access limitations on the same sources, all the

mentioned works rely on the assumption that for each source a single access limitation only is specified. While in our setting we also enforce this syntactic restriction, as remarked in Section 4, we may use SFWIDs to assert that various sources are equivalent, and thus enforce on them multiple access limitations. Also, to the best of our knowledge, no previous work addresses the problem of dynamic optimization of accesses, in particular for the case of sources with multiple access limitations or exploiting source dependencies.

Finally, we remark that the problem of implication of functional and inclusion dependencies has been studied extensively in the 80's and early 90's (see, e.g., [Mitchell, 1983; Beeri and Vardi, 1984; Chandra and Vardi, 1985; Cosmadakis et al., 1990; Cosmadakis and Kanellakis, 1986]). It is well known that it is undecidable in the general case [Chandra and Vardi, 1985], but efficiently solvable for special cases. Among these, we mention the case where the inclusion dependencies are restricted to be unary [Cosmadakis et al., 1990] and the case where the graph representation of a set of inclusion dependencies can be captured by a forest [Levene and Loizou, 2001] with the so-called *tree-like inclusion dependencies*. Neither case subsumes the classes of dependencies that we consider in this paper; in particular, a graph representation of the inclusion dependencies by a forest requires acyclicity among inclusion dependencies, while we allow SFWIDs to be cyclic.

7 Conclusions

In this paper, we have studied the problem of query planning for conjunctive queries over sources with access limitations. We have surveyed existing query planning techniques that can be applied to this context statically, before query execution, to restrict the query plan to those accesses that are relevant for at least one instance of the sources. Once a query plan is at hand, we can exploit dependencies on the schema in order to determine dynamically, during query execution, whether an access has the potential to extract data that have not been retrieved earlier by previous accesses of the query plan. In particular, we have studied dynamic optimization under functional dependencies and simple full-width inclusion dependencies, which are typical constraints used to model data sources accessible through Web forms. For these dependencies, we have provided an algorithm that decides their implication in polynomial time. Based on this result, we have further provided a necessary and sufficient condition for deciding dynamic relevance of accesses that can be tested in polynomial time in the size of the schema and of the data extracted until the time of the test, assuming a fixed maximum source arity.

Future directions of research include the integration of the results discussed in this paper with other optimization opportunities that might stem from the

query containment problem for this context. In particular, upper bounds for testing conjunctive query containment are known in the context of sources with access limitations, but with no dependencies; algorithms, if they exist, for deciding containment under access limitations *and* dependencies (such as FDs and SFWIDs) still need to be studied. In particular, in [Calì and Martinenghi, 2008a] it is shown that conjunctive query containment can be decided in coNExpTime with a technique based on a particular kind of chase, called *crayfish chase* since it somehow proceeds backwards. It may be possible that the crayfish chase can be combined with the traditional chase based on inclusion and functional dependencies in order to test containment under both access limitations and dependencies. It should also be noted that the coNExpTime bound is not known to be tight, since the best hardness result known for the problem of conjunctive query containment under access limitations is NP-hardness, as a consequence of the NP-completeness of the ordinary conjunctive query containment problem.

Another natural direction of research is to study and extend the algorithms presented in this paper for dynamic optimization to more expressive classes both of queries, by adding, e.g., union and negation, and of dependencies, by considering, e.g., more general kinds of inclusion dependencies than simple full-width inclusion dependencies.

Acknowledgments

Andrea Calì was partially supported by the EPSRC project "Schema Mappings and Automated Services for Data Integration and Exchange" (EP/E010865/1). Diego Calvanese and Davide Martinenghi acknowledge support from Italian PRIN project "New technologies and tools for the integration of Web search services". Diego Calvanese has also been partially supported by FET project TONES (Thinking ONtologiES), funded within the EU 6th Framework Programme under contract FP6-7603.

References

[Abiteboul et al., 1995] Abiteboul, S., Hull, R., and Vianu, V. (1995). *Foundations of Databases*. Addison Wesley Publ. Co., Reading, Massachussetts.

[Armstrong, 1974] Armstrong, W. W. (1974). Dependency structures of data base relationships. In *IFIP Congress*, pages 580–583.

[Beeri and Bernstein, 1979] Beeri, C. and Bernstein, P. A. (1979). Computational problems related to the design of normal form relational schemas. *ACM Trans. on Database Systems*, 4(1):30–59.

[Beeri and Vardi, 1984] Beeri, C. and Vardi, M. Y. (1984). A proof procedure for data dependencies. *J. of the ACM*, 31(4):718–741.

[Calì and Calvanese, 2002] Calì, A. and Calvanese, D. (2002). Optimized querying of integrated data over the Web. In *Proc. of the IFIP WG8.1 Working Conference on Engineering Information Systems in the Internet Context (EISIC 2002)*, pages 285–301. Kluwer Academic Publishers.

[Calì and Martinenghi, 2008a] Calì, A. and Martinenghi, D. (2008a). Conjunctive query containment under access limitations. In *Proc. of ER 2008*, page to appear.

