Complexity of Conjunctive Query Answering under Access Limitations (preliminary report)

Andrea Calì[1,2] and Igor Razgon[1]

[1]Dept of Computer Science
Birkbeck, Univ. of London, UK

[2]Oxford-Man Inst. of Quantitative Finance
University of Oxford, UK

{andrea,igor}@dcs.bbk.ac.uk

Abstract. The Deep Web consists of data accessible through HTML forms but not as web pages; usually such data are modelled as relations that can be queried only by operating a selection on certain attributes — such restrictions are called access limitations. In this paper we illustrate the problem of Boolean conjunctive query answering under access limitations; we define and motivate the problem's two main cases, we provide some preliminary results on its computational complexity and we suggest some research directions.

1 Introduction

Data Exchange and Integration [9, 12, 17] are core problems in data management, especially in the case where heterogeneous data sources, possibly on the Web, are integrated as a single database. In this scenario, it is often the case that data sources impose *access limitations*, i.e., they require that the query that is executed on them has a special form. In particular, in the relational case, certain (fixed) attributes are required to be selected, i.e., associated to a constant. This is true, for instance, when the data source is accessible through a web form, that requires some fields to be filled in, or in some legacy databases.

The presence of access limitations complicates query processing, as they prevent data to be queried freely. Therefore, limitations need to be circumvented with *recursive* query plans [16, 14], expressible in Datalog. The following example is taken from [4].

Example 1. Consider the following relational sources: $r_1(\mathit{Title}, \mathit{City}, \mathit{Artist})$, representing information about concerts, with song title, city of performance, and artist name, and requiring the second attribute to be selected; $r_2(\mathit{Artist}, \mathit{Nation}, \mathit{City})$, representing name, nationality and city of birth of artists, and requiring the first attribute to be selected. In this case, given the conjunctive query $q(A) \leftarrow r_2(A, \mathit{italian}, \mathit{modena})$ asking for names of Italian artists born in Modena, we notice that q cannot be immediately evaluated, since r_2 requires the first attribute to be bound to a constant (selected). However,

the two attributes named *City* in r_1 and r_2 both represent city names, and similarly the attributes named *Artist* represent artist names[1]. In such a case, we can use names of artists extracted from r_1 to access r_2 and thus extract tuples that may contribute to the answer. More precisely, we start from the constant *modena* present in the query, and access r_1; this will return tuples with new artist names; such constants (artist names) can be used to access r_2. In turn, new tuples from r_2 may provide new constants representing city names, that can be used to access r_1, and so on. Once this recursive process has terminated, we can evaluate the query on the retrieved tuples. □

The issue of processing queries under access limitations has been widely investigated in the literature [1, 16, 14, 13, 10, 8]; in particular, [10] considers the optimization of non-recursive plans, [8] addresses the problem in the case of query answering using views, and [16] presents a polynomial-time algorithm to decide whether a CQ can be answered in the presence of access limitations. Recursive query plans were introduced in [15], where the query containment problem is addressed. Several works [5, 2, 3] deal with reducing (or optimising) query processing under access limitations, both dynamically (at query processing time) and statically (at query plan generation time) — see [1] for a survey of the related work on this.

In this paper we address the problem of Boolean conjunctive query (BCQ) containment in the presence of access limitations, and provide some results on its complexity. In particular:

1. We state and motivate two variants of the BCQ answering (decision) problem in the case of access limitations: the restricted case, where relations are to be accessed only according to their limitations, and the restricted case, where relations can be freely accessed.
2. We show that the introduction of access limitations does not increase the complexity of BCQ answering if the predicate arities are bounded.
3. We show that in the unrestricted case, even in the case of unbounded arities, the complexity of BCQ answering is the same as in the absence of limitations.
4. For the unrestricted case, we have devised a backward-style algorithm for determining whether a fact of the database is obtainable by querying the data according to the access limitations. We argue that this algorithm is, in practice, more efficient than the obvious forward-style one.
5. We argue that our preliminary results could lead us to extend the results on tractable classes of BCQs in the absence of limitations to our case with access limitations.

