Containment of Conjunctive Object Meta-Queries

Andrea Calì
Faculty of Computer Science
Free University of Bozen-Bolzano
piazza Domenicani 3
I-39100 Bolzano, Italy
ac@andreacali.com

Michael Kifer
Department of Computer Science
State University of New York at Stony Brook
Stony Brook, NY 11794-4400, U.S.A.
kifer@cs.sunysb.edu

ABSTRACT

We consider the problem of query containment over an object data model derived from F-logic. F-logic has generated considerable interest commercially, in the academia, and within various standardization efforts as a means for building ontologies and for reasoning on the Semantic Web. Solution to the containment problem for F-logic queries can help with query optimization as well as the classification problem in information integration systems. An important property of F-logic queries, which sets them apart from database queries, is that they can mix the data-level and the meta-level in simple and useful ways. This means that such queries may refer not only to data but also schema information. To the best of our knowledge, the containment problem for such queries has not been considered in the literature. We show that, even for queries over meta-information together with data, this problem is decidable for non-recursive conjunctive queries. We also provide relevant complexity results.

1. INTRODUCTION

The query containment problem is a question of whether the result of one query is always contained in the result of another. This problem has attracted considerable interest in the database and knowledge representation communities. In databases, query containment is key to query optimization and schema integration [2, 16, 25], and in knowledge representation it has been widely used in Description Logic [4] for object classification, schema integration, service discovery, and more [8, 23].

The most interesting and practically significant instance of the problem arises when queries are posed against databases that satisfy constraints. In this case, query results that are otherwise not contained within each other might become contained if we restrict the attention to databases that satisfy a given set of constraints. A study of this problem was pioneered by Johnson and Klug in [16] for functional and inclusion dependencies, and then further studied in other works [7, 6].

In dealing with this problem, an important issue is the form of the constraints and where they are coming from. If no restriction on the form of the constraints is placed, the containment problem may be computationally hard or even undecidable. However, in practice, constraints typically come from design tools that follow certain methodology, such as the Entity-Relationship Model [10]. For instance, in a previous work by one of the authors [6], it was shown that the containment problem under the constraints that arise from the E-R design methodology has better computational complexity than in the general case that was previously investigated in [7].

The present paper takes the same approach and considers the constraints that typically arise from object-oriented design. The specific data model that we use comes from F-logic [19]—a knowledge representation formalism that has generated considerable interest in the academia, within various standardization efforts, and commercially as a means for building ontologies and for reasoning on the Semantic Web. Query containment over object databases has been studied before [22]. However, F-logic queries have one property that is not present in the query classes considered so far: it has *meta-querying* capability, i.e., it can query data *and schema* in a uniform way. This property is considered important in knowledge integration and service discovery on the Semantic Web. The need to access schema information has been recognized in other languages as well, albeit the facilities that are made available to the user are often rather awkward. For instance, SQL databases provide access to the system catalog and Java has reflection API for the same purpose.

In this paper, we show that query containment is decidable for a subset of conjunctive F-logic meta-queries, and is in NP. This subset, which we call *F-logic lite*, excludes negation and default inheritance, and allows only a limited form of cardinality constraints. This result complements the earlier works on the subject, such as [7, 6], because F-logic queries and the associated constraints are different. For instance, decidability does not follow from [7] because conjunctive F-logic queries involve certain recursion, unions, and joins of ternary predicates, which are not allowed in [7].

The practical upshot of these results is that they open the door to the use of query containment for F-logic based appli-

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the VLDB copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Very Large Data Base Endowment. To copy otherwise, or to republish, to post on servers or to redistribute to lists, requires a fee and/or special permission from the publisher, ACM.
VLDB '06, September 12-15, 2006, Seoul, Korea.
Copyright 2006 VLDB Endowment, ACM 1-59593-385-9/06/09

cations in query processing, ontology integration, and Web service modeling and discovery [3, 17, 21, 11, 5]. Our results are also relevant to the Semantic Web language RDF [20] and the recently proposed query language for it, called SPARQL [27]. RDF has many of the meta-data features of F-logic and SPARQL can query them. Thus, our results apply to SPARQL as well.

This paper is organized as follows. Section 2 introduces the background material on F-logic queries. Section 3 provides introductory results and definitions, and Section 4 contains the main results of the paper. Section 5 concludes the paper.

2. PRELIMINARIES

F-logic was introduced in [18, 19] as a formalism for object-oriented deductive databases. Since then it received further development and was implemented in the FLORA-2 and FLORID knowledge representation systems [29, 13, 15, 14] as well as commercially [26]. Fortunately, for this paper we need only a very limited amount of background on F-logic, which will be introduced in this section. However, we assume familiarity with Datalog and Logic Programming.

Basic F-logic notation. Unlike classical predicate calculus, F-logic has special atomic formulas to represent the various object-oriented concepts that are common to object-oriented and frame-based systems. For instance, `john:student` states that object `john` is a member of class `student`; `freshman::student` and `student::person` state that class `freshman` is a subclass of the class `student` and `student` is a subclass of `person`. These statements imply, for instance, that `john:person` (`john` is a student) and `freshman::person` (class `freshman` is a subclass of `person`) are true statements.

A statement of the form `john[age->33]` means that object `john` has an attribute, `age`, whose value is `33`. Actually, this really means that `33` is just *one of the values* of the attribute `age` — to say that `33` is the *only* value, one would need a cardinality constraint, as explained later.

Common constraints such as *type constraints* and *cardinality constraints* are specified via so called *signature* statements. A typical signature has the form `person[age*=>number]`. It says that the attribute `age` of class `student` has the *type* `number` and that this type is inherited by subclasses and class instances of `person`. This acts as a constrain on the statements of the form `john[age->33]`. That is, for every object in class `person` the value of the attribute `age` must be an object of class `number`.

Cardinality constraints can also be specified using signature statements. For instance, to say that the attribute `age` has at most one value, one would write `person[age {0:1} *=> number]`. Another frequently used cardinality constraint states that a certain attribute in a class is `mandatory`, i.e., it must have at least one value on any object in the class. For instance, to say that the `name` attribute is mandatory in class `person`, we write `person[name {1:*} *=> string]`.

As mentioned in the introduction, F-logic treats object data and meta-data in a uniform way. This is primarily manifested in the following two ways:

- Classes are also objects, so, for example, statements like `student:class` are correct. Here `student`, which was previously seen in a context of a class (`john:student`), now occurs in a context of an object—a member of the class `class`. (Note that it *does not* follow from this and the previous statements that `john:class`, `freshman:class`, or `student::class`.)
- Variables can occur anywhere an object, an attribute, or a class is allowed. In this paper, we will be using capitalized words as variable names. For instance, `john:X`, `Y::person`, `john[Att->33]`, and `person[Att*=>Val]` are all allowed statements where X, Y, Att, and Val are variables.

The above properties enable simple and natural formulation for a wide variety of meta-queries—queries that return information about the schema of the database instead of the data stored in it. For instance, the answer to the meta-query

```
?- X::person.
```

could be `X = employee` and `X = student`, and the answer to the meta-query

```
?- student[Att*=>string].
```

could be `Attr = name` and `Attr = major`. Queries can also mix the meta and the object levels. For instance, the mixed query (where "," denotes "and")

```
?- student[Att*=>string], john[Att->Val].
```

will return a set of attribute-value pairs for the attributes of type `string` in class `student`. Only the attributes that have defined values for object `john` are returned. Incidentally, `john` does not need to be a member of class `student` for such a query to return a result.

Examples of meta-queries. Meta-querying is a commonly acknowledged need in knowledge representation — especially on the Semantic Web. To show that it is not such an esoteric idea, we give some examples of meaningful meta-queries and the corresponding query containments. Consider the following rule:

```
q(A,B) :- T1[A*=>T2], T2::T3, T3[B*=>_].
```

As usual, `:-` here denotes Datalog implication. The part to the left of `:-` is the *head* of the rule and the part to the right is the rule `body`. The comma separating the statements in the rule body is an abbreviation for conjunction, and the symbol _ is a common notation for a completely new variable

that does not appear anywhere else in the rule. (Different occurrences of _ denote different variables.)

The above rule defines a set of pairs `(A,B)` of attributes that are *joinable* in a path expression of the form `A.B` (that is, the range of `A` is contained in the domain of `B`). If we now examine the following rule

```
qq(A,B) :- T1[A*=>T2], T2[B*=>_].
```

we will see that the query containment $\mathtt{q} \subseteq \mathtt{qq}$ holds.

For another example, consider the following rule:

```
q(Att,Class,Type) :-
     Class[Att {1,*} *=> _],
     Class[Att*=>Type],
     _:Class.
```

Recall that `{1,*}` is a cardinality constraint that says that `Att` must have at least one value, i.e., it is a *mandatory* attribute. This rule defines a set of triples `(Att,Class,Type)` such that `Att` is a mandatory attribute in `Class` of type `Type` and, furthermore, `Class` is nonempty. Note that here we are querying meta-data that is subject to certain constraints: only the mandatory attributes are of interest and only the classes that have at least one member in the database. Consider now the following rule:

```
qq(Att,Class,Type) :-
      O:Class, O[Att->V], V:Type.
```

It is easy to see that the containment $\mathtt{q} \subseteq \mathtt{qq}$ holds.

Low-level encoding of F-logic primitives. The above notation was used to motivate the problem and to define the target class of queries. The actual theoretical development will use an encoding of the semantics of a subset of F-logic using the standard logic programing notation. The encoding relies on the equivalence result of [19].

The subset of F-logic, which we will consider in this paper will be referred to as *F-logic Lite*. This subset is characterized by the absence of negation and nonmonotonic features of F-logic (such as default inheritance) and by allowing only the cardinality constraints of the form `{1:*}` — a constraint that says that the corresponding attribute in mandatory — and of the form `{0:1}` — a constraint that marks functional (single-valued) attributes. Some other features, such as default values and non-inheritable types, are also ignored.

The encoding, uses the following predicates, whose set we will denote by P_{FL}:

- $\mathsf{member}(O, C)$: object O is a member of class C. This is the encoding for $O : C$.
- $\mathsf{sub}(C_1, C_2)$: class C_1 is a subclass of class C_1. This encodes the statement $C_1 :: C_2$.
- $\mathsf{data}(O, A, V)$: attribute A has value V on object O. This is the encoding for $O[A\texttt{->}V]$.
- $\mathsf{type}(O, A, T)$: attribute A has type T for object O (recall that in F-logic classes are also objects). This encodes the statements of the form $O[A\texttt{*=>}T]$.
- $\mathsf{mandatory}(A, O)$: attribute A is mandatory for object (class) O, i.e., it must have at least one value for O. This is an encoding of statements of the form $O[A\,\texttt{\{1:*\}*=>\_}]$.
- $\mathsf{funct}(A, O)$: A is a functional attribute for the object (class) O, i.e., it can have at most one value for O. This statement encodes $O[A\,\texttt{\{0:1\}*=>\_}]$.

Note that this encoding places meta-data (classes and attributes) and object data at the same level, which is needed for supporting F-logic meta-queries. This encoding is also related to, but is slightly different from, the usual encoding of RDF in first-order logic.

We can now formulate the axioms that represent the low-level encoding of the F-logic primitives discussed above in standard predicate notation. We annotate each rule in the encoding to make it easier to follow.

(1) Type correctness:
$\mathsf{member}(V, T)$ `:-` $\mathsf{type}(O, A, T), \mathsf{data}(O, A, V)$.
This encodes the semantics of the constraint that an F-logic signature like `O[A *=> T]` imposes on a statement like `O[A -> V]`; that is, that `V` must be of type `T`.

(2) Subclass transitivity:
$\mathsf{sub}(C_1, C_2)$ `:-` $\mathsf{sub}(C_1, C_3), \mathsf{sub}(C_3, C_2)$.
This rule encodes the fact that the subclass relationship is transitive.

(3) Membership property:
$\mathsf{member}(O, C_1)$ `:-` $\mathsf{member}(O, C), \mathsf{sub}(C, C_1)$.
This is the usual property that relates class membership and subclass relationship: `O:C` and $\mathtt{C::C}_1$ imply $\mathtt{O:C}_1$.

(4) Functional attribute property:
$V = W$ `:-`
$\mathsf{data}(O, A, V), \mathsf{data}(O, A, W), \mathsf{funct}(A, O)$.
This rule states that if we have `O[A->V]`, `O[A->W]`, and the attribute `A` is single-valued then X must equal W.

(5) Mandatory attributes definition:
$\forall O, A\ \exists V\ \mathsf{data}(O, A, V)$ `:-` $\mathsf{mandatory}(A, O)$. This states that mandatory attributes must have at least one value. Note that this is *not* a Datalog rule (not even a Horn rule) because it has an existentially quantified variable in the head.

(6) Inheritance of types from classes to members:
$\mathsf{type}(O, A, T)$ `:-` $\mathsf{member}(O, C), \mathsf{type}(C, A, T)$.
This rule expresses the usual property of type inheritance: the type of an attribute is inherited from superclasses to class members.

(7) Inheritance of types from classes to subclasses:
$\mathsf{type}(C, A, T)$ `:-` $\mathsf{sub}(C, C_1), \mathsf{type}(C_1, A, T)$.
This states that subclasses inherit types from superclasses.

(8) Supertyping:
$\mathsf{type}(C, A, T)$ `:-` $\mathsf{type}(C, A, T_1)$, $\mathsf{sub}(T_1, T)$.
This states that $\mathtt{T_1::T}$ and $\mathtt{C[A *=> T_1]}$ entails $\mathtt{C[A *=> T]}$. This is also one of the usual properties of typing: if an attribute has certain type then any supertype of that type will also do.

(9) Inheritance of mandatory attributes to subclasses:
$\mathsf{mandatory}(A, C)$ `:-` $\mathsf{sub}(C, C_1), \mathsf{mandatory}(A, C_1)$.
This states that a mandatory attribute of a class is also a mandatory attribute of its subclasses.

(10) Inheritance of mandatory attributes from classes to their members:
$\mathsf{mandatory}(A, O)$ `:-`
$\mathsf{member}(O, C), \mathsf{mandatory}(A, C)$.
Like in (9), but this time inheritance of the mandatory property is to class members rather than subclasses.

(11) Inheritance of functional property to subclasses:
$\mathsf{funct}(A, C)$ `:-` $\mathsf{sub}(C, C_1)$, $\mathsf{funct}(A, C_1)$.
If `A` is a single-valued attribute in a class then it must be single-valued in the subclasses of that class.

(12) Inheritance of functional property to members:
$\mathsf{funct}(A, O)$ `:-` $\mathsf{member}(O, C), \mathsf{funct}(A, C)$.
Like (11), but this time inheritance of the single-valued property is to class members.

