Containment of Conjunctive Queries under Access Limitations (extended abstract)

Andrea Calì and Diego Calvanese

Faculty of Computer Science
Free University of Bolzano-Bozen
piazza Domenicani 3, I-39100 Bolzano, Italy
ac@andreacali.com, calvanese@inf.unibz.it

Abstract. Relational data may have access limitations, i.e., relations may require certain attributes to be selected when they are accessed; this happens, for instance, while querying web data sources (wrapped in relational form) or legacy databases. It is known that the evaluation of a conjunctive query under access limitations requires a recursive algorithm that is encoded into a Datalog program. In this paper we address the problem of containment of conjunctive queries under access limitations, which is highly relevant in query optimization. Checking containment in this case would amount to check containment of recursive Datalog programs, which is undecidable in general. We show however, that due to the specific form of the Datalog programs resulting from encoding access limitations, the containment problem is in fact decidable. We provide a decision procedure based on chase techniques, and study its computational complexity.

1 Introduction

Integrated access of data over the Web is a prominent application field for information integration [6]. Information Integration, in general, is the problem of combining data residing at different, heterogeneous sources, by providing the user with a uniform access to the data [8, 14]. Often, in the context of Web data integration, no direct access to the underlying database is provided: data are accessible only via forms, where typically certain fields are required to be filled in by the user in order to obtain a result. Consider for example the DBLP site[1], where, in order to obtain a list of publications extracted from an underlying database, either an author or a title has to be specified, while it is not possible to ask directly, e.g., for all publications of a certain conference[2]. Also, another case where (relational) sources have limited access is the case of legacy data sources. Limitations on how sources can be accessed significantly complicate query processing [13, 10, 7, 5, 12, 4], since in this case the known techniques for query answering are in general not sufficient. As shown in [13, 10, 11], query

[1] http://dblp.uni-trier.de/
[2] We do not consider here advanced search, which is also supported at DBLP.

answering in the presence of access limitations in general requires the evaluation of a recursive query plan, which can be suitably expressed in Datalog.

Since source accesses are costly, an important issue is how to minimize the number of accesses to the sources while still being guaranteed to obtain all possible answers to a query. [10, 11] discuss several optimisations that can be made at compile time, during query plan generation. However, the presented techniques are not applicable in the case where user queries and view definitions are arbitrary conjunctive queries. A technique for optimising query answering for the full class of conjunctive queries is presented in [2], together with a runtime optimisation technique.

To the best of our knowledge, optimisation techniques based on *query containment* [9, 3] have not been yet studied in the case of access limitations. Containment of queries is a well-recognised problem in query evaluation and optimisation. In this paper we address the problem of checking containment of conjunctive queries in the presence of access limitations on the data sources. In particular:

1. We clearly state the problem in the case of access limitations, showing that it amounts to checking containment between two recursive Datalog programs (problem that is in general undecidable).
2. We introduce a novel formal tool to check containment of a query Q_1 into another query Q_2 under access limitations, namely the *backward-chase*, that is a set of databases that are representative of all database that provide an answer to Q_1. The backward chase is in general an infinite set.
3. We show the decidability of the problem of containment in this setting, by proving that, in order to check containment, only database of a limited size in the chase need to be considered.
4. Finally, we show an upper bound to the complexity of the query containment problem.

2 Preliminaries

In this section we present the formal framework in which we address the problem of query containment. We will consider *conjunctive queries*, for the syntax and semantics of which we refer the reader, for instance, to [1]; we shall use Datalog notation for such queries. We will consider (relational) database whose values belong to a domain Δ of constants, and to another domain Δ_F of *fresh* constants, needed for technical reasons, and such that $\Delta \cap \Delta_F = \emptyset$. The answer to a conjunctive query Q evaluated over a database B is denoted $Q(B)$. The head and the body of a conjunctive query Q are denoted with $\text{head}(Q)$ and $\text{body}(Q)$ respectively; the set of constants appearing in Q is denoted $\text{const}(Q)$. We denote with $|Q|$ the number of atoms in a query Q.

In the following, we shall need the notion of *homomorphism*, which can be used to define the answers $Q(B)$ to a query Q evaluated over a database B. A homomorphism is a function from the symbols of a query to $\Delta \cup \Delta_F$, that sends constants of Δ to themselves, and induces a well-defined map from conjuncts of Q to tuples of the corresponding relations in B, i.e. it maps atoms of the form

$R(d_1, \ldots, d_n)$ to tuples of the relation R in B. We recall that a tuple t is in $Q(B)$ if and only if there is a homomorphism that sends body(Q) to tuples of B and head(Q) to t.

