An Introduction to Ontology-Based Query Answering with Existential Rules

Marie-Laure Mugnier[1] and Michaël Thomazo[2]

[1] Université Montpellier 2
mugnier@lirmm.fr
[2] TU Dresden
michael.thomazo@tu-dresden.de

Abstract. The need for an ontological layer on top of data, associated with advanced reasoning mechanisms able to exploit ontological knowledge, has been acknowledged in the database, knowledge representation and Semantic Web communities. We focus here on the ontology-based data querying problem, which consists in querying data while taking ontological knowledge into account. To tackle this problem, we consider a logical framework based on existential rules, also called Datalog$^{\pm}$.
In this course, we introduce fundamental notions on ontology-based query answering with existential rules. We present basic reasoning techniques, explain the relationships with other formalisms such as lightweight description logics, and review decidability results as well as associated algorithms. We end with ongoing research and some challenging issues.

Table of Contents

1 Ontology-Based Query Answering

Novel intelligent methods are required to manage and exploit the huge amounts of data nowadays available. The interest of considering *ontological knowledge* when accessing data has been widely acknowledged, both in the database and knowledge representation communities. Indeed, ontologies[3], which typically encode general domain knowledge, can be used to infer data that are not explicitely stored, hence palliating incompleteness in databases. They can also be used to enrich the vocabulary of data sources, which allows a user to abstract from the specific way data are stored. Finally, when several data sources use different vocabularies, ontologies can be used to unify these vocabularies.

Example 1. In this simple example, we consider data on movies, described with unary relations *MovieTitle* (titles of movies) and *MovieActor* (movie actors), and a binary relation *Play*, encoding that a given person plays a role in a movie identified by its title. Let q be a query asking if a given person, whose identifier

[3] We will reserve the term "ontology" to general domain knowledge—also called terminological knowledge—in order to clearly distinguish it from the data—or assertional knowledge—called here facts.

is B, plays in a movie. If the data do not explicitely contain the information that B plays in a movie, the answer to q will be no. Now, assume that the data contain the information that B is a movie actor (B is in the extension of *MovieActor* relation) and we have the knowledge that "every movie actor plays a role in some movie". We can infer that B plays in a movie. Hence, the answer to q should be yes. In the following, we will encode knowledge in First-Order logic. Then, the query will be encoded as $q = \exists y\ Play(B, y)$ and the piece of ontological knowledge as the following rule:

$$R = \forall x(MovieActor(x) \rightarrow \exists y(Play(x, y) \wedge MovieTitle(y)))$$

We can check that the ground atom *MovieActor(B)* and the rule R entail q, hence the positive answer to q.

The issue of proposing formalisms able to express ontological knowledge, associated with querying mechanisms able to exploit this knowledge when accessing data, is known as ontology-based data access. In this paper, we will more precisely consider the following problem, called *ontology-based query answering*: given a knowledge base composed of an ontology and facts, and a query, compute the set of answers to the query on the facts, while taking implicit knowledge represented in the ontology into account. We will consider the basic queries called conjunctive queries.

In the Semantic Web, ontological knowledge is usually represented with formalisms based on *description logics* (DLs). However, DLs are restricted in terms of expressivity, in the sense that terminological knowledge can only be expressed by tree-like structures. Moreover, only unary and binary predicates are generally supported. Historically, DLs focused on reasoning tasks about the terminology itself, for instance classifying concepts; querying tasks were restricted to ground atom entailment [BCM$^+$07]. Conjunctive query answering with classical DLs appeared to be extremely complex (e.g., for the classical DL $\mathcal{ALCI}$, it is 2ExpTime-complete, and still NP-complete in the size of the data). Hence, less expressive DLs specially devoted to conjunctive query answering on large amounts of data have been designed, beginning with the DL-Lite family. Another family of lightweight DLs used for query answering is the $\mathcal{EL}$ family, which was originally designed for polynomial time terminological reasoning. These DLs form the core of so-called tractable profiles of the Semantic Web language OWL 2, namely OWL 2 QL and OWL2 EL (the third tractable profile being OWL 2 RL, which is closely related to the rule-based language Datalog).

On the other hand, querying large amounts of data is the fundamental task of databases. Therefore, the challenge in this domain is now to access data while taking ontological knowledge into account. The deductive database language *Datalog* allows to express some ontological knowledge. However, in Datalog rules, variables are range-restricted, i.e., all variables in the rule head necessarily occur in the rule body. Therefore, these rules can produce knowledge about already known individuals, but cannot infer the existence of unknown individuals, a feature sometimes called "value invention" in databases. This feature has been

recognized as crucial in an open-domain perspective, where it cannot be assumed that all individuals are known in advance.

Existential rules have been proposed to meet these two requirements, i.e., value invention and the ability to express complex structures. Existential rules extend First-Order Horn clauses, i.e., plain Datalog rules, by allowing to introduce new existentially quantified variables. They are also known as Datalog$^{\pm}$ family, in reference to Datalog. More preciselly, an existential rule is a positive rule of the form $body \rightarrow head$, where *body* and *head* are any conjunctions of atoms, and variables occurring only in the *head* are existentially quantified. The rule in Example 1 is an existential rule; it allows to infer the existence of a movie in which B plays a role, even if this movie is not identified.

This paper provides an introduction to ontological query answering with existential rules. In Section 2, we present basic logical foundations for representing and reasoning with facts, conjunctive queries and plain Datalog rules. Section 3 is devoted to existential rules and their relationships with other formalisms (tuple-generating dependencies in databases, conceptual graph rules, positive logic progams via skolemization, lightweight description logics). Section 4 introduces the main approaches to solve the problem. Entailment with general existential rules being undecidable, Section 5 presents the main decidability criteria currently known, as well as associated algorithmic techniques. Section 6 ends with ongoing research and open issues.

We purposely omitted bibliographical references in this introductive section. References will be given later, in the appropriate sections.

2 Fundamental Notions on Conjunctive Query Answering and Positive Rules

Data can be stored in various forms, for instance in a relational database, an RDF triple store or a graph database. We abstract from a specific language or technology by considering first-order logic (FOL). Ontological knowledge will also be expressed as first-order logical formulas. In this section, we present basic theoretical notions for representing and reasoning with facts, conjunctive queries, as well as simple positive rules, namely Datalog rules.

2.1 Basic Logical Notions

We consider first-order vocabularies with constants but no other function symbols. A *vocabulary* is a pair $\mathcal{V} = (\mathcal{P}, \mathcal{C})$, where $\mathcal{P}$ is a finite set of predicates (or relations) and $\mathcal{C}$ is a possibly infinite set of constants. Each predicate has an *arity*, which is its number of arguments. A *term* (on $\mathcal{V}$) is a variable or a constant (in $\mathcal{C}$). An *atom* (on $\mathcal{V}$) is of the form $p(t_1, \ldots, t_k)$ where p is a predicate of arity k (from $\mathcal{P}$) and the t_i are terms (on $\mathcal{V}$). An atom is *ground* if it has no variable. A formula *on* $\mathcal{V}$ has all its atoms on $\mathcal{V}$. A variable in a formula is *free* if it is not in the scope of a quantifier. A formula is *closed* if it has no free variable.

Given a formula F, we note $terms(F)$ (respectively $vars(F)$, $csts(F)$) the set of terms (respectively variables, constants) that occur in F. In the following, we always assume that distinct formulas (representing facts, queries or rules) have *disjoint sets of variables*.

An *interpretation* I of a vocabulary $\mathcal{V} = (\mathcal{P}, \mathcal{C})$ is a pair $(D, .^I)$, where D is a non-empty (possibly infinite) set, called the *domain* of I, and $.^I$ defines the semantics of the elements in $\mathcal{V}$ with respect to D:

- $\forall c \in \mathcal{C}$, $c^I \in D$
- $\forall p \in \mathcal{P}$ with arity k, $p^I \subseteq D^k$.

An interpretation I of $\mathcal{V}$ is a *model* of a formula F on $\mathcal{V}$ if F is true in I. A formula G *entails* a formula F (we also say that F is a *semantic consequence* of G) if every model of G is a model of F, which is denoted by $G \models F$. We note $G \equiv F$ if $G \models F$ and $F \models G$.

2.2 The Positive Existential Conjunctive Fragment of FOL

The *positive existential conjunctive* fragment of first-order logic, denoted by FOL$(\wedge, \exists)$, is composed of formulas built with the single connector $\wedge$ and the single quantifier $\exists$. Without loss of generality, we consider that these formulas are in prenex form, i.e., all existential quantifiers are in front of the formula. Then, it is often convenient to see them as *sets of atoms*. As we will see later on, this fragment allows to represent facts and conjunctive queries.

A fundamental notion in FOL$(\wedge, \exists)$ is that of a homomorphism. We recall that a *substitution* s of a set of variables V by a set of terms T is a mapping from V to T. Given a set of atoms F, $s(F)$ denotes the set obtained from F by applying s, i.e., by replacing each variable $v \in V$ with $s(v)$.

Definition 1 (Homomorphism, notation $\geq$). *Given two sets of atoms F and G, a* homomorphism *h from F to G is a substitution of $vars(F)$ by $terms(G)$ such that, for all atom $p(t_1 \dots t_k) \in F$, $p(h(t_1) \dots h(t_k)) \in G$, i.e., $h(F) \subseteq G$. We note $F \geq G$ if there is a homomorphism from F to G, and say that F is* more general than *G.*

Example 2. Let us consider the facts $F_1 = \{p(x_1, y_1), p(y_1, z_1), p(z_1, x_1)\}$ and $F_2 = \{p(x_2, y_2), p(y_2, z_2), p(z_2, u_2)\}$, where all terms are variables. There are three homomorphisms from F_2 to F_1. For instance, $h = \{x_2 \mapsto x_1, y_2 \mapsto y_1, z_2 \mapsto z_1, u_2 \mapsto x_1\}$.

Homomorphism can also be defined among interpretations. Given two interpretations $I_1 = (D_1, .^{I_1})$ and $I_2(D_2, .^{I_2})$ of a vocabulary $\mathcal{V} = (\mathcal{P}, \mathcal{C})$, a homomorphism from I_1 to I_2 is a mapping from D_1 to D_2 such that:

- for all $c \in \mathcal{C}$, $h(c^{I_1}) = c^{I_2}$;
- for all $p \in \mathcal{P}$ and $(t_1 \dots t_k) \in p^{I_1}$, $(h(t_1) \dots h(t_k)) \in p^{I_2}$.

We first point out that an interpretation I is a model of a FOL$(\wedge, \exists)$ formula, if and only if there is a mapping v from $terms(F)$ to D such that:

- for all $c \in consts(F)$, $v(c) = c^I$;
- for all atom $p(e_1 \dots e_k) \in F$, $(v(e_1) \dots v(e_k)) \in p^I$.

Such a mapping is called a *good assignment* of F to I.

A nice property of FOL$(\wedge, \exists)$ is that each formula has a *canonical model*, which is a representative of all its models, and can be used to check entailment, as we will see below. This model has the same structure as F, hence the name "isomorphic model".

Definition 2 (Isomorphic model). *Let F be a FOL$(\wedge, \exists)$-formula built on the vocabulary $\mathcal{V} = (\mathcal{P}, \mathcal{C})$. The* isomorphic model *of F, denoted by $M(F)$, :*

- *D is in bijection[4] with $terms(F) \cup \mathcal{C}$ (to simplify notations, we consider that this bijection is the identity);*
- *for all $c \in \mathcal{C}$, $M(c) = c$;*
- *for all $p \in \mathcal{P}$, $M(p) = \{(t_1 \dots t_k) | p(t_1 \dots t_k) \in F\}$ if p occurs in F, otherwise $M(p) = \emptyset$.*

We check that $M(F)$ is indeed a model of F, by choosing the identity as good assignment.

Property 1. For any FOL$(\wedge, \exists)$ formula F, $M(F)$, the model isomorphic to F, is a universal model, i.e., for all model M' of F, it holds that $M(F) \geq M'$.

Proof. If M' is a model of F, then there is a good assignment v from F to M'. Since $M(F)$ is isomorphic to F, v defines a homomorphism from $M(F)$ to M'.

Given two interpretations I_1 and I_2, with $I_1 \geq I_2$, if I_1 is a model of F, then I_2 also is. Indeed, the composition of a homomorphism from I_1 to I_2 and of a good assignment from F to I_1 yields a good assignment. Hence, to check if G entails F, it is sufficient to check that $M(G)$ is a model of F, i.e., there is a good assignment from F to $M(G)$.

Note that a good assignment from F to $M(G)$ defines a homomorphism from F to G. Reciprocally, a homomorphism from F to G defines a good assignment from F to $M(G)$. It follows that checking entailment in the FOL$(\wedge, \exists)$ fragment amounts to a homomorphism check:

Theorem 1. *Let F and G be two FOL$(\wedge, \exists)$ formulas. It holds that $G \models F$ iff there is a homomorphism from F to G (i.e., $F \geq G$).*

Proof. Follows from previous definitions and Property 1.

If $F \geq G$ and $G \geq F$, we say that F and G are (homomorphically) *equivalent*. According to the preceding theorem, this equivalence notion corresponds to logical equivalence.

Definition 3 (Core). *Given a set of atoms F, the* core *of F is a minimal subset of F equivalent to F.*

[4] A bijection is a one-to-one correspondence.

It is well-known that among all equivalent sets of atoms on a vocabulary, there is a unique core, up to variable renaming (where a *variable renaming*, or *isomorphism*, from F to F', is a bijective substitution s of $vars(F)$ by $vars(F')$ such that $F = F'$).

2.3 Facts and Conjunctive Queries

Classically, a fact is a ground atom. We extend this notion, so that a fact may contain existentially quantified variables and not only constants. Hence, a fact becomes a closed FOL$(\wedge, \exists)$ formula. This allows to represent in a natural way null values in relational databases or blank nodes in RDF. Moreover, this is in line with existential rules, which produce existential variables. It follows that a conjunction of facts can be seen as a single fact when it is put in prenex form.

Example 3. Let $F_1 = \exists x \exists y (p(x, y) \wedge q(y, a))$, where a is a constant, and $F_2 = \exists x p(x, x)$. F_1 and F_2 are facts. The formula $F_1 \wedge F_2$ can be seen as a single fact obtained by considering a prenex form, which involves renaming the variable x in F_1 or in F_2. One obtains for instance $\exists x \exists y \exists z (p(x, y) \wedge q(y, a) \wedge p(z, z))$, which can also be seen as the set of atoms $\{p(x, y), q(y, a), p(z, z)\}$.

Conjunctive queries are the basic and more frequent queries in databases. There can be expressed in FOL$(\wedge, \exists)$. They correspond to SELECT-FROM-WHERE queries in SQL and to basic pattern queries in SPARQL.

Definition 4 (Facts, queries, answers). *A* fact *is an existentially closed conjunction of atoms. A* conjunctive query *(CQ) is an existentially quantified conjunction of atoms with possibly free variables. A* Boolean *conjunctive query (BCQ) has no free variable. Let $\{x_1 \dots x_k\}$ be the free variables in a CQ q; an* answer *to q in a fact F is a substitution s of $\{x_1 \dots x_k\}$ by constants in F, such that $F \models s(q)$ (in other words, s is the restriction to $\{x_1 \dots x_k\}$ of a homomorphism from q to F). Given an ordering $(x_1 \dots x_k)$ of the free variables in q, we often denote an answer s by $(s(x_1) \dots s(x_k))$. A Boolean query q has only the empty substitution as possible answer, in which case q is said to have a positive answer, otherwise q has a negative answer.*

Several equivalent basic decision problems can be considered. Let $\mathcal{K}$ be a knowledge base (composed of facts for now, but later enriched with rules); then the following decision problems are polynomially equivalent (see e.g. [BLMS11]):

- CQ ANSWERING decision problem: given a KB $\mathcal{K}$ and a CQ q, is there an answer to q in $\mathcal{K}$?
- CQ EVALUATION decision problem: given a KB $\mathcal{K}$, a CQ q and a list of constants t, is t an answer to q in $\mathcal{K}$?
- BCQ ANSWERING problem: given a KB $\mathcal{K}$ and a BCQ q, is () an answer to q in $\mathcal{K}$?
- BCQ ENTAILMENT problem: given a KB $\mathcal{K}$ and a BCQ q, is q entailed by $\mathcal{K}$?

In the following, we consider BCQ ENTAILMENT as a reference problem.

Checking homomorphism is an NP-complete problem. Hence, BCQ ENTAILMENT is NP-complete. Instead of the classical complexity measure, also called combined complexity, we may also consider *data complexity*, in which case only the data are part of the problem input. Then, BCQ ENTAILMENT becomes polynomial: indeed, a naive algorithm for checking if there is a homomorphism from q to F is in $\mathcal{O}(|F|^{|q|})$.

2.4 Adding Positive Range-Restricted Rules

A basic kind of ontological knowledge is that of *range-restricted* positive rules, where "range-restricted" means that all variables in the head of a rule also occur in the body of this rule [AHV95]. It follows that such rules allows to entail knowledge on individuals that *already exist* in the data. In ontologies, such rules typically express taxonomies (like a schema in RDFS), or properties of relations (like symmetry or transitivity). We will also refer to these rules as *Datalog rules*, as they exactly correspond to plain Datalog rules (i.e., without negation).

Definition 5 (Range-restricted rule, Datalog rule). *A* range-restricted (positive) rule*, or (plain) Datalog rule, is a formula* $R = \forall \boldsymbol{x} \forall \boldsymbol{y}(B[\boldsymbol{x}, \boldsymbol{y}] \rightarrow H[\boldsymbol{y}])$*, where* $\boldsymbol{x}, \boldsymbol{y}$ *are sets of variables,* B *and* H *are conjunctions of atoms, respectively called the* body *and the* head *of* R*, also denoted by* $body(R)$ *and* $head(R)$*.*

A rule R is *applicable* to a fact F if there is a homomorphism h from $body(R)$ to F. Applying R to F with respect to h consists in adding $h(head(R))$ to F. By iteratively applying rules in all possible ways, one obtain a unique fact, called the *saturation* of F, and denoted by F^*. The process stops in finite time since no new variable is created.

Let us now consider a knowledge base composed of facts (seen as a single fact) and range-restricted rules, $\mathcal{K} = (F, \mathcal{R})$. To check if $\mathcal{K} \models q$, we can rely on notions similar to the positive existential case. Indeed, the model isomorphic to F^* is a model of $(F, \mathcal{R})$ and it keeps the property of being a universal model.

Hence, $\mathcal{K} \models q$ if and only if there is a homomorphism from q to F^*. In combined complexity, this test is still NP-complete if the arity of the predicates is bounded (then the size of F^* is polynomial in the size of F), otherwise it is EXPTIME-complete [AHV95]. It is polynomial with respect to data complexity.

In the next section, we extend positive rules to existential rules, by relaxing the constraint of being range-restricted.

3 Existential Rules

In this section, we present existential rules, as well as their relationships to other database or KR formalisms.

3.1 The Framework of Existential Rules

Definition 6 (Existential rule). *An* existential rule *(or simply a* rule *hereafter) is a formula* $R = \forall \boldsymbol{x} \forall \boldsymbol{y}(B[\boldsymbol{x}, \boldsymbol{y}] \rightarrow \exists \boldsymbol{z} H[\boldsymbol{y}, \boldsymbol{z}])$*, where* $\boldsymbol{x}, \boldsymbol{y}$ *and* $\boldsymbol{z}$ *are sets of variables,* B *and* H *are conjunctions of atoms, respectively called the* body *and the* head *of* R*, also denoted by* $body(R)$ *and* $head(R)$*. The* frontier *of* R*, denoted by* $fr(R)$*, is the set* $vars(B) \cap vars(H) = \boldsymbol{y}$*. The set of* existential variables *in* R *is the set* $vars(H) \setminus fr(R) = \boldsymbol{z}$*.*

In the following, we will omit quantifiers in rules since there is no ambiguity. For instance, $p(x, y) \rightarrow q(y, z)$ denotes the rule $R = \forall x \forall y \ (p(x, y) \rightarrow \exists z \ q(y, z))$.

Example 4. Consider the following predicates, with their arity mentioned in parentheses; unary predicates can be seen as concept names, i.e. types of entities, and the other predicates as relation names: Area(1), Project(1), Researcher(1), isProject(3), hasExpertise(2), isMember(2)
Here are some examples of existential rules composing the ontology:
"The relation isProject associates a project, the area of this project and the leader of this project, who is a researcher" [signature of isProject]
$R_0 = isProject(x, y, z) \rightarrow Project(x) \wedge \ Area(y) \wedge \ Researcher(z)$
"Every leader of a project is a member of this project"
$R_1 = isProject(x, y, z) \rightarrow isMember(z, x)$
"Every researcher expert in an area is member of a project in this area"
$R_2 = Researcher(x) \wedge hasExpertise(x, y) \rightarrow isProject(u, y, z) \wedge isMember(x, u)$
"Every researcher is expert in an area"
$R_3 = Researcher(x) \rightarrow hasExpertise(x, y)$
R_0 and R_1 are range-restricted, but not R_2 and R_3.

Definition 7 (Application of an existential rule). *An existential rule* R *is* applicable *to a fact* F *if there is a homomorphism* h *from* $body(R)$ *to* F*; the result of the* application of R to F with respect to h *is a fact* $\alpha(F, R, \pi) = F \cup \pi^{\text{safe}}(head(R))$ *where* π^{safe} *is a substitution of* $head(R)$*, that replaces each* $x \in fr(R)$ *with* $h(x)$*, and each other variable with a "fresh" variable, i.e., not introduced before.*

Example 5. We consider the vocabulary and rules from Example 4. Let $F = \{Researcher(a), Researcher(b), hasExpertise(a, \text{“}KR\text{”}), Area(\text{“}KR\text{”})\}$ be a fact. R_2 is applicable to F with respect to $h_0 = \{(x, a), (y, \text{“}KR\text{”})\}$, which yields atoms $\{isProject(u_0, \text{”}KR\text{”}, z_0), isMember(a, u_0)\}$, where u_0 and z_0 are fresh existential variables. R_3 is applicable to F as well, with $h_1 = \{(x, b)\}$, which produces the atom $hasExpertise(b, y_0)$. Then, R_2 could be applied again, to the obtained fact, with respect to $h_2 = \{(x, b), (y, y_0)\}$, which would produce atoms $\{isProject(u_1, y_0, z_1), isMember(b, u_1)\}$.

Existential rules have a double origin. On the one hand, they can be seen as an extension of plain Datalog to enable value invention, yielding the Datalog$^{\pm}$ family [CGK08,CGL09]. It is important to note, however, that rules make a query in Datalog, while, in Datalog$^{\pm}$ (and in the present framework), they form an

ontology, and the query itself does not embed deductive knowledge. On the other hand, existential rules come from earlier studies on a graph-based knowledge representation framework, inspired by conceptual graphs [CM09]; indeed, the logical translation of conceptual graph rules yields exactly existential rules, as defined in [SM96].

We also point out that existential rules have the same form as a very general kind of dependencies, which has long been studied in databases, namely *Tuple-Generating Dependencies* (TGD) [AHV95]. Intuitively, a database instance D satisfies a TGD t if, each time the body of t is found in D, the head of t is found in D as well; formally, if t is applicable to D by homomorphism h, then h has to be extensible to a homomorphism h' from $\mathit{head}(t)$ to D, i.e., such that $h(x) = h'(x)$ for each $x \in \mathit{fr}(t)$. Existential rules benefit from theoretical results obtained on TGDs (such as results on the chase and on decidability issues, see Sections 4 and 5).

Note that an existential rule is not a Horn clause because of existential variables in its head. However, both are closely related, since by *skolemisation*, an existential rule can be transformed into a set of Horn clauses with functions. The skolemization of a rule R is a rule $\mathit{skolem}(R)$ built from R by replacing each occurrence of an existential variable y with a functional term $f_y^R(\boldsymbol{x})$, where $\boldsymbol{x} = \mathit{fr}(R)$. We remind that skolemization does not produce an equivalent formula, however a formula is (un)satisfiable if and only if its skolem form is.

Example 6 (Skolemization). Let $R = \mathit{Researcher}(x) \wedge \mathit{hasExpertise}(x, y) \rightarrow \mathit{isProject}(u, y, z) \wedge \mathit{isMember}(x, u)$ (Rule R_2 in Example 4). The frontier of R_2 is $\{x, y\}$. Then, $\mathrm{skolem}(R) = \mathit{Researcher}(x) \wedge \mathit{hasExpertise}(x, y) \rightarrow \mathit{isProject}(f_u^R(x, y), y, f_z^R(x, y)) \wedge \mathit{isMember}(x, f_u^R(x, y))$, which yields two Horn clauses.

A set of skolemized existential rules can be seen as a specific positive logic program, hence nonmonotonic negation can be added while benefitting from results obtained in logic programming. For instance, if nonmonotonic negation is added with stable model semantics, one obtains a specific case of Answer Set Programming.

This framework can be extended to *equality rules* and *negative constraints*. An equality rule is a rule of the form $B \rightarrow x = t$, where x and t are distinct terms, $x \in \mathit{vars}(B)$ and $t \in \mathit{vars}(B)$ or is a constant. When the unique name assumption is made, i.e., distinct constants refer to distinct individuals, the application of an equality rule is said to fail if it leads to set the equality between distinct constants. This kind of failure corresponds to an inconsistency of the knowledge base. Equality rules generalize functional dependencies, which are widely used in conceptual modeling.

Constraints are another kind of construct specifically devoted to the definition of the consistency or inconsistency of the knowledge base. A *negative* constraint is a rule of the form $C \rightarrow \bot$, where $\bot$ denotes the absurd symbol (i.e., a propositional atom whose value is false). It is satisfied if C is not entailed by $(F, \mathcal{R})$. Negative constraints are typically used to express disjointness

of concepts/classes or incompatibility of relations. See [CGL09] and [BLMS11] for the integration of equality rules and negative constraints in the existential rule framework.

3.2 Relationships to Lightweight Description Logics

Interestingly, existential rules generalize lightweight description logics. We focus here on $\mathcal{EL}$ [BBL05,LTW09] and DL-Lite [CGL$^+$07,ACKZ09]. For instance, DL-Lite$_\mathcal{R}$, which underlines OWL2 QL profile [OWL09], can be expressed by linear existential rules (whose body and head are restricted to a single atom) and negative constraints, see e.g. [CGL09]. Other DL-Lite fragments allow to declare functional roles, which can be translated by equality rules. Table 1 summarizes the translation from DL-Lite axioms to existential rules. Example 7 illustrates this translation on a concrete case. The DL $\mathcal{EL}$ can be expressed by "pure" existential rules, as shown in Table 2.

DL-Axiom	Translated rule
$A \sqsubseteq B$	$A(x) \rightarrow B(x)$
$A \sqsubseteq \exists R$	$A(x) \rightarrow R(x, y)$
$\exists R \sqsubseteq \exists S^-$	$R(x, y) \rightarrow S(z, x)$
$B \sqsubseteq \exists R.C$	$B(x) \rightarrow R(x, y) \wedge C(y)$
$R \sqsubseteq S$	$R(x, y) \rightarrow S(x, y)$
$B \sqsubseteq \neg C$	$B(x) \wedge C(x) \rightarrow \bot$
$funct(R)$	$R(x, y) \wedge R(x, z) \rightarrow y = z$

Table 1. Translation of DL-Lite axioms

DL-Axiom	Translated rule
$B \sqcap C \sqsubseteq D$	$B(x) \wedge C(x) \rightarrow D(x)$
$B \sqsubseteq C$	$B(x) \rightarrow C(x)$
$B \sqsubseteq \exists R.C$	$B(x) \rightarrow R(x, y) \wedge C(y)$
$\exists R.B \sqsubseteq C$	$R(x, y) \wedge B(y) \rightarrow C(x)$

Table 2. Translation of (normal) $\mathcal{EL}$-axioms

Example 7. This example borrows a DL-Lite$_\mathcal{R}$ TBox from [CGL$^+$07]. Consider the atomic concepts *Professor* and *Student*, and the roles *TeachesTo* and *HasTutor*. The TBox is listed below with its translation into existential rules.

DL-Lite$_\mathcal{R}$ TBox	Existential rules
$Professor \sqsubseteq \exists TeachesTo$	$Professor(x) \rightarrow TeachesTo(x, y)$
$Student \sqsubseteq \exists HasTutor$	$Student(x) \rightarrow HasTutor(x, y)$
$\exists TeachesTo^- \sqsubseteq Student$	$TeachsTo(y, x) \rightarrow Student(x)$
$\exists HasTutor^- \sqsubseteq Professor$	$HasTutor(y, x) \rightarrow Professor(x)$
$Professor \sqsubseteq \neg Student$	$Professor(x) \wedge Student(x) \rightarrow \bot$
$HasTutor^- \sqsubseteq TeachesTo$	$HasTutor(y, x) \rightarrow TeachesTo(x, y)$

In DL-Lite$_\mathcal{F}$, the role *HasTutor* could be declared functional (by the statement (*funct* ($HasTutor$)), which would be translated into the following equality rule: $HasTutor(x, y) \wedge HasTutor(x, z) \rightarrow y = z$.

More generally, existential rules allow to overcome some limitations of light-weight DLs. First, they have unrestricted predicate arity (while DLs consider unary and binary predicates only), which allows for a natural coupling with database schemas, in which relations may have any arity. Moreover, adding pieces of information, for instance to take contextual knowledge into account, such as data provenance for instance, is made easy by the unrestricted predicate arity, since these pieces of information can be added as new predicate arguments. Second, the body and the head of a rule may have any structure, and there is no constraint on frontier variables, hence rules allow to represent cyclic structures, while DLs are fundamentally restricted to tree-like structures.

Example 8. The following rule cannot be expressed in DLs:

$$p(x, y) \rightarrow q(x, z) \wedge q(y, z)$$

Unsurprisingly, there is a price to pay for this expressivity. Indeed, BCQ Entailment is undecidable for general existential rules (e.g., [BV81,CLM81] for an equivalent problem on TGDs, and [BM02] for fact entailment with conceptual graph rules). However, many classes of rules for which it remains decidable have been studied. The main classes are reviewed in Section 5.

4 Main Approaches to Ontology-Based Query Answering

We now consider a knowledge base (KB) $\mathcal{K} = (F, \mathcal{R})$, composed of a set of facts, seen as a single fact F, and a finite set of existential rules $\mathcal{R}$. We recall that the BCQ entailment problem takes as input a KB $\mathcal{K} = (F, \mathcal{R})$ and a BCQ q, and asks if $\mathcal{K} \models q$ holds, where $\mathcal{K}$ is seen as the conjunction of F and the rules in $\mathcal{R}$.

To solve this problem, there are two main approaches, which are related to the classical paradigms for processing rules, namely *forward chaining* and *backward chaining*. In databases, these paradigms are also called *bottom-up* and *top-down*, respectively. Forward chaining consists in iteratively applying the rules starting from the initial fact, while trying to produce a fact to which the query can be mapped by homomorphism. Backward chaining consists in iteratively using the

rules to rewrite the query, starting from the initial query, while trying to produce a query that can be mapped to the initial fact by homomorphism.

In the OBQA context, these two paradigms are recast as follows. Forward chaining is used to *materialize* all inferences in the data, then the query is evaluated against this materialization. Backward chaining is decomposed into two steps as well. First, the query is rewritten into another query using the rules. Then, the rewritten query is evaluated against the initial data. Both approaches can be seen as ways of integrating the rules, respectively into the data and into the query, in order to come back to a classical database query evaluation problem.

Materialization has the advantage of enabling efficient query answering but may be not appropriate, because the saturated data may be too large, but also because of data access rights or data maintenance reasons. Query rewriting has the advantage of avoiding changes in the data, however its drawback is that the rewritten query may be large, even exponential in the size of initial query, hence less efficiently processed, at least with current database techniques.

Since BCQ ENTAILMENT is not decidable, none of these techniques leads to a procedure that terminates in *all* cases. Various conditions on rules ensuring the decidability of BCQ ENTAILMENT have been exhibited. These conditions ensure the termination of algorithms based on materialization or on query rewriting, or on a combination of both. See the next section for an overview.

We now present the main notions and results that underly these two approaches.

4.1 Materialization-based approach

We already pointed out that existential rules have the same form as TGDs. Forward chaining on TGDs is known as the *chase*. It was initially designed to repair a database that violates some TGDs. Indeed, when the database does not satisfy a TGD (according to homomorphism h), this TGD can be applied to the database (according to h) to add missing data. However, these local repairs may lead to a new TGD violation, hence the forward chaining process.

Definition 8 (Derivation Sequence). *Let F be a fact, and $\mathcal{R}$ be a set of rules. An* $\mathcal{R}$-derivation *of F is a finite sequence $(F_0 = F), \ldots, F_k$ such that for all $0 \leq i < k$, there is $R_i \in \mathcal{R}$ and a homomorphism h from $body(R_i)$ to F_i such that $F_{i+1} = \alpha(F_i, R_i, h)$.*

Theorem 2 (Soundness and completeness of $\mathcal{R}$-derivation). *Let $\mathcal{K} = (F, \mathcal{R})$ be a KB and q be a Boolean conjunctive query. Then $\mathcal{K} \models q$ iff there exists an $\mathcal{R}$-derivation $(F_0 = F), \ldots, F_k$ such that $F_k \models q$.*

It follows that a breadth-first forward chaining mechanism yields a positive answer in finite time when $\mathcal{K} \models q$. This mechanism, called the *saturation* hereafter (and the *chase* in databases) works as follows. Let $F_0 = F$ be the initial fact. Each step is as follows: (1) check if q maps to the current fact, say F_{i-1} at

step i ($i > 0$): if it is the case, q has a positive answer; (2) otherwise, produce a fact F_i from F_{i-1}, by computing all new homomorphisms from each rule body to F_{i-1}, then performing all corresponding rule applications. A homomorphism to F_{i-1} is said to be *new* if it has not been already computed at a previous step, i.e., it uses at least an atom added at step $i-1$. The fact F_k obtained after the step k is called the k-saturation of F and is denoted by $\alpha_k(F, \mathcal{R})$. Formally: let $\alpha(F, \mathcal{R}) = F \cup \{\pi_{safe}(head(R)), \forall R \in \mathcal{R}$ and homomorphism $\pi : body(R) \to F\}$; then, $\alpha_0(F, \mathcal{R}) = F$ and for $i > 0$, $\alpha_i(F, \mathcal{R}) = \alpha(\alpha_{i-1}(F, \mathcal{R}), \mathcal{R})$.

We define $\alpha_\infty(F, \mathcal{R}) = \cup_{k \geq 0} \alpha_k(F, \mathcal{R})$ and we denote it by $F^*_{\mathcal{R}}$, and simply F^* when there is no doubt on $\mathcal{R}$. F^* can be infinite, as illustrated by the following example.

Example 9. Let $F = p(a)$ and $R = p(x) \to q(x, y) \wedge p(y)$. $\alpha_\infty(F, \mathcal{R})$ is infinite, since each application of R leads to a new application of R. Note that for all $i \geq 0$, $\alpha_{i+1}(F, \mathcal{R})$ is not equivalent to $\alpha_i(F, \mathcal{R})$.

We recall the nice property that holds for range-restricted rules: M^*, the interpretation isomorphic to F^*, is a representative of all models of $(F, \mathcal{R})$. More precisely, M^* is a *universal* model of $(F, \mathcal{R})$. This property is kept for existential rules. The difference with the range-restricted rule case is that M^* can be infinite.

Theorem 3 (Soundness and completeness of saturation). *Let $(F, \mathcal{R})$ be a KB and q be a Boolean conjunctive query. Let $F^* = \alpha_\infty(F, \mathcal{R})$. The following four statements are equivalent:*

- *$F, \mathcal{R} \models q$;*
- *M^* is a model of q;*
- *there is a homomorphism from q to F^**
- *there is an integer $k \geq 0$ and a homomorphism from q to $\alpha_k(F, \mathcal{R})$.*

Obviously, the saturation terminates when F^* is finite. Further work has proposed mechanisms related to forward chaining to build a finite representation of F^*, even when F^* is infinite, for restricted classes of existential rules [CGK08,TBMR12]. The developed techniques are close in spirit to blocking techniques in DL tableau procedures [BCM+07].

4.2 Query Rewriting Approach

In the OBQA context, query rewriting was first proposed for DL-Lite [CGL+05]: the ontology is used to rewrite the initial conjunctive query into a union of conjunctive queries, which can then be passed to a relational database management system. More generally, the input query can be rewritten into a *first-order query* (e.g., a union of semi-conjunctive queries [Tho13]). First-order queries are exactly the logical counterpart of SQL queries. Query rewriting has been further generalized by considering rewriting into a *Datalog query* (a UCQ can be seen as a specific case of such a query). See the last section for more details.

We focus here on the basic rewriting technique, which outputs a UCQ, seen as a set of CQs. We present a conceptually simple and generic approach to query

rewriting with existential rules, namely piece-based query rewriting, which can be applied to any kind of existential rule (but, of course, it is ensured to stop only for some classes of rules). This technique has been introduced in [BLMS09] (and [BLMS11] for the journal version). A slightly different technique has been proposed in [GOP11].

For simplicity reasons, we focus on Boolean conjunctive queries hereafter. To be applicable to arbitrary CQs, the definitions should be extended to process free variables in a special way. However, considering BCQs only is not a restriction, since we can use a simple transformation from any CQ to a BCQ: let q be a CQ with free variables $x_1 \dots x_q$; q is translated into a BCQ q' by adding an atom $ans(x_1 \dots x_q)$, where ans is a special predicate not occurring in the knowledge base; the technique presented hereafter ensures that, if we rewrite q' into a UCQ q'', then remove from q'' the atoms with predicate ans (and consider their arguments as free variables), we get the UCQ that should be obtained by rewriting q.

Example 10. Consider again Example 4. Let $q = \exists x_1\ isProject(x_1, \text{“}KR\text{”}, x_2)$ asking for the leaders of projects in KR. The associated Boolean query is $q' = \exists x_1 \exists x_2 (ans(x_2) \wedge isProject(x_1, \text{“}KR\text{”}, x_2))$. The key point is that the predicate ans does not appear in the knowledge base, hence the added atom will never be rewritten and its variables will be correctly processed (as detailed in following Example 15).

Query rewriting relies on the notion of a *unification* between a query and a rule head. We first recall here the usual definition of unification, used for instance in plain Datalog (or range-restricted rules), then explain why it has to be extended in the case of existential rules.

Definition 9 (Datalog Unification). *Let q be a Boolean conjunctive query, and R be a Datalog rule. A* unifier *of q with R is a pair $\mu = (a, u)$, where a is an atom of q and u is a substitution of $vars(a) \cup vars(head(R))$ by $terms(head(R)) \cup consts(a)$ such that $u(a) = u(head(R))$.*

When a query and a rule unify, it is possible to rewrite the query with respect to that unification, as specified in Definition 10.

Definition 10 (Datalog rewriting). *Let q be a Boolean conjunctive query, R be a Datalog rule and $\mu = (a, u)$ be a unifier of q with R. The rewriting of q according to μ, denoted by $\beta(q, R, \mu)$, is $u(body(R) \cup \bar{q'})$, where $\bar{q'} = q \setminus q'$.*

Please note that these classical notions have been formulated in order to stress similarities with the notions we introduce hereafter.

Example 11 (Datalog Unification and Rewriting). Let us consider $q_e = t(x_1, x_2) \wedge s(x_1, x_3) \wedge s(x_2, x_3)$ and $R = s_1(x, y) \rightarrow s(x, y)$. A Datalog unifier of q_e with R is $\mu_d = (s(x_1, x_3), \{u(x_1) = x, u(x_3) = y\})$. The rewriting of q_e according to μ is the following query:

$$t(x, x_2) \wedge s_1(x, y) \wedge s(x_2, y).$$

Let us stress that this query is equivalent to the following query:

$$t(x_1, x_2) \wedge s_1(x_1, x_3) \wedge s(x_2, x_3),$$

where x has been renamed by x_1 and y by x_3. In the following, we will allow ourselves to use such a variable renaming without prior notice.

Applying the same steps without paying attention to the existential variables in existential rule heads would lead to erroneous rewritings, as shown by Example 12.

Example 12 (Wrong Unification). Let us consider $q_e = t(x_1, x_2) \wedge s(x_1, x_3) \wedge s(x_2, x_3)$ and $R = f(x) \rightarrow s(x, y)$. A Datalog unification of q_e with R is $\mu_{error} = (s(x_1, x_3), \{u(x_1) = x, u(x_3) = y\})$. According to Definition 10, the rewriting of q_e with R would be q_r:

$$q_r = t(x, x_2) \wedge f(x) \wedge s(x_2, y).$$

However, q_r is not a sound rewriting of q_e, which can be checked by considering the following fact (obtained by instantiating q_r):

$$F = t(a, b) \wedge f(a) \wedge s(b, c).$$

We have $F \models q_r$, however $F, \mathcal{R} \not\models q_e$. Indeed, $F^*_{\mathcal{R}} \equiv F$ and q_e cannot be mapped by homomorphism to F.

For that reason, the notion of *piece unifier* has been introduced, originally in the context of conceptual graph rules [SM96], then recast in the framework of existential rules [BLMS09]. Instead of unifying only one atom at once, one may have to unify a whole "piece", that is, a set of atoms that should have been created by the same rule application. The following definitions and the algorithm are mainly taken from [KLMT12]. Alternative definitions can be found in [KLMT13]. Given a Boolean conjunctive query q and $q' \subseteq q$, we call *separating variables* of q' (w.r.t. q) the set of variables that belong to both q' and $q \setminus q'$, and we denote this set by $sep_q(q')$. The other variables of q' are said to be non-separating variables.

Definition 11 (Piece Unifier). *Let q be a Boolean conjunctive query and R be an existential rule. A* piece unifier *of q with R is a pair $\mu = (q', u)$ with $q' \subseteq q, q' \neq \emptyset$, and u is a substitution of $fr(R) \cup vars(q')$ by $terms(head(R)) \cup consts(q')$ such that:*

1. *for all $x \in fr(R)$, $u(x) \in fr(R)$ or $u(x)$ is a constant (for technical convenience, we allow $u(x) = x$);*
2. *for all $x \in sep_q(q'), u(x) \in fr(R)$ or $u(x)$ is a constant;*
3. *$u(q') \subseteq u(head(R))$;*

It follows from this definition that existential variables from R can only be unified with non-separating variables of q'. In other words, when a variable x of q' is unified with an existential variable of R, all the atoms in which x occur must be part of the unification (i.e., x cannot be a separating variable).

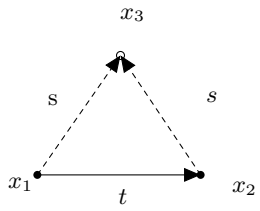

Fig. 1. Since x_3 is unified with an existential variable (Example 13), dashed atoms must be part of the unification.

Let us consider the unification attempted in Example 12 from this point of view.

Example 13 (Piece Unifier). We consider again $q_e = t(x_1, x_2) \wedge s(x_1, x_2) \wedge s(x_2, x_3)$ and $R = f(x) \rightarrow s(x, y)$. $\mu_{error} = (q' = \{s(x_1, x_3)\}, \{u(x_1) = x, u(x_3) = y\})$ is not a piece unifier. Indeed, x_3 belongs to $sep_{q_e}(q')$ since it appears in $s(x_2, x_3)$, which does not belong to q', hence violating the second condition of the piece unifier definition.

A correct choice of piece is illustrated by Figure 1. Indeed, let us define μ by $((\{s(x_1, x_3), s(x_2, x_3)\}, \{u(x_1) = x, u(x_3) = y, u(x_2) = x\})$. μ is a piece unifier of q_e with R, which can be checked by verifying that Conditions 1 to 3 are fulfilled.

Given the above definition of piece unifiers, the definition of rewritings remains syntactically the same as in the Datalog case.

Definition 12 (Rewriting). *Given a Boolean conjunctive query q, an existential rule R and a piece unifier $\mu = (q', u)$ of q with R, the* direct rewriting *of q according to μ, denoted by $\beta(q, R, \mu)$ is $u(body(R) \cup \bar{q'})$, where $\bar{q'} = q \setminus q'$.*

Example 14 (Direct rewriting). Let μ be the unifier of q_e with R defined in Example 13. The direct rewriting of q_e with respect to μ is:

$$\beta(q_e, R_2, \mu) = t(x, x) \wedge f(x).$$

The notion of $\mathcal{R}$-rewriting allows to denote queries that are obtained thanks to successive rewriting operations.

Definition 13 ($\mathcal{R}$-rewriting of q). *Let q be a Boolean conjunctive query and $\mathcal{R}$ be a set of existential rules. An $\mathcal{R}$-rewriting of q is a conjunctive query q_k*

obtained by a finite sequence $q_0 = q, q_1, \ldots, q_k$ *such that for all* i *such that* $0 \leq i < k$*, there is* $R_i \in \mathcal{R}$ *and a piece unifier* μ_i *of* q_i *with* R_i *such that* $q_{i+1} = \beta(q_i, R, \mu_i)$*.*

We are now able to illustrate how non-Boolean queries can be processed by translation into Boolean queries, thus avoiding to consider answer variables in a specific way (see the discussion associated with Example 10).

Example 15. Consider the set of rules $\mathcal{R}$ from Example 4.
Let $q = \exists x_1 isProject(x_1, \text{“}KR\text{”}, x_2)$, which asks for the leaders of projects in KR. Let $q' = \exists x_1 \exists x_2 (ans(x_2) \wedge isProject(x_1, \text{“}KR\text{”}, x_2))$ be the Boolean query obtained from q. The only rule that contains an atom with predicate $isProject$ is $R_2 = Researcher(x) \wedge hasExpertise(x, y) \rightarrow isProject(u, y, z) \wedge isMember(x, u)$. However, q' cannot be piece-unified with the head of R_2 because x_2 would be unified with the existential variable z, whereas it also appears in $ans(x_2)$ which cannot be unified. In this case, the only $\mathcal{R}$-rewriting of q' with the rules from Example 4 is q' itself. Hence, the only rewriting of the original query q would be q itself, obtained from q' by removing the ans atom and making x_2 free.

We now present the fundamental theorem justifying the notion of $\mathcal{R}$-rewriting. This theorem was originally written in the framework of conceptual graph rules [SM96]. Since the logical translation of conceptual graph rules is exactly existential rules, it can be immediately recast in the framework of existential rules.

Theorem 4 (Soundness and completeness of $\mathcal{R}$-rewriting). *Let* F *be a fact,* $\mathcal{R}$ *be a set of existential rules, and* q *be a Boolean conjunctive query. Then* $F, \mathcal{R} \models q$ *iff there is an* $\mathcal{R}$*-rewriting* q' *of* q *such that* $F \models q'$*.*

Of course, this does not always provide a halting procedure since there may be an infinite number of $\mathcal{R}$-rewritings, as illustrated by the following example. Note that this example actually considers a Datalog rule.

Example 16. Let $R = p(x, y) \wedge p(y, z) \rightarrow p(x, z)$. Let $q = p(a, b)$, where a and b are constants. Hence q is a ground atom. A direct rewriting of q with R is $q_1 = p(a, y) \wedge p(y, b)$. Each atom of q_1 unifies with $head(R)$, which yields two isomorphic queries. Let us consider $q_2 = p(a, y') \wedge p(y', y) \wedge p(y, b)$. The same process can be repeated indefinitely, producing increasingly longer “paths” from a to b. Hence, the set of $\{R\}$-rewritings of q is infinite.

For some classes of rules, there exists a *finite* set of $\mathcal{R}$-rewritings (in other words a UCQ), which is both sound and complete, as formally defined below:

Definition 14 (Sound and complete set of $\mathcal{R}$-rewritings). *Let* $\mathcal{R}$ *be a set of existential rules and* q *be a Boolean conjunctive query. Let* $\mathcal{Q}$ *be a set of BCQs.* $\mathcal{Q}$ *is said to be* sound *with respect to* q *and* $\mathcal{R}$ *if, for all fact* F*, and all* $q' \in \mathcal{Q}$*, if* $F \models q'$ *then* $F, \mathcal{R} \models q$*. Reciprocally,* $\mathcal{Q}$ *is said to be* complete *with respect to* q *and* $\mathcal{R}$ *if, for all fact* F*, if* $F, \mathcal{R} \models q$*, then there is* $q' \in \mathcal{Q}$ *such that* $F \models q'$*.*

See the next section for conditions on rules ensuring that a finite sound and complete set of rewritings exists, whatever the input conjunctive query is.

5 Decidable Classes of Rules and Algorithmic Techniques

In this section, we present several criteria that ensures decidability of BCQ ENTAILMENT. First, we consider the case where the saturation introduced in Section 4 is equivalent to a finite fact. This allows to apply classical forward chaining techniques. However, sets of rules ensuring such a property are not recognizable, hence the definition of several recognizable sufficient conditions, also known as "concrete" classes. In this paper, we present two of them, and we provide links to other relevant work. Similarly, we consider the case where query rewriting into a union of conjunctive queries can be performed, by presenting a generic algorithm and several concrete cases where it is applicable. Last, we briefly mention and provide relevant links to other decidable cases.

5.1 Finite Expansion Sets

We have already seen how to compute a saturation by applying rules in a breadth-first fashion. A set of rules that produces new information in only a finite number of steps for every initial fact F is called a *finite expansion set*, as formalized in Definition 15.

Definition 15 (Finite Expansion Set). *A set of rules $\mathcal{R}$ is called a* finite expansion set *(*fes*) if and only if, for every fact F, there exists an integer $k = f(F, \mathcal{R})$ such that $\alpha_k(F, \mathcal{R}) \equiv \alpha_\infty(F, \mathcal{R})$.*

Since one can not decide if a given set of rules is a *fes*[5], it is crucial to design expressive specific cases. All the cases known so far are based on some notion of acyclicity. Several of them have been proposed, and we present here two incomparable notions of acyclicity.

Weak-Acyclicity The first notion is based on the notion of *position* of a predicate.

Definition 16 (Position). *Let p be a predicate of arity k. A position of p is a pair (p, i), with i from 1 to k.*

We now present the graph of position dependencies. The vertices of this graph are all the positions that appear in a rule set. Intuitively, it tracks how variables are propagated from one position to another one. Moreover, it also tracks how new existentially quantified variables are introduced, and in which positions. Definition 17 formalizes this intuition.

Definition 17 (Graph of Position Dependencies [FKMP05]). *Let $\mathcal{R}$ be a set of rules. The (oriented)* graph of position dependencies *$(V, A \cup A^*)$ of $\mathcal{R}$ is defined as follows:*

- *V is the set of all positions for all predicates appearing in $\mathcal{R}$;*

[5] We say that *fes* is an *abstract class*.

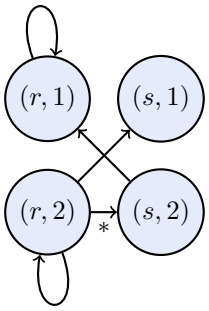

Fig. 2. The graph of position dependencies associated with $\mathcal{R}$ (Example 17)

- *there is an arc from* (p, i) *to* (q, j) *in* A *if there exist a rule* $R \in \mathcal{R}$*, and a variable* $x \in fr(R)$ *such that* x *appears in position* (p, i) *in the body of* R *and in position* (q, j) *in the head of* R*;*
- *there is an arc from* (p, i) *to* (q, j) *in* A^* *if there exist a rule* $R \in \mathcal{R}$ *and a variable* $x \in fr(R)$ *such that* x *appears in position* (p, i) *in the body of* R *and an existentially quantified variable appears in position* (q, j) *in the head of* R*. The arcs of* A^* *are called special arcs.*

The rank *of a position is the maximum number (possibly infinite) of special arcs on a path leading to that position.*

Example 17 illustrates the construction of the graph of position dependencies.

Example 17. Let $\mathcal{R}$ be a set containing the following rules:

- $r(x, y) \to s(y, z)$
- $s(x, y) \to r(y, x)$
- $r(x, y) \wedge r(y, z) \to r(x, z)$

The graph of position dependencies of $\mathcal{R}$ is shown in Figure 2. Special arcs are labelled by a star.

The graph of position dependencies is used to defined "weak-acyclicity", where some cycles are forbidden.

Definition 18 (Weak-Acyclicity [FKMP05]). *Let* $\mathcal{R}$ *be a set of rules.* $\mathcal{R}$ *is said to be* weakly-acyclic *if there is no cycle in the graph of position dependencies of* $\mathcal{R}$ *that goes through a special arc.*

Let us point out that a set of rules containing no existentially quantified variables in rule heads is trivially weakly acyclic (because there is no special arc). Such sets of rules (which can be seen as Datalog programs) are sometimes called *range-restricted*.

Property 2. A weakly-acyclic set of rules is a finite expansion set.

The proof is done by upper-bounding for any fact F and any weakly-acyclic set of rules $\mathcal{R}$ the number of fresh existential variables in the core of the saturation of F with $\mathcal{R}$ (by a double exponential with respect to $\mathcal{R}$; the upper-bound is polynomial if $\mathcal{R}$ is fixed).

Example 18. In the graph of Figure 2 (Example 17), no cycle goes through a special edge, thus $\mathcal{R}$ is weakly acyclic. As such, $\mathcal{R}$ is a finite expansion set.

This condition is sufficient to ensure the finiteness of forward chaining, but not necessary, as witnessed by the following example.

Example 19. Let R be the following rule:

$$r(x, y) \wedge s(x, y) \rightarrow r(x, v) \wedge r(w, y) \wedge s(x, w) \wedge s(v, y).$$

The graph of position dependencies is a clique of special edges, but an application of R cannot trigger a novel application of R —hence, $\{R\}$ is a finite expansion set.

Acyclic Graph of Rule Dependency The latter example motivates the notion of *rule dependency* [BLMS11], which has originally been introduced for conceptual graph rules [Bag04]. The main idea here is to *characterize* which rule can effectively lead to trigger another rule. Preventing such cycles of dependencies naturally ensures the finiteness of forward chaining.

Definition 19 (Dependency). *Let R_1 and R_2 be two existential rules. R_2* depends *on R_1 if there exist a fact F, a homomorphism π_1 from body(R_1) to F and a homomorphism π_2 from body(R_2) to $\alpha(F, R_1, \pi_1)$ such that π_2 is not a homomorphism from body(R_2) to F.*

This definition means that an application of R_1 may, on some fact, trigger a *new* application of R_2. All rule dependencies are summarized in the graph of rule dependencies, whose definition is given below. It is possible to decide if a rule depends on another, by using the notion of piece-unifier introduced in Section 4 [SM96,BLMS11]. The associated decision problem is NP-complete.

Definition 20 (Graph of Rule Dependencies). *Let $\mathcal{R}$ be a set of rules. The* graph of rule dependencies *of $\mathcal{R}$, denoted by GRD$(\mathcal{R})$ is defined as follows:*

- *its vertices are the rules of $\mathcal{R}$,*
- *there is an arc from R_1 to R_2 if and only if R_2 depends on R_1.*

A set of rules $\mathcal{R}$ is said to have an acyclic graph rule of dependencies (aGRD) if *GRD*$(\mathcal{R})$ is acyclic. This is in particular the case for Example 20.

Example 20. Let us consider the following two rules:

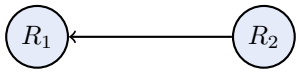

Fig. 3. The graph of rule dependencies of Example 20

- $R_1 = p(x) \to r(x, y) \wedge r(y, z) \wedge r(z, x)$,
- $R_2 = r(x, y) \wedge r(y, x) \to p(x)$.

Their graph of rule dependencies is given Figure 3.

Let us last notice that Examples 17 and 20 show that weak-acyclicity and aGRD are incomparable criteria.

Related Work The two presented notions are only two examples of the various acyclicity notions that have been introduced so-far. They have indeed been generalized in a variety of ways, such as super-weak acyclicity [Mar09], join-acyclicity [KR11], aGRD$_k$ [BMT11], as well as model-summarizing acyclicity and model-faithful acyclicity [GHK$^+$12]. The interested reader is invited to consult [GHK$^+$13], which among others contains a nice overview of the introduced acyclicity notions.

5.2 Finite Unification Sets

As already noticed in Section 4, materializing the saturation, even when it is theoretically possible, may not be practical due to its large size. Approaches based on query reformulation have thus been proposed. We rely here on piece-based query rewriting, presented in Section 4.

We recall that $q_2 \models q_1$ if and only if there is a homomorphism from q_1 to q_2, which we denote by $q_1 \geq q_2$. Let q be a BCQ, and $\mathcal{Q}$ be a sound and complete UCQ-rewriting of q. If there exist q_1 and q_2 in $\mathcal{Q}$ such that $q_1 \geq q_2$, then $\mathcal{Q} \setminus \{q_2\}$ is also a sound and complete rewriting of q. This observation motivates the definition of *cover* of a set of first-order queries.

Definition 21 (Cover). *Let $\mathcal{Q}$ be a set of Boolean conjunctