Answering Queries using Views: a KRDB Perspective for the Semantic Web

FRANÇOIS GOASDOUÉ
France Télécom R&D and Université Paris Sud
and
MARIE-CHRISTINE ROUSSET
Université Paris Sud

In this paper, we investigate a first step towards the long-term vision of the Semantic Web by studying the problem of answering queries posed through a mediated ontology to multiple information sources whose content is described as *views* over the ontology relations. The contributions of this paper are twofold. We first offer a uniform logical setting which allows us to encompass and to relate the existing work on answering and rewriting queries using views. In particular, we make clearer the connection between the problem of rewriting queries using views and the problem of answering queries using extensions of views. Then we focus on an instance of the problem of rewriting *conjunctive queries* using views through an ontology expressed in a description logic, for which we exhibit a complete algorithm.

Categories and Subject Descriptors: H.1.1 [**Systems and Information Theory**]: General systems theory; Information theory; I.2.4 [**Knowledge Representation Formalisms and Methods**]: Representation languages

General Terms: Theory, Algorithms

Additional Key Words and Phrases: Information integration, knowledge representation, semantic web

1. INTRODUCTION

The Semantic Web ([Berners-Lee et al. 2001]) envisions a worldwide distributed architecture where highly distributed and possibly heterogeneous data or computational resources will easily interoperate to coordinate complex tasks such as answering queries or distributed computing. Semantic marking up of Web resources using ontologies is expected to provide the necessary glue for making this vision work.

In this paper, we investigate a first step towards this long-term vision of the

François Goasdoué, Artificial intelligence and inference systems, LRI, Bâtiment 490, Université Paris Sud, 91405 Orsay Cedex, France. This work has been performed while the author was seconded to Information management and retrieval, France Télécom R&D, 2 avenue Pierre Marzin, 22307 Lannion Cedex, France.
Marie-Christine Rousset, Artificial intelligence and inference systems, LRI, Bâtiment 490, Université Paris Sud, 91405 Orsay Cedex, France.

Semantic Web by studying the problem of answering queries posed through a mediated ontology to multiple information sources. In this information integration vision, users do not pose queries directly to the (possibly remote) sources in which data are stored but to the set of virtual relations that have been designed to model an ontology of a domain of interest. The contents of the sources that are relevant to that domain are described as *views* over the ontology relations. In such a setting, the problem of answering users queries becomes a problem of *answering queries using view extensions*. This problem is closely related to the problem of *query rewriting using views*, which consists in reformulating a user query into an (equivalent or maximally contained) query whose definition refers only to the views.

The instances of the answering or rewriting problems using views that have been considered vary depending on the languages used for describing the queries, the views, and the rewritings. The setting that has been extensively studied concerns the pure relational setting where queries, views and rewritings are expressed as conjunctive or Datalog queries (see [Halevy 2001] for a survey). Recently, instances of the answering or rewriting problems have been studied, in which the queries, views or rewritings may be expressed using description logics [Beeri et al. 1997; Rousset 1999; Baader et al. 2000; Calvanese et al. 2000]. Description logics are a family of logics that have been developed for modeling complex hierarchical structures, and can also be viewed as query languages with an interesting trade-off between complexity and expressive power (see [Nardi et al. 1995] for a survey). A description logic deals with unary predicates (referred to as *concepts*), representing sets of objects, and binary predicates (referred to as *roles*), representing properties holding on pairs of objects. A description logic is defined according to the *constructors* that are allowed to define complex *concept descriptions* starting from a set of atomic concepts and roles. The different constructors express varied restrictions on concepts and roles. For example, the *value restriction* constructor allows to build concept descriptions of the form $\forall r.C$: they define the set of individuals that are related through the role r to individuals of the concept C only. Varying the set of allowed constructors enables to tailor a description logic with a desired tradeoff between expressiveness and computational complexity.

In this paper, we consider the problems of answering and rewriting queries using views in the uniform logical setting of CARIN [Levy and Rousset 1998a], which combines description logics and Datalog. The added expressive power of both formalisms is especially useful for applications in information integration. On the one hand, description logics are the leading candidates for the future ontology Web language (OWL) recommended by the W3C Web Ontology Working Group's charter (see www.w3.org/2001/sw/WebOnt/charter). On the other hand, Datalog rules correspond to the most essential relational database queries – SelectProjectJoin queries. The Information Manifold [Levy et al. 1996] is the first information integration system that pointed out the usefulness of extending Datalog with some features of description logics for building mediators. PICSEL [Goasdoué et al. 2000] benefits from the use of the full expressive power of CARIN-$\mathcal{ALN}$, and has shown the practical interest of such an hybrid formalism for modeling in a natural way and with a reasonable expressiveness a real application of information integration related to the domain of tourism. The use of the $\mathcal{ALN}$ description logic added to

Datalog restricted forms of negation, value restriction and cardinality constraints. The combination of those $\mathcal{ALN}$ constructors provides an expressive power allowing for example the definitions of "hotels located in Lannion, France, with at most 12 rooms having air conditioning but no minibar.".

The contributions of this paper are twofold. We first offer a uniform logical setting which allows us to encompass and to relate the existing work on answering and rewriting queries using views. In particular, we make clearer the connection between the problem of rewriting queries using views, and the problem of answering queries using extensions of views. Then we focus on an instance of the problem of rewriting *conjunctive queries* using views through an ontology expressed in a description logic. The simplicity of the description logic that we consider is compensated for by the possibility of expressing inclusion and disjointness axioms on atomic concepts. This choice has resulted from an analysis of the description logics constructors that have been actually used by the designers of the mediated ontology on the tourism domain [Reynaud and Safar 2002] in the PICSEL information integration system. While the expressive power of $\mathcal{ALN}$ was available, the atomic negation and the number restrictions (except $(\geq\ 1\ r)$) have hardly been used, while the value restriction constructor $\forall r.C$ was extensively used as a way of expressing typing constraints on roles that they consider as necessarily associated with some concepts. For instance, flights whose airline is American can be defined by the $\mathcal{ALN}$ concept description: $Flight \sqcap \forall Airline.American$. It is important to notice that most of the knowledge engineers using description logics as a modeling tool employ the $\forall$ constructor in a rather specific way and with a semantics in mind which is not exactly its actual logical semantics: they usually use $\forall r.C$ within a conjunction $D \sqcap \forall r.C$ with the intended meaning that filler(s) of the role r exist for any instance of the concept D, and that for all those instances, the filler(s) of the role r are instances of the concept C. This has also been noticed in the ontology design done in the MKBEEM project [Mkbeem]. In our proposal for a language for modeling ontologies, we account for this intuitive semantics by replacing the $\forall$ constructor by a constructor $\forall+$ which forces the combined use of value restriction on roles with the unqualified existential restriction on those roles. More generally, we think that the ontologies that will be used in the setting of the Semantic Web, will be built by communities of users or practioners who are not knowledge engineers or logicians. Therefore, they must be provided with appropriate modeling tools associated with query processors. We think that our proposal allowing answering conjunctive queries over ontologies defined as hierarchies of classes described by their names and a set of associated typed roles is a basic building block useful for the future infrastructure of the Semantic Web.

The paper is organized as follows. In section 2, we present an illustrative example of information integration through an ontology. In section 3, we define precisely the formal background of our study: the languages that we consider in this paper to define queries, views and rewritings are (possibly different) languages of CARIN. In section 4, we formally define the general rewriting problem using views that we consider, we relate it to the problem of answering queries using extensions of views, and we show how it encompasses instances that have been previously studied in the literature. In section 5, we define an instance which is adapted to the setting of the Semantic Web: we provide a rewriting algorithm for that instance, we show its soundness and completeness, and we establish its complexity. Finally, in the last section, we conclude with related work and discussion.

2. ILLUSTRATIVE EXAMPLE

Let us consider an ontology describing touristic products using concepts and properties. This ontology is organized according to geographical locations and different kinds of places related to those geogaphical locations by the property LocatedIn. Suppose that we have the following concepts: Hotel, Apartment, HousingPlace, Place, SportResort, SeasideResort, GeographicalLocation, Caraibbean, Guadeloupe, Martinique. Figure 2 illustrates the way they are organized within a hierarchy. Note that this hierarchy can be manually or automatically constructed, depending on the representation formalism that is used to define the ontology concepts.

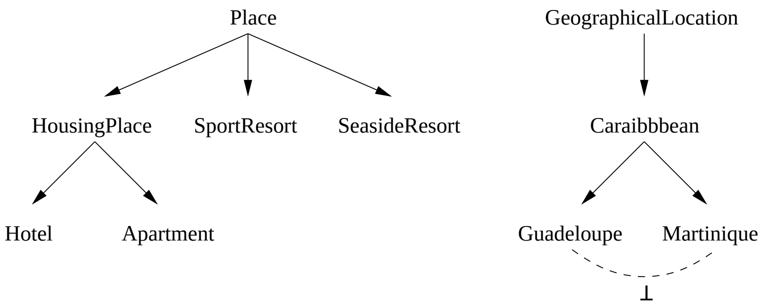

Fig. 1. Hierarchy

Suppose that it is stated in addition that Martinique and Guadeloupe are disjoint concepts (while SportResort and SeasideResort are not).

In an information integration setting, such an ontology plays a central role by serving as a pivot vocabulary to express both the contents of available data sources related to tourism, and the users queries. The ontology serves as a query interface between end-users and a collection of pre-existing data sources. The goal is to give users the illusion that they interrogate a centralized and homogeneous information system, instead of a collection of distributed and possibly heterogeneous data sources.

For instance, suppose that three data sources are available, whose content can be described in function of the concepts and properties of the ontology as follows:

—S_1 stores instances of the concept Hotel that are related by the property LocatedIn to locations situated in Martinique only.

—S_2 stores instances of the concept Apartment that are related by the property LocatedIn to locations situated in Guadeloupe only.

—S_3 stores instances of the concept SportResort that are related by the property LocatedIn to locations situated in Martinique only.

—S_4 stores instances of the binary property LocatedIn, that is pairs of (x,y) such that the place x is located in the geographical location y.

Formally, the source contents are expressed as *views* over the ontology concepts and properties, as it will be explained in the following section. Views are named queries defined over a given set of predicates using a given language.

Users will use the concepts and properties of the domain ontology to express their queries. For instance, consider a query asking for *housing places and sport resorts located at the same place in a Caraibbean island.* It can be formally expressed as the following *conjunctive query*:

$\mathsf{q}(X,U)$:- $\mathsf{HousingPlace}(X) \wedge \mathsf{LocatedIn}(X,Y) \wedge \mathsf{Caraibbean}(Y) \wedge \mathsf{SportResort}(U) \wedge \mathsf{LocatedIn}(U,Y)$

The query processing involves *rewriting* the users query into queries expressed in terms of views, which correspond to query plans that are directly executable against the data sources.

In our example, we would obtain two rewritings for the query q using the views S_1, S_2, S_3, and S_4.

$\mathsf{q}_{r_1}(X,U)$:- $\mathsf{S}_1(X) \wedge \mathsf{S}_3(U) \wedge \mathsf{S}_4(X,Y) \wedge \mathsf{S}_4(U,Y)$.

$\mathsf{q}_{r_2}(X,U)$:- $\mathsf{S}_2(X) \wedge \mathsf{S}_3(U) \wedge \mathsf{S}_4(X,Y) \wedge \mathsf{S}_4(U,Y)$.

The first rewriting expresses a query plan that consists of querying the sources S_1 and S_3 to get candidate instances of the distinguished variables X and U, and then of querying the source S_4 with those instances, to perform a join on the second attributes of the answers that are obtained.

The second rewriting is actually inconsistent and will therefore be discarded as a query plan. Its inconsistency comes from the disjointness between the concepts Guadeloupe and Martinique and the definitions of the views S_2 and S_3, from which Guadeloupe(Y) and Martinique(Y) can be infered.

The execution of the query plans obtained as rewritings provides answers that are correct w.r.t the original query, but not necessarily complete. The question of knowing whether we can obtain *all* the answers by rewriting is a core issue in information integration, which is investigated in Section 4.

3. FORMAL BACKGROUND

In this paper, we consider the problem of rewriting queries using views when the queries, the views and the rewritings are expressed in languages of CARIN.

3.1 CARIN

CARIN is a family of hybrid logical languages in which we can define two kinds of logical sentences over base predicates:

—Concept descriptions using description logic constructors,

—Rules whose bodies may contain conjuncts that are built using concept descriptions.

We now define precisely the syntax and the semantics of those two kinds of sentences. They are built over a set $\mathcal{P}$ of *base predicates* including a set N_R of binary predicates and a set N_P of unary predicates. An element r of N_R will be called a role, and an element of N_P will be called an atomic concept.

3.1.1 *Concept descriptions.* Concept descriptions over $N_R \cup N_P$ are inductively defined using description logics constructors as follows:

—Any atomic concept $A \in N_P$ is a concept description.
—$\top$ is a concept description (called the top or universal concept).
—$\perp$ is a concept description (called the bottom or empty concept).
—If A is an atomic concept, then $\neg A$ is a concept description (negation). Restricting the negation to atomic concepts is called *atomic negation.*
—If C and D are concept descriptions, then $C \sqcap D$ is a concept description (called the conjunction of C and D).
—If C is a concept description and r is a role, then $\forall r.C$ and $\exists r.C$ are concept descriptions (respectively called value restriction and existential restriction). Restricting the use of existential restriction to the $\top$ concept ($\exists r.\top$) is called *unqualified existential restriction.*
—If n is an integer and r is a role, then $(\geq n\, r)$ and $(\leq n\, r)$ are concept descriptions (called number restrictions).