[Calì and Martinenghi, 2008b] Calì, A. and Martinenghi, D. (2008b). Querying data under access limitations. In *Proc. of ICDE 2008*, pages 50–59. IEEE Computer Society.

[Casanova et al., 1984] Casanova, M. A., Fagin, R., and Papadimitriou, C. H. (1984). Inclusion dependencies and their interaction with functional dependencies. *J. of Computer and System Sciences*, 28(1):29–59.

[Chandra and Vardi, 1985] Chandra, A. K. and Vardi, M. Y. (1985). The implication problem for functional and inclusion dependencies is undecidable. *SIAM J. on Computing*, 14(3):671–677.

[Cosmadakis and Kanellakis, 1986] Cosmadakis, S. S. and Kanellakis, P. C. (1986). Functional and inclusion dependencies - A graph theoretical approach. In Kanellakis, P. C. and Preparata, F. P., editors, *Advances in Computing Research, Vol. 3*, pages 163–184. JAI Press.

[Cosmadakis et al., 1990] Cosmadakis, S. S., Kanellakis, P. C., and Vardi, M. (1990). Polynomial-time implication problems for unary inclusion dependencies. *J. of the ACM*, 37(1):15–46.

[Dembinski and Maluszynski, 1985] Dembinski, P. and Maluszynski, J. (1985). AND-parallelism with intelligent backtracking for annotated logic programs. In *Proc. of SLP 1985*, pages 29–38.

[Deutsch et al., 2005] Deutsch, A., Ludäscher, B., and Nash, A. (2005). Rewriting queries using views with access patterns under integrity constraints. In *Proc. of the 10th Int. Conf. on Database Theory (ICDT 2005)*, pages 352–367.

[Duschka and Levy, 1997] Duschka, O. M. and Levy, A. Y. (1997). Recursive plans for information gathering. In *Proc. of the 15th Int. Joint Conf. on Artificial Intelligence (IJCAI'97)*, pages 778–784.

[Florescu et al., 1999] Florescu, D., Levy, A. Y., Manolescu, I., and Suciu, D. (1999). Query optimization in the presence of limited access patterns. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 311–322.

[Floyd, 1962] Floyd, R. W. (1962). Algorithm 97: Shortest path. *Communications of the ACM*, 5(6):345.

[Halevy, 2001] Halevy, A. Y. (2001). Answering queries using views: A survey. *Very Large Database J.*, 10(4):270–294.

[Jones et al., 1976] Jones, N., Lien, Y., and Lasser, W. (1976). New problems complete for nondeterministic log space. *Math. Systems Theory*, 10:1–17.

[Levene and Loizou, 2001] Levene, M. and Loizou, G. (2001). Guaranteeing no interaction between functional dependencies and tree-like inclusion dependencies. *Theor. Comput. Sci.*, 254(1-2):683–690.

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

[Li and Chang, 2000] Li, C. and Chang, E. (2000). Query planning with limited source capabilities. In *Proc. of the 16th IEEE Int. Conf. on Data Engineering (ICDE 2000)*, pages 401–412.

[Li and Chang, 2001a] Li, C. and Chang, E. (2001a). Answering queries with useful bindings. *ACM Trans. on Database Systems*, 26(3):313–343.

[Li and Chang, 2001b] Li, C. and Chang, E. (2001b). On answering queries in the presence of limited access patterns. In *Proc. of the 8th Int. Conf. on Database Theory (ICDT 2001)*, pages 219–233.

[Ludäscher and Nash, 2004] Ludäscher, B. and Nash, A. (2004). Processing union of conjunctive queries with negation under limited access patterns. In *Proc. of the 9th Int. Conf. on Extending Database Technology (EDBT 2004)*, pages 422–440.

[Maier, 1980] Maier, D. (1980). Minimum covers in the relational database model. *J. of the ACM*, 27(4):664–674.

[Millstein et al., 2003] Millstein, T. D., Halevy, A. Y., and Friedman, M. (2003). Query containment for data integration systems. *J. of Computer and System Sciences*, 66(1):20–39.

[Millstein et al., 2000] Millstein, T. D., Levy, A. Y., and Friedman, M. (2000). Query containment for data integration systems. In *Proc. of the 19th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS 2000)*, pages 67–75.

[Mitchell, 1983] Mitchell, J. C. (1983). The implication problem for functional and inclusion dependencies. *Information and Control*, 56:154–173.

[Nash and Ludäscher, 2004] Nash, A. and Ludäscher, B. (2004). Processing first-order queries under limited access patterns. In *Proc. of the 23rd ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS 2004)*, pages 307–318.

[Rajaraman et al., 1995] Rajaraman, A., Sagiv, Y., and Ullman, J. D. (1995). Answering queries using templates with binding patterns. In *Proc. of the 14th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'95)*.

[Yang et al., 2006] Yang, G., Kifer, M., and Chaudhri, V. K. (2006). Efficiently ordering subgoals with access constraints. In *Proc. of the 25th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS 2006)*, pages 22–22.

[Yerneni et al., 1999] Yerneni, R., Li, C., Garcia-Molina, H., and Ullman, J. D. (1999). Computing capabilities of mediators. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 443–454.