In the following, we will denote the above set of rules by Σ_{FL}. We will also refer to the i-th rule as ρ_i. The above statements are all Datalog rules except ρ_4 and ρ_5. Rule ρ_4 uses equality in the head and ρ_5 has an existential quantifier in the rule head and thus invents new (fresh) values. Also, most of the rules are recursive. Additional properties of this encoding are studied in the next section.

Thanks to the above encoding, we can express any F-logic Lite database and its schema as a relational database augmented with a set of rules for deriving new information and for expressing constraints. We shall consider *only* the databases that satisfy the above set of rules.

Query containment. The query containment problem for F-logic Lite can be now stated as follows. Given a pair of meta-queries, q_1 and q_2, over the predicates P_{FL} of the above encoding of F-logic Lite, we say that q_1 *is contained in* q_2 with respect to Σ_{FL}, denoted $q_1 \subseteq_{\Sigma_{FL}} q_2$, if for every database B that satisfies Σ_{FL} we have $q_1(B) \subseteq q_2(B)$, where $q(B)$ denotes the result of query q on B.

In this paper we will focus on positive *conjunctive queries* [1], i.e., the queries that are conjunctions of the predicates in P_{FL} and no negation is allowed.

3. CHASE AND CONTAINMENT

In this section we study the problem of query containment for queries expressed over the schema P_{FL} derived from the low-level encoding of F-logic Lite. We recall that the encoding is entirely relational plus the rules Σ_{FL} shown in Section 2. First, we introduce the notion of *homomorphism*, which is of fundamental importance in conjunctive query containment [9]. We remind that for conjunctive queries over generic relational schemata without constraints, a homomorphism from the body of a query q_2 to the body of another query, q_1, which also maps the head of q_2 to the head of q_1, implies $q_1 \subseteq q_2$. Indeed, if the body of q_1 is mapped by a homomorphism to facts of a database, B, so that the head of q_1 is mapped to tuple t, then by composing the homomorphisms $q_2 \longrightarrow q_1$ and $q_1 \longrightarrow B$ we get a homomorphism that maps the body of q_2 to the facts of B and the head of q_2 to t. Thus, every tuple produced by q_1 is also produced by q_2.

Definition 1 Given a database B and a query q, a *homomorphism* from q to B is a function from the symbols of q to the values of B that maps every constant occurring in q to itself and variables of q to constants in B. This induces a well defined map from the conjuncts of q to the tuples of the *corresponding* relations in B. In particular, a conjunct $r(c_1, \ldots, c_n)$ in q is mapped by a homomorphism μ to a fact $r(\mu(c_1), \ldots, \mu(c_n))$ in B. We also extend the definition to sets of conjuncts: given a set of conjuncts $C = \{c_1, \ldots, c_n\}$, we define $\mu(C) = \{\mu(c_1), \ldots, \mu(c_n)\}$.

Now we come to the notion of *chase* of a query with respect to our set of rules Σ_{FL}. The chase [24, 16] is a tool for representing databases that satisfy certain dependencies, and it is used to check implication of dependencies and containment of queries under dependencies. Given a database, the chase is constructed by a sort of *repair* of the database w.r.t. the rules that are not satisfied. In particular, in our case, violations of all rules except ρ_4 are repaired with the *addition* of suitable tuples, while violations of ρ_4 are repaired by *equating* constants that are not equal. The rules in $\Sigma_{FL} - \{\rho_4\}$ are called *tuple-generating dependencies*, while ρ_4 is an *equality generating dependency*. The query q to be "chased" is treated as a database, and new tuples (or conjuncts) are added according to the rules. The chase of a query q, denoted $chase_{\Sigma_{FL}}(q)$, is a database constructed starting from $body(q)$.

Henceforth, we will use the terms *tuple* and *conjunct* interchangeably when talking about chasing queries. Homomorphisms will map tuples in a chase to other tuples of the same chase. We will use $\mathcal{R}(c)$ to denote the relation symbol associated with a conjunct c in the chase; for example, if $c = r(a_1, \ldots, a_m)$ then $\mathcal{R}(c) = r$.

Definition 2 Given a conjunctive query q over P_{FL} and the set Σ_{FL} of F-logic Lite rules, we define the *chase* of q w.r.t. Σ_{FL}, denoted $chase_{\Sigma_{FL}}(q)$, as follows. Initially, $chase^*_{\Sigma_{FL}}(q)$ is the set of all conjuncts in the body of q. Then we modify $chase^*_{\Sigma_{FL}}(q)$ according to the following rules:

(1) If there is a homomorphism μ that sends the atoms in the $body(\rho_4)$ to conjuncts of $chase^*_{\Sigma_{FL}}(q)$, then we say that the rule ρ_4 is *applicable* and we proceed as follows: *(a)* If $\mu(V)$ and $\mu(W)$ are distinct constants (recall that V and W are the two variables appearing

in the head of ρ_4), stop the chase—the construction fails; *(b)* If $\mu(V)$ precedes $\mu(W)$ in lexicographic order (we assume that all constants precede all variables in this order), change $\mu(W)$ into $\mu(V)$ everywhere in $chase^*_{\Sigma_{FL}}(q)$ (and in $head(q)$ if $\mu(W)$ appears there); if instead $\mu(W)$ precedes $\mu(V)$, change $\mu(V)$ into $\mu(W)$ (if they are equal then nothing needs to be done).

(2) For every rule $\rho \in \Sigma_{FL} - \{\rho_4\}$: if there is a homomorphism μ that maps the atoms in the $body(\rho)$ to tuples of $chase^*_{\Sigma_{FL}}(q)$, then *(i)* If $\rho \neq \rho_5$, and $\mu(head(\rho)) \notin chase^*_{\Sigma_{FL}}(q)$, we say that the rule ρ is *applicable* and we add $\mu(head(\rho))$ to $chase^*_{\Sigma_{FL}}(q)$; *(ii)* If $\rho = \rho_5$, let us say that μ' *extends* μ if it is like μ everywhere except the existential variable V in the head of ρ_5. If there is no extension of μ that sends $head(\rho)$ to some conjunct in $chase^*_{\Sigma_{FL}}(q)$, we say that the rule ρ_5 is *applicable*. In this case, let μ' extend μ by mapping V to a fresh constant that lexicographically follows all other constants in the segment of the chase constructed so far (but still precedes all variables). Add $\mu'(head(\rho_5))$ to the chase as a conjunct.

The construction of the chase proceeds by iteratively executing the following two steps: *(a)* while rule 1 is applicable, apply it repeatedly; *(b)* if rule 2 is applicable for some ρ, apply it *once*.

Notice that it may happen that, in the application of rule 1, the head of the query q may be modified. This is because rule ρ_4 and the chase rule *(1)* may cause a change of a variable that appears in the head.

Example 1 Consider the following meta-query:

$$q(V_1, V_2) \quad :- \quad \mathsf{data}(O, A, V_1), \mathsf{data}(O, A, V_2), \mathsf{funct}(A, C), \mathsf{member}(O, C)$$

In the construction of $chase_{\Sigma_{FL}}(q)$, rule ρ_{12} will add the conjunct $\mathsf{funct}(A, O)$ and then, by rule ρ_4, we will replace V_2 with V_1 and obtain

$$q(V_1, V_1) \quad :- \quad \mathsf{data}(O, A, V_1), \mathsf{data}(O, A, V_1), \mathsf{funct}(A, O), \mathsf{funct}(A, C), \mathsf{member}(O, C)$$

□

Therefore, the chase procedure may have side effects on the head of the query. Henceforth we shall use $head(chase_{\Sigma_{FL}}(q))$ to denote the head of the query q as it is transformed by the construction of $chase_{\Sigma_{FL}}(q)$ according to Σ_{FL}.

We now give the definition of the *chase graph*. In this graph, the notion of *level* roughly indicates how far we need to go in the chase starting from the initial query. This is crucial for our purposes, since we will show that in order to test containment we will need to examine the chase only up to a certain level. The chase graph is defined as follows.

Definition 3 Given a conjunctive query q on P_{FL}, we define the *chase graph* $G(q)$ as follows.

(1) The nodes of the graph are the conjuncts in $chase_{\Sigma_{FL}}(q)$.

(2) If a conjunct, c, is generated in the chase by the application of a rule $\rho \in \Sigma_{FL}$, then there is an arc in $G(q)$ from each of the tuples that are involved in generation of c by the application of ρ (i.e., tuples on which the body of ρ is mapped) to c. Such arcs are labelled with ρ.

(3) The *level* of a a conjunct c, written as $level(c)$, is defined as follows; *(i)* if $c \in body(q)$, then $level(c) = 0$; *(ii)* if c is generated by the application of rule ρ on conjuncts c_{p1}, ..., c_{pn} (for our rules $N \leq 3$) then $level(c) = \max\{level(c_{p1}), ..., level(c_{pn})\} + 1$.

(4) If in the construction of the chase, for some rule $\rho \in \Sigma_{FL} - \{\rho_4\}$, there is a homomorphism μ that sends the atoms in $body(\rho)$ to tuples of $chase^*_{\Sigma_{FL}}(q)$, where $chase^*_{\Sigma_{FL}}(q)$ is the fragment of the chase constructed so far, then *(i)* if $\rho \neq \rho_5$, and $\mu(head(\rho)) \in chase^*_{\Sigma_{FL}}(q)$, there will be special arcs in $G(q)$ from each of the tuples in $\mu(body(\rho))$ (the tuples on which the body of ρ is mapped) to c; such arcs are called *cross-arcs*, and they are labelled as the ordinary arcs; *(ii)* if $\rho = \rho_5$ and there *is* an extension μ' of μ that maps $head(\rho)$ to a conjunct in $chase^*_{\Sigma_{FL}}(q)$, there will be a cross-arc in $G(q)$ from each of the tuples in $\mu'(body(\rho))$ to $\mu'(head(\rho))$.

(5) Arcs (cross-arcs or non-cross-arcs) from a node at a level k to a node at level $k + 1$ are called *primary* arcs, while the others are called *secondary*.

The following theorem provides the basic tool for checking containment of queries over P_{FL}. Intuitively, in the containment check between two queries q_1 and q_2, $body(q_1)$ represents the set of tuples in a generic database B that lead to an answer tuple in $q(B)$. Since we need to check containment under Σ_{FL}, we need to take into account not only $body(q_1)$, but also further tuples that the rules guarantee to be in B; therefore, we need to consider $chase_{\Sigma_{FL}}(q_1)$. This is stated as follows.

Theorem 4 Let q_1 and q_2 be two conjunctive queries over P_{FL} with the same arity. Then $q_1 \subseteq_{\Sigma_{FL}} q_2$ if and only if there exists a homomorphism that sends the conjuncts of $body(q_2)$ to conjuncts in $chase_{\Sigma_{FL}}(q_1)$ and $head(q_2)$ to $head(chase_{\Sigma_{FL}}(q_1))$.

Proof (sketch). The proof of this theorem can be derived from [12]. In that paper, *tuple-generating dependencies (TGDs)* and *equality-generating dependencies (EGDs)* are considered together, and it is shown that any successful chase of a database B constructed according to a set Σ of TGDs and EGDs is a *universal solution*, i.e., a database that is a representative of all databases that include B and satisfy σ. Our rules in Σ_{FL} have the same form as TGDs and EGDs; therefore, we obtain the claim from [12] by considering q_1 as a database. Although [12] deals with *finite* universal solutions and we are dealing with infinite chases, the same techniques apply in our setting. □

The previous theorem establishes a criterion for checking containment, but it is not directly applicable, since it does not suggest an algorithm for deciding containment. In fact, the iterative construction of the chase by application of the chase rules may not terminate. In the following section we will first show some properties of the chase, and then we will show that, in order to test containment of meta-queries, only a finite portion of the chase is necessary.

4. DECIDABILITY OF CONTAINMENT

In this section we prove that containment of object meta-queries is decidable, and we give a nondeterministic polynomial time algorithm for checking containment between two object meta-queries. We show that only a finite portion of the possibly infinite chase is necessary to check containment with the technique suggested by Theorem 4, and then we present an algorithm for query containment, showing an upper bound for the complexity of the problem.

For technical reasons, we will do the chase in a special way. This will isolate some of the peculiarities of the chase and allow us to concentrate on more important properties. Namely, we shall first proceed with the chase with respect to all the rules except for ρ_5; such a preliminary chase always terminates, since no new constant is generated. To simplify matters, we will view all tuples in $chase_{\Sigma^-_{FL}}(q)$ as being at level 0, where $\Sigma^-_{FL} = \Sigma_{FL} - \{\rho_5\}$. This will allow us to isolate the initial part of the chase from the cyclic phase (if the latter takes place).

We now illustrate some properties of the chase that will be useful in the proof of the main result. First of all it is not difficult to notice that, in the construction of the chase for a query q with respect to the set Σ_{FL}, the only way to have an infinite chase is the iterative application of rules ρ_5-ρ_1-ρ_6-ρ_{10}. This happens when q contains at least a set of atoms specifying a cycle of mandatory attributes $A_1, \ldots, A_k$ belonging to classes $T_1, \ldots, T_k$, respectively, where A_i is of type T_{i+1} for all $i \in \{1, \ldots, k-1\}$ and A_k is of type T_1. More precisely, we need q to have conjuncts of the following form:

$$\begin{array}{l}
\mathsf{mandatory}(A_1, T_1)\\
\mathsf{type}(T_1, A_1, T_2)\\
\ldots\\
\mathsf{mandatory}(A_{k-1}, T_{k-1})\\
\mathsf{type}(T_{k-1}, A_{k-1}, T_k)\\
\mathsf{mandatory}(A_k, T_k)\\
\mathsf{type}(T_k, A_k, T_1)
\end{array}$$

In such a case, if there is no atom in q of the form $\mathsf{data}(T_1, A_1, v)$, where v is a constant or variable, the chase process yields the following series of conjuncts:

$$\begin{array}{rl}
\text{cycle } 1: & \mathsf{data}(T_1, A_1, v_1)\\
& \mathsf{member}(v_1, T_2)\\
& \mathsf{type}(v_1, A_2, T_3)\\
& \mathsf{mandatory}(A_2, v_1)
\end{array}$$

$$\begin{array}{rl}
\text{cycle } 2: & \mathsf{data}(v_1, A_2, v_2)\\
& \mathsf{member}(v_2, T_3)\\
& \mathsf{type}(v_2, A_3, T_4)\\
& \mathsf{mandatory}(A_3, v_2)\\
& \ldots\\
& \ldots
\end{array}$$

$$\begin{array}{rl}
\text{cycle } k-1: & \mathsf{data}(v_{k-2}, A_{k-1}, v_{k-1})\\
& \mathsf{member}(v_{k-1}, T_k)\\
& \mathsf{type}(v_{k-1}, A_k, T_1)\\
& \mathsf{mandatory}(A_k, v_{k-1})
\end{array}$$

$$\begin{array}{rl}
\text{cycle k}: & \mathsf{data}(v_{k-1}, A_k, v_k)\\
& \mathsf{member}(v_k, T_1)\\
& \mathsf{type}(v_k, A_1, T_2)\\
& \mathsf{mandatory}(A_1, v_k)
\end{array}$$

In the rest of the chase, at levels greater than 0, the only other applications of a rule in Σ_{FL} occur due to the application of ρ_3, ρ_7 or ρ_8. In this case, depending on the conjuncts al level 0, new cycles may start. All other rules are applied in $chase_{\Sigma^-_{FL}}(q)$ (i.e., in the initial construction of level 0 of the chase graph), and they are never applied again at higher levels.

Example 2 Consider the query q defined as $q() \text{ :- } \mathsf{mandatory}(A, T), \mathsf{type}(T, A, T), \mathsf{sub}(T, U)$. Part of the chase graph $G(q)$ is shown in Figure 1. Notice that an infinite chain is created by the conjuncts

$$\begin{array}{l}
\mathsf{mandatory}(A, T)\\
\mathsf{type}(T, A, T)\\
\mathsf{data}(T, A, v_1)\\
\mathsf{member}(v_1, T)\\
\mathsf{type}(v_1, A, T)\\
\mathsf{mandatory}(A, v_1)\\
\mathsf{data}(v_1, A, v_2)\\
\mathsf{member}(v_2, T)\\
\mathsf{type}(v_2, A, T)\\
\ldots
\end{array}$$

This chain is formed by conjuncts that never interact with other ones except for those at level 0. Because of rules ρ_3, ρ_8 and ρ_8, it is possible that branches depart from this chain; in our case, for example, we obtain the conjunct $\mathsf{member}(v_1, U)$ from ρ_3. It may be possible, depending on the conjuncts at level 0, that the new conjuncts generated by rules ρ_3, ρ_8 and ρ_8 start new cycles; however, it is easy to see that such cycles (again due to the iterative application of rules ρ_5-ρ_1-ρ_6-ρ_{10}) do not interfere with each other, and they do not interact with the existing cycles in the chase. □

The previous considerations easily lead to the following result, stating that the above described cycles do not interfere with each other.

Lemma 5 (Locality) Consider a meta-query q, and its chase, $chase_{\Sigma_{FL}}(q)$. Consider a conjunct c at $level(c) \geq 1$. Then every secondary arc going into c starts at a conjunct d such that either: *(i)* $level(d) = 0$, or *(ii)* $level(d) = level(c) - 2$. Moreover, the parent e of d (i.e., the node e such that (e, d) is a primary arc) is such that there is a path from e to c that traverses 3 arcs.

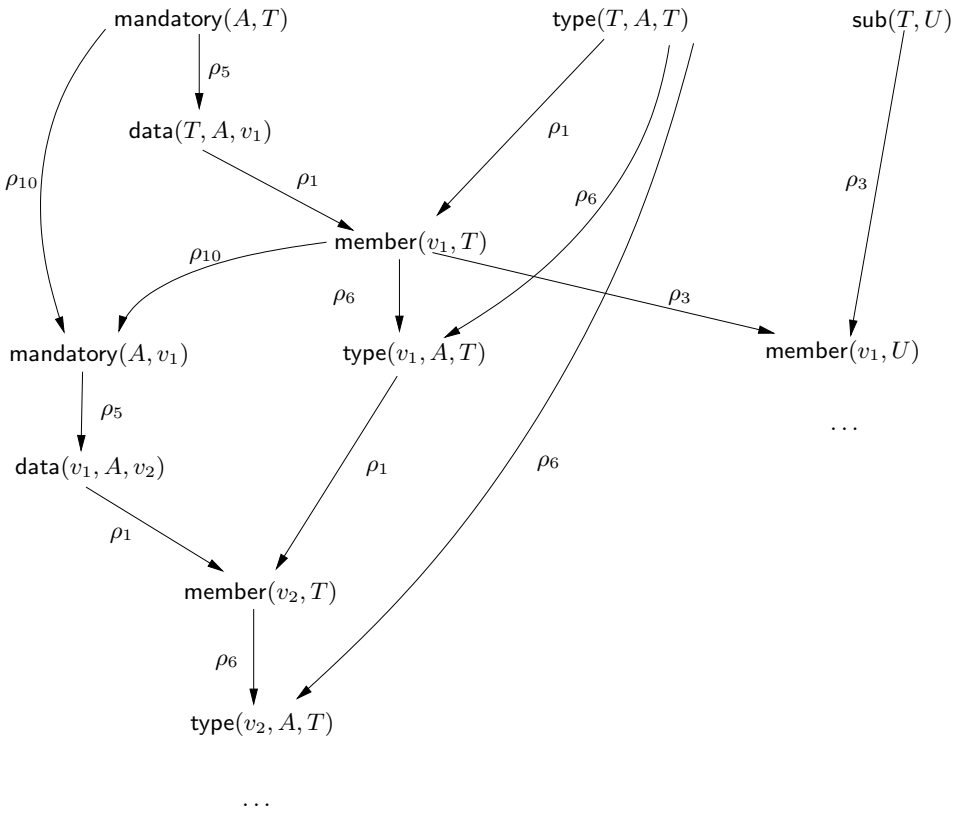

Figure 1: Chase graph for Example 2

The above locality property in the lemma is crucial for proving the decidability of containment. Intuitively, the conjuncts at level 0 in $chase_{\Sigma_{FL}}(q)$ establish the way the chase develops, while the new conjuncts are added depending *only* on the conjuncts at level 0 in $chase_{\Sigma_{FL}}(q)$ and on predecessors conjuncts that are one or two levels back in the chase. Notice that case *(ii)* in Lemma 5 refers to the application of rule ρ_1. Consider the conjunct $c = \mathsf{member}(v_2, T)$ in Example 2. It is generated by the application of ρ_1 on $d_1 = \mathsf{data}(v_1, A, v_2)$ and $d_2 = \mathsf{type}(v_1, A, T)$, where (d_1, c) is a primary arc and $level(c) = level(d_2) + 2$. Also, consider the conjunct $e = \mathsf{member}(v_1, T)$, which is a parent of d_2, i.e., such that (e, d_2) is a primary arc. Then there is a path from e to c, which traverses 3 primary arcs. Intuitively, this makes d_2 stay "local" with respect to c, and the chain of conjuncts remains "isolated" from the rest of the chase.

Now we come to the notion of *equivalence* between conjuncts in the chase. The idea is that, when the chase is infinite, its structure is *cyclic*, and after a certain number of levels, the structure of the chase becomes similar to that in the earlier levels. In our case, this means that if we are looking for a homomorphism from q_2 to $chase_{\Sigma_{FL}}(q_1)$ in order to check whether $q_1 \subseteq_{\Sigma_{FL}} q_2$, we can limit our attention to an a priori bounded prefix of the chase, and later conjuncts add no new information.

Definition 6 (Equivalent conjuncts) Consider two conjuncts c_1, c_2 of the same arity k, in the chase $chase_{\Sigma_{FL}}(q)$ of a query q. We say that c_1 and c_2 are *equivalent*, denoted $c_1 \sim c_2$, if for every i s.t. $1 \leq i \leq k$ it holds that if $c_1[i]$ or $c_2[i]$ are constants, then $c_1[i] = c_2[i]$, where $c[i]$ denotes the i-th component of a conjunct c. In other words, c_1 and c_2 must agree on the components that are (non-fresh) constant symbols in order to be equivalent.

We now define the notion of *primary path*—a path in the chase graph that consists of "almost" primary arcs only.

Definition 7 (Primary path) Consider a chase graph $G(q)$ for a conjunctive meta-query q. A *primary path* is a path in $G(q)$ from a conjunct c_1 to a conjunct c_2, where each arc is either *(i)* a primary arc or *(ii)* an arc from c_1 to a conjunct c s.t. $\mathcal{R}(c_1) = \mathsf{type}$ and $level(c) = level(c_1) + 2$.

The notion of primary path is needed as a sort of thread on which the chains of the chase develop, from level 0 to higher levels. Unlike the case of inclusion dependencies in relational databases [16], here the chains in the chase graph are not chains of conjuncts; instead, they have a more complicated structure. However, as stated by the locality principle, there are chains that are independent from each other in the construction of the chase, and that occasionally branch out. Primary paths identify such pseudo-chains. The case *(ii)* in Definition 7 is necessary because, if we allow only primary arcs in primary paths, such paths would not be able to start from a conjunct c where $\mathcal{R}(c) = \mathsf{type}$. This should be clear from Example 2, where one can see that, in infinite cycles due to the iterative application of rules ρ_5-ρ_1-ρ_6-ρ_{10},

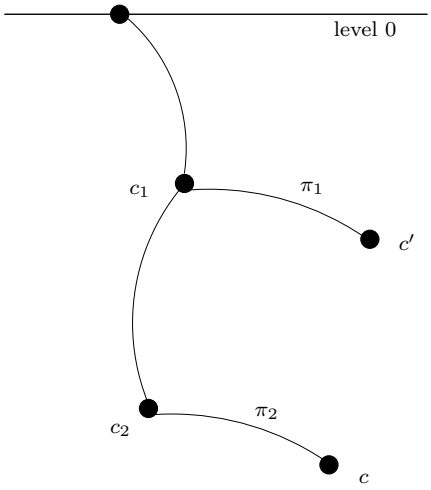

Figure 2: Figure for Lemma 9.

the arcs going from every conjunct c with $\mathcal{R}(c) = \mathsf{type}$ lead to a conjunct d such that $level(d) = level(c) + 2$.

In order to characterize primary paths that develop in similar ways, we use the notion of *parallel* primary paths.

Definition 8 Consider two paths, π_1 and π_2, of the same length with c_1 and c_1' as initial conjuncts. We say that π_1 and π_2 are *parallel*, denoted $\pi_1 \parallel \pi_2$, if the following property holds. Let $c_1, \ldots, c_n$ be the conjuncts in π_1, in this order, and let $c_1', \ldots, c_n'$ be the conjuncts in π_2. Then for all i, $\leq i \leq n$, the arcs (c_i, c_{i+1}) and (c_i', c_{i+1}') must be labeled with the same rule symbol ρ_i (and therefore $\mathcal{R}(c_{i+1}) = \mathcal{R}(c_{i+1}'))$.

We now show that if a conjunct is at a level of the chase $chase_{\Sigma_{FL}}(q)$ of a query q that is greater than a certain bound (which depends on q), it can be mapped by a homomorphism to a conjunct that has level lower than that bound. This result will allow us to prove an analogous result for sets of conjuncts.

Lemma 9 Consider a conjunct $c \in chase_{\Sigma_{FL}}(q)$; there exists a homomorphism μ from $\{c\}$ to $chase_{\Sigma_{FL}}(q)$ such that $level(\mu(c)) \leq 2 \cdot |q|$.

Proof (sketch). To prove this result, we use the property of the chains of conjuncts, which was established by Lemma 5. After some number of steps δ following primary path arcs, we will have completed a cycle of mandatory attributes, and at least two equivalent conjuncts will be generated on the path. The value of δ is bounded by $2 \cdot |q|$ because in order to have a cycle of k mandatory attributes, we need $2k$ conjuncts in q and also $4k$ levels to complete the cycle in the chase. We consider a primary path from a conjunct at level 0 to c (see Figure 2). It is not difficult to see, from Definition 7, that this path must be unique; if the path is shorter than δ, we are done; otherwise, there must be at least two equivalent conjuncts in it, c_1, c_2, such that $level(c_1) < level(c_2)$. Now consider the primary path π_2 from c_2 to c, and the path π_1 from c_1 to c' s.t. $\pi_1 \parallel \pi_2$; it is easy to see that the final conjunct c' on π_2 is equivalent to c. Moreover, we have $level(c') \leq level(c) - (level(c_2) - level(c_1))$. If $level(c') \leq \delta$ we are done, otherwise we proceed again with the same operation until we get a conjunct $\bar{c}$ such that $level(\bar{c}) \leq \delta$ and $\bar{c} \sim c$. The equivalence between $\bar{c}$ and c ensures the existence of a homomorphism with the desired properties. □

Next, we show a result analogous to that of Lemma 9 for pairs of conjuncts.

Lemma 10 Consider two conjuncts c_1, c_2 in $chase_{\Sigma_{FL}}(q)$, such that there is a primary path from c_1 to c_2, in the chase graph $G(q)$, and assume there is a homomorphism μ that sends c_1 to a conjunct c_1' in $chase_{\Sigma_{FL}}(q)$, where $c_1' \sim c_1$. Then there exists a homomorphism μ' s.t. $\mu'(c_1) = c_1'$ and $\mu'(c_2) = c_2'$, with $level(c_2') \leq level(c_1') + \delta$, where $\delta = 2 \cdot |q|$.

Proof (sketch). Consider the primary path π_1, from c_1 to c_2 (see Figure 3); now consider a path π_2 from c_1' to some conjunct c_2' such that $\pi_1 \parallel \pi_2$; it is easy to see that $c_1' \sim c_2'$. Now, if $level(c_2') - level(c_1') \leq \delta$ we are done; otherwise, π_2 has two equivalent conjuncts, $\bar{c}_1$ and $\bar{c}_2$, and we can perform excisions of the path between these two conjuncts as in the previous lemma, until the "clipped" path runs across a number of levels that is less or equal than δ. The new "clipped" path will terminate in a conjunct c_3, as shown in Figure 3 where the path from $\bar{c}_1$ to $\bar{c}_2$ has been clipped. Moreover, the homomorphism μ' sending c_1 to c_1' and c_2 to c_3 will exist due to the equivalence between c_2' and c_3, and due to the fact that the only symbols shared by c_1 and c_2 are those that also appear in q. □

We now present the key result, mentioned earlier, that a set of n conjuncts in the chase of a query can be mapped by a homomorphism to another set of conjuncts at levels lower than a certain limit, which depends on n and q.