2 Preliminaries

In this section we present a formalisation of the problem of answering queries under access limitations.

[1] This is captured by the notion of *abstract domain* (see Section 2), which we do not take into account in this paper.

We consider relations as sets of facts whose arguments (a.k.a. attributes) are values belonging to an infinite domain, which we denote with Δ. In some cases (see e.g. [5]) some more specific domains, called *abstract domains*, are associated to attributes; these attributes are used to distinguish, for instance, strings representing names from strings representing car registration numbers. In this paper, however, this distinction is not relevant, and we assume that each attribute can have values in Δ.

Access limitations on a relation are constraints that impose that certain attributes must be *selected* (bound to a constant) for the relation to be accessed. More formally, a schema with access limitations is a pair $\langle \mathcal{R}, \Lambda \rangle$, where

(i) $\mathcal{R}$ is a set of relational predicates (i.e. a relational schema), each with an associated *arity* (number of arguments); a predicate r of arity n is denoted r/n; its j-th argument (or position, or attribute), with $1 \leq j \leq n$, is denoted by $r[j]$.

(ii) Λ is a set of access limitations that specifies, for every attribute of every relational predicate, whether it is an *input* or an *output* attribute; in order to access a relation, all input attributes must be selected[2].

For convenience of notation, we indicate the access limitations of each relation as a sequence, of 'i' and 'o' symbols written as a superscript in the signature of the relation; an 'i' (resp., 'o') indicates that the corresponding argument is an input (resp., output) argument. A signature has therefore the form r^{Λ_r}/n, where Λ_r the access limitation on r. We sometimes indicate a sequence of terms (i.e., variables or constants) $t_1, \ldots, t_n$ as $\mathbf{t}$ and its length n as $|\mathbf{t}|$. An atom $\mathbf{a}$ is a formula of the form $\mathbf{a} = r(t_1, \ldots, t_n)$; if all $t_1, \ldots, t_n$ are constants of Δ, $\mathbf{a}$ is said to be *ground*; a ground atom is also called *fact*. The *instance* of a predicate $r \in \mathcal{R}$ is a set of (ground) facts of the form $r(c_1, \ldots, c_n)$ such that $c_i \in \Delta$ for all i such that $1 \leq i \leq n$. A *database instance*, or simply *database*, of a schema $\mathcal{R}$ is the union of the instances of all predicates r in $\mathcal{R}$. Given a database D, the instance of r in D is denoted $r(D)$; it is constituted of all facts of D having r as predicate. Given the set of atoms S, we denote with $\mathrm{dom}(S)$ the *active domain* of S, that is, the set of all values appearing as arguments of atoms in S. In the following, we shall implicitly assume that we are considering a schema $\mathcal{S} = \langle \mathcal{R}, \Lambda \rangle$, if not otherwise stated.

A *conjunctive query* (CQ) q of arity n over a schema $\mathcal{S}$ is written in the form $q(\mathbf{X}) \leftarrow conj(\boldsymbol{X}, \boldsymbol{Y})$, where $|\mathbf{X}| = n$, $q(\mathbf{X})$ is called the *head* of q, $conj(\mathbf{X}, \mathbf{Y})$ is called the *body* of q and is a conjunction of atoms involving the variables in $\mathbf{X}$ and $\mathbf{Y}$ and possibly some constants, and the predicate symbols of the atoms are in $\mathcal{R}$; $q(\mathbf{X})$ is denoted as $\mathrm{head}(q)$, the set of atoms in the body is denoted as $\mathrm{body}(q)$, and $|q|$ denotes $|\mathrm{body}(q)|$ (number of atoms in q); notice that the head predicate q is not in $\mathcal{R}$. The set of constants appearing in q is denoted $\mathrm{const}(q)$, the set of variables $\mathrm{var}(q)$.

