Conjunctive Query Containment under Access Limitations

Andrea Calì[1,3] and Davide Martinenghi[2]

[1] Oxford-Man Institute of Quantitative Finance
University of Oxford, UK
[2] Dip. di Elettronica e Informazione
Politecnico di Milano, Italy
[3] Computing Laboratory
University of Oxford, UK
andrea.cali@comlab.ox.ac.uk, martinen@elet.polimi.it

Abstract. Access limitations may occur when querying data sources over the web or heterogeneous data sources presented as relational tables: this happens, for instance, in Data Exchange and Integration, Data Warehousing, and Web Information Systems. Access limitations force certain attributes to be selected in order to access the tables. It is known that evaluating a conjunctive query under such access restrictions amounts to evaluating a possibly recursive Datalog program. We address the problem of checking containment of conjunctive queries under access limitations, which is highly relevant in query optimization. Checking containment in such a setting would amount to checking containment of recursive Datalog programs of a certain class, while, for general Datalog programs, this problem is undecidable. We propose a decision procedure for query containment based on the novel notion of crayfish-chase, showing that containment can be decided in co-NEXPTIME, which improves upon the known bound of 2EXPTIME. Moreover, by means of a direct proof, our technique provides a new insight into the structure of the problem.

1 Introduction

In Data Exchange and Integration [11,14,24], Data Warehousing, and Web Information Systems, querying heterogeneous data sources, possibly on the web, is a crucial issue. 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 significantly complicates query processing; in particular, as shown in [23,17,19], it requires the evaluation of a recursive query plan, which can be suitably expressed in Datalog.

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 have retrieved all obtainable tuples that contribute to the answer. $\square$

Since accessing sources may be costly, especially on the web, an important issue is how to optimize query evaluation. Query containment [15,6] is a well-recognized problem in query evaluation and optimization, in particular in Data Integration and Exchange; containment between two queries q_1, q_2 holds if the result of q_1 is always a subset of the result of q_2, independently of the database on which the queries are evaluated.

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 conjunctive query into another under access limitations, namely the *crayfish-chase*, that is a set of databases that are representative of all databases that provide an answer to a query. The crayfish-chase is in general an infinite set.
3. We give a direct proof of the decidability of containment in this setting, by showing that, in order to check containment, it is sufficient to consider databases in the crayfish-chase whose size does not exceed a certain limit.
4. We provide an upper bound to the complexity of conjunctive query containment, showing that it can be decided in co-NEXPTIME, which improves the known bound of 2EXPTIME.

Finally, besides achieving a better worst-case complexity upper bound, the new technique provides an insight into the query containment problem, that paves

[1] In the following, this will be represented by the notion of *abstract domain*.

the way to the investigation of the containment problem under limitations for more expressive classes of queries, and under database dependencies.

2 Preliminaries

In this section we present the formal framework in which we address the problem of query containment.

We consider relations as sets of facts whose arguments are values belonging to given domains. 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, referring to Example 1, distinguishes, e.g., strings representing artist names from strings representing song titles. 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, each with an associated *arity*; *(ii)* every attribute of a relational predicate $r \in \mathcal{R}$ has exactly one *abstract domain*; *(iii)* 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 in a query, all input attributes must be selected. 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 the form $r^{\Lambda_r}(A_1, \ldots, A_n)$, where r is the relation name, n is the arity of r, Λ_r its access limitations, and each A_i is an abstract domain. A *relation* over such a signature is a set of facts of the form $r(c_1, \ldots, c_n)$ such that each c_i is a value belonging to abstract domain A_i. A (*database*) *instance* of a schema $\mathcal{S}$ is a union of relations, one over each signature in $\mathcal{S}$, i.e., it is a set of facts. In the following, we assume two fixed domains: a non-empty set Δ of constants and, for technical reasons, an infinite domain Δ_F of *fresh* constants. We call *concrete* those databases whose values belong to Δ and *virtual* those databases whose values belong to Δ_F; we also assume that constants in Δ_F cannot appear in queries. We sometimes indicate a sequence of terms (i.e., variables or constants) $t_1, \ldots, t_n$ as $\boldsymbol{t}$, its length n as $|\boldsymbol{t}|$, and similarly a tuple $\langle t_1, \ldots, t_n \rangle$ as $\langle \boldsymbol{t} \rangle$, and its length n as $|\langle \boldsymbol{t} \rangle|$. A *conjunctive query* (CQ) q of arity n over a schema $\mathcal{S}$ is written in the form

$$q(\boldsymbol{X}) \leftarrow \mathit{conj}(\boldsymbol{X}, \boldsymbol{Y})$$

where $|\boldsymbol{X}| = n$, $q(\boldsymbol{X})$ is called the *head* of q, $\mathit{conj}(\boldsymbol{X}, \boldsymbol{Y})$ is called the *body* of q and is a conjunction of atoms involving the variables in $\boldsymbol{X}$ and $\boldsymbol{Y}$ and possibly some constants, and the predicate symbols of the atoms are in $\mathcal{S}$; $\langle \boldsymbol{X} \rangle$ is denoted as $\mathrm{head}(q)$, the set of atoms in the body is denoted as $\mathrm{body}(q)$, and $|q|$ denotes $|\mathrm{body}(q)|$. The set of constants appearing in q is denoted $\mathrm{const}(q)$, the set of variables $\mathrm{var}(q)$. A set of atoms $\mathcal{N}$ is *connected* if the non-directed graph $(\mathcal{N}, \mathcal{A})$ is connected, where $\mathcal{N}$ is the set of nodes, and $\mathcal{A}$ is the set containing exactly