Lemma 11 Given a query q over P_{FL}, consider a set of conjuncts $C = \{c_1, \ldots, c_n\}$ in $chase_{\Sigma_{FL}}(q)$. Then there exists a

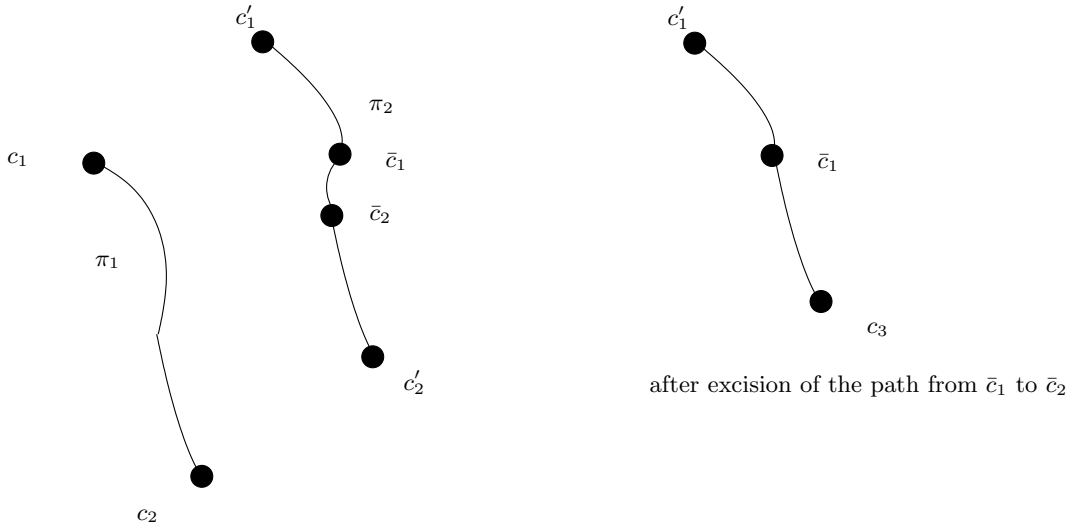

Figure 3: Figure for Lemma 10.

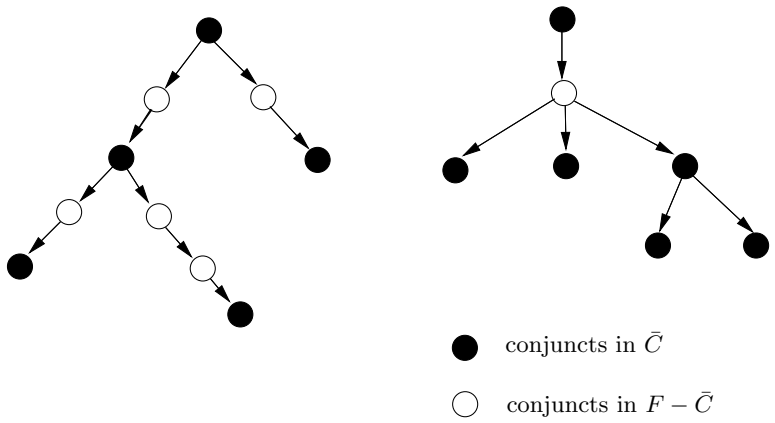

Figure 4: Figure for Lemma 11.

homomorphism h from C to $chase_{\Sigma_{FL}}(q)$ such that for every i, $1 \leq i \leq n$, we have $level(h(c_i)) \leq n \cdot \delta$, where $\delta = 2 \cdot |q|$.

Proof (sketch). Consider all conjuncts in C, and the union of all arcs belonging to primary paths from some conjunct at level 0 to every $c \in C$ in the chase graph of q. It is easy to see that, due to the fact that paths do not interact with each other, the primary paths here are unique. The conjuncts in all such paths in general constitute, together with the arcs among them, a *forest* F (see Figure 4). Let $\bar{C}$ be the union of C with the conjuncts in F that have at least two outgoing edges in F. Figure 4 shows such a forest, where conjuncts in $\bar{C}$ are drawn as solid circles. Now we proceed by iteration on the structure of the forest F; if $|\bar{C}| = 1$ the claim is true by Lemma 9. Now, consider a tree in F. The conjunct c in F that reside at the lowest level > 0 can be mapped with a homomorphism to another conjunct, c', at level $\leq \delta$; consider a descendant d of c in $F \cap \bar{C}$; it can be mapped (by another homomorphism) to a conjunct that is at most δ levels deeper than c', i.e. at a level $\leq 2\delta$; the same holds for successors of d and so on. It is not difficult to see that all homomorphisms determined in this way are *compatible* with each other, therefore they can be combined in a single homomorphism h. Since the maximum number of nodes in $\bar{C}$ that belong to a single primary path in F is $|C| = n$, it is not difficult to see that h maps C to conjuncts that have depth at most $\delta \cdot n$. □

As a consequence of the previous lemma, to check $q_1 \subseteq_{\Sigma_{FL}} q_2$ we only need to find a homomorphism from q_2 to an initial, a priori bounded finite segment of $chase_{\Sigma_{FL}}(q_1)$.

Theorem 12 Let q_1 and q_2 be two conjunctive queries over P_{FL} with the same arity. Then $q_1 \subseteq_{\Sigma_{FL}} q_2$ if and only if there exists a homomorphism that sends the conjuncts of $body(q_2)$ to conjuncts in the first $|q_2| \cdot \delta$ levels of $chase_{\Sigma_{FL}}(q_1)$, and $head(q_2)$ to $head(chase_{\Sigma_{FL}}(q_1))$, where $\delta = 2 \cdot |q_1|$.

Proof (sketch). This theorem is a direct consequence of Lemma 11 and Theorem 4. We use Theorem 4 to check the containment $q_1 \subseteq_{\Sigma_{FL}} q_2$. Suppose we find a homomorphism μ from $body(q_2)$ to $chase_{\Sigma_{FL}}(q_1)$ that has the desired properties, and such that it maps the conjuncts in $body(q_2)$

to a set $C = \{c_1, \ldots, c_n\}$ of conjuncts in $chase_{\Sigma_{FL}}(q_1)$. If all conjuncts in C have level less or equal than $|q_2| \cdot \delta$, we are done; otherwise, we apply Lemma 11 and obtain a new homomorphism, λ, that maps the conjuncts of C to another set of conjuncts C' such that all its conjuncts have levels less or equal than $|q_2| \cdot \delta$. By Theorem 4, the composition $\lambda \circ \mu$ of the two homomorphisms establishes the containment. Clearly, this homomorphism only considers the levels of $chase_{\Sigma_{FL}}(q_1)$ that are $\leq |q_2| \cdot \delta$. □

Finally, we characterize the computational complexity of the problem of checking containment of queries by giving an upper bound for the problem. We do this by exhibiting an algorithm for checking containment, which obeys the bounds.

Theorem 13 Consider two conjunctive queries q_1, q_2 on P_{FL}. Containment $q_1 \subseteq_{\Sigma_{FL}} q_2$ of q_1 in q_2 can be decided by a nondeterministic algorithm running in time polynomial in $|q_1|$ and $|q_2|$.

Proof (sketch). We prove the result by providing a nondeterministic algorithm that checks whether $q_1 \subseteq_{\Sigma_{FL}} q_2$ and runs in time polynomial in the input. First, we calculate level 0 of $chase_{\Sigma_{FL}}(q_1)$, i.e. $chase_{\Sigma^-_{FL}}(q_1)$; this is done in time polynomial in $|q_1|$ and independently of q_2. What we need to do now it to check the existence of a homomorphism μ from $body(q_2)$ to $chase_{\Sigma_{FL}}(q_1)$ that sends $head(q_2)$ to $head(chase_{\Sigma_{FL}}(q_1))$; if such homomorphism exists, we can apply Lemma 11 and combine our homomorphism μ with the homomorphism λ from $\mu(body(q_2))$ to $chase_{\Sigma^-_{FL}}(q_1)$; thus we deduce the existence of a homomorphism $\mu \circ \lambda$ from $body(q_2)$ to the first $\delta \cdot |q_2|$ levels of the chase, where $\delta = 2 \cdot |q_1|$. In order to check the existence of such a homomorphism, the algorithm nondeterministically guesses $|q_2|$ conjuncts in the first segment of the chase, i.e. up to level $|q_2| \cdot \delta$. This is done again in polynomial time since, for each conjunct to be guessed, the algorithm needs to guess a primary path from level 0 to some conjunct at depth less or equal than $|q_2| \cdot \delta$. Due to the locality of the application of the rules Σ_{FL} (Lemma 5), this can be done by retaining only level 0 and two levels of the path that have been guessed at a certain point. This needs to be done in parallel for all $|q_2|$ conjuncts to be guessed, because the guessed conjuncts may have symbols in common, and therefore it is not sufficient to guess each one of them separately. We do not detail the technique used for such a guess. Finally, we guess a homomorphism from $body(q_2)$ to the guessed conjuncts, that sends $head(q_2)$ to $head(chase_{\Sigma_{FL}}(q_1))$. If it exists, then $q_1 \subseteq_{\Sigma_{FL}} q_2$; otherwise the containment does not hold. □

5. CONCLUSION

In recent years, F-logic [19] has become a popular tool for building ontologies, information integration, and semantic Web services and a number of systems—both academic and commercial—have become available [14, 13, 28, 26]. In this paper, we considered the problem of query containment for conjunctive meta-queries over F-logic knowledge bases and have shown that the problem is decidable and is in NP. This important class of queries has not been covered by the known results on query containment and, we believe, a solution to this problem will open the door to new F-logic based applications in the areas of ontology modeling, information integration, and semantic Web services.

For future work, we plan to obtain a tight lower bound for the complexity results. We will also investigate extending the decidability results to more expressive query languages. These possible extensions include inheritance, negation, and aggregates. Another area where further research might be fruitful is finding a general class of queries, including the F-logic queries discussed here, for which our proof techniques still apply. Finding such a class would broaden the practical impact of our work.

Acknowledgments. The work of Michael Kifer was supported in part by NSF grants CCR-0311512, IIS-0534419, and by U.S. Army Medical Research Institute under a subcontract through BNL. Andrea Calì's work was supported by the EU FET Project IST-2005-7603 TONES (Thinking ONtologiES).

6. REFERENCES

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[2] A. Aho, Y. Sagiv, and J. D. Ullman. Equivalence of relational expressions. *SIAM Journal of Computing*, 8(2):218–246, 1979.

[3] J. Angele and G. Lausen. Ontologies in F-logic. In S. Staab and R. Studer, editors, *Handbook on Ontologies in Information Systems*, pages 29–50. Springer Verlag, Berlin, Germany, 2004.

[4] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider. *The Description Logic Handbook*. Cambridge Univ. Press, Cambridge, UK, 2003.

[5] S. Battle, A. Bernstein, H. Boley, B. Grosof, M. Gruninger, R. Hull, M. Kifer, D. Martin, S. McIlraith, D. McGuinness, J. Su, and S. Tabet. Semantic Web Services Language (SWSL), September 2005. W3C member submission.

[6] A. Calì. Containment of conjunctive queries over conceptual schemata. In *The 11th International Conference on Database Systems for Advanced Applications (DASFAA 2006)*, pages 628–643, April 2006.

[7] D. Calvanese, G. De Giacomo, and M. Lenzerini. On the decidability of query containment under constraints. In *ACM Symposium on Principles of Database Systems*, pages 149–158, 1998.

[8] D. Calvanese, G. De Giacomo, and M. Lenzerini. Description logics for information integration. In A. Kakas and F. Sadri, editors, *Computational Logic: Logic Programming and Beyond, Essays in Honour of Robert A. Kowalski*, volume 2408 of *Lecture Notes in Computer Science*, pages 41–60. Springer, 2002.

[9] A. K. Chandra and P. M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *ACM Symposium on Theory of Computing*, pages 77–90, 1977.

[10] P. Chen. The entity-relationship model - toward a unified view of data. *ACM Transactions on Database Systems*, 1(1):9–36, 1976.

[11] J. de Bruijn. The WSML specification, February 2005.

[12] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data exchange: Semantics and query answering. In *International Conference on Database Theory (ICDT)*, pages 207–224, 2003.

[13] FLORA-2. The FLORA-2 Web site. http://flora.sourceforge.net.

[14] FLORID. The FLORID system. http://www.informatik.uni-freiburg.de/∼dbis/florid/.

[15] J. Frohn, R. Himmerder, G. Lausen, W. May, and C. Schlepphorst. Managing semistructured data with FLORID: A deductive object-oriented perspective. *Information Systems*, 23(8):589–613, 1998.

[16] D. Johnson and A. Klug. Testing containment of conjunctive queries under functional and inclusion dependencies. *Journal of Computer and System Sciences*, 28:167–189, 1984.

[17] M. Kifer. Rules and ontologies in f-logic. In *Reasoning Web*, number 3564 in Lecture Notes in Computer Science, pages 22–34, July 2005.

[18] M. Kifer and G. Lausen. F-Logic: A higher-order language for reasoning about objects, inheritance and schema. In *ACM SIGMOD Conference on Management of Data*, pages 134–146, New York, 1989. ACM.

[19] M. Kifer, G. Lausen, and J. Wu. Logical foundations of object-oriented and frame-based languages. *Journal of ACM*, 42:741–843, July 1995.

[20] O. Lasilla and R. S. (editors). Resource description framework (RDF) model and syntax specification. Technical report, W3C, February 1999. http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/.

[21] A. Lefebvre, P. Bernus, and R. Topor. Query transformation for accessing heterogeneous databases. In *Proceedings of the JICSLP-92 Workshop on Deductive Databases*, November 1992.

[22] A. Levy and D. Suciu. Deciding containment for queries with complex objects (extended abstract). In *ACM Symposium on Principles of Database Systems*, pages 20–31, 1997.

[23] L. Li and I. Horrocks. A software framework for matchmaking based on semantic web technology. In *Twelfth International World Wide Web Conference (WWW 2003)*, 2003.

[24] D. Maier, A. O. Mendelzon, and Y. Sagiv. Testing implications of data dependencies. *ACM Transactions on Database Systems*, 4(4):455–469, 1979.

[25] T. Millstein, A. Levy, and M. Friedman. Query containment for data integration systems. In *ACM Symposium on Principles of Database Systems*, pages 67–75, New York, NY, USA, 2000. ACM Press.

[26] Ontoprise, GmbH. Ontobroker. http://www.ontoprise.com/.

[27] E. Prud'hommeaux and A. Seaborne. SPARQL query language for RDF. http://www.w3.org/TR/rdf-sparql-query/, 2005.

[28] M. Sintek and S. Decker. TRIPLE – A query, inference, and transformation language for the Semantic Web. In *International Semantic Web Conference (ISWC)*, June 2002.

[29] G. Yang, M. Kifer, and C. Zhao. FLORA-2: A rule-based knowledge representation and inference infrastructure for the Semantic Web. In *International Conference on Ontologies, Databases and Applications of Semantics (ODBASE-2003)*, November 2003.