[2] In general, there could be more than one annotation for each predicate, that is, more than one way of accessing the corresponding relation. However, in this paper we assume there is exactly one access limitation (or pattern) per predicate. Our results are easily extended to the general case.

$$
\begin{array}{ll}
\rho_1: & q() \leftarrow \hat{r}(X,Y), \hat{s}(Z,Y) \\
\rho_2: & \hat{r}(X,Y) \leftarrow dom(X), r(X,Y) \\
\rho_3: & \hat{s}(X,Y) \leftarrow dom(X), s(X,Y) \\
\rho_4: & dom(X) \leftarrow \hat{s}(X,Y)
\end{array}
\qquad
\begin{array}{ll}
\rho_5: & dom(Y) \leftarrow \hat{s}(X,Y) \\
\rho_6: & dom(X) \leftarrow \hat{r}(X,Y) \\
\rho_7: & dom(Y) \leftarrow \hat{r}(X,Y) \\
\rho_8: & dom(a)
\end{array}
$$

Fig. 1. Datalog program for Example 2

In the following we shall extensively use the notion of *mapping* from terms to terms, and typically we will map variables to terms. The term resulting from the application of such a mapping μ to a term t is written $\mu(t)$; note that μ also induces a mapping from an atom $\mathbf{f} = r(t_1, \ldots, t_n)$ to another atom indicated $\mu(\mathbf{f}) = r(\mu(t_1), \ldots, \mu(t_n))$, and from a set of atoms $S = \{\mathbf{f}_1, \ldots, \mathbf{f}_m\}$ to another set of atoms indicated $\mu(S) = \{\mu(\mathbf{f}_1), \ldots, \mu(\mathbf{f}_m)\}$. An *homomorphism* μ from a set of atoms S_1 to another set of atoms S_2 is a mapping that sends every constant into itself and all atoms of S_1 into atoms of S_2, that is, $\mu(S_1) \subseteq S_2$.

Given a database D, the *answer* $q(D)$ to a CQ q on D is the set of tuples $\langle \mathbf{c} \rangle$ of constants, with $|\mathbf{c}| = |\text{head}(q)|$, such that there is an homomorphism that sends body(q) to atoms (facts) of D and head(q) to $q(\mathbf{c})$. A a *Boolean* CQ (BCQ) is a CQ whose head predicate has arity zero. A BCQ is often represented, for convenience, as the set of its body atoms. A BCQ q has positive answer on a database D if there exists a homomorphism μ from body(q) to D (or better from q to D, being q represented as a set of atoms). In this case we write $D \models q$. In the following we will concentrate, w.l.o.g., on BCQs only.

In the presence of access limitations on the sources, queries cannot be usually evaluated as in the traditional case, as we show in Example 2. Given a query, an algorithm exists (see, e.g., [5]) that retrieves, starting from a set of initial constants of $\mathcal{A}$, all the *obtainable* facts in a database, according to the access limitations. The query is then evaluated on the obtainable facts, as all non-obtainable ones cannot contribute to the answer. Roughly speaking, the computation of the obtainable facts starts from an initial sets of constants, that must include those appearing in the query; with such constants, we access all the relations we can, according to their access limitations. The new obtained facts (if any exists) contain other constants, that are used again to access relations in all possible ways, and the process goes on until no new facts are obtained.

Definition 1. *Given a BCQ q posed over a schema $\mathcal{S} = \langle \mathcal{R}, \Lambda \rangle$, a set of constants $I \subseteq \mathcal{A}$, and a database D for $\mathcal{R}$, we denote the set of (Λ, I)-obtainable tuples of D under Λ and I as $\rho_{\Lambda,I}(D)$. We say that q has positive answer on D under Λ and I if $\rho_{\Lambda,I}(D) \models q$. In this case, we write $D \models_{\Lambda,I} q$.*

Example 2. Consider a schema with predicates $r^{io}/2$ and $s^{io}/2$, a set of initial constants $I = \{a\}$, and the BCQ $q = \{r(X,Y), s(Z,Y)\}$. The Datalog program Π_q for q is shown in Figure 1. The query is rewritten over the *cache* relations $\hat{r}, \hat{s}$ (rule ρ_1) defined in the cache rules ρ_2 and ρ_3; the cache relations contain the obtainable facts. Rules ρ_2 and ρ_3 also ensure that the facts that are stored in the caches are retrieved from the sources according to the access limitations