Note that other constructors can be used to build concept descriptions. We only mention the set of constructors that are used in this article. It corresponds to the $\mathcal{ALEN}$ description logic.

The semantics of concept descriptions is defined in terms of *interpretations.* An interpretation I is a pair $(\Delta^I, .^I)$ where Δ^I is a non-empty set of individuals called the domain of interpretation of I, and $.^I$ is an interpretation function, which assigns a subset of Δ^I to every concept description. An interpretation function maps each atomic concept $A \in N_P$ to a subset A^I of Δ^I and each role $r \in N_R$ to a subset r^I of $\Delta^I \times \Delta^I$. The interpretation of an an arbitrary concept description is inductively defined as follows:

—$\top^I = \Delta^I$, $\perp^I = \emptyset$
—$(\neg A)^I = \Delta^I \setminus A^I$
—$(C \sqcap D)^I = C^I \cap D^I$
—$(\forall r.C)^I = \{d \in \Delta^I \mid \forall e \in \Delta^I((d,e) \in r^I \rightarrow e \in C^I)\}$
—$(\exists r.C)^I = \{d \in \Delta^I \mid \exists e \in \Delta^I s.t(d,e) \in r^I\}$
—$(\geq n\, r)^I = \{d \in \Delta^I \mid \#\{e \mid (d,e) \in r^I\} \geq n\}$
—$(\leq n\, r)^I = \{d \in \Delta^I \mid \#\{e \mid (d,e) \in r^I\} \leq n\}$

An interpretation I is a model of a concept description C iff C^I is not empty.

Definition 3.1 *Subsumption and equivalence of concept descriptions.* Let C and D be concept descriptions.

—C is *subsumed* by D (denoted $C \preceq D$) iff for every interpretation I, $C^I \subseteq D^I$.
—C is *equivalent* to D(denoted $C \equiv D$) iff for every interpretation I, $C^I = D^I$.

In this paper, we mention different description logics:

—$\mathcal{ALN}$, in which concept descriptions are restricted to be top, bottom, conjunctions, atomic negations, value restrictions, and number restrictions.
—$\mathcal{FLE}$, in which concept descriptions are restricted to be the top, conjunctions, existential restrictions and value restrictions.

—$\mathcal{ALE}$, which, in addition of the $\mathcal{FLE}$ concept descriptions, allows bottom and atomic negations.

3.1.2 CARIN *rules.* A CARIN rule is a logical sentence of the form:

$$q(\vec{X}) : - \bigwedge_{i=1}^{n} p_i(\overrightarrow{X_i \cup Y_i})^1.$$

$q(\vec{X})$ is the *head* of the rule, and q is called the *head predicate* of the rule: it cannot be a concept description or a role, and can be of any arity. $\bigwedge_{i=1}^{n} p_i(\overrightarrow{X_i \cup Y_i})$ is the *body* of the rule, and is made of the *conjuncts* $p_i(\overrightarrow{X_i \cup Y_i})$, where $p_1, \ldots, p_n$ are either roles, or concept descriptions, or head predicates of some other rules. $X = \bigcup_{i=1}^{n} X_i$ are variables that are called the *distinguished* variables of the rule. The other variables $Y = \bigcup_{i=1}^{n} Y_i$ are called *existential* variables of the rule.

Definition 3.2 *Concept-atom and role-atom.* A *concept-atom* is an atom $p(U)$ such that p is a concept description. A *role-atom* is an atom $r(U_1, U_2)$ such that r is a role name.

EXAMPLE 1. Consider the following rule:

$$\begin{aligned} \mathsf{q}(X_1, X_2) : - \ & \mathsf{Flight}(X_1) \wedge \mathsf{Flight}(X_2) \wedge \mathsf{Airline}(X_1, U) \wedge \mathsf{Airline}(X_2, U) \\ & \wedge (\forall \mathsf{Stop.AmCity})(X_1) \wedge \mathsf{Stop}(X_2, S) \wedge \mathsf{EurCity}(S). \end{aligned}$$

Its distinguished variables are X_1, X_2. The rule defines the relation q of arity 2, as being the set of pairs X_1, X_2 that are flights depending on the same airline and such that the flight X_1 only stops (if it stops) in American cities, and the flight X_2 has a stop in an European city. □

Given a set of CARIN rules, a head predicate p is said to *depend* on a predicate q if q appears in the body of a rule whose head predicate is either p or a predicate which p depends on. A set of rules is said to be *recursive* if there is a cycle in the dependency relation among head predicates.

Rules can be seen as a way of defining predicates in the following sense.

Definition 3.3 *Definition of a predicate by rules.* Let $\mathcal{R}$ be a set of CARIN rules, and let p be a predicate. The *definition* of p, denoted by $def(p)$, is the subset of rules of $\mathcal{R}$ whose head predicate is p or a predicate which p depends on. If the definition of p is the whole set $\mathcal{R}$, we say that $\mathcal{R}$ defines the predicate p.

In the same way that an interpretation of the atomic concepts and roles determines the interpretation of complex descriptions, the interpretation of base predicates uniquely determines the interpretation of the predicates defined by a set of non recursive rules.

Definition 3.4 *Semantics of a predicate defined by non recursive rules.* Let q be a predicate and let $def(q)$ be its rule definition. Given an interpretation I, the interpretation q^I of q in I is defined inductively as follows:

[1]For any set S, we denote by $\vec{S}$ a vector of its elements. Conversely, for any vector $\vec{V}$, we denote by V the set made of its elements.

—If the arity of q is not null:
$\vec{o} \in q^I$ iff $def(q)$ contains a rule $q(\vec{X}) : - \bigwedge_{i=1}^{n} p_i(\overrightarrow{X_i \cup Y_i})$ for which there exists a mapping α from the variables of the rule to the domain Δ^I such that: $\alpha(\vec{X}) = \vec{o}$ and $\alpha(\overrightarrow{X_i \cup Y_i}) \in p_i^I$ for every atom of the body of the rule.

—If the arity of q is 0 (i.e., q is a boolean query):
q^I is true iff $def(q)$ contains a rule $q(\vec{X}) : - \bigwedge_{i=1}^{n} p_i(\overrightarrow{X_i \cup Y_i})$ for which there exists a mapping α from the variables of the rule to the domain Δ^I such that: $\alpha(\overrightarrow{X_i \cup Y_i}) \in p_i^I$ for every atom of the body of the rule.

—An interpretation I is a model of q iff q^I is not empty.

The following definition extends to predicates defined by a set of non recursive rules the notions of subsumption and equivalence previously defined for concept descriptions.

Definition 3.5 *Subsumption and equivalence of predicates defined by rules.* Let q_1 and q_2 be two predicates defined by non recursive CARIN rules.

—q_1 is *subsumed* by q_2 iff for every interpretation I, $q_1^I \subseteq q_2^I$.

—q_1 is *equivalent* to q_2 iff for every interpretation I, $q_1^I = q_2^I$.

3.1.3 *Languages encompassed by* CARIN. Three kinds of languages can be distinguished, depending on which restrictions we impose on the sentences that are allowed

3.1.3.1 *Relational languages* . A relational language does not permit sentences corresponding to concept descriptions other than atomic concepts. Allowed sentences are Datalog rules. For instance, the query language considered in [Calvanese et al. 2000a] is a relational language since regular path queries over graphs can be expressed as a set of (recursive) rules built over a set of base binary predicates.

3.1.3.2 *Terminological languages* . A terminological language is restricted to sentences that are only concept descriptions or roles. No rule is allowed in such a language. $\mathcal{ALN}$, $\mathcal{FLE}$ or $\mathcal{ALE}$ are terminological languages of CARIN. They are considered in [Baader et al. 2000] as the basis for defining the problem of rewriting concepts using terminologies.

3.1.3.3 *Hybrid languages* . A hybrid language of CARIN imposes some restrictions on the sentences but allows concept descriptions to appear in bodies of rules. Such languages are useful to define conjunctive queries over concept descriptions and roles. We denote by CARIN-$\mathcal{L}$ the hybrid CARIN language for which the description logic is $\mathcal{L}$. For instance, the queries and views considered in [Beeri et al. 1997] are defined by non recursive CARIN-$\mathcal{ALN}$ rules. In that setting, CARIN is restricted in two ways: a first restriction concerns the description logic component, since only concept descriptions of $\mathcal{ALN}$ are allowed; a second restriction concerns the rule component since only non recursive rules are allowed.

3.2 Ontologies in CARIN

Two components can cohabit in an ontology $\mathcal{O}$ defined in CARIN over a given set of base predicates $\mathcal{P}$: a *terminological component* and a *deductive component*.

The sentences in the terminological component $\mathcal{T}_{\mathcal{O}}$ are either *concept definitions,* or *inclusion statements.* A concept definition is a statement of the form $A := D$, where A is a concept name and D is a concept description. We assume that a concept name appears on the left hand side of at most one concept definition. An inclusion statement is of the form $C \sqsubseteq D$, where C and D are concept descriptions. Intuitively, a concept definition associates a definition with a name of a concept. An inclusion states that every instance of the concept C must be an instance of D.

A concept name A is said to *depend* on a concept name B if B appears in the concept definition of A. A set of concept definitions is said to be *cyclic* if there is a cycle in the dependency relation. When the terminology contains only concept definitions and has no cycles we can *unfold* the terminology by iteratively substituting every concept name with its definition. As a result, we obtain a set of concept definitions where all the concepts that appear in the right hand sides are concepts of $\mathcal{P}$.

An interpretation I is a model of $\mathcal{T}_{\mathcal{O}}$ if $C^I \subseteq D^I$ for every inclusion $C \sqsubseteq D$ in the terminology and $A^I = D^I$ for every concept definition $A := D$. We say that C *is subsumed by* D w.r.t. $\mathcal{T}_{\mathcal{O}}$ ($C \preceq_{\mathcal{T}_{\mathcal{O}}} D$) if $C^I \subseteq D^I$ in every model I of $\mathcal{T}_{\mathcal{O}}$.

Given a terminology $\mathcal{T}_{\mathcal{O}}$, we call its vocabulary the set of predicates composed by the base concepts and roles appearing in $\mathcal{T}_{\mathcal{O}}$, and by the concepts that are defined in $\mathcal{T}_{\mathcal{O}}$.

EXAMPLE 2. Consider the following terminology:
$\mathsf{Caraibbean} \sqsubseteq \mathsf{GeographicalLocation}$
$\mathsf{Martinique} \sqsubseteq \mathsf{Caraibbean}$
$\mathsf{Guadeloupe} \sqsubseteq \mathsf{Caraibbean}$
$\mathsf{Martinique} \sqcap \mathsf{Guadeloupe} \sqsubseteq \bot$
$\mathsf{PrivateBuilding} \sqcap \mathsf{CollectiveBuilding} \sqsubseteq \bot$
$\mathsf{Hotel} := \mathsf{HousingPlace} \sqcap (\geq\ 5\ \mathsf{AssRoom}) \sqcap\ (\forall \mathsf{AssBuilding}.\mathsf{CollectiveBuilding})$
$\mathsf{Apartment} := \mathsf{HousingPlace} \sqcap (\geq\ 1\ \mathsf{AssRoom}) \sqcap\ (\forall \mathsf{AssBuilding}.\mathsf{PrivateBuilding})$
The vocabulary defined by that terminology is: {GeographicalLocation,Caraibbean, Martinique, Guadeloupe, PrivateBuilding, CollectiveBuilding, Hotel, HousingPlace, AssRoom, AssBuilding, Apartment}. □

The deductive component $\mathcal{R}_{\mathcal{O}}$ is a set of CARIN rules defined over a set of base predicates and possibly over the vocabulary of a terminology. The vocabulary defined by $\mathcal{R}_{\mathcal{O}}$ is the set of predicates appearing in the left hand side or the right hand side of rules of $\mathcal{R}_{\mathcal{O}}$.

Given an ontology $\mathcal{O} = \langle \mathcal{T}_{\mathcal{O}}, \mathcal{R}_{\mathcal{O}} \rangle$, its vocabulary $\mathcal{P}_{\mathcal{O}}$ is the union of the vocabularies of its two components.

A model of $\mathcal{O}$ is an interpretation I which is a model of the two components $\mathcal{T}_{\mathcal{O}}$ and $\mathcal{R}_{\mathcal{O}}$.

3.3 Queries and views in CARIN

Queries and views are defined over a given set of predicates, using a certain (query or view) language. In the setting of this paper, we consider queries and views defined:

—using a language of CARIN,

—over the vocabulary $\mathcal{P}_\mathcal{O}$ of an ontology $\mathcal{O}$, consisting of base predicates and possibly of predicates defined over those base predicates within a terminology or a rule base. Note that one of its (terminological or deductive) components, or the ontology itself, can be empty. In the latter case, $\mathcal{P}_\mathcal{O}$ is reduced to a set of given base predicates, which is the classical setting for defining relational queries and views.

Definition 3.6 *Queries.* Let $\mathcal{L}$ be a language of CARIN. A *query* expressed in $\mathcal{L}$ over the vocabulary $\mathcal{P}_\mathcal{O}$ of an ontology is a predicate q whose definition, $def(q)$, is:

—either a $\mathcal{L}$ concept description over $\mathcal{P}_\mathcal{O}$, or a role, if $\mathcal{L}$ is a terminological language,

—or a set of $\mathcal{L}$ rules over $\mathcal{P}_\mathcal{O}$, if $\mathcal{L}$ is a relational or hybrid language of CARIN.