$$\begin{array}{lrl}
\rho_1 : & q(A) & \leftarrow \hat{r}_2(A, \mathit{italian}, \mathit{modena}) \\
\rho_2 : & \hat{r}_1(T, C, A) & \leftarrow r_1(T, C, A), \mathit{dom}_C(C) \\
\rho_3 : & \hat{r}_2(A, N, C) & \leftarrow r_2(A, N, C), \mathit{dom}_A(A) \\
\rho_4 : & \mathit{dom}_T(T) & \leftarrow \hat{r}_1(T, C, A) \\
\rho_5 : & \mathit{dom}_C(C) & \leftarrow \hat{r}_1(T, C, A) \\
\rho_6 : & \mathit{dom}_A(A) & \leftarrow \hat{r}_1(T, C, A) \\
\rho_7 : & \mathit{dom}_A(A) & \leftarrow \hat{r}_2(A, N, C) \\
\rho_8 : & \mathit{dom}_N(N) & \leftarrow \hat{r}_2(A, N, C) \\
\rho_9 : & \mathit{dom}_C(C) & \leftarrow \hat{r}_2(A, N, C) \\
\rho_{10} : & \mathit{dom}_N(\mathit{italian}) & \\
\rho_{11} : & \mathit{dom}_C(\mathit{modena}) &
\end{array}$$

Fig. 1. Datalog program for Example 2

all arcs between any two atoms in $\mathcal{N}$ that share a variable or a constant. A CQ q is *connected* if body(q) is. Every maximal subset of body(q) that is connected is called a *connected part* of q.

In the following we shall extensively use the notion of *mapping* from terms to terms, and typically we will map variables to terms, or fresh constants in Δ_F to constants in Δ. 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 a tuple $\theta = \langle t_1, \ldots, t_n \rangle$ to another tuple indicated $\mu(\theta) = \langle \mu(t_1), \ldots, \mu(t_n) \rangle$, from a fact $f = r(t_1, \ldots, t_n)$ to another fact indicated $\mu(f) = r(\mu(t_1), \ldots, \mu(t_n))$, and from a database $D = \{f_1, \ldots, f_m\}$ to another database indicated $\mu(D) = \{\mu(f_1), \ldots, \mu(f_m)\}$. A *substitution mapping* (or, simply, substitution) is a mapping from terms to terms that sends every constant into itself[2]; a substitution is *grounding* for a set of variables $\mathcal{V}$ if it sends each variable in $\mathcal{V}$ into a constant.

Given a database D, the *answer* $q(D)$ to a CQ q on D is the set of tuples $\langle \boldsymbol{c} \rangle$ of constants, with $|\boldsymbol{c}| = |\text{head}(q)|$, such that there is a substitution that sends body(q) to facts of D and head(q) to $\langle \boldsymbol{c} \rangle$.

In the presence of access limitations on the sources, queries cannot be evaluated as in the traditional case, as will be shown in Example 2. Given a query over the data sources, an algorithm exists [17] 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, each with an associated abstract domain, as described in Example 1. The Datalog program, whose construction is sketched in Example 2, encodes 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 access limitations. With the new facts obtained (if any), we obtain new values with which we can repeat the process and access the relations again, until we have no way of making new accesses. The program extracts all facts obtainable while respecting the access limitations, but there may be facts in the sources that cannot be retrieved.

[2] Substitutions are sometimes written in postfix notation. Here we use infix notation.

Given a query q posed over a schema $\mathcal{S} = \langle \mathcal{R}, \Lambda \rangle$, a set of constants $I \subseteq \Delta$, and a database D for $\mathcal{S}$, we denote the answers obtained through the recursive evaluation described above as $\mathrm{ans}(q, \mathcal{S}, D, I)$. The tuples or facts extracted from D starting from I and respecting Λ are said to be Λ, I*-obtainable.* Notice that in general $\mathrm{ans}(q, \mathcal{S}, D, I) \subseteq q(D)$.

Example 2. Consider again Example 1, with $r_1^{oio}(T, C, A)$ and $r_2^{ioo}(A, N, C)$. The Datalog program generated by the algorithm of [17] for the query $q(A) \leftarrow r_2(A, \mathit{italian}, \mathit{modena})$ is shown in Figure 1. The query is rewritten over the caches (rule ρ_1) 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 $\rho_4 - \rho_9$ are the domain rules. Finally, ρ_{10}, ρ_{11} are facts assigning the right abstract domain to the initial constants. $\square$

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 facts recursively as described above, we will define the containment between two CQs 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 CQs* q_1, q_2 *over a schema* $\mathcal{S} = \langle \mathcal{R}, \Lambda \rangle$*, and a set* I *such that* $\mathrm{const}(q_1) \cup \mathrm{const}(q_2) \subseteq I \subseteq \Delta$*; we say that* q_1 *is* contained *in* q_2 *under* Λ *with respect to* I*, denoted* $q_1 \subseteq_{\Lambda,I} q_2$*, if, for every database* D *for* $\mathcal{R}$*, we have* $\mathrm{ans}(q_1, \mathcal{S}, D, I) \subseteq \mathrm{ans}(q_2, \mathcal{S}, D, I)$.

From the previous definition, it follows that checking containment would amount to checking 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

We start by observing that query containment under access limitations is essentially different from ordinary query containment, because, although the latter entails the former, the converse does not hold, as shown in Proposition 1.