— in our case, we access r and s only if we have a constant in *dom* (relation that containts all extracted or initial constants) with which to "bind" (select) the input argument. Rules $\rho_4 - \rho_7$ put all extracted constants in *dom*. □

Notice that if Π_q entails a fact $\mathbf{a}$ in a cache relation ($\hat{r}(D)$ or $\hat{s}(D)$, in Example 2 above), then $\mathbf{a}$ is in $\rho_{\Lambda,I}(D)$ or, equivalently, $\mathbf{a}$ is (Λ, I)-obtainable.

Example 3. Consider again the schema of Example 2, where $I = \{a\}$, and the database $D = \{r(a,b), r(c,d), s(b,c), s(e,a)\}$. In computing $\rho_{\Lambda,I}(D)$ we start from a and obtain $r(a,b)$, then with b we obtain $s(b,c)$, and finally with c we obtain $r(c,d)$. The fact $s(e,a)$ is not (Λ, I)-obtainable. □

3 The Query Answering Problem

In this section we formally define the problem of answering BCQs on databases under access limitations, given a set of initial constants. We identify two variants of such problem, depending on the setting.

Definition 2. *Given a database D, a set of initial constants $I \subseteq \Delta$, a set Λ of access limitations and a BCQ q, the problem of answering q on D under Λ and I amounts to determine whether $D \models_{\Lambda,I} q$. We have two variants of the problem, which we define below.*

(i) BCQ answering under access limitations, unrestricted *case: this is the problem of determining whether $D \models_{\Lambda,I} q$, while having* arbitrary *access to the relations of D.*
(ii) BCQ answering under access limitations, restricted *case: this is the problem of determining whether $D \models_{\Lambda,I} q$, while having access to the relations of D* only *according to Λ.*

Notice that the problem in the restricted case is the "classic" case [5, 4], where we are computing the answers to a BCQ having only limited access to the data, according to Λ. The BCQ answering problem in the unrestricted case is also relevant in real-world scenarios; we now explain why. Assume that access limitations are enforced by an organisation in order to limit access to data by external users (e.g., those outside the organisation). The organisation has arbitrary access to the data, and it is interested in determining what external users, whose access to the data is limited by Λ, can retrieve by posing certain queries; in order to determine this, of course, an algorithm will have the advantage of freely accessing the data, regardless of access limitations. In the next section we shall see how these two settings affect the complexity of BCQ answering.

4 Complexity

In this section we illustrate some preliminary results about the computational complexity of BCQ answering under access limitations. We consider instances

of the BCQ answering problem of the form $\langle D, \Lambda, I, q\rangle$, where D is a database for $\mathcal{R}$, Λ the access limitations on $\mathcal{R}$ ($\mathcal{S} = \langle \mathcal{R}, \Lambda\rangle$ is implicit), $I \subseteq \Delta$ a set of initial constants, and q a BCQ. We now present some preliminary results, first in the restricted case, and then in the unrestricted case. We will see that, not surprisingly, the latter case allows for more efficient BCQ processing. In particular, let W be the maximum arity of predicates in $\mathcal{R}$. In the restricted case, to check whether an atom is (Λ, I)-obtainable requires time exponential in W, while in the unrestricted case the same check can be executed in polynomial time.

4.1 Restricted Case

In the restricted case, one can only explore $\rho_{\Lambda,I}(D)$ in a "blind" fashion, trying all possible accesses to the relations — see Example 2. From the considerations in Example 2 and [7], we obtain the following.

Proposition 1. *Given an instance a database D for a schema $\mathcal{R}$ with access limitations Λ, $\rho_{\Lambda,I}(D)$ can be computed in time polynomial in $|D|$ and exponential in W.*

As an immediate corollary, we obtain that, if W is fixed, the complexity of BCQ answering is not altered by the presence of access limitations. This holds because we can simply compute $\rho_{\Lambda,I}(D)$ in polynomial time and then evaluate q on it. Alternatively, we can guess an homomorphism $\mu : \mathrm{var}(q) \to \mathrm{dom}(D)$, compute $\rho_{\Lambda,I}(D)$, and then check whether $\mu(q) \subseteq \rho_{\Lambda,I}(D)$.

Corollary 1. *BCQ answering under access limitations in the restricted case, for fixed W, is* NP*-complete.*

Notice that in the restricted case there is no hope of checking (deterministically) whether a fact $\mathbf{a}$ is in $\rho_{\Lambda,I}(D)$ in time less than exponential in W, as shown by the following example.

Example 4. Consider the schema $\mathcal{R} = \{r^{i\cdots i}/k\}$ (with k input arguments and no output arguments), the database $D = \{r(a_1, \ldots, a_k)\}$, and the set $I = \{c_1, \ldots, c_m\} \supseteq \{a_1, \ldots, a_{k+1}\}$. In order to determine whether $\mathbf{a} = r(a_1, \ldots, a_k)$ is (Λ, I)-obtainable (in fact, it is), we need to try to access the relation $r(D)$ with all possible k-tuples of constants of I; in the worst case, this will necessarily require m^k accesses (queries) to $r(D)$. As a consequence, every deterministic BCQ answering algorithm, even for atomic BCQs, will require time exponential in W; consider, for instance, the query $q = \{r(X_1, \ldots, X_{k+1})\}$. □

4.2 Unrestricted Case

We now come to BCQ answering in the unrestricted case. We first show that $\rho_{\Lambda,I}(D)$ can be computed in time polynomial in $|D|$ and W. This can be shown

by exhibiting an algorithm that computes all (Λ, I)-obtainable facts similarly to the restricted case, but with the following difference. Consider again Example 4, but with a different D. Instead of trying all possible k-tuples of constants of I, the algorithm gets all facts of $r(D)$ and considers as obtainable those whose input arguments are in $\{c_1, \ldots, c_m\}$. The constants in the output arguments of the so obtained facts are then added to $\{c_1, \ldots, c_m\}$ and the new set is then used again to discover new obtainable facts, and so on.

Corollary 2. *BCQ answering under access limitations in the restricted case is* NP*-complete.*

The above technique, like the one for the restricted case, has the disadvantage of "blindly" trying to reach all possible obtainable facts. Indeed, a smarter technique exists, which proceeds backwards from a fact $\mathbf{r}$ to determine if it is (Λ, I)-obtainable. The algorithm constructs a graph whose nodes are labelled with facts of D, and the children of each node labelled with $\mathbf{a}$ are those atoms of D that *could* provide constants to retrieve $\mathbf{a}$. The graph is then visited to check whether there actually is a "proof" of the (Λ, I)-obtainability of $\mathbf{r}$.

Example 5. Assume a schema $\mathcal{R} = \{r_1^{iio}/3, r_2^{ioo}/3\}$, and that we have the atom $\mathbf{r} = r_1(a, b, c)$ labelling a node β. To determine the children of β, the following queries are executed on the database D, using $C = \{a, b\}$ (we use the head predicate q):

$$\begin{array}{ll} q(X, Y, a) \leftarrow r_1(X, Y, a) & q(X, b, Z) \leftarrow r_2(X, b, Z) \\ q(X, Y, b) \leftarrow r_1(X, Y, b) & q(X, Y, a) \leftarrow r_2(X, Y, a) \\ q(X, a, Z) \leftarrow r_2(X, a, Z) & q(X, Y, b) \leftarrow r_2(X, Y, b) \end{array}$$

Each obtained tuple (without repetitions), will label a successor of β. □

5 Discussion

We have presented some preliminary results on the complexity BCQ answering under access limitations. First, we have identified and motivated two principal settings: the restricted case, where all relations are accessible only according to the access limitations, and the unrestricted one, where relations are freely accessible. We have shown that access limitations do not increase the complexity of BCQ answering, provided that in the restricted case the arity of the predicates is bounded. We have shown that in the unrestricted case we can check whether a fact of the database is obtainable under limitations ((Λ, I)-obtainable) by means of an algorithm that performs a backward search for a "proof" that the fact can be obtained.

Future work and directions. We argue that the aforementioned backward algorithm is more efficient, in real world scenarios, than the simple forward one, as

the latter proceeds "blindly" in a breadth-first fashion. We are setting up experiments to validate our claim. We suggest that the backward algorithm might be "plugged" into known algorithms for CQ answering in the case of tractable queries — possibly acyclic, bounded-treewidth [6] or bounded-hypertree-width queries [11]. This will allow us to study tractable cases of BCQ answering in the presence of access limitations, verifying whether the introduction of such limitations increases the computational complexity of the problem.

Acknowledgments. Andrea Calì acknowledges support by the EPSRC project "Logic-based Integration and Querying of Unindexed Data" (EP/E010865/1).

References

1. Vince Bárány, Michael Benedikt, and Pierre Bourhis. Access patterns and integrity constraints revisited. In *Proc. of ICDT 2013*, pages 213–224, 2013.
2. Michael Benedikt, Pierre Bourhis, and Clemens Ley. Querying schemas with access restrictions. *PVLDB*, 5(7):634–645, 2012.
3. Michael Benedikt, Georg Gottlob, and Pierre Senellart. Determining relevance of accesses at runtime. In *Proc. of PODS*, pages 211–222, 2011.
4. Andrea Calì and Davide Martinenghi. Conjunctive query containment under access limitations. In *Proc¿ of ER 2008*, pages 326–340, 2008.
5. Andrea Calì and Davide Martinenghi. Querying data under access limitations. In *Proc. of ICDE 2008*, pages 50–59, 2008.
6. Chandra Chekuri and Anand Rajaraman. Conjunctive query containment revisited. *Theor. Comput. Sci.*, 239(2):211–229, 2000.
7. Evgeny Dantsin, Thomas Eiter, Georg Gottlob, and Andrei Voronkov. Complexity and expressive power of logic programming. *ACM Comput. Surv.*, 33(3):374–425, 2001.
8. Oliver M. Duschka and Alon Y. Levy. Recursive plans for information gathering. In *Proc. of IJCAI 1997*, pages 778–784, 1997.
9. Ronald Fagin, Phokion G. Kolaitis, Renée J. Miller, and Lucian Popa. Data exchange: Semantics and query answering. *Ther. Comp. Sci.*, 336(1):89–124, 2005.
10. Daniela Florescu, Alon Y. Levy, Ioana Manolescu, and Dan Suciu. Query optimization in the presence of limited access patterns. In *Proc. of SIGMOD 1999*, pages 311–322, 1999.
11. Georg Gottlob, Nicola Leone, and Francesco Scarcello. Hypertree decompositions and tractable queries. *J. Comput. Syst. Sci.*, 64(3):579–627, 2002.
12. Richard Hull. Managing semantic heterogeneity in databases: A theoretical perspective. In *Proc. of PODS 1997*, pages 51–61, 1997.
13. Chen Li and Edward Chang. Answering queries with useful bindings. *Trans. on Datab. Syst.*, 26(3):313–343, 2001.
14. Chen Li and Edward Chang. On answering queries in the presence of limited access patterns. In *Proc. of ICDT 2001*, pages 219–233, 2001.
15. Todd D. Millstein, Alon Y. Levy, and Marc Friedman. Query containment for data integration systems. In *Proc of PODS 2000*, pages 67–75, 2000.
16. Anand Rajaraman, Yehoshua Sagiv, and Jeffrey D. Ullman. Answering queries using templates with binding patterns. In *Proc. of PODS 1995*, 1995.
17. Jeffrey D. Ullman. Information integration using logical views. In *Proc. of ICDT 1997*, pages 19–40, 1997.