The *arity* of a query is the arity of the predicate defining it.

Definition 3.7 *Conjunctive queries.* A query whose definition contains a single rule is called a *conjunctive query.*

—$\mathcal{CQ}$-$\mathcal{L}$ will denote the query language restricted to conjunctive queries expressed within CARIN-$\mathcal{L}$ where $\mathcal{L}$ is a description logic.

—$\mathcal{CQ}$ will denote the relational language of any $\mathcal{CQ}$-$\mathcal{L}$ forbidding any concept description other than atomic concepts to appear in conjunctive queries.

—$\mathcal{CQ}^{\neq}$ will denote the relational query language allowing conjunctive queries possibly with inequalities.

We identify a conjunctive query with the head of the rule defining it, and its definition with the body of that rule. For example, the rule:

$$\begin{aligned}\mathsf{q}(X_1, X_2) :- \ &\mathsf{Flight}(X_1) \wedge \mathsf{Flight}(X_2) \wedge \mathsf{Airline}(X_1, U) \wedge \mathsf{Airline}(X_2, U)\\ &\wedge (\forall \mathsf{Stop.AmCity})(X_1) \wedge \mathsf{Stop}(X_2, S) \wedge \mathsf{EurCity}(S)\end{aligned}$$

in Example 1 defines a $\mathcal{CQ}$-$\mathcal{FLE}$ query $q(X_1, X_2)$ of arity 2 whose definition is:

$$\begin{aligned}def(\mathsf{q}(X_1, X_2)) = \ &\mathsf{Flight}(X_1) \wedge \mathsf{Flight}(X_2) \wedge \mathsf{Airline}(X_1, U) \wedge \mathsf{Airline}(X_2, U)\\ &\wedge (\forall \mathsf{Stop.AmCity})(X_1) \wedge \mathsf{Stop}(X_2, S) \wedge \mathsf{EurCity}(S)\end{aligned}$$

The set of answers of a query is defined relatively to a database storing a set of ground facts that are *extensions* of some predicates. Those predicates are called *extensional predicates.*

Definition 3.8 *Extension of predicates.* Let p be an extensional predicate. Its *extension*, denoted $ext(p)$, is a set of facts of the form $p(\vec{a})$ where $\vec{a}$ is a tuple of constants. For a set $\mathcal{E}$ of extensional predicates, we denote $ext(\mathcal{E})$ the union of the extensions of the predicates in $\mathcal{E}$.
An interpretation I is extended to a set of extensions by assigning an individual a^I of Δ^I to every constant appearing in the extensions. I is a model of $ext(\mathcal{E})$ if for every fact $p(\vec{a})$ of $ext(\mathcal{E})$, $\vec{a}^I \in p^I$.

Note that with this definition, the extensions are assumed to be sound but not necessarily complete: for a model I of $ext(\mathcal{E})$, there may exist an extensional predicate p for which there exists a tuple $\vec{t} \in p^I$ and there does not exist any $p(\vec{a})$ of $ext(\mathcal{E})$ such that $\vec{a}^I = \vec{t}$.

In the standard setting of relational databases, the extensional predicates are those which are defined in the schema, which, in our setting, correspond to the base predicates or, if the schema is defined by a domain ontology, to the predicates of the ontology vocabulary. However, in the general case, the extensional predicates may be predicates of queries called *views*, i.e., queries whose answers have been precomputed (materialized views) or are known to be available from some data sources (virtual views).

Definition 3.9 *Views.* A *view* v is a query which has a definition $def(v)$ and possibly an extension $ext(v)$. Let $\mathcal{V}$ be a set of views. An interpretation I is a *model* of $\mathcal{V}$ iff I is a model of all the view predicates in $\mathcal{V}$.

We can now define formally the answer set of a query given a set of extensions. We must distinguish whether those extensions come from base predicates or from view predicates. Definition 3.10 is the standard semantic definition of the answer set of a query in relational databases: given a database instance consisting of extensions of base predicates (possibly defined within an ontology), the answers to q are tuples $\vec{a}$ such that the fact $q(\vec{a})$ is logically entailed by the facts in the database instance (and the sentences of the ontology) .

Definition 3.10 *Answer set of a query over extensions of ontology predicates.* Let q be a query of arity n expressed in a language $\mathcal{L}$ of CARIN over the vocabulary $\mathcal{P}_{\mathcal{O}}$ of an ontology $\mathcal{O}$. Let $ext(\mathcal{P}_{\mathcal{O}})$ be the union of their extensions. Let $\mathcal{C}$ be the set of constants appearing in $ext(\mathcal{P}_{\mathcal{O}})$. The *answer set of the query over* $ext(\mathcal{P}_{\mathcal{O}})$ is defined as follows:
$Ans(q, ext(\mathcal{P}_{\mathcal{O}})) = \{\vec{a} \in \mathcal{C}^n \mid \text{for every model } I \text{ of } \mathcal{O} \text{ and } ext(\mathcal{P}_{\mathcal{O}}),\ \vec{a}^I \in q^I\ \}$

When the database instance is unknown and extensions of some *view* predicates are available instead, Definition 3.11 states that the answers to a query q are those tuples $\vec{a}$ such that the fact $q(\vec{a})$ is logically entailed by the facts in the views extensions, the statements in the ontology *and* the view definitions. This conveys that the view extensions together with the view definitions (and the ontology statements) infer (incomplete) information about the unknown database instance.

Definition 3.11 *Answer set of a query over extensions of views.* Let q be a query of arity n expressed in a language $\mathcal{L}_1$ of CARIN over the vocabulary $\mathcal{P}_{\mathcal{O}}$ of an ontology $\mathcal{O}$. Let $\mathcal{V} = \{v_1, \ldots, v_k\}$ be a set of views defined in a language $\mathcal{L}_2$ of CARIN over the same vocabulary. Let $ext(\mathcal{V})$ be the union of the extensions of the views $v_1, \ldots, v_k$. Let $\mathcal{C}$ be the set of constants appearing in $ext(\mathcal{V})$. The *answer set of the query against* $ext(\mathcal{V})$ is defined as follows:
$Ans(q, ext(\mathcal{V})) = \{\vec{a} \in \mathcal{C}^n \mid \text{for every model } I \text{ of } \mathcal{V},\ \mathcal{O} \text{ and of } ext(\mathcal{V}),\ \vec{a}^I \in q^I\}.$

The difference between answering queries using views (Definition 3.11) and answering queries over a database instance (Definition 3.10) is illustrated in the following example.

EXAMPLE 3. Consider the conjunctive query $q(X,Y)$ defined by
$q(X,Y) : -p_1(X,Y) \wedge p_2(Y,X)$,
and the following view definitions
$v_1(X) : -p_1(X,X)$

$v_2(X) : -p_2(X, X)$
$v_3(X) : -p_1(X, Y)$
$v_4(X) : -p_2(Y, X)$
and assume that the views extensions consists of $\{v_1(a), v_2(a), v_3(b), v_4(c)\}$.

According to Definition 3.11, $< a, a >$ is the only answer for the query q given those views. In fact, it is the only certain answer of q over any database instance, which can be inferred from the views extensions and the views definitions. In particular, from the extensions of v_3 and v_4 (i.e., the facts $v_3(b)$ and $v_4(c)$), we only know that there exists at least one fact of the form $p_1(b, Y)$ in the extension of the base predicate p_1, and at least one fact of the form $p_2(Z, c)$ in the extension of the base predicate p_2. However, we cannot conclude whether $< b, c >$ is an answer to q because we do not know whether the constant specified by Y is the same as the constant specified by Z. □

Note that our definition of answering queries using views (Definition 3.11) does not take into account any specific assumption about the completeness of the set of constants appearing in the extensions of views and in the extensions of base predicates. The interested readers are referred to [Calvanese et al. 2000] and [Halevy 2001] in which those assumptions are formalized as *sound, complete and exact view assumptions* and *closed domain versus open domain assumption* respectively, and studied for computing the set of *certain answers* [Abiteboul and Duschka 1998] to queries using views depending on those assumptions. An answer is a certain answer of the query if it is an answer for all possible databases that are consistent with the given extensions of the views. A database $ext(\mathcal{P})$ is consistent with a set $ext(\mathcal{V})$ of extensions of views iff every ground fact $v(\vec{a})$ in a view extension is an answer of the query v over the database $ext(\mathcal{P})$. It is important to note that the extension of a set of views does not entail a unique extension of the base predicates $ext(\mathcal{P})$. It provides only partial information about it.

In order to situate our setting within the general setting of answering queries using views taking into account those assumptions, we just have to say that our definition follows the *open domain assumption* and considers that all the views are *sound*. Under those assumptions, the answer set for a query, as defined in Definition 3.11, is guaranteed to be the set of *certain* answers of the query. Such assumptions are appropriate in the information integration setting: a sound view v corresponds to a data source which is known to produce only (but not necessarily all) data that are logically specified by its associated definition $def(v)$.

The two following definitions of query subsumption (also called query containment), and query equivalence, possibly modulo a set of views, are simple generalizations of Definition 3.1 and Definition 3.5.

Definition 3.12 *Query subsumption (a.k.a. containment) and equivalence.* Let q_1 and q_2 be two queries of same arity defined in two languages of CARIN over the same vocabulary of an ontology $\mathcal{O}$.

—q_1 is *subsumed* by q_2 iff $q_1^I \subseteq q_2^I$, for every model I of $\mathcal{O}$.
—q_1 is *equivalent* to q_2 iff $q_1^I = q_2^I$, for every model I of $\mathcal{O}$.

Definition 3.13 *Query subsumption (a.k.a. containment) and equivalence modulo views.* Let q be a query defined in $\mathcal{L}_1$ over the vocabulary of an ontology $\mathcal{O}$, and $\mathcal{V} =$

$\{v_1, \ldots, v_k\}$ a set of views defined in $\mathcal{L}_2$ over the same vocabulary. Let $q_{\mathcal{V}}$ be a query defined in $\mathcal{L}_3$ such that the base predicates of $def(q_{\mathcal{V}})$ are the view predicates $v_1, \ldots, v_k$.

—$q_{\mathcal{V}}$ is *subsumed by* q *modulo* $\mathcal{V}$ iff $q_{\mathcal{V}}^I \subseteq q^I$, for every model I of $\mathcal{O}$ and $v_1, \ldots, v_k$,
—$q_{\mathcal{V}}$ is *equivalent to* q *modulo* $\mathcal{V}$ iff $q_{\mathcal{V}}^I = q^I$, for every model I of $\mathcal{O}$ and $v_1, \ldots, v_k$.

Note that Definition 3.13 does not involve extensions of views but just their definitions.

4. REWRITING QUERIES USING VIEWS

We start with presenting a general framework for defining *rewritings* of a query in terms of views and for stating the problem of query rewriting using views. We then relate it to the *problem of answering queries using extensions of views*, and we situate within that general framework some instances of the rewriting and the answering problem that have been previously studied.

4.1 A general framework for defining the rewriting problem

Let $\mathcal{L}_1$, $\mathcal{L}_2$ and $\mathcal{L}_3$ be three languages of CARIN (which can be identical or different). Given a query defined in $\mathcal{L}_1$ over a certain vocabulary, and given a set of views defined in $\mathcal{L}_2$ over the same vocabulary, the problem of rewriting a query using views consists of finding queries (called rewritings) that are defined in $\mathcal{L}_3$ using the view predicates as base predicates and that are equivalent to (or maximally contained in) the original query.

Definition 4.1 *Rewriting, maximally contained and equivalent rewriting.* Let q be a query defined in $\mathcal{L}_1$ over a given vocabulary, and $\mathcal{V} = \{v_1, \ldots, v_k\}$ be a set of views defined in $\mathcal{L}_2$ over the same vocabulary.

—An $\mathcal{L}_3$ *rewriting of* q *using* $\mathcal{V}$ is a query $q_{\mathcal{V}}$ defined in $\mathcal{L}_3$ over the base predicates $v_1, \ldots, v_k$ such that $q_{\mathcal{V}}$ is subsumed by q modulo $\mathcal{V}$.
—An $\mathcal{L}_3$ rewriting $q_{\mathcal{V}}$ of q using $\mathcal{V}$ is a *maximally contained rewriting* iff there is no $\mathcal{L}_3$ rewriting $q'_{\mathcal{V}}$ such that $q_{\mathcal{V}}$ is strictly subsumed by $q'_{\mathcal{V}}$ (i.e., not equivalent to $q'_{\mathcal{V}}$).
—An $\mathcal{L}_3$ rewriting $q_{\mathcal{V}}$ of q using $\mathcal{V}$ is an *equivalent rewriting* iff $q_{\mathcal{V}}$ is equivalent to q modulo $\mathcal{V}$.

EXAMPLE 4. Consider the vocabulary composed of the roles Stop, HotelLocation and Airline, and of the atomic concepts EurCity, Hotel, Flight and AmCity. Consider the query q defined by the concept description Flight $\sqcap \exists$Stop.EurCity and the set of views $\mathcal{V} = \{\mathsf{v}_1, \mathsf{v}_2, \mathsf{v}_3, \mathsf{v}_4\}$ with the associated $\mathcal{FLE}$ definitions:

$def(\mathsf{v}_1) = \mathsf{Stop}$
$def(\mathsf{v}_2) = \mathsf{HotelLocation}$
$def(\mathsf{v}_3) = \mathsf{Hotel} \sqcap \forall \mathsf{HotelLocation.EurCity}$
$def(\mathsf{v}_4) = \mathsf{Flight} \sqcap \forall \mathsf{Airline.AmCity}$

The conjunctive query $\mathsf{q}_{\mathcal{V}}(X) :- \mathsf{v}_4(X) \wedge \mathsf{v}_1(X,Y) \wedge \mathsf{v}_2(U,Y) \wedge \mathsf{v}_3(U)$ is a $\mathcal{CQ}$ rewriting of q using $\mathcal{V}$. To see why, let us show that for any model I of $\mathcal{V}$ for any $a \in \Delta^I$, if $a \in \mathsf{q}_{\mathcal{V}}^I$ then $a \in \mathsf{q}^I$. Consider $a \in \mathsf{q}_{\mathcal{V}}^I$:

—Since $def(\mathsf{q}_{\mathcal{V}})$ is restricted to the single rule $\mathsf{q}_{\mathcal{V}}(X) : -\mathsf{v}_4(X) \wedge \mathsf{v}_1(X,Y) \wedge \mathsf{v}_2(U,Y) \wedge \mathsf{v}_3(U)$, that means that there exists b, c in Δ^I such that: $a \in \mathsf{v_4}^I$, $(a,b) \in \mathsf{v_1}^I$, $(c,b) \in \mathsf{v_2}^I$ and $c \in \mathsf{v_3}^I$.

—Since I is a model of $\mathcal{V}$, according to the view definitions, we have: $a \in (\mathsf{Flight} \sqcap \forall \mathsf{Airline.AmCity})^I$, $(a,b) \in \mathsf{Stop}^I$, $(c,b) \in \mathsf{HotelLocation}^I$ and $c \in (\mathsf{Hotel} \sqcap \forall \mathsf{HotelLocation.EurCity})^I$.

—According to the semantics of the concept descriptions, it follows from $c \in (\mathsf{Hotel} \sqcap \forall \mathsf{HotelLocation.EurCity})^I$ that $c \in (\forall \mathsf{HotelLocation.EurCity})^I$, and since $(c,b) \in \mathsf{HotelLocation}^I$, that $b \in \mathsf{EurCity}^I$.
It follows from $a \in (\mathsf{Flight} \sqcap \forall \mathsf{Airline.AmCity})^I$ that $a \in \mathsf{Flight}^I$.
Finally, we have the three assertions $a \in \mathsf{Flight}^I$, $(a,b) \in \mathsf{Stop}^I$ and $b \in \mathsf{EurCity}^I$, which entails that $a \in (\mathsf{Flight} \sqcap \exists \mathsf{Stop.EurCity})^I$.

—Since $def(\mathsf{q}) = \mathsf{Flight} \sqcap \exists \mathsf{Stop.EurCity}$, and since $a \in (\mathsf{Flight} \sqcap \exists \mathsf{Stop.EurCity})^I$, we obtain that $a \in \mathsf{q}^I$.

□

As illustrated in the above example, the languages for defining queries, views and rewritings may be different. In Example 4, the query and the views are expressed in the same (terminological) language $\mathcal{FLE}$, while the rewriting is defined in the relational language $\mathcal{CQ}$. It is important to note that the existence of a rewriting depends on the rewriting language that is considered. For instance, in Example 4, the query q defined by $\mathsf{Flight} \sqcap \exists \mathsf{Stop.EurCity}$ does not have any $\mathcal{FLE}$ rewriting using the set of views $\mathcal{V}$, while we have exhibited a $\mathcal{CQ}$ rewriting.

In addition, as pointed out in [Baader et al. 2000], it is important to observe that decidability results and algorithms for checking the existence of $\mathcal{L}_3$ rewritings cannot be transferred to rewriting languages that are sublanguages of $\mathcal{L}_3$.

4.2 Connection between rewriting queries using views and answering queries using views

The problem of answering queries using views (Definition 3.11) is the problem of computing the answers to a query having data only in extensions of views. The problem of rewriting queries using views (Definition 4.1) is the problem of reformulating an original query into queries whose definition uses only view predicates. As pointed out in [Abiteboul and Duschka 1998; Calvanese et al. 2000; Calvanese et al. 2000b], those two problems though tightly related, are not equivalent. It follows immediately from Definitions 3.11, 3.13 and 4.1 that evaluating rewritings on the views extensions provides answers of the original query (w.r.t. those views extensions). However, in the general case, it is not guaranteed that a maximally contained (or even an equivalent [Calvanese et al. 2000b]) rewriting of a query using a set of views provides *all* the answers that can be obtained for the query from the extensions of the views. It depends on the restrictions imposed on the languages $\mathcal{L}_1$, $\mathcal{L}_2$ and $\mathcal{L}_3$ for defining the queries, the views and the rewritings. In particular, there may be a rewriting in a more expressive language than $\mathcal{L}_3$ that may provide more answers than the answers of any rewriting in $\mathcal{L}_3$.

In an information integration setting where the sources are on the network, the view extensions are not directly available. Rewriting queries using the view definitions and then evaluating the rewritings is the only effective method for getting answers to users queries. It is therefore important to exhibit cases in which the evaluation of rewritings provides *all* the answers of the original query. Theorem 4.3 exhibits such a case, based on the result provided by Proposition 4.2 which shows the central role of *conjunctive queries* as the basis for a rewriting language. As a corollary of Theorem 4.3, Corollary 4.4 outlines a sufficient condition characterizing settings in which the problem of answering queries using views has a polynomial data complexity.

PROPOSITION 4.2. *Let q be a query expressed in a language $\mathcal{L}_1$ of* CARIN *over the vocabulary of an ontology $\mathcal{O}$. Let $\mathcal{V}$ be a set of views defined in a language $\mathcal{L}_2$ of* CARIN *over the same vocabulary. Let $ext(\mathcal{V})$ be the union of the extensions of the view predicates.*

If there exists $\vec{a} \in Ans(q, ext(\mathcal{V}))$ then there exists a $\mathcal{CQ}^{\neq}$ rewriting $q_{\mathcal{V}}$ of q using $\mathcal{V}$ such that $\vec{a} \in Ans(q_{\mathcal{V}}, ext(\mathcal{V}))$.

PROOF. Let $\vec{a}$ be an answer of q over $\mathcal{V}$. Let $\mathcal{C}$ be the set of all the constants appearing in $ext(\mathcal{V})$: every constant a of $\vec{a}$ belongs to $\mathcal{C}$, but other constants than those of $\vec{a}$ may belong to $\mathcal{C}$ too.

Let Var be an infinite set of fresh variables, and let ϕ be an injective mapping from $\mathcal{C}$ to Var such that every constant $c \in \mathcal{C}$ is mapped to a fresh variable $\phi(c)$ in Var. For any vector $\vec{c} = (c_1, \ldots, c_n)$ we denote by $\phi(\vec{c})$ the vector of variables $(\phi(c_1), \ldots, \phi(c_n))$. Let $\vec{X}$ be the vector of variables resulting from the mapping of the tuple $\vec{a}$: $\phi(\vec{a}) = \vec{X}$. Let Y be the variables of $\phi(\mathcal{C})$ that do not belong to X. $\vec{X}$ will be the vector of the distinguished variables of the rule that will define our rewriting, while Y will be its existential variables.

Let $q_{\mathcal{V}}$ be the conjunctive query defined by: $q_{\mathcal{V}}(\vec{X}) :- \bigwedge_{v(\vec{c}) \in ext(\mathcal{V})} v(\phi(\vec{c}))$. Since ϕ is an injective mapping, all the pairs of variables of that query must be stated as different: $q_{\mathcal{V}}$ is therefore a conjunctive query with inequalities. We omit writing those inequality atoms $\phi(c_i) \neq \phi(c_j)$ for clarity reasons.

Let us show that $q_{\mathcal{V}}$ is a $\mathcal{CQ}^{\neq}$ rewriting of q using $\mathcal{V}$, i.e., that for any interpretation I which is a model of every $v \in \mathcal{V}$ and of the ontology $\mathcal{O}$, if $\vec{o} \in q^I_{\mathcal{V}}$, then $\vec{o} \in q^I$.

Consider $\vec{o} \in q^I_{\mathcal{V}}$: there exists a mapping α from $X \cup Y$ to the domain Δ^I such that $\alpha(\phi(\vec{c})) \in v^I$ for every conjunct $v(\phi(\vec{c}))$ in the body of the rule defining $q_{\mathcal{V}}$, and such that for every $i \neq j$, $\phi(c_i) \neq \phi(c_j)$.

We extend the interpretation I to the constants of $\mathcal{C}$ as follows: for every $c \in \mathcal{C}$, $c^I = \alpha(\phi(c))$. I is therefore a model of $ext(\mathcal{V})$ since for every $v(\vec{c}) \in ext(\mathcal{V})$, $\vec{c}^I \in v^I$.

Therefore, I is a model of $\mathcal{O}$, $\mathcal{V}$ and of $ext(\mathcal{V})$. According to Definition 3.11, since $\vec{a}$ is an answer of q using $\mathcal{V}$, we have: $\vec{a}^I \in q^I$.

By construction of the extension of I to the constants of $\mathcal{C}$, we have $\vec{a}^I = \alpha(\phi(\vec{a}))$. Since $\phi(\vec{a}) = \vec{X}$, $\vec{a}^I = \alpha(\vec{X}) = \vec{o}$. Therefore, we obtain: $\vec{o} \in q^I$. □

The following theorem is a consequence of Proposition 4.2 . It shows that when a query has a finite number of *maximally contained conjunctive rewritings*, then the complete set of its answers can be obtained as the union of the answer sets of its rewritings.

THEOREM 4.3. *Let $\mathcal{V}$ be a set of views defined in a language $\mathcal{L}_2$ of* CARIN *over the vocabulary of an ontology. Let q be a query expressed in a language $\mathcal{L}_1$ of* CARIN *over the same vocabulary. If q has a* finite number of maximally contained $\mathcal{CQ}^{\neq}$ rewritings, $q_{\mathcal{V}}^1, \cdots, q_{\mathcal{V}}^n$, *then:*

$$Ans(q, ext(\mathcal{V})) = \bigcup_{i=1}^{n} Ans(q_{\mathcal{V}}^i, ext(\mathcal{V})).$$

PROOF. It follows immediately from Definitions 3.11, 3.13 and 4.1 that $Ans(q_{\mathcal{V}}^i, ext(\mathcal{V}))$ is included in $Ans(q, ext(\mathcal{V}))$, and then so is their union.

For showing the inclusion of $Ans(q, ext(\mathcal{V}))$ in $\bigcup_{i=1}^{n} Ans(q_{\mathcal{V}}^i, ext(\mathcal{V}))$, let us consider an answer $\vec{a}$ belonging to $Ans(q, ext(\mathcal{V}))$. According to Proposition 4.2, there exists a $\mathcal{CQ}^{\neq}$ rewriting $q_{\mathcal{V}}$ of q which, by construction is such that $\vec{a} \in Ans(q_{\mathcal{V}}, ext(\mathcal{V}))$.

$q_{\mathcal{V}}$ is necessarily subsumed by one of the maximally contained $\mathcal{CQ}^{\neq}$ rewritings $q_{\mathcal{V}}^i$ of q. According to Definition 3.12, $Ans(q_{\mathcal{V}}, ext(\mathcal{V})$ is therefore included in $Ans(q_{\mathcal{V}}^i, ext(\mathcal{V}))$, and then since $\vec{a} \in Ans(q_{\mathcal{V}}, ext(\mathcal{V}))$, $\vec{a}$ belongs to $Ans(q_{\mathcal{V}}^i, ext(\mathcal{V}))$, and thus to $\bigcup_{i=1}^{n} Ans(q_{\mathcal{V}}^i, ext(\mathcal{V}))$ too. □

The following corollary of Theorem 4.3 results from the fact that answering conjunctive queries (possibly with inequalities) over a given database instance has a polynomial data complexity.

COROLLARY 4.4. *Let $\mathcal{L}_1$ be a query language and $\mathcal{L}_2$ be a view language such that every $\mathcal{L}_1$ query has a finite number of maximally contained $\mathcal{CQ}^{\neq}$ rewritings using $\mathcal{L}_2$ views. Then, the problem of answering $\mathcal{L}_1$ queries using only extensions of $\mathcal{L}_2$ views as input has a polynomial data complexity.*

4.3 Couching previous work within our framework

Before describing the specific instance of the rewriting problem that we study in detail in this paper, we situate previous work dealing with answering or rewriting queries using views within our general framework. We point out when algorithms have been implemented in information integration systems.

—In [Halevy 2001], the *bucket*, *inverse rules* and *minicon* algorithms are described and compared. They allow to rewrite queries using views in a setting where the languages $\mathcal{L}_1$, $\mathcal{L}_2$ and $\mathcal{L}_3$ for defining queries, views and rewritings are *relational languages*, possibly extended with arithmetic comparison predicates. Those algorithms have been developed specifically for the context of information integration: the bucket algorithm has been implemented in *The Information Manifold* system [Levy et al. 1996] and the inverse rules algorithm has been implemented in the *InfoMaster* system [Duschka and Genesereth 1997].

—In [Abiteboul and Duschka 1998], the complexity of answering queries using views is studied when queries and views are expressed in variants of *relational languages* allowing inequalities or negation in the view and/or query definitions. It is shown that the complexity depends on whether views are assumed to store all the tuples that satisfy the view definition, or only a subset of it (which is the case corresponding to the setting considered in this paper).

—The query and view languages $\mathcal{L}_1$, $\mathcal{L}_2$ that are considered in [Calvanese et al. 2000a] for defining the problem of answering queries using views are the language $\mathcal{L}$ of regular expressions on a set of binary predicates (those regular expressions represent paths within a graph in which the edges are labelled). $\mathcal{L}$ can be easily transformed into the *relational language* which is recursive and which is built on binary predicates only.

—The first extension to description logics for defining the problem of rewriting queries using views has been presented in [Beeri et al. 1997] . Two instances of the rewriting problem were studied there:

—The *datalog-rewriting* problem considers non recursive CARIN-$\mathcal{ALN}$, which is a *hybrid language* of CARIN, as the query and the view languages, while the rewriting language is the *relational language* of CARIN. It is shown that if the view definitions do not contain existential variables, then it is always possible to find a finite set of conjunctive rewritings maximally contained in the query. However, if the views have existential variables, it is not always possible to find a maximally contained Datalog rewriting.

—The *dl-rewriting* problem considers a purely *terminological language* of CARIN as the query, view and rewriting languages: queries, views and rewritings are expressed in the description logics $\mathcal{ALCNR}$.

—In [Baader et al. 2000], a general framework is presented for defining rewritings that are expressed in a description logic $\mathcal{L}_3$ for queries that are concept descriptions of a description logic $\mathcal{L}_1$, using views that are defined in a description logic $\mathcal{L}_2$. Our framework encompasses it since the query, view and rewriting languages considered in [Baader et al. 2000] are *terminological languages* of CARIN. The specific rewriting problem which is then studied in detail and for which a rewriting algorithm is provided, is the *minimal rewriting problem* when $\mathcal{L}_1$, $\mathcal{L}_2$ and $\mathcal{L}_3$ are all $\mathcal{ALE}$: it consists of finding an equivalent $\mathcal{ALE}$ rewriting of minimal size for a query which is a $\mathcal{ALE}$ concept description, using view definitions that are roles or $\mathcal{ALE}$ concept descriptions.

—[Rousset 1999] deals with a new decidable instance of the *datalog-rewriting* problem introduced in [Beeri et al. 1997], in which the query language is a *hybrid language* of CARIN, while the view language is a *terminological language* of CARIN, and the rewriting language is a *relational language*. In fact, the query and rewriting languages are the same as in [Beeri et al. 1997], i.e., non recursive CARIN-$\mathcal{ALN}$ and $\mathcal{CQ}$ respectively while the view language is $\mathcal{ALN}$. This restriction guarantees that it is always possible to compute the (finite) set of maximally contained conjunctive rewritings of a query, and therefore to obtain (Theorem 4.3) the set of all its possible answers using views from its rewritings. An incomplete implementation of the rewriting algorithm presented in [Rousset 1999] has been used for answering users queries in the PICSEL information integration system [Goasdoué et al. 2000].

—In [Calvanese et al. 2000], the complexity of answering queries using views is studied for queries and views that are unions of conjunctives queries over descriptions of the description logics $\mathcal{DLR}$. $\mathcal{DLR}$ [Calvanese et al. 1998] is a very expressive description logics embedding *n-ary relations*. The complexity is studied under different assumptions about the objects in the view extensions (closed versus open domain assumption) and the accurateness of the view definitions for representing their actual content (sound, complete, or exact views). It is shown that under the open domain assumption (which corresponds to the setting of this paper), checking whether a tuple is an answer to a query using views extensions is decidable (in double exponential time) when the queries and the views are unions of conjunctives queries over $\mathcal{DLR}$ expressions.

5. A CASE STUDY APPROPRIATE FOR THE SEMANTIC WEB

While there is a general agreement on the central role of ontologies in the Semantic Web for mediating between users and distributed data, the discussion is still open on which kinds of ontologies are the most appropriate. In particular, it is difficult to find the right tradeoff between simplicity of use for end-users, expressive power for a fine-grained modeling of the data sources, and computational complexity for query answering.

In this section, based on our experience both on theoretical and practical aspects of information integration, we define a setting for information integration centered on ontologies that are simple enough for users or practitioners who are not knowledge engineers or logicians. The design of this simple setting has resulted from the lessons drawn from the PICSEL [Picsel] and MKBEEM [Mkbeem] projects: the ontology designers did not exploit the full expressive power of $\mathcal{ALN}$ which was offered to them for modeling their application domain. In particular, they hardly used the number restrictions constructors while they extensively used the value restriction constructor but with the implicit assumption that the roles r involved in an expression $\forall r.C$ have atleast a filler (which does not correspond to the logical semantics of the $\forall$ constructor). We introduced the $\forall^+$ constructor (see Section 5.1) in order to capture the intuitive meaning of value restrictions that most of knowledge engineers have in mind, e.g., when they write $Hotel \sqcap \forall^+ AssociateRoom.WithAirConditionning$, they assume that hotels has at least one room. Replacing the $\forall$ constructor by the $\forall$+ constructor results in the $\mathcal{AL}^+$ description logic, which is a sublanguage of $\mathcal{ALN}$. The rewriting algorithm that will be described in Section 5.3 is therefore a restriction of the one described in [Rousset 1999], compared to which the worst case time complexity is significantly reduced.

We start by presenting precisely this setting by defining it as a particular instance of the formal framework presented in Section 3. We then state the rewriting problem that we consider for that instance. Finally, we provide a rewriting algorithm, and we show that it is sound and complete. According to Theorem 4.3 and Corollary 4.4, this provides a method for anwering queries using views which is complete while having a polynomial data complexity.

5.1 The particular instance

We first define the ontologies that we consider, and then the queries and views that are allowed over those ontologies.

5.1.1 *Ontologies.* The ontologies that we deal with are CARIN ontologies (Section 3.2) with empty deductive components. They are reduced to terminologies

(a.k.a. TBox) with restricted inclusion statements and concept definitions using a limited description logic simply enabling stating concept conjunction and elementary typing information for necessary roles.
The inclusion statements that are allowed are of the two following forms:

—$A \sqsubseteq B$,
—or $A \sqcap B \sqsubseteq \bot$,

where A and B are *atomic* concepts.

We restrict the inclusion statements of a TBox to be acyclic: there is no chain $L = L_0, L_1, \ldots, L_n = L$ such that the inclusion statements $L_{i-1} \sqsubseteq L_i$ are included in the TBox.

The concept definitions use the description logic (denoted $\mathcal{AL}^+$) whose constructors are: top ($\top$), bottom ($\bot$), atomic negation ($\neg A$), conjunction ($C_1 \sqcap C_2$) and the necessary value restriction ($\forall^+ r.C$). The latter constructor is not standard in description logics. Its semantics is given by the equivalences:

$$(\forall^+ r.C) \equiv (\forall r.C) \sqcap (\exists r.\top) \equiv (\forall r.C) \sqcap (\geq 1 r).$$

We consider *normalized* terminologies. The normalization of a TBox $\mathcal{T}$ is performed by:

(1) adding the inclusion statement $\neg B \sqsubseteq \neg A$ for each $A \sqsubseteq B$ belonging to $\mathcal{T}$,
(2) replacing each statement $A \sqcap B \sqsubseteq \bot$ of $\mathcal{T}$ by the two inclusion statements: $A \sqsubseteq \neg B$ and $B \sqsubseteq \neg A$,
(3) unfolding the concept definitions as mentioned in section 3.2, and exhaustively applying the following normalization rule on the unfolded definitions:

$$(\forall^+ r.(C_1 \sqcap C_2)) \rightarrow (\forall^+ r.C_1) \sqcap (\forall^+ r.C_2)$$

As a result, each concept description defining a concept name has the form of a conjunction $C_1 \sqcap \ldots \sqcap C_k$ such that each conjunct C_i has the form L or $\forall^+ r_1. \cdots .\forall^+ r_n.L$ where L is either an atomic concept or the negation of an atomic concept.

This allow us to state:

PROPOSITION 5.1. *Given a normalized $\mathcal{AL}^+$ TBox $\mathcal{T}$, checking the satisfiability of the $\mathcal{T}$ can be done in polynomial time.*

PROOF (SKETCH). The proof relies on showing that $\mathcal{T}$ is unsatisfiable iff:

—there are atomic concepts or negations of atomic concepts $L = L_0, L_1, \ldots, L_n = \neg L$ such that the inclusion statements $L_{i-1} \sqsubseteq L_i$ are included in $\mathcal{T}$,
—or, if a concept description C defining a concept name is unsatisfiable. C is unsatisfiable iff:
  —it has a conjunct of the form $\bot$ or $\forall^+ r_1. \cdots .\forall^+ r_n.\bot$,
  —or it has two conjuncts $\forall^+ r_1. \cdots .\forall^+ r_n.L$ and $\forall^+ r_1. \cdots .\forall^+ r_n.LL$, or L and LL, such that L and LL are incompatible. L and LL are incompatible if there exists $L = L_0, L_1, \ldots, L_n = L'$ and $LL = LL_0, LL_1, \ldots, LL_k = \neg L'$ such that the inclusion statements $L_{i-1} \sqsubseteq L_i$ and $LL_{j-1} \sqsubseteq LL_j$ are included in $\mathcal{T}$.

□

From now, we consider that the TBoxes that we deal with have been checked as being satisfiable.
Given a satisfiable normalized TBox $\mathcal{T}$, and two normalized $\mathcal{AL}^+$ satisfiable concept descriptions C and D , checking the subsumption modulo $\mathcal{T}$ between C and D can be done in polynomial time as well:

— $C \preceq_{\mathcal{T}} D$ iff every conjunct of C is subsumed modulo $\mathcal{T}$ by one conjunct of D.
— $\forall^+ r_1.\cdots.\forall^+ r_n.L \preceq_{\mathcal{T}} \forall^+ r'_1.\cdots.\forall^+ r'_p.L'$ iff $n = p$, $\forall i$, $r_i = r'_i$, and $L \preceq_{\mathcal{T}} L'$.
— $L \preceq_{\mathcal{T}} L'$ iff there are atomic concepts or negations of atomic concepts $L = L_0, L_1, \ldots, L_n = L'$ such that the inclusion statements $L_{i-1} \sqsubseteq L_i$ are included in $\mathcal{T}$.

5.1.2 *Queries.* The queries that are allowed are $\mathcal{CQ}$-$\mathcal{AL}^+$ expressions built over the vocabulary of the ontology. This language is interesting in practice since it extends the language of conjunctive queries to the one of typed conjunctive queries.

We consider normalized queries. A query is normalized by:

(1) replacing in the query each pair of atoms $C_1(U) \wedge C_2(U)$ by the atom $(C_1 \sqcap C_2)(U)$,
(2) replacing in the query each occurence of a concept defined in $\mathcal{T}$ by its definition,
(3) normalizing the concept descriptions appearing in the query in the same way as the concept definitions of the ontology were.

5.1.3 *Views.* The views are defined either by roles or $\mathcal{AL}^+$ concept descriptions built over the vocabulary of the ontology.
As for the queries, the views are normalized in the same way as queries are in the above steps (2) and (3).

5.2 The rewriting problem

Based on the results of Theorem 4.3 and Corollary 4.4, in order to guarantee that query answering can be soundly and completely performed with a polynomial data complexity through a first step of query rewriting, we choose the purely relational language of conjunctive queries ($\mathcal{CQ}$) as the rewriting language.

The problem of rewriting queries using views that we study in the remainder of the paper can be formally stated as follows.

Given:

— an $\mathcal{AL}^+$ normalized and satisfiable TBox $\mathcal{T}$,
— a normalized and satisfiable $\mathcal{CQ}$-$\mathcal{AL}^+$ query q over $\mathcal{T}$, and
— a set of normalized and satisfiable $\mathcal{AL}^+$ views $\mathcal{V}$ over $\mathcal{T}$.

Question:
Does there exist a *finite* set of maximally contained $\mathcal{CQ}$ rewritings of q using $\mathcal{V}$?

5.3 The rewriting algorithm

First, we present and study the *RewriteAtom* algorithm that rewrites atomic queries of arity 1 (defined by a concept-atom) or of arity 2 (defined by a role-atom). Then, we will focus on the *RewriteQuery* algorithm that rewrites a whole conjunctive query of any arity.

5.3.1 *RewriteAtom.* We introduce the $\bigotimes$ operator that acts as a cartesian product of conjunctions. This binary operator, which will be used to combine rewritings, works on set of atom conjunctions as follows:

—$\emptyset \bigotimes \{cj_1, \ldots, cj_n\} = \emptyset$
—$\{cj^1, \ldots, cj^m\} \bigotimes \{cj_1, \ldots, cj_n\} = \{cj^1 \wedge cj_1, \ldots, cj^1 \wedge cj_n, \ldots, cj^m \wedge cj_1, \ldots, cj^m \wedge cj_n\}$

The *RewriteAtom* algorithm computes at least all the maximal rewritings of a query.

> *RewriteAtom* algorithm
> **input:** an $\mathcal{AL}^+$ normalized and satisfiable TBox $\mathcal{T}$, an normalized and satisfiable atomic query q over $\mathcal{T}$ defined by either a role-atom or a concept-atom, and a set of normalized and satisfiable views $\mathcal{V}$ also over $\mathcal{T}$.
> **output:** a set of $\mathcal{CQ}$ rewritings of q including all the maximal rewritings.
>
> **if** q is defined by a role-atom $r(U_1, U_2)$ **then**
> $result := \{v(U_1, U_2) \mid v \in \mathcal{V}$ and $def(v) = r\}$
> **else if** q is defined by a concept-atom $(\bigsqcap_{i=1}^{n} C_i)(U_1)$ **then**
> $result := \bigotimes_{i=1}^{n} RewriteAtom(\mathcal{T}, C_i(U_1), \mathcal{V})$
> **else** $result := \{v_0(U_{m+1}) \wedge \bigwedge_{i=1}^{m} v_i(U_{m+2-i}, U_{m+1-i})$
> $\mid \forall i \in [0..m]\ v_i \in \mathcal{V}$,
>   and $def(v_0) = ((\forall^+ r_1. \cdots .\forall^+ r_m.D) \sqcap \cdots)$ with $D \preceq_{\mathcal{T}} C$,
>   and $\forall i \in [1..m]\ def(v_i) = r_i\}$
> **return** *result*

The first **if** concerns the rewriting of binary atomic queries defined by role-atoms: those using views explicitly defined as the corresponding roles.

The **else if** deals with conjunctions of concept descriptions and is based on the following logical entailment (actually an equivalence):

$$\bigwedge_{i=1}^{n} C_i(U_1) \models (\bigsqcap_{i=1}^{n} C_i)(U_1).$$

The final **else** is dedicated to concept-atoms of the form: $\top$, $A(U_1)$, $(\neg A)(U_1)$, $(\forall^+ r'_1. \cdots .\forall^+ r'_n\ \top)(U_1)$, $(\forall^+ r'_1. \cdots .\forall^+ r'_n\ A)(U_1)$ and $(\forall^+ r'_1. \cdots .\forall^+ r'_n\ \neg A)(U_1)$. It results from the validity of the following logical entailment, which holds for any concept description C and D such that $D \preceq_{\mathcal{T}} C$:

$$(\forall^+ r_1. \cdots .\forall^+ r_m.D)(U_{m+1}) \wedge \bigwedge_{i=1}^{m} r_i(U_{m+2-i}, U_{m+1-i}) \models C(U_1).$$

Note that if $m = 0$ this case reduces to subsumption between two concept descriptions $(D \preceq_{\mathcal{T}} C)$.

5.3.1.1 *Properties of RewriteAtom.* We consider in turn the termination, soundness and completeness of *RewriteAtom*. Finally, we study its worst case time complexity.

Termination When *RewriteAtom* is applied to a role-atom (**if**) or a concept-atom that does not contain a conjunction (**else**), its termination is immediate since there is no recursive call at all and $\mathcal{V}$ is finite. Otherwise, when it is applied to a concept-atom that contains a conjunction (**else if**), the termination is guaranteed because all the recursive calls lead to the previous case (**else**).

Soundness and completeness Theorem 5.2 states the soundness and completeness of *RewriteAtom*.

THEOREM 5.2. *Let $\mathcal{T}$ be a normalized and satisfiable $\mathcal{AL}^+$ TBox. Let q be a normalized and satisfiable atomic query over $\mathcal{T}$ defined by a role-atom or an $\mathcal{AL}^+$ concept-atom, and $\mathcal{V}$ be a set of normalized and satisfiable $\mathcal{AL}^+$ views also over $\mathcal{T}$. Let result be the output of RewriteAtom$(\mathcal{T}, q, \mathcal{V})$:*

—*each element of result is a $\mathcal{CQ}$ rewriting of q (Soundness)*

—*every $\mathcal{CQ}$ rewriting of q has a body that includes (up to a renaming of existential variables) all the view-atoms of at least one element of result (Completeness).*

The proof of soundness is straightforward. It suffices to check that the rewriting rules encoded in *RewriteAtom* (i.e., the above logical entailments) are true.

The proof of completeness is more complex. We first summarize the general strategy that we follow. Then, we explain the technical machinery that we employ, which is based on the computation of completions from a given knowledge base using a set of propagation rules as described in [Nardi et al. 1995]. Finally, we state the different lemmas that compose the proof and we explain their role in the proof.

GENERAL STRATEGY We need to characterize the conjunctive rewritings q_v of an atomic query q which is defined either by a role-atom $r(U_1, U_2)$ or a concept-atom $C(U)$. By definition, q_v is a rewriting of q iff for every model I of the TBox $\mathcal{T}$ and of the views, $q_v^I \subseteq q^I$. According to Definitions 4.1 and 3.13, if we identify the distinguished variables of q_v and q, this is equivalent to: $expand(body(q_v)), \mathcal{T} \models C(U)$ (respectively $r(U_1, U_2)$), where $expand(body(q_v))$ is the conjunction of $\mathcal{AL}^+$ concept-atoms and role-atoms obtained by replacing in the definition of q_v every view predicate v by its definition.
Therefore, for each kind of atom $C(U)$ or $r(U_1, U_2)$ defining an atomic query possibly appearing in the setting of application of *RewriteAtom* we have to characterize the different knowledge bases made of $\mathcal{AL}^+$ concept-atoms and role-atoms which entail it, while satisfying the inclusion statements of $\mathcal{T}$.

TECHNICAL MACHINERY Given a knowledge base S_0 made of $\mathcal{AL}^+$ concept-atoms and role-atoms such that $S_0, \mathcal{T} \models C(U)$ (respectively $r(U_1, U_2)$), we consider the (unique) completion of S_0 using the following propagation rules:

(1) $S \rightarrow_{\sqcap} \{C_1(s), C_2(s)\} \cup S$ if
  (a) $(C_1 \sqcap C_2)(s)$ is in S,
  (b) $C_1(s)$ and $C_2(s)$ are not both in S.
(2) $S \rightarrow_{\forall} \{C(t)\} \cup S$ if
  (a) $(\forall^{+} r.C)(s)$ is in S,
  (b) $r(s,t)$ is S,
  (c) $C(t)$ is not in S.
(3) $S \rightarrow_{\forall^{+}} \{r(s,t), C(t)\} \cup S$ if
  (a) $(\forall^{+} r.C)(s)$ is in S,
  (b) t is a fresh variable,
  (c) $r(s,t)$ is not in S.
(4) $S \rightarrow_{\sqsubseteq} \{L'(s)\} \cup S$ if
  (a) $L(s)$ is in S,
  (b) $L \sqsubseteq L'$ is an inclusion statement in $\mathcal{T}$,
  (c) $L'(s)$ is not in S.

Starting from S_0, we obtain S_i by successively applying i propagation rules. We stop when no propagation rule applies on S_n: S_n is called the *completion.* Note that there is a unique completion because we are in $\mathcal{AL}^{+}$. In a knowledge base S obtained from an initial knowledge base S_0, we must distinguish the *generated variables*, which are created by the application of the propagation rule $\rightarrow_{\forall^{+}}$, from the *initial variables* which are the individuals appearing in the atoms of S_0. We will denote the initial variables by capital letters U, V, X, Y, Z while we will denote the generated variables by letters u, v, w. An *object* of S is a (initial or generated) variable, and will be denoted s.

For a knowledge base S, we define the dependency graph of its variables as follows: its nodes are the (initial or generated) variables appearing in the atoms of S ; there exists an edge from s_1 to s_2 iff $r(s_1, s_2)$ is an atom of S. Examining the propagation rules reveals that the generated variables form a forest of trees in that graph, each tree being rooted in an initial variable.

Given a completion S, we define its canonical interpretation I_S as follows:

(1) $\mathcal{O}^{I_S} := \{s \mid s \text{ is an object in } S\}$,
(2) $s^{I_S} := s$,
(3) for an atomic concept A, $s \in A^{I_S}$ if and only if $A(s) \in S$,
(4) $(s_1, s_2) \in r^{I_S}$ if and only if $r(s_1, s_2) \in S$.

The extensions of the complex concepts are computed using the equations defining the semantics of $\mathcal{AL}^{+}$ constructors.
The main existing results that we will use in our proof are the following ones (from [Levy and Rousset 1998a]):

(1) Let S be the completion of S_0, and let I_S be its canonical interpretation. Then, I_S is a model of S. We can therefore refer to the canonical interpretation of a completion as its *canonical model.*
(2) Let S be the completion of S_0: $S_0, \mathcal{T} \models p(\vec{U})$ iff $S \models p(\vec{U})$.

STRUCTURE OF THE PROOF OF COMPLETENESS Lemma 5.3 characterizes the knowledge bases, made of $\mathcal{AL}^+$ concept-atoms and role-atoms, which entail a role-atom $r(U_1, U_2)$. It follows from this lemma that every rewriting of an atomic query of arity 2 defined by a role-atom $r(U_1, U_2)$ must include an atom of the form $v(U_1, U_2)$ such that $def(v) = r$. This corresponds to the rewritings computed in the **if** in *RewriteAtom*.

LEMMA 5.3. *Let $r(U_1, U_2)$ be a role-atom. If S_0 is a knowledge base made of $\mathcal{AL}^+$ concept-atoms and role-atoms such that $S_0, \mathcal{T} \models r(U_1, U_2)$, then $r(U_1, U_2)$ belongs to S_0.*

PROOF. Let S be the completion computed from S_0, and I_S its canonical model. I_S is a model of $r(U_1, U_2)$: there exists a mapping α from $\{U_1, U_2\}$ to the domain $\mathcal{O}^{I_S}$ of I_S such that $\alpha(U_1) = U_1$, $\alpha(U_2) = U_2$, and $(\alpha(U_1), \alpha(U_2)) \in r^{I_S}$. By definition of I_S, $(\alpha(U_1), \alpha(U_2)) \in r^{I_S}$ means that $r(U_1, U_2) \in S$. Considering the propagation rules, $r(U_1, U_2)$ cannot have been produced by the application of a propagation rule. Therefore, it must be the case that $r(U_1, U_2) \in S_0$. □

We focus now on the entailment of concept-atoms $C(U_1)$.
Lemma 5.4 deals with the case where a concept-atom $C(U_1)$ is entailed by a knowledge base S_0 because there exists a concept D which is subsumed by C such that $D(U_1) \in S$ where S is the completion obtained from S_0. This case leads to the rewritings computed in the **else** of *RewriteAtom*.

LEMMA 5.4. *Let S_0 be a knowledge base made of $\mathcal{AL}^+$ concept-atoms and role-atoms such that $S_0, \mathcal{T} \models C(U_1)$. Let S be its completion. Let $D(U_1)$ be a concept-atom of S such that $D \preceq_{\mathcal{T}} C$. Then, either $D(U_1) \in S_0$ or there exists a concept of the form $(\forall^+ r_1. \cdots .\forall^+ r_m.D)$ and variables $U_{m+1}, \ldots, U_1$ such that:*
$(\forall^+ r_1. \cdots .\forall^+ r_m.D)(U_{m+1}) \in S_0$ and $r_i(U_{m+2-i}, U_{m+1-i}) \in S_0$, for every $i \in [1..m]$.

PROOF. Let S_i be the knowledge base resulting of the application of i propagation rules to S_0. We consider the case where $D(U_1) \in S$ and $D(U_1) \notin S_0$. There exists $m > 0$ such that $D(U_1) \in S_m$ and $D(U_1) \notin S_{m-1}$. One propagation rule has been applied in order to produce $D(U_1)$. Since $D(U_1) \in S_m$ and $D(U_1) \notin S_{m-1}$, it cannot be the rule $\rightarrow_{\sqcap}$ or $\rightarrow_{\sqsubseteq}$. It cannot be the rule $\rightarrow_{\forall^+}$ since U_1 is an initial variable. Therefore, the rule $\rightarrow_{\forall}$ has applied, and that means that there exists a role r_m and a variable U_2 s.t. $(\forall^+ r_m.D)(U_2)$ and $r_m(U_2, U_1)$ are in S_{m-1}. U_2 is necessarily an initial variable too, and therefore $r_m(U_2, U_1)$ must belong to S_0 in order to be in S_{m-1}. While $m - 1 \neq 0$, we can iterate the same reasoning on $(\forall^+ r_m.D)(U_2) \in S_{m-1}$ by induction on $m - 1$, which ends the proof of the lemma. □

Lemma 5.5 concludes the proof of Theorem 5.2. It deals with the case where a concept-atom $C(U_1)$ is entailed by a knowledge base S_0 and $\mathcal{T}$ while there does not exist any concept D subsumed by C such that $D(U_1) \in S$. It can be the case when C is a conjunction. This case leads to the rewritings computed in the **else if** of *RewriteAtom*.

LEMMA 5.5. *Let $C = \sqcap_{j=1}^{n} C_j$ be a normalized and satisfiable concept description. Let S_0 be a knowledge base made of $\mathcal{AL}^+$ concept-atoms and role-atoms such*

that $S_0, \mathcal{T} \models C(U_1)$. If the completion S of S_0 does not include any atom $D(U_1)$ such that $D \preceq_{\mathcal{T}} C$, then for $j \in [1..n]$ there exists either $D_j(U_1) \in S_0$ such that $D_i \preceq_{\mathcal{T}} C_i$ or there exists a concept of the form $\forall^+ r_{1_j}.\cdots.\forall^+ r_{m_j}.D_j$ and variables $U_{m_j+1}, \ldots, U_{1_j}$ such that:
$(\forall^+ r_{1_j}.\cdots.\forall^+ r_{m_j}.D_j)(U_{m_j+1}) \in S_0$ and $r_{i_j}(U_{m_j+2-i_j}, U_{m_j+1-i_j}) \in S_0$, for every $i_j \in [1..m_j]$.

Proof. Let S be the completion of S_0. Since $S_0, \mathcal{T} \models C(U_1)$ then $\forall j \in [1..n]$ $S \models C_j(U_1)$. Given any $C_k(U_1)$ among $C_1(U_1), \ldots, C_n(U_1)$, it can be of the form $\top(U_1)$, $A(U_1)$, $(\neg A)(U_1)$, $(\forall^+ r_1.\cdots.\forall^+ r_m.\top)(U_1)$, $(\forall^+ r_1.\cdots.\forall^+ r_m.A)(U_1)$ or $(\forall^+ r_1.\cdots.\forall^+ r_m.\neg A)(U_1)$. It is easy to see that S must contain an atom $D_k(U_1)$ such that $D_k \preceq_{\mathcal{T}} C_k$ in order to have $S \models C_k(U_1)$. This case is handled by Lemma 5.4. □

Worst case time complexity We give an upper bound of the worst case time complexity of *RewriteAtom*. In order to simplify the complexity analysis, we suppose that the set $\mathcal{V}$ of views is not redundant in the following sense: distinct views have distinct definitions.

Theorem 5.6. *The time complexity of RewriteAtom is at worst:*

- *polynomial in the size of the TBox,*
- *polynomial in the number of views and in their size, and*
- *exponential in the size of the query.*

Proof. In this complexity analysis, we use the following parameters:

- $s_{\mathcal{T}}$ is the size of the TBox $\mathcal{T}$
- v is the number of views
- c_v is the maximal number of conjuncts in the normalized view definitions
- d_v is the maximal depth of the normalized view definitions (i.e., the maximal number of nested $\forall^+$)
- c_q is the maximal number of conjuncts in the query, when the query is a concept-atom

In order to prove Theorem 5.6, let us review the possible queries.

(1) If the query is a role-atom $r(U_1, U_2)$, *RewriteAtom* checks views in $\mathcal{V}$ ($\leq v$) in order to find one whose definition is r (**if**).
(2) If the query is a concept-atom $C(U_1)$ whose predicate is not a conjunction (**else**), for each conjunct ($\leq c_v$) of each view ($\leq v$), *RewriteAtom* has to:
    (a) check if $D \preceq_{\mathcal{T}} C$ ($\leq s_{\mathcal{T}}$),
    (b) find views ($\leq d_v$), whose definitions are roles, among the views ($\leq v$), and
    (c) combine them ($\leq d_v$).
    It follows that the rewriting time is at worst less than $v.c_v.(s_{\mathcal{T}} + d_v.v + d_v) \leq s_{\mathcal{T}}.c_v.d_v.v^2$.
(3) If the query is a concept-atom whose predicate is a conjunction $(\prod_{i=1}^{c_q} C_i)(U_1)$, *RewriteAtom* has to perform c_q recursive calls that lead to the above case 2 and to combine the rewritings produced by these recursive calls (cost of $\bigotimes$).

An upper bound of the number of rewritings produced by such a call is the time it takes to perform this call. The reason is that, in the worst case, each time *RewriteAtom* performs an operation, it builds a rewriting. Thus, it follows that the rewriting time is less than $(s_{\mathcal{T}}.c_v.d_v.v^2)^{c_q} + c_q.(s_{\mathcal{T}}.c_v.d_v.v^2)$ and even $2.(s_{\mathcal{T}}.c_v.d_v.v^2)^{c_q}$.

We can end this proof by stating that an upper bound of the worst case time complexity of *RewriteAtom* is in $O((s_{\mathcal{T}}.c_v.d_v.v^2)^{c_q})$. □

5.3.2 *RewriteQuery*. We reuse an algorithm introduced in [Goasdoué 2001] in order to extend the atomic queries managed by *RewriteAtom* to conjunctive queries. The underlying idea is to combine (using $\bigotimes$) the rewritings of every atoms defining a query (computed by *RewriteAtom*) in order to obtain (maximal) rewritings of that query. However, as it is shown in the forthcoming example, such a strategy may miss some maximal rewritings if we only consider the query to rewrite. The *approximation* of some subqueries by concept-atoms may lead to additional rewritings. *RewriteQuery* computes first a set of approximations of the initial query, then rewrite them following the above strategy in order to obtain all the maximal rewritings of the initial query.

The approximation of a query is defined thanks to the notions of *binding graph* and *tree query*. The *binding graph* accounts for the connection existing between variables within a query.

Definition 5.7 *Binding graph of a conjunctive query.* Let q be a conjunctive query. Its *binding graph* (denoted by $\mathcal{G}(q)$) is a directed graph defined as follows:

—the nodes of $\mathcal{G}(q)$ are the variables of q and

—there exists an edge labelled by r from U_1 to U_2 in $\mathcal{G}(q)$ iff $r(U_1, U_2)$ is in q.

The binding graph allows to define several classes of conjunctive queries like tree, forest, dag, connected queries ([Goasdoué and Rousset 2002]). Here, we are interested in *tree queries*.

Definition 5.8 *Tree query.* Let q be a conjunctive query with a unique distinguished variable X. q is a *tree query* iff its binding graph is a tree rooted in X.

In [Goasdoué and Rousset 2002], we have exhibited an operator, denoted $Approx_{\uparrow}$, that computes the maximally subsumed concept-atom, whose concept description is in $\mathcal{AL}^+$, of a $\mathcal{CQ}$-$\mathcal{AL}^+$ tree query. Given this operator, the notion of approximation of a query is the following.

Definition 5.9 *Approximation of* $\mathcal{CQ}$-$\mathcal{AL}^+$ *queries.* Let q be a $\mathcal{CQ}$-$\mathcal{AL}^+$ query. An *approximation* q' of q is a $\mathcal{CQ}$-$\mathcal{AL}^+$ query obtained by replacing in q some subtree queries $q_1, \ldots, q_n$ by $Approx_{\uparrow}(q_1), \ldots, Approx_{\uparrow}(q_n)$.

However, an $\mathcal{AL}^+$ concept-atom can be subsumed by a query which is not necessarily a tree but a query for which a restriction is a forest query, i.e., it contains only tree subqueries ([Goasdoué and Rousset 2002]).

Definition 5.10 *Restriction of a query.* Let q be a query. A *restriction* of q is a query q' obtained by equating some variables of q with some other variables of q.

It follows from this definition that a query subsumes each of its restrictions. Rewriting approximations of a query's restrictions might lead to obtain maximal rewriting for that query.

Finally, a particular case concerns queries having boolean subqueries, i.e., a subquery that constains only existential variables that do not appear elsewhere in the query. Rewriting approximations of a complement of such a query or of one of its restrictions might lead to obtain maximal rewriting for that query.

Definition 5.11 *Complement of a query w.r.t a set of views.* A *complement* of a query q is a query obtained by adding in the body of q for some concept-atoms of the form $C(Y)$ *such that* Y *is an existential variable and there is no role-atom* $r(U,Y)$ *in* q, a conjunction of role-atoms $\bigwedge_{i=1}^{n} r_i(U_i,U_{i+1}) \wedge r(U_{i+1},Y)$ where $U_1,\ldots,U_{n+1}$ are fresh existential variables and the nesting $\forall^+ r_1.\cdots.\forall^+ r_{n+1}.\forall^+ r$ appears in the view definitions.

The following example intends to show how the above notions are used in our rewriting algorithm.

Example 5. Let us consider the two queries

$\mathsf{q}_1(X) :- \ \mathsf{Flight}(X) \wedge \mathsf{DeparturePlace}(X,Y) \wedge \mathsf{EurCity}(Y)$
$\qquad\qquad \wedge\ \mathsf{NightFlight}(Z) \wedge \mathsf{DeparturePlace}(Z,Y)$

$\mathsf{q}_2 :- \ \mathsf{HousingPlace}(Y)$

and assume that they are built over an ontology including the statements $\mathsf{NightFlight} \sqsubseteq \mathsf{Flight}$, $\mathsf{Hotel} \sqsubseteq \mathsf{HousingPlace}$ and $\mathsf{FrenchCity} \sqsubseteq \mathsf{EurCity}$.

In q_1, the user indicates that he wants flights whose departure places are european cities and such that there is some night flight that leaves those places.

q_2 is a boolean query expressing that the user wants to know if there exists sources providing housing places. Note that this query is not an atomic query that can be handled by *RewriteAtom* because its arity is 0 and not 1.

Let $\mathcal{V} = \{\mathsf{v}_1, \mathsf{v}_2\}$ be the set of available views such that:

$def(\mathsf{v}_1) = \ \mathsf{NightFlight} \sqcap \forall^+\mathsf{DeparturePlace.FrenchCity}$,

$def(\mathsf{v}_2) = \ \mathsf{Excursion} \sqcap \forall^+\mathsf{RestingPlace.Hotel}$.

v_1 stores instances of night flights whose departure places are french cities and v_2 stores instances of excursions whose resting places are hotels.

RewriteQuery will compute the following rewritings for q_1 and q_2 respectively:

$\mathsf{q}^1_{\mathcal{V}}(X) :- \ \mathsf{v}_1(X)$,

$\mathsf{q}^2_{\mathcal{V}} :- \ \mathsf{v}_2(Z)$.

$\mathsf{q}^1_{\mathcal{V}}$ is obtained by first computing the restriction q_1' of q_1:

$\mathsf{q}_1'(X) :- \ \mathsf{NightFlight}(X) \wedge \mathsf{DeparturePlace}(X,Y) \wedge \mathsf{EurCity}(Y)$.

This new query, which is subsumed by q_1, is a tree query and thus has an approximation:

$\mathsf{q}_1''(X) :- \ (\mathsf{NightFlight} \sqcap \forall^+\mathsf{DeparturePlace.EurCity})(X)$.

It is now easy to see that q_1'', which is subsumed by q_1', has a direct rewriting using $\mathcal{V}$: $\mathsf{q}^1_{\mathcal{V}}$.

$\mathsf{q}^2_{\mathcal{V}}$ is obtained by first computing the complement q_2' of q_2:

$\mathsf{q}_2' :- \ \mathsf{RestingPlace}(Z,Y) \wedge \mathsf{HousingPlace}(Y)$.

q_2' is subsumed by q_2 as well as its approximation $\mathsf{q}_2'' :- \ (\forall^+\mathsf{RestingPlace.HousingPlace})(Z)$.

It is now easy to see that q_2'' has a direct rewriting: $\mathsf{q}^2_{\mathcal{V}}$. $\square$

Given the above definitions, we can state the algorithm *RewriteQuery* that computes at least all the maximal $\mathcal{CQ}$ rewritings of $\mathcal{CQ}$-$\mathcal{AL}^+$ queries using $\mathcal{AL}^+$ views.

RewriteQuery algorithm
input: an $\mathcal{AL}^+$ normalized and satisfiable TBox $\mathcal{T}$, a normalized and satisfiable $\mathcal{CQ}$-$\mathcal{AL}^+$ query q over $\mathcal{T}$, and a set of normalized and satisfiable views $\mathcal{V}$ over $\mathcal{T}$.
output: a representative set of all the $\mathcal{CQ}$ rewritings of q.

$result = \emptyset$
for each q' among q and its normalized restrictions **do**
  for each q'' among q' and its normalized complements **do**
    for each q''' among q'' and its normalized approximations **do**
      let $q'''(\vec{X}) : - \bigwedge_{i=1}^{n} a_i$ be the definition of q'''
      for each conjunction $\mathcal{CJ} \in \bigotimes_{i=1}^{n} RewriteAtom(\mathcal{T}, a_i, \mathcal{V})$ **do**
        $result = result \cup \{q_v\}$ where q_v is defined by $q_v(\vec{X}) : -\mathcal{CJ}$
return *result*

We consider in turn the termination, soundness and compleness of *RewriteQuery*. Finally, we study its worst case time complexity.

Termination The termination of *RewriteQuery* follows from the termination of *RewriteAtom* and from the finite number of the restrictions (finite number of variables), complements (finite number of views) and approximations (finite number of atoms and thus of subtree queries) of a given $\mathcal{CQ}$-$\mathcal{AL}^+$ query.

Soundness and completeness Theorem 5.12 states the soundness and completeness of *RewriteQuery*.

THEOREM 5.12. *Let $\mathcal{T}$ be a normalized and satisfiable $\mathcal{AL}^+$ TBox. Let q be a normalized and satisfiable $\mathcal{CQ}$-$\mathcal{AL}^+$ query over $\mathcal{T}$, and $\mathcal{V}$ be a set of normalized and satisfiable views also over $\mathcal{T}$.*
Let result be the output of RewriteQuery$(\mathcal{T}, q, \mathcal{V})$:

—*each element of result is a $\mathcal{CQ}$ rewriting of q (Soundness) and*
—*every $\mathcal{CQ}$ rewriting of q has a body that includes (up to a renaming of existential variables) all the view-atoms of at least one element of result (Completeness).*

The proof of soundness is straightforward. It suffices to check that a restriction, a complement or an approximation of a $\mathcal{CQ}$-$\mathcal{AL}^+$ query q is subsumed by q. Thus, the $\mathcal{CQ}$ queries over $\mathcal{V}$ computed in the inner **for** of *RewriteQuery* are rewritings of the input query.

The full proof of completeness is in [Goasdoué 2001]. Here we only sketch it by mentioning the main lemmas.

It has to be shown that the body of any rewriting q_v of q includes (up to a renaming of the existential variables) the atoms appearing explicitly in a rewriting of the input query, or of a restriction, complement or approximation. Since *RewriteAtom* computes all the maximal rewritings of any atomic query (Theorem 5.2), the completeness of *RewriteQuery* then follows.

LEMMA 5.13. *Let q be a query of arity n defined by a conjunction of $\mathcal{AL}^+$ concept-atoms and role-atoms (denoted by body(q)). Let q_v be the query of same arity defined by a conjunction (denoted by body(q_v)) of view predicates of $\mathcal{V}$. Let $\vec{V}_q = [X_1, \ldots, X_n]$ and $\vec{V}_{q_v} = [Z_1, \ldots, Z_n]$ be the respective vectors of their distinguished variables, appearing in the head of their respective definitions. Let σ be the mapping from V_q to V_{q_v} such that $\sigma(X_i) = Z_i$ for every $i \in [1..n]$. Let expand(body(q_v)) be the conjunction of $\mathcal{AL}^+$ concept-atoms and role-atoms obtained by replacing in the definition of q_v every view predicate v by its definition. Then: q_v is a rewriting of q using $\mathcal{V}$ iff expand(body(q_v)), $\mathcal{T} \models \exists Y_1, \ldots, Y_k\, \sigma(body(q))$, where $Y_1, \ldots, Y_k$ are the existential variables of q and $\sigma(body(q))$ denotes the conjunction of atoms obtained from body(q) by replacing each distinguished variable X_i by $\sigma(X_i)$.*

Let q_v be a rewriting of q. Let $\vec{V}_q = [X_1, \cdots, X_n]$ and $\vec{V}_{q_v} = [Z_1, \cdots, Z_n]$ be the respective vectors of their distinguished variables, appearing in the head of their respective definitions. According to Lemma 5.13: $expand(body(q_v)), \mathcal{T} \models \exists Y_1, \ldots, Y_k\, \sigma(body(q))$, where $Y_1, \ldots, Y_k$ are the existential variables of q, and $\sigma(X_i) = Z_i$ for every $i \in [1..n]$.
Let q' be obtained from q by replacing its distinguished variables $[X_1, \cdots, X_n]$ by $[\sigma(X_1), \cdots, \sigma(X_n)]$. Depending on whether σ is injective or not, q' is equivalent to q or to one of its restrictions (obtained by equating those distinguished variables that have the same image by σ). By construction: $expand(body(q_v)), \mathcal{T} \models \exists Y_1, \ldots, Y_k\, body(q')$, and thus by Lemma 5.13, q_v is a rewriting of q'.
Therefore, without loss of generality, we can assume that the distinguished variables of q_v are included in the distinguished variables of q and that: $expand(body(q_v)), \mathcal{T} \models \exists Y_1, \ldots, Y_k\, body(q')$, where q' is either q or one of its restrictions. In any case, q_v is a rewriting of q'.

Let S be the completion of $expand(body(q_v))$, and I_S its canonical model. Let Δ_S be the objects of S. Δ_S is composed of the initial variables appearing in $expand(body(q_v))$ and of some generated variables which do not appear in *expand* $(body(q_v))$ but which have been added by the application of the $\rightarrow_{\forall+}$ propagation rule. I_S is a model of $\exists Y_1, \ldots, Y_k\, body(q')$. Therefore, there exists a mapping α from $\{Y_1, \ldots, Y_k\}$ to Δ_S such that for every role-atom $r(U_1, U_2)$ and every concept-atom $C(U)$ in $body(q')$ $r(\alpha(U_1), \alpha(U_2))$ is in S and $S \models C(\alpha(U))$.

If α is injective. We consider without loss of generality that, by renaming existential variables of the query, α maps every existential variable of q' to itself or to a generated variable.
Case 1: There is no existential variable that is mapped to a generated variable. It follows that every role-atom of the query appears in $expand(body(q_v))$ and $expand(body(q_v))$ entails every concept-atom of the query. q_v is then obtained by $\bigotimes_{i=1}^{n} RewriteAtom(a_i, \mathcal{V}, \mathcal{T})$ where $body(q') = \bigwedge_{i=1}^{n} a_i$.
Case 2: There exists some existential variables that are mapped to generated variables. Let Y be such a variable and v the generated variable such that $\alpha(Y) = v$.

—If Y has a predecessor U using r (i.e., there exists a role-atom $r(U, Y)$), U is its only predecessor since v, which is a generated variable, can have only one

predecessor in S. It follows from the choice of Y that $\alpha(U) = U$, $r(U,Y)$ is in $body(q')$ and $r(U,v)$ is in S.
We have shown in [Goasdoué 2001] that q' contains necessarily a tree subquery sq' rooted in U since α, which is injective, maps it to a tree subquery of S. It follows that sq' can be replaced by its approximation. Let q'' be q' in which we have performed the replacement. Let $\psi(U) = \{D \mid D(U) \in S\}$, we have $\bigwedge_{D\in\psi(U)} D(U) \models sq'$. Since $(\bigsqcap_{D\in\psi(U)} D)(U) \models Approx_{\uparrow}(sq')$ then $S \models q''$ and thus, from lemma 5.13, q_v is a rewriting of q''.

—Y does not have a predecessor. Let X be a variable of S that is not generated and such that there exists role-atoms of the form $r_0(X, v_1), \ldots, r_n(v_n, v)$ where $v_1, \ldots, v_n$ are generated variables. Let q_1 be the complement of q' obtained by adding the role-atoms $r_0(X, Y_1), \ldots, r_n(Y_n, Y)$ to its definition. This has been possible only if $\forall^+ r_0. \cdots .\forall^+ r_n.$ appears in a definition of a view of $\mathcal{V}$. By construction, q_v is a rewriting of q_1.

If the mapping α is not injective, let q'' be the restriction of q' obtained by equating the existential variables that have the same mapping. By construction, $expand(body(q_v)), \mathcal{T} \models \exists Y_{i_1}, \ldots, Y_{i_k}\ body(q'')$, where $Y_{i_1}, \ldots, Y_{i_k}$ are the existential variables of q", and thus (Lemma 5.13) q_v is rewriting of q''. The restriction of the mapping α to the existential variables of q'' is now injective. So, we are back to the previous case.

Worst case time complexity We give an upper bound of the worst case time complexity of *RewriteQuery*.

THEOREM 5.14. *The time complexity of RewriteQuery is at worst:*

—*polynomial in the size of the TBox,*

—*polynomial in the number of views and in their size,*

—*exponential in the size of the query, and*

—*factorial in the number of variables in the query.*

PROOF (SKETCH). We use the following parameters in addition of those already defined for *RewriteAtom*:

—n_a is the number of atoms in the query

—n_{var} is the number of variables in the query

In [Goasdoué 2001], we have shown that if we consider independently each **for** statement in *RewriteQuery*:

—an upper bound of the number of iterations performed in the first **for** is $n_{var}!$

—an upper bound of the number of iterations performed in the second **for** is $n_a.(c_v.d_v.v + 1)$

—an upper bound of the number of iterations performed in the third **for** is $2^{n_a.d_v}$

—an upper bound of the number of iterations performed in the fourth **for** is $2.(T_{RewriteAtom})^{n_a.d_v}$, where $T_{RewriteAtom}$ is the worst case time complexity of *RewriteAtom*.

It follows that if we consider the nesting of the **for** statements, the worst case complexity of *RewriteQuery* is: $n_{var}!.n_a.(c_v.d_v.v+1).2^{n_a.d_v+1}.(T_{RewriteAtom})^{n_a.d_v}$. □

6. CONCLUSION

In this paper, we have considered the problems of answering and rewriting queries using views in a uniform logical setting combining Datalog and description logics, and we have situated within that general setting most of the existing related work ([Baader et al. 2000; Beeri et al. 1997; Calvanese et al. 2000; Halevy 2001; Rousset 1999]). In particular, we have made clearer the connection between the problems of answering queries using views and rewriting queries using views.

Then, we have described a mediator framework centered on an ontology serving as a pivot vocabulary both for expressing users queries and for describing contents of varied and distributed information sources. This simple framework can be seen as a first-step towards the long-term vision of a real Semantic Web. Within this framework, we have studied from an algorithmic point of view the problem of answering queries by rewritings. In particular, we have provided an algorithm that generates a finite set of conjunctive rewritings and we have proved that the corresponding query plans are guaranteed to compute all the certain answers that can be obtained from the available distributed information sources.

The framework that we have presented is based on the $\mathcal{AL}^+$ description logic, which forces the combined use of value restriction and existential restriction for a given role through the constructor that we have denoted $\forall^+$. We have introduced this constructor because we have noticed that the standard formal semantics of value restrictions in description logics does not fit the intuitive semantics they are given by knowledge engineers. The constructor $\forall^+$ is appropriate for expressing typing information on roles in a natural way. In addition, by restricting the use of existential restrictions to be associated with value restrictions, it enabled us to bypass the inherent complexity raised by existential restrictions. Existential restrictions are a source of complexity which is well known in description logics. The subsumption problem is NP-complete for $\mathcal{FLE}$ while it is polynomial for description logics like $\mathcal{ALN}$ which do not allow existential restrictions. The presence of existential restriction makes the problem of instance checking strictly harder than the problem of subsumption ([Donini et al. 1994]). In particular, it has been shown in [Donini et al. 1994] that instance checking in $\mathcal{FLE}$ is coNP-hard even w.r.t. to data complexity only. This means that answering queries using views cannot be done with a polynomial data complexity except if some limitation on the use of existential restriction is imposed.

The centralized approach of mediation which we have considered in this paper is a particular case of a more general architecture of a distributed information system. The next step that we plan to investigate is a Peer-to-Peer architecture based on *distributed* ontologies serving as schemas of distributed data or services. The additional problem will be to describe and reason on relationships between ontologies.

REFERENCES

Abiteboul, S. and Duschka, O. M. 1998. Complexity of answering queries using materialized views. In *PODS '98. Proceedings of the Seventeenth ACM SIG-SIGMOD-SIGART Symposium* *on Principles of Database Systems*, ACM, Ed. ACM Press, New York, NY 10036, USA.

Baader, F., Küsters, R., and Molitor, R. 2000. Rewriting concepts using terminologies. In *Proceedings of the Seventh International Conference on Knowledge Representation and Reasoning (KR2000)*, A. Cohn, F. Giunchiglia, and B. Selman, Eds. Morgan Kaufmann Publishers, San Francisco, CA, 297–308.

Beeri, C., Levy, A., and Rousset, M.-C. 1997. Rewriting queries using views in description logics, editor = acm. In *PODS '97. Proceedings of the Sixteenth ACM SIG-SIGMOD-SIGART Symposium on Principles of Database Systems, May 12–14, 1997, Tucson, Arizona.* ACM Press, New York, NY 10036, USA.

Berners-Lee, T., Hendler, J., and Lassila, O. 2001. The semantic web. In *Scientific American.*

Calvanese, D., De Giacomo, G., and Lenzerini, M. 1998. On the decidability of query containment under constraints. In *PODS '98. Proceedings of the Seventeenth ACM SIG-SIGMOD-SIGART Symposium on Principles of Database Systems*, ACM, Ed. ACM Press, New York, NY 10036, USA.

Calvanese, D., De Giacomo, G., and Lenzerini, M. 2000. Answering queries using views in description logics. In *Proceedings of AAAI 2000.*

Calvanese, D., De Giacomo, G., Lenzerini, M., and Vardi, M. 2000b. Answering regular path queries using views. In *Proceedings of ICDE 2000.*

Calvanese, D., De Giacomo, G., Lenzerini, M., and Vardi, M. 2000a. View-based query processing and constraint satisfaction. In *Proceedings of LICS 2000.*

Donini, F. M., Lenzerini, M., Nardi, D., and Schaerf, A. 1994. Deduction in concept languages: From subsumption to instance checking. *Journal of Logic and Computation 4,* 4.

Duschka, O. M. and Genesereth, M. 1997. Query planning in infomaster. In *Proceedings of ACM Symposium on Applied Computing.*

Goasdoué, F. 2001. Réécriture de requêtes en termes de vues dans carin et intégration d'informations. Ph.D. thesis, Université Paris Sud XI - Orsay.

Goasdoué, F., Lattes, V., and Rousset, M.-C. 2000. The use of carin language and algorithms for information integration: the picsel system. *International Journal on Cooperative Information Systems 9*, 383–401.

Goasdoué, F. and Rousset, M.-C. 2002. Compilation and approximation of conjunctive queries by concept descriptions. In *ECAI 2002*, F. van Harmelen, Ed. 267–271. IOS Press.

Halevy, A. Y. 2001. Answering queries using views: A survey. *VLDB Journal: Very Large Data Bases 10,* 4, 270–294.

Levy, A., Rajamaran, A., and Ordille, J. 1996. Query-answering algorithms for information agents. In *Proceedings of the thirtenth AAAI conference on Artificial Intelligence: AAAI'96.*

Levy, A. and Rousset, M.-C. 1998a. Combining horn rules and description logics in carin. *Artificial Intelligence 101.*

Levy, A. Y. and Rousset, M.-C. 1998b. Verification of knowledge bases based on containment checking. *Artificial Intelligence 101,* 1–2, 227–250.

Mkbeem. http://www.mkbeem.com.

Nardi, D., Donini, F., Lenzerini, M., and Schaerf, A. 1995. *Principles of Artificial Intelligence.* Springer-Verlag, Chapter Reasoning in description logics.

Picsel. http://www.lri.fr/~picsel.

Reynaud, C. and Safar, B. 2002. Representation of ontologies for information integration. In *Proc. of the 13th International Conference on Knowledge Engineering and Knowledge Management (EKAW 2002).*

Rousset, M.-C. 1999. Backward reasoning in aboxes for query answering. In *Proceedings of DL'99 and KRDB'99.*