Proposition 1. *Let* q_1 *and* q_2 *be two CQs over a schema* $\langle \mathcal{R}, \Lambda \rangle$*, and* I *a set of constants such that* $I \supseteq \mathrm{const}(q_1) \cup \mathrm{const}(q_2)$*. If* $q_1 \subseteq q_2$ *then* $q_1 \subseteq_{\Lambda,I} q_2$*, but the converse does not hold.*

Proof. Assume, w.l.o.g., that q_1 and q_2 have no variables in common. For each obtainable answer tuple $\langle \boldsymbol{t} \rangle$ to q_1, there is a corresponding instance of $\mathrm{body}(q_1)$ whose facts are Λ, I-obtainable, i.e., there is a grounding substitution μ_t for $\mathrm{var}(q_1)$ such that the facts in $\mu_t(\mathrm{body}(q_1))$ are Λ, I-obtainable and

$\mu_t(\text{head}(q_1)) = \langle \boldsymbol{t} \rangle$. Since containment is assumed, there exists a substitution λ such that $\lambda(\text{body}(q_2)) \subseteq \text{body}(q_1)$, and $\lambda(\text{head}(q_2)) = \text{head}(q_1)$. But then, for each answer tuple $\langle \boldsymbol{t} \rangle$ to q_1 there is also an instance of body(q_2) whose facts are Λ, I-obtainable that generates the same answer tuple $\langle \boldsymbol{t} \rangle$. To see this, it suffices to note that the facts in $\mu_t(\lambda(\text{body}(q_2)))$ are Λ, I-obtainable since those in $\mu_t(\text{body}(q_1))$ are, and that $\langle \boldsymbol{t} \rangle = \mu_t(\lambda(\text{head}(q_2)))$.

To see that the converse does not hold, consider a schema with two relations $r_1^{ii}(A, B)$ and $r_2^{oi}(B, C)$ and the queries $q_1(B) \leftarrow r_1(a, B)$ and $q_2(B) \leftarrow r_1(a, B), r_2(B, C)$. For every $I \supseteq \{a\}$ that does not contain any constant of abstract domain B, we have that $q_1 \subseteq_{\Lambda,I} q_2$. Indeed, when evaluating q_1, the only Λ, I-obtainable facts for r_1 are those whose second argument is some constant b that occurs as first argument in a fact for r_2, i.e., b is also an answer to q_2. However $q_1 \not\subseteq q_2$, since there is at least one database D such that $q_1(D) \not\subseteq q_2(D)$ (take, e.g., $D = \{r_1(a, b)\}$). ■

We now present the foundations of our novel technique to check containment of CQs under access limitations. Similarly to what is done for containment of CQs under inclusion and functional dependencies [15], in order to check the containment of a query q_1 into another query q_2, we characterize the set of all 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. With the chase at hand, we can evaluate q_2 over a finite set of databases in the chase of q_1 in order to check the existence of a counterexample to containment, i.e., a database D that provides an answer tuple to q_1 that is not in the answer to q_2 in D.

The chase of a CQ under access limitations is defined as follows. Each database of the chase starts from the *frozen* body of the query, i.e., the image of the body of the query according to some grounding substitution that sends variables to fresh constants. Then, according to the access limitations, we go back in the extraction process, adding facts that may lead to the extraction of the previous ones, and we continue to do that until all the facts we choose to add come from relations whose input arguments are filled in by initial values. Since we proceed somehow backwards, we call our chase *crayfish-chase*.

For convenience, we first need a preprocessing step (`constElim`) to eliminate constants in the query, as illustrated in Figure 2. The intuition is that a constant acts as a relation, called *artificial relation*, whose content is accessible and amounts only to the constant itself. Under this assumption, the constant-free query and the original one are equivalent, as specified in Proposition 2.

Proposition 2. *Let q be a CQ over a schema $\mathcal{S}$, I a set such that* $\text{const}(q) \subseteq I \subseteq \Delta$, *and* $(\mathcal{S}', q') = \texttt{constElim}(\mathcal{S}, q, I)$. *Let D be a database for $\mathcal{S}$ and let D' be as D plus one fact $\ell_c(c)$ for each artificial relation ℓ_c in $\mathcal{S}'$ with associated constant c. Then $q(D) = q'(D')$.*

Definition 2 (crayfish-chase). *Consider a CQ q over a schema $\mathcal{S} = \langle \mathcal{R}, \Lambda \rangle$ and a set I such that* $\text{const}(q) \subseteq I \subseteq \Delta$. *The* crayfish-chase *of q, denoted* cchase$(q, \mathcal{S}, I)$, *is the set of all finite databases that can be constructed as follows.*

```
INPUT: a schema $\mathcal{S}$, a CQ $q$ over $\mathcal{S}$, and a set $I$ such that const$(q) \subseteq I \subseteq \Delta$
OUTPUT: a schema $\mathcal{S}'$, a CQ $q'$ over $\mathcal{S}'$

  – Let $\mathcal{S}' := \mathcal{S}$, $q' := q$
  – For each constant $a \in I$ with abstract domain $A$
      • Add signature $\ell_a^o(A)$ for the new artificial relation $\ell_a$ to $\mathcal{S}'$
      • Replace all occurrences of $a$ in $q'$, if any, with a fresh new variable $X_a$
      • If $a$ occurs in $q$, add the conjunct $\ell_a(X_a)$ to the body of $q'$
  – Return $(\mathcal{S}', q')$
```

Fig. 2. Algorithm `constElim` for elimination of constants

Each database $D \in$ cchase$(q, \mathcal{S}, I)$ *is represented as a* forest*, the nodes of which are facts of D; each node n has a* level*, denoted* level(n)*, that is a non-negative integer. The set of nodes at level h in a database D will be called* level h *of D. The* depth *of D, denoted* depth(D)*, is the maximum level of nodes in D.*

1. *Let $(\mathcal{S}', q') =$ `constElim`$(\mathcal{S}, q, I)$.*
2. *We fix a single injective substitution mapping μ for all databases of* cchase$(q, \mathcal{S}, I)$ *that sends each variable in* var(q') *into a fresh new constant in Δ_F, and thus* body(q') *into a set of facts; such facts will be level* 0 *of D. Each tree of the forest will be rooted at a node of level* 0*.*
3. *We call* μ(head(q)) *the* head *of the* crayfish-chase*, denoted* head(cchase$(q, \mathcal{S}, I)$)*.*
4. *For each fact $f = r(c_1, \ldots, c_n)$ at level k, and for each input attribute of r, say the i-th, there is exactly one fact $f' = r'(c'_1, \ldots, c'_m)$ such that $c_i = c'_j$, for some position j corresponding to an output attribute of r' having the same abstract domain as c_i's. If f is at level k, then f' must be at level $k+1$, and an arc (f, f') is in D. All other constants in f' must be fresh new constants in Δ_F, not appearing elsewhere in any of the levels less than or equal to $k+1$.*
5. *Each leaf of D is a fact of a (possibly artificial) relation without input arguments.*

The databases in the crayfish-chase of a query q, as stated in Lemma 1, are representative of all concrete databases that return an answer to q while respecting the access limitations, i.e., they are sufficient to retrieve all obtainable answers to q and yet do not add any other answer.

Definition 3. *A mapping λ from Δ_F to Δ is said to be* compatible *with a virtual database D (in short, D-*compatible*), if λ sends each constant ζ occurring in D in a fact $\ell_c(\zeta)$ of an artificial relation ℓ_c into the corresponding constant c.*

Lemma 1. *Let $\mathcal{S}$ be a schema, q a query over $\mathcal{S}$, and I a set of constants such that* const$(q) \subseteq I \subseteq \Delta$*. (a) For every concrete database D such that there exists a tuple $t \in$* ans$(q, \mathcal{S}, D, I)$*, there exists a database $D' \in$* cchase$(q, \mathcal{S}, I)$ *and a D'-compatible mapping λ from Δ_F to Δ such that $\lambda(D') \subseteq D$ and $t \in$* ans$(q, \mathcal{S}, \lambda(D'), I)$*.*

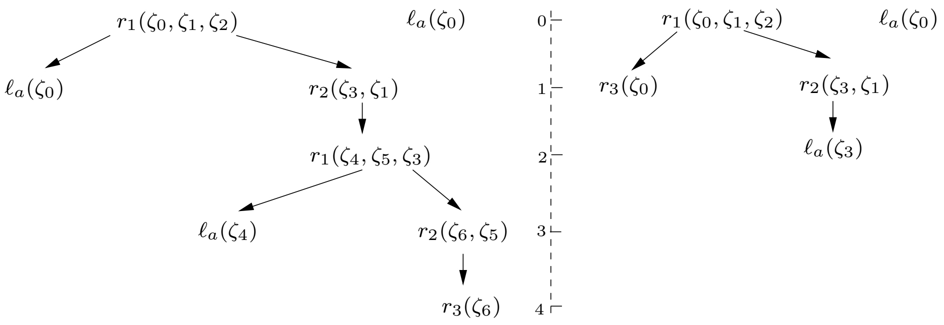

Fig. 3. Database forests in the crayfish-chase $\text{cchase}(q,\mathcal{S},I)$ of Example 3

(b) Conversely, for every database $D' \in \text{cchase}(q,\mathcal{S},I)$ *and for every* D'*-compatible mapping* λ *from* Δ_F *to* Δ *such that* $\lambda(D')$ *is concrete, if there exists a tuple* $t' \in \text{ans}(q,\mathcal{S},\lambda(D'),I)$*, then there is a database* D *such that* $t' \in \text{ans}(q,\mathcal{S},D,I)$.

Proof (sketch). (a) If t can be obtained in D, this means that there is a grounding substitution μ for $\text{var}(q)$ such that $\mu(\text{head}(q)) = t$ and all the facts in $\mu(\text{body}(q))$ are obtainable and in D. Therefore, each fact in $\mu(\text{body}(q))$ either has constants from I in all its input positions, or, inductively, for each constant c not from I in an input position there is some obtainable fact in D with c in an output position. But this models exactly a tree of a database of $\text{cchase}(q,\mathcal{S},I)$, with the only difference that here there may be more than one constant not from I in common in two facts, which can be captured by a mapping λ.
(b) This holds by construction: by applying λ to D' one obtains a database D with t' in the answer to q. ■

Example 3. Assume we have a schema $\mathcal{S}$ with the following relations: $r_1^{iio}(A,B,A), r_2^{io}(A,B), r_3^{o}(A)$. Consider the query $q(X_2) \leftarrow r_1(a,X_1,X_2)$ and the set $I = \{a\}$. First of all, we transform the query by eliminating the constants: we get $q'(X_2) \leftarrow r_1(X_0,X_1,X_2), \ell_a(X_0)$, where ℓ_a is an auxiliary predicate with signature $\ell_a^o(A)$; no other auxiliary predicates are introduced. After freezing the query, we obtain two facts in the frozen body: $r_1(\zeta_0,\zeta_1,\zeta_2)$ and $\ell_a(\zeta_0)$; the head of $\text{cchase}(q,\mathcal{S},I)$ is $\langle\zeta_2\rangle$. Every database in $\text{cchase}(q,\mathcal{S},I)$ is a forest of exactly two trees rooted at $r_1(\zeta_0,\zeta_1,\zeta_2)$ and $\ell_a(\zeta_0)$ respectively, since these two facts constitute the level 0 of every database in the chase; every tree rooted in $\ell_a(\zeta_0)$ will consist of only one node; two possible databases (forests) are depicted in Figure 3, separated by a dashed vertical line on which we have indicated the depth of the different levels. □

We now show that, when considering $q_1 \subseteq_{\Lambda,I} q_2$, once we have the crayfish-chase of q_1, the evaluation of q_2 over a database in the above chase can ignore the access limitations, as long as the same set of initial constants is used.

Lemma 2. *Consider two CQs q_1, q_2 over a schema $\mathcal{S} = \langle \mathcal{R}, \Lambda \rangle$, a set $I \subseteq \Delta$, and a database $D \in \text{cchase}(q_1, \mathcal{S}, I)$; then $\text{ans}(q_2, \mathcal{S}, D, I) = q_2(D)$.*

Proof. Straightforward, since all facts in D are, by construction, Λ, I-obtainable. ■

Lemma 3. *Consider two CQs q_1, q_2 over a schema $\mathcal{S} = \langle \mathcal{R}, \Lambda \rangle$, and a set I such that $\text{const}(q_1) \cup \text{const}(q_2) \subseteq I \subseteq \Delta$. Then $q_1 \subseteq_{\Lambda,I} q_2$ if and only if, for every database $D \in \text{cchase}(q_1, \mathcal{S}, I)$, $\text{head}(\text{cchase}(q_1, \mathcal{S}, I)) \in \text{ans}(q_2, D, \mathcal{S}, I)$.*

Proof (sketch). "$\Leftarrow$" Consider a generic concrete database B such that there exists a tuple t in $\text{ans}(q_1, \mathcal{S}, B, I)$; by Lemma 1, there exist a database $D \in \text{cchase}(q_1, \mathcal{S}, I)$ and a mapping λ from Δ_F to Δ compatible with D such that $\lambda(D) \subseteq B$ and $t \in \text{ans}(q_1, \mathcal{S}, \lambda(D), I)$. Now, by hypothesis, there exists a mapping μ that sends $\text{body}(q_2)$ to facts of D, and $\text{head}(q_2)$ to $\text{head}(\text{cchase}(q_1, \mathcal{S}, I))$; we have that $\lambda(\mu(\text{body}(q_2))) \subseteq B$ and $\lambda(\mu(\text{head}(q_2))) = t$. This proves that $t \in \text{ans}(q_2, \mathcal{S}, B, I)$ and thus that $q_1 \subseteq_{\Lambda,I} q_2$, since B was generic.

"$\Rightarrow$" Trivial, from the definition of containment under access limitations. ■

Now we come to the main result in this section, that follows trivially as a corollary of Lemma 2 and Lemma 3, stating that examining the databases of a crayfish-chase provides us with a necessary and sufficient condition to test containment of CQs under access limitations.

Theorem 1. *Consider two CQs q_1, q_2 over a schema $\mathcal{S} = \langle \mathcal{R}, \Lambda \rangle$, and a set I such that $\text{const}(q_1) \cup \text{const}(q_2) \subseteq I \subseteq \Delta$. Then, $q_1 \subseteq_{\Lambda,I} q_2$ if and only if, for every database $D \in \text{cchase}(q_1, \mathcal{S}, I)$, $\text{head}(\text{cchase}(q_1, \mathcal{S}, I)) \in q_2(D)$.*

Notice that the previous theorem does not provide any direct strategy for checking containment; indeed, given a CQ over a schema $\mathcal{S}$, and a set I of initial constants, the number of databases in $\text{cchase}(q_1, \mathcal{S}, I)$ may be infinite. Also, notice that, although all databases in $\text{cchase}(q_1, \mathcal{S}, I)$ are of finite size, there is in general no fixed bound on such size.

4 Decidability and Complexity

In this section we give a direct proof of decidability of checking containment between CQs under access limitations that exploits the notion of crayfish-chase. This will be done by showing that, while checking $q_1 \subseteq_{\Lambda,I} q_2$, when we look for a substitution that sends $\text{body}(q_2)$ to facts in some database $D \in \text{cchase}(q_1, \mathcal{S}, I)$ (and $\text{head}(q_2)$ to $\text{head}(\text{cchase}(q_1, \mathcal{S}, I))$), it is sufficient to consider databases in $\text{cchase}(q_1, \mathcal{S}, I)$ whose depth does not exceed a certain limit, depending on the schema and the queries. In particular, Lemma 5 states that in order to find a counterexample showing that $q_1 \not\subseteq_{\Lambda,I} q_2$, we need to consider only databases of the crayfish-chase of q_1 of limited depth. This allows us to provide an improved upper bound for the complexity of this problem, as shown in Theorem 2. This requires some preparatory lemmas and definitions.

We first show that two facts sharing a constant in a database of a crayfish-chase cannot be more than one level apart.

Lemma 4. *Consider a database D in a crayfish-chase. If two nodes n_1 and n_2 in D have a constant in common, then* $|\mathrm{level}(n_1) - \mathrm{level}(n_2)| \leq 1$.

Proof. If $n_1 = n_2$ the claim trivially holds. By construction of the crayfish-chase, the constants that appear at some level k of D cannot appear in any level greater than $k + 1$. In particular, all constants in output arguments are not propagated to the next level, while those in input arguments occur only in output fields in the next level, and therefore disappear after two levels. Therefore, either n_1 and n_2 are connected by an arc (and thus their levels are at a distance of 1), or each of them lies on a level less than 2 (not necessarily connected by an arc, since different nodes of level 0 may share constants). ∎

As a consequence of Lemma 4, a connected part of n atoms of a query cannot be mapped on more than n contiguous levels.