To each attribute in a relation 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, which is needed to distinguish, e.g., strings representing person names from strings representing plate numbers.

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, we have the following:

1. we consider a relational schema $\mathcal{S} = \langle \mathcal{R}, \Pi \rangle$ with access limitations;
2. $\mathcal{R}$ is a set of relational predicates, each with an associated arity;
3. every attribute of a relational predicate R has an *abstract domain*;
4. Π is a set of access limitations, that specifies, for every atribute of some relational predicate, whether it is free or *bound*; in order to access a relation in a query, all bound attributes (denoted with the subscript b here) must be selected.

In the presence of access limitations on the sources, evaluating a query in the ordinary way does not yeld the same results as in the case without limitations as shown by the following example.

Example 1. Consider a relation $S_1(\mathit{Title}, \mathit{Year}^b, \mathit{Artist})$, representing information about title, year and artist (composer or performer) of songs, and another relation $S_2(\mathit{Artist}^b, \mathit{Nation}, \mathit{YOB})$, representing name, nationality and year of birth of artists. In this case, given the conjunctive query

$$Q(A) \leftarrow S_2(A, \mathit{italian}, 1950)$$

asking for names of Italian artists born in 1950, we notice that Q cannot be immediately evaluated, since S_2 requires the first attribute to be bound to a constant (selected). Therefore, simple evaluation produces an empty answer to Q, for every database. However, supppose that the both the two attributes *Year* and *YOB* have the same abstract domain, representing years; similarly, the attributes named *Artist* share the same abstract domain; in such a case, likely to happen in practice, we could use names of artists extracted from S_1 to access S_2 and extract tuples that may contribute to the answer. More precisely, we start from the constant 1950, present in the query, and access S_1; this will return tuples with new artist names; such constants (artist names) can be used to access S_2. In turn, new tuples from S_2 may provide new constants representing years, that can be used to access S_1, and so on. Once this recursive process has terminated, we have retrieved the maximum number of tuples that can contribute to the answer.

Given a query over the data sources, an algorithm exists [10] that retrieves all the *obtainable* tuples in the answer to the query. Such an algorithm consists in

the evaluation of a suitable Datalog program which extracts all obtainable tuples starting from a set of initial values, as described in the previous example. The Datalog program, which we do not describe in detail due to space limitations, is constructed by encoding in Datalog clauses the limitations on the sources that must be respected during evaluation of the query. The evaluation of the Datalog program is done as follows: starting from a set of initial values, that must include those appearing in the query, we access all the relations we can, according to their binding patterns. With the new tuples obtained (if any), we obtain new values with which to access the relations again, getting from them new tuples, and so on, until we have no way of doing accesses with new constants. The program extracts all tuples obtainable while respecting the access limitations, but there may be tuples in the sources that cannot be retrieved. Given a query Q posed over a schema $\mathcal{S} = \langle \mathcal{R}, \Pi \rangle$ with access limitations, a set of constants $I \subseteq \Delta$, and a database B for $\mathcal{R}$, we denote the answers obtained through the recursive evaluation described above as $\text{ans}(Q, \mathcal{S}, B, I)$. Notice that in general $\text{ans}(Q, \mathcal{S}, B, I) \subseteq Q(B)$.

We now come to the problem of containment. Since, in the presence of access limitations, the only way of accessing the sources to answer a query is to extract the tuples recursively as described above, we will define the containment between two conjunctive queries by considering this query answering technique. As for the set of initial constants, in principle we may have additional constants with respect to those appearing in the two queries; therefore, as set of initial constants, we shall consider a superset of the union of the constants appearing in the two queries.

Definition 1. *Consider two conjunctive queries Q_1, Q_2 over a schema $\mathcal{S} = \langle \mathcal{R}, \Pi \rangle$ with access limitations, and a set of constants I such that $I \supseteq \text{const}(Q_1) \cup \text{const}(Q_2)$; we say that Q_1 is contained in Q_2 under Π with respect to I, denoted $Q_1 \subseteq_{\Pi,I} Q_2$, if for every database B for $\mathcal{R}$ we have $\text{ans}(Q_1, \mathcal{S}, B, I) \subseteq \text{ans}(Q_2, \mathcal{S}, B, I)$.*

From the previous definition it follows that checking containment amounts to check containment between two recursive Datalog programs, which in general is an undecidable problem [1]. However, in the following we will show that, due to the special form of the programs, checking containment under access limitations is indeed decidable.

3 Containment under Access Limitations

In this section we present the foundations of our novel technique to check containment of conjunctive queries under access limitations. Analogously to what is done for containment of conjunctive queries under inclusion and functional dependencies [9], in order to check the containment of a query Q_1 into another query Q_2, we characterise the set of databases that provide an answer tuple for Q_1 by constructing, starting from Q_1, a set of databases called *chase*. In our case, the chase is constructed according to the access limitations is a *set* of databases. With the chase at hand, we can evaluate Q_2 over the chase of Q_1 in

order to check the existance of a counterexample to containment, i.e., a database C that provides an answer tuple to Q_1 that is not in the answer set of Q_2 when evaluated over C.

We now give the definition of the chase of a conjunctive query. The chase starts from the *frozen* body of the query, i.e., the image of the body of the query according to some homomorphism; then, according to the access limitations, we go back in the extraction process, adding to the chase tuples that may lead to the extraction of the previous ones, and we proceed until we decide to stop, adding tuples that can be extracted with the initial constants in the query. Since we proceed somehow backwards, we call our chase *backward-chase*.

Definition 2 (Backward-chase). *Consider a conjunctive query Q over a schema $\mathcal{S} = \langle \mathcal{R}, \Pi \rangle$ with access limitations, and a set of non-fresh constants $I \supseteq \text{const}(Q)$. The* backward-chase *of Q, denoted* $\text{bchase}(Q, S, I)$ *is a set of databases; a database $C \in \text{bchase}(Q, S, I)$ is constructed as follows. For technical reasons, we shall consider C as a* graph *where nodes are tuples of the database, and each node c has a* level*, denoted* $\text{level}(c)$*, that is a non-negative integer; the set of nodes at level h in C will be called* level h *of C.*

1. *We start by* freezing $\text{body}(Q)$*; this is done by choosing a homomorphism μ that sends* $\text{body}(Q)$ *to a set of facts; such facts will stay at level* 0 *of C. The homomorphism μ must send variables in Q to fresh constants in Δ_F.*
2. *We use an auxiliary predicate H (that without loss of generality we can consider to be in $\mathcal{R}$), of same arity as Q, for the head of Q; we will call $\mu(\text{head}(Q))$ the* head of the backward-chase*, denoted* $\text{head}(\text{bchase}(Q, S, I))$.
3. *For each fact $c = R(d_1, \ldots, d_n)$ at level k, and for each bound attribute of R, say the i-th, we have that there is a fact $c' = R'(d'_1, \ldots, d'_m)$ such that d'_i appears in a position corresponding to a free attribute or R', and the attribute associated to d'_i has the same abstract domain as d_i. If c is at level k, the fact c' must be at level $k+1$, and an arc (c', c) is in C. All other constants in c' that do not appear at level k must be* fresh *constants in Δ_F, not appearing in any of the levels lower than k. Notice that c' may have more than one constant that is used to "fill" bound attributes of c, but not of facts at level different from k. Also, the same tuple can appear in different nodes of the graph.*
4. *All the leaves of C are facts of the form $L(d)$, where L is an auxiliary unary predicate (that without loss of generality we can consider to be in $\mathcal{R}$), and $d \in I$.*

Notice that every database belonging to a backward chase is a *forest* of trees, each rooted at a node at level 0. When we evaluate a query over a database C of some backward-chase, we consider C as the set of its tuples (removing duplicates).

Now we come to the main result in this section, stating that the backward-chase of a query Q is a representative for all databases that return an answer tuple when Q is evaluated over them (considering the access limitations). Before presenting the main result, we need an auxiliary lemma that shows that, once we consider the chase of the left-hand side query, the evaluation of the right-hand side one over a database in the above chase can ignore the access limitations,

as long as the set of inital constants includes the constants appearing in both queries.

Lemma 1. *Consider two conjunctive queries Q_1, Q_2 over a schema $\mathcal{S} = \langle \mathcal{R}, \Pi \rangle$ with access limitations, a set of constants I such that $I \supseteq \text{const}(Q_1) \cup \text{const}(Q_2)$, and a database $C \in \text{bchase}(Q_1, \mathcal{S}, I)$; we have that $\text{ans}(Q_2, \mathcal{S}, C, I) = Q_2(C)$.*

Proof (sketch). Since $I \supseteq \text{const}(Q_1) \cup \text{const}(Q_2)$, all tuples in C can be extracted in the recursive evaluation of Q_2; therefore, such an evaluation produces the same results as the ordinary conjunctive query evaluation. □

Theorem 1. *Consider two conjunctive queries Q_1, Q_2 over a schema $\mathcal{S} = \langle \mathcal{R}, \Pi \rangle$ with access limitations, and a set of constants I such that $I \supseteq \text{const}(Q_1) \cup \text{const}(Q_2)$; then, we have $Q_1 \subseteq_{\Pi,I} Q_2$ if and only if for every database $C \in \text{bchase}(Q_1, S, I)$ there exists a homomorphism that sends $\text{body}(Q_2)$ to facts of C, and $\text{head}(Q_2)$ to $\text{head}(C)$.*

Proof (sketch).

"⇐" Consider a generic database B such that there exists a tuple t in $\text{ans}(Q_1, \mathcal{S}, B, I)$, with $t \in (\Delta \cup \Delta_F)^n$ (let n be the arity of Q_1); it is not difficult to prove that there exists a database $C \in \text{bchase}(Q_1, S, I)$ such that there exists a homomorphism λ such that $\lambda(C) \subseteq B$; this holds because the steps of the extraction of the tuples that lead to the retrieval of t are mimicked by the steps in the construction of C (the former go in the reverse order with respect to the latter). Now, by hypothesis, there exists a homomorphism μ that sends $\text{body}(Q_2)$ to facts of C, and $\text{head}(Q_2)$ to $\text{head}(C)$; by considering the composition of homomorphisms $\eta = \mu \circ \lambda$, we have that $\eta(\text{body}(Q_2)) \subseteq B$ and $\eta(headQ_2) = t$. This proves that $Q_1 \subseteq_{\Pi,I} Q_2$.

"⇒" This is straightforward, from the definition of containment under access limitations. □

With the notion of chase at hand, we are able to face the problem of decidability of checking containment of conjunctive queries under access limitations. Notice that the previous theorem does not provide any direct strategy for checking containment; indeed, given a conjunctive query over a schema $\mathcal{S}$, and a set I of initial constants, the number of databases in $\text{bchase}(Q_1, \mathcal{S}, I)$ may be infinite. Also, notice that the databases in $\text{bchase}(Q_1, \mathcal{S}, I)$ may be of infinite size, though, for obvious reasons, henceforth we shall only consider finite databases.

4 Decidability and Complexity

In this section we prove that the problem of checking containment of two conjunctive queries under access limitations is decidable. This will be proved by showing that, while checking $Q_1 \subseteq_{\Pi,I} Q_2$, when we look for a homomorphism that sends $\text{body}(Q_2)$ to facts in some database $C \in \text{bchase}(Q_1, S, I)$ (and $\text{head}(Q_2)$ to $\text{head}(C)$), it is sufficient to consider databases in $\text{bchase}(Q_1, S, I)$ that have a number of levels that does not exceed a certain limit, depending on the schema and the queries.

Henceforth, we shall denote with $\text{subtree}(c)$ the subtree having c as root, and containing *all* descendants of c. We need some preliminary definitions.

Definition 3. *Consider a graph C in some backward-chase for a query over a schema with access limitations. Two subtrees $C_1 = \text{subtree}(c_1)$ and $C_2 = \text{subtree}(c_1)$ of C, where c_1 and c_2 are nodes of C, are said to be k-isomorphic if the trees rooted at c_1 and c_2 and having only levels up to k (where k is not more than the maximum level of C_2) are identical modulo the renaming of the fresh constants.*

We now provide a theorem that shows the decidability of the problem of checking containment (or better, non-containment). The theorem shows that, in order to find a counterexample showing that $Q_1 \not\sqsubseteq_{\Pi,I} Q_2$, we need to consider only databases of the backward-chase of Q_1 that have a number of levels that does not exceed a certain limit.

Theorem 2. *Consider two conjunctive queries Q_1, Q_2 over a schema $S = \langle \mathcal{R}, \Pi \rangle$ with access limitations, and a set of constants I such that $I \supseteq \text{const}(Q_1) \cup \text{const}(Q_2)$; if there exists a finite database $C \in \text{bchase}(Q, \mathcal{S}, I)$ such that $Q_1(C) \not\sqsubseteq_{\Pi,I} Q_2(C)$, then there exists another finite database $C' \in \text{bchase}(Q, \mathcal{S}, I)$ such that $Q_1(C') \not\sqsubseteq_{\Pi,I} Q_2(C')$, and such that C' has maximum level $\delta = |Q_2|^2 + |\mathcal{R}|$.*

Proof (sketch). In order to guarantee that the databases that we consider in the chase are representatives of every counterexample, we show that we can obtain a counterexample $\bar{C}$ with number of levels less or equal than δ from a counterexample C of arbitrary depth (number of levels). It is possible to show that $|Q_2|$ levels in $\bar{C}$ are sufficient to ensure that, if Q_2 maps onto C, it maps also onto $\bar{C}$; in fact, the connected parts of Q_2 map onto at most $|Q_2|$ subtrees of depth at most $|Q_2|$, and the fact that we check databases in the backward-chase with at least $|Q_2|^2$ levels ensures that we find some $\bar{C}$ that has subtrees that are $|Q_2|$-isomorphic to those on which the connected parts of Q_2 map. The remaining $|\mathcal{R}|$ levels are sufficient to "close off" the database with facts of the form $L(d), d \in I$, and make it become a member of the backward-chase. □

Finally, we characterise the computational complexity of our query containment problem, by providing an upper bound for it.

Theorem 3. *The complexity of checking containment of conjunctive queries under access limitations is in co-*NEXPTIME.

Proof (sketch). The proof is done by showing a nondeterministic algorithm that guesses a database $C \in \text{bchase}(Q, \mathcal{S}, I)$ with maximum number of levels $\delta = |Q_2| + |\mathcal{R}|$; in the worst case the guessed database has $O(W^\delta)$ nodes, each of which can be chosen in $|\mathcal{R}| \cdot W$ ways. The database C can therefore be guessed by a nondeterministic algorithm in exponential time; after that, checking whether $Q_2(C)$ yields $\text{head}(\text{bchase}(Q, \mathcal{S}, I))$ can be done in nondeterministic polynomial time. This proves that the non-containment is in NEXPTIME, from which the thesis follows. □

5 Conclusions

We have addressed the problem of containment of conjunctive queries in the case where access limitations are present on the relational schema. This prob-

lem is highly relevant in query optimisation. We have shown that, since in the presence of access limitations the evaluation of a query, which is in general inherently recursive, needs to be encoded in a Datalog program, the problem of containment amounts to checking containment between two recursive Datalog programs. Though the problem of containment of recursive Datalog program is in general undecidable, we have shown that in our case containment checking is indeed decidable, and we have provided an upper bound to the complexity of the problem by exhibiting a nondeterministic algorithm that solves it.

We plan to extend the results presented in this paper by finding a lower complexity bound for the problem of query containment. Also, we intend to implement the containment algorithm in order to test it on real cases, and to optimise the algorithm to achieve efficiency in real-world cases.

Acknowledgments. This work has been funded by the EU STREP FET project TONES (Thinking ONtologiES), FP6-7603.

References

1. Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison Wesley Publ. Co., 1995.
2. Andrea Calì and Diego Calvanese. 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 Publisher, 2002.
3. Ashok K. Chandra and Philip M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *Proc. of STOC'77*, pages 77–90, 1977.
4. Alin Deutsch, Bertram Ludäscher, and Alan Nash. Rewriting queries using views with access patterns under integrity constraints. In *Proc. of ICDT 2005*, pages 352–367, 2005.
5. Oliver M. Duschka and Alon Y. Levy. Recursive plans for information gathering. In *Proc. of IJCAI'97*, pages 778–784, 1997.
6. Daniela Florescu, Alon Levy, and Alberto Mendelzon. Database techniques for the World-Wide Web: A survey. *SIGMOD Record*, 27(3):59–74, 1998.
7. Daniela Florescu, Alon Y. Levy, Ioana Manolescu, and Dan Suciu. Query optimization in the presence of limited access patterns. In *Proc. of ACM SIGMOD*, pages 311–322, 1999.
8. Richard Hull. Managing semantic heterogeneity in databases: A theoretical perspective. In *Proc. of PODS'97*, pages 51–61, 1997.
9. David S. Johnson and Anthony C. Klug. Testing containment of conjunctive queries under functional and inclusion dependencies. *J. of Computer and System Sciences*, 28(1):167–189, 1984.
10. Chen Li and Edward Chang. Query planning with limited source capabilities. In *Proc. of ICDE 2000*, pages 401–412, 2000.
11. Chen Li and Edward Chang. On answering queries in the presence of limited access patterns. In *Proc. of ICDT 2001*, pages 219–233, 2001.
12. Alan Nash and Bertram Ludäscher. Processing first-order queries under limited access patterns. In *Proc. of PODS 2004*, pages 307–318, 2004.
13. Anand Rajaraman, Yehoshua Sagiv, and Jeffrey D. Ullman. Answering queries using templates with binding patterns. In *Proc. of PODS'95*, 1995.
14. Jeffrey D. Ullman. Information integration using logical views. In *Proc. of ICDT'97*, volume 1186 of *LNCS*, pages 19–40. Springer, 1997.