Corollary 1. *Consider a CQ q over a schema $\mathcal{S} = \langle \mathcal{R}, \Lambda \rangle$, a set I such that* $\mathrm{const}(q) \subseteq I \subseteq \Delta$*, and a database $D \in \mathrm{cchase}(q, \Lambda, I)$. Let $\mathcal{P}$ be any connected part of q', where $(\mathcal{S}', q') = \mathtt{constElim}(\mathcal{S}, q, I)$; if there exists a substitution μ sending variables to constants in Δ_F that sends the atoms in $\mathcal{P}$ into facts of D, then $max_{\{p_i,p_j\}\subseteq\mathcal{P}}(|\mathrm{level}(\mu(p_i)) - \mathrm{level}(\mu(p_j))|) \leq |\mathcal{P}|$, i.e., $\mu(\mathcal{P})$ lies onto at most $|\mathcal{P}|$ contiguous levels on D.*

Henceforth, we shall denote with subtree(c) the subtree (of a given tree) having node c as root, and containing *all* descendants of c; with k-subtree(c), k a positive integer, we denote the subtree rooted in c, and containing all descendants of c up to level $\mathrm{level}(c) + k - 1$. Lemma 5, below, shows that if a query does not map onto a database of a crayfish-chase, then there is a (possibly different) database of the chase which has limited depth and onto which the query still cannot be mapped. To construct this database, we trim redundant parts by using the notion of subtree replacement.

Definition 4 (Subtree replacement). *Let D be a virtual database of a crayfish-chase, and consider two nodes $n_1 = r(c_1, \ldots, c_k)$ and $n_2 = r(d_1, \ldots, d_k)$ in D, such that n_2 is a descendant of n_1. Let μ be a mapping from Δ_F to Δ_F that sends d_i into c_i for $1 \leq i \leq k$ and every other constant into itself. Then, a* replacement *of* subtree(n_1) *with* subtree(n_2) *in D is the result of replacing* subtree(n_1) *with* μ(subtree(n_2)).

Lemma 5. *Consider two CQs q_1, q_2 over a schema $\mathcal{S} = \langle \mathcal{R}, \Lambda \rangle$, and a set I such that* $\mathrm{const}(q_1) \cup \mathrm{const}(q_2) \subseteq I \subseteq \Delta$*; if there exists a database* $D \in \mathrm{cchase}(q_1, \mathcal{S}, I)$ *such that* $\mathrm{head}(\mathrm{cchase}(q_1, \mathcal{S}, I)) \notin q_2(D)$*, then there exists a database* $D' \in \mathrm{cchase}(q, \mathcal{S}, I)$ *such that* $\mathrm{head}(\mathrm{cchase}(q_1, \mathcal{S}, I)) \notin q_2(D')$*, and such that* $\mathrm{depth}(D') \leq 2 \cdot |\mathcal{R}| + |q_2| - 3$.

Proof.
Case *(1)*: q_2 is connected.

Subcase *(1a)*. There is only one relation r among those in q_2 such that, for every database $B \in \mathrm{cchase}(q_1, \mathcal{S}, I)$, if q_2 can be mapped onto facts of B, the

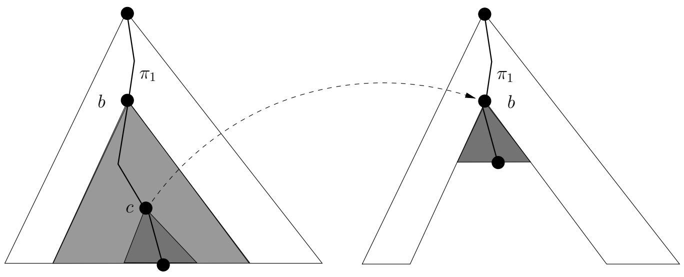

Fig. 4. Subtree replacement of subtree(b) with subtree(c) (Lemma 5)

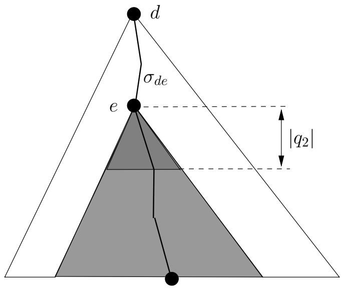

Fig. 5. Second phase of iterative subtree replacement (Lemma 5)

mapped fact f with smallest level has relation r and no mapped fact with a different relation has level equal to level(f).

Since q_2 is connected, if it is mapped onto a database, by Corollary 1, it will be mapped onto facts whose level is between level(f) and level(f) + $|q_2| - 1$.

Take now any path π_1 from a node at level 0 to a leaf of D. For simplicity, we say that a relation r occurs in a path π (and that π contains r), if a fact of the form $r(\boldsymbol{\zeta})$ occurs in it π. Since q_2 cannot be mapped onto D by hypothesis, a fortiori q_2 cannot be mapped onto any of the $|q_2|$-subtrees rooted in any of the occurrences of r in π_1. Let b be the node of π_1 with the occurrence (if any) of r with the smallest level (call a its parent if b is not at level 0) and let c be the one with the greatest level. We apply the replacement of subtree(b) with subtree(c), as shown in Figure 4. In the obtained database, q_2 continues to be not mappable, since *(i)* facts have been removed from D, and *(ii)* only one potential "join" has been added (that between c and a in D), which is irrelevant to q_2, since a's relation is certainly different from r, and r was assumed to be the only possible predicate at the smallest level of facts of the image of q_2 (and r does not occur above b in π_1). This step is repeated for every path in which r occurs. After this, in all paths from a node at level 0 to a leaf of the obtained database, r occurs at most once and q_2 is still not mappable.

To complete the transformation, we apply the following steps as long as possible to every path π_2 from a node at level 0 and a leaf.

- If π_2 does not contain r, we a apply subtree replacement, in the same way as was done for r above, for any relation occurring more than once in π_2; again, non-mappability of q_2 onto the database is preserved. At the end of the process, each such path will have at most length $|\mathcal{R}| - 1$.
- If π_2 contains r, let d be the node at level 0, and e the node with r; the segment σ_{de} of π_2 from d to e contains r only in e. If there is another relation s occurring more than once in σ_{de}, we apply the replacement of the subtree rooted at the occurrence of s in σ_{de} closest to d with the one with the occurrence in σ_{de} closest to e, as shown in Figure 5; again, non-mappability is preserved; besides, after all such replacements, σ_{de} has length at most $|\mathcal{R}|$. We apply in the same way all possible replacements to remove multiple occurrences of a relation in all subtrees rooted in a node of subtree(e) lying at level level(e) + $|q_2| - 1$. Thus, the $|q_2|$-subtree(e) (shaded in Figure 5) on which q_2 cannot be mapped is kept and the distance between e and the leaves will eventually be at most $(|q_2| - 1) + (|\mathcal{R}| - 1) = |q_2| + |\mathcal{R}| - 2$.

In total, the obtained database D' has depth at most $|\mathcal{R}|+(|q_2|+|\mathcal{R}|-2)-1 = 2 \cdot |\mathcal{R}| + |q_2| - 3$ and q_2 cannot be mapped onto D'.

Subcase *(1b)*: There is more than one relation that can be on the smallest level of the mapped facts of q_2 in a crayfish-chase database; let call $\mathcal{F}$ the set of such relations. For each path in D from a node of level 0 to a leaf, consider the occurrence with smallest level among the relations in $\mathcal{F}$. For that relation, we apply the replacement of the first occurrence with the last occurrence on the path. Since it was the first occurrence of a relation in $\mathcal{F}$, the join added with the replacement will not introduce mappability of q_2. After all such replacements are applied, in each path from level 0 to a leaf, the relation, say r, of the occurrence, say a, with smallest level among the relations in $\mathcal{F}$ will occur only once. With this in mind, we proceed as in subcase 1a to eliminate all multiple occurrences of relations above a in the path, still preserving non-mappability, so that eventually level(a) will be at most $|\mathcal{R}| - 1$. Also, we safely apply all possible replacements to remove multiple occurrences of a relation in all subtrees rooted in a node of subtree(a) lying at level level(a) + $|q_2| - 1$, since in any such subtree r does not occur, and thus q_2 cannot be mapped onto (and we know that q_2 cannot be mapped onto the $|q_2|$-subtree(a), which is kept). In the end we still obtain a database D' with depth at most $2 \cdot |\mathcal{R}| + |q_2| - 3$ such that q_2 cannot be mapped onto D'.

Case *(2)*: q_2 is not connected. We proceed as in the case of a connected query, and apply the same argument on one of the connected parts of q_2. Clearly, if a connected part of q_2 cannot be mapped, q_2 cannot be mapped either. ■

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

Theorem 2. *Containment of conjunctive queries under access limitations is decidable in co-*NEXPTIME.

Proof (sketch). By virtue of Theorem 1 and Lemma 5, in order to check containment, only databases of a limited depth need to be checked. There are only finitely many such databases that are different modulo isomorphism.

Let $q_1 \subseteq_{\Lambda,I} q_2$ be the containment to be decided for a schema $\mathcal{S} = \langle \mathcal{R}, \Lambda \rangle$. We use a nondeterministic algorithm that guesses a database $D \in \text{cchase}(q_1, \mathcal{S}, I)$ with maximum number of levels $\delta = 2 \cdot |\mathcal{R}| + |q_2| - 3$ that is a witness of non-containment; by Lemma 5, we know that we do not need to consider databases of bigger depth for this purpose. The guessed database has at most $O(W^\delta)$ nodes, where W is the maximum arity of the relations in $\mathcal{R}$; notice that each node has at most W children, each of which can be chosen in at most $|\mathcal{R}| \cdot W$ different ways. The database D can therefore be guessed in exponential time (w.r.t. δ) by a nondeterministic algorithm. After that, checking, on the same nondeterministic branch, whether $q_2(D)$ yields $\text{head}(\text{cchase}(q, \mathcal{S}, I))$ can be done in polynomial time in the size of D, i.e., in exponential time w.r.t. δ. Therefore a witness for non-containment can be guessed in NEXPTIME, from which the thesis follows. ■

5 Related Work

The issue of processing queries under access limitations has been widely investigated in the literature [23,17,19,18,12,10]; in particular, [12] considers the optimization of non-recursive plans, [10] addresses the problem in the case of query answering using views, and [23] 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 [22,17]; in particular, [22] addresses the problem of query containment under access limitations.

The problem of checking containment of two CQs under access limitations was shown to be decidable in [22] in the setting of data integration systems using the local-as-view approach by reducing this problem to containment of a recursive Datalog program in a non-recursive one; the optimal complexity for this problem is 3EXPTIME [7]. In [18], the authors propose an encoding of CQs with access limitations into monadic Datalog programs; containment between monadic Datalog programs was shown to be decidable in 2EXPTIME in [8], which immediately provides a 2EXPTIME upper bound for containment of CQs under access limitations. The same upper bound is easily obtained by combining the results from Section 3 with the complexity of checking containment of a Datalog program in a CQ; such problem was shown to be decidable in 2EXPTIME (tight bound) in [7]. In this paper, we improve upon this upper bound by providing an algorithm that checks containment in co-NEXPTIME, as mentioned in the position paper [4] and informally presented in [3].

In [16], the author addresses the issue of *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. [25] addresses the problem of ordering subgoals for non-recursive Datalog queries in oder to make the query executable from left to right complying with the access limitations. In [2], a run-time optimization technique, that exploits the information about database dependencies that hold on the sources, is presented; [5]

uses the structure of the query to minimize the accesses needed to retrieve all obtainable answers to a query. [9] solves the (quite general) problem of query answering using views [13] under integrity constraints and under access limitations by reducing it to the same problem under integrity constraints only; various extensions to the query languages are provided. In [20], the authors analyze the complexity of determining the *feasibility* of a query, i.e., determining whether there exists an equivalent query that is executable as is, while respecting the access limitations. [21] studies the complexity of the feasibility problem for CQs, UCQs, $\mathrm{CQ}^{\neg}$s and $\mathrm{UCQ}^{\neg}$s.

6 Conclusions

We have addressed the problem of containment of CQs in the case where access limitations are present on the relational schema. This problem is highly relevant in query optimization. In the presence of access limitations, the evaluation of a query is in general inherently recursive and can be encoded in a Datalog program. The problem of containment would then amount to checking containment between two Datalog programs, which is undecidable. However, in this particular case, containment checking is indeed decidable, and we have provided an improved upper bound to the complexity of the problem by exhibiting a nondeterministic algorithm that solves it.

With our crayfish-chase technique we have provided a direct proof of decidability that we plan to use for further investigations. In particular, we intend to extend our results to more general classes of queries, and to extend the problem by introducing integrity constraints on the schema. The combination of our crayfish-chase with the well-known chase based on inclusion and functional dependencies seems a promising direction of research. We also plan to extend the results presented in this paper by finding a lower complexity bound for the problem of query containment.

Acknowledgments. A. Calì was supported by the EPSRC project "Schema Mappings and Automated Services for Data Integration and Exchange" (EP/E010865/1). D. Martinenghi acknowledges support from Italian PRIN project "New technologies and tools for the integration of Web search services".

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison Wesley Publ. Co., Reading (1995)
2. Calì, A., Calvanese, D.: 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), pp. 285–301. Kluwer Academic Publisher, Dordrecht (2002)
3. Calì, A., Calvanese, D.: Containment of conjunctive queries under access limitations (extended abstract). In: Proc. of SEBD 2006, pp. 131–138 (2006)

4. Calì, A., Calvanese, D.: Optimising query answering in the presence of access limitations (position paper). In: Proc. of the 2nd Workshop on Logical Aspects and Applications of Integrity Constraints (LAAIC 2006). IEEE Computer Society, Los Alamitos (2006)
5. Calì, A., Calvanese, D., Martinenghi, D.: Optimization of query plans in the presence of access limitations. In: Arenas, M., Hidders, J. (eds.) EROW 2007 (ICDT workshop), Informal proceedings, pp. 33–47 (2007)
6. Chandra, A.K., Merlin, P.M.: Optimal implementation of conjunctive queries in relational data bases. In: Proc. of STOC 1977, pp. 77–90 (1977)
7. Chaudhuri, S., Vardi, M.Y.: On the equivalence of recursive and nonrecursive datalog programs. J. of Computer and System Sciences 54(1), 61–78 (1997)
8. Cosmadakis, S.S., Gaifman, H., Kanellakis, P.C., Vardi, M.Y.: Decidable optimization problems for database logic programs. In: Proc. of STOC 1988, pp. 477–490 (1988)
9. Deutsch, A., Ludäscher, B., Nash, A.: Rewriting queries using views with access patterns under integrity constraints. In: Proc. of ICDT 2005, pp. 352–367 (2005)
10. Duschka, O.M., Levy, A.Y.: Recursive plans for information gathering. In: Proc. of IJCAI 1997, pp. 778–784 (1997)
11. Fagin, R., Kolaitis, P.G., Miller, R.J., Popa, L.: Data exchange: Semantics and query answering. Theor. Comp. Sci. 336(1), 89–124 (2005)
12. Florescu, D., Levy, A.Y., Manolescu, I., Suciu, D.: Query optimization in the presence of limited access patterns. In: Proc. of ACM SIGMOD, pp. 311–322 (1999)
13. Halevy, A.Y.: Answering queries using views: A survey. VLDB Journal 10(4), 270–294 (2001)
14. Hull, R.: Managing semantic heterogeneity in databases: A theoretical perspective. In: Proc. of PODS 1997, pp. 51–61 (1997)
15. Johnson, D.S., Klug, A.C.: Testing containment of conjunctive queries under functional and inclusion dependencies. J. of Computer and System Sciences 28(1), 167–189 (1984)
16. Li, C.: Computing complete answers to queries in the presence of limited access patterns. VLDB Journal 12(3), 211–227 (2003)
17. Li, C., Chang, E.: Query planning with limited source capabilities. In: Proc. of ICDE 2000, pp. 401–412 (2000)
18. Li, C., Chang, E.: Answering queries with useful bindings. ACM Trans. on Database Systems 26(3), 313–343 (2001)
19. Li, C., Chang, E.: On answering queries in the presence of limited access patterns. In: Van den Bussche, J., Vianu, V. (eds.) ICDT 2001. LNCS, vol. 1973, pp. 219–233. Springer, Heidelberg (2000)
20. Ludäscher, B., Nash, A.: Processing first-order queries under limited access patterns. In: Proc. of PODS 2004, pp. 307–318 (2004)
21. Ludäscher, B., Nash, A.: Processing union of conjunctive queries with negation under limited access patterns. In: Lindner, W., Mesiti, M., Türker, C., Tzitzikas, Y., Vakali, A.I. (eds.) EDBT 2004. LNCS, vol. 3268, pp. 422–440. Springer, Heidelberg (2004)
22. Millstein, T.D., Levy, A.Y., Friedman, M.: Query containment for data integration systems. In: Proc. of PODS 2000, pp. 67–75 (2000)
23. Rajaraman, A., Sagiv, Y., Ullman, J.D.: Answering queries using templates with binding patterns. In: Proc. of PODS 1995 (1995)
24. Ullman, J.D.: Information integration using logical views. In: Afrati, F.N., Kolaitis, P.G. (eds.) ICDT 1997. LNCS, vol. 1186, pp. 19–40. Springer, Heidelberg (1996)
25. Yang, G., Kifer, M., Chaudhri, V.K.: Efficiently ordering subgoals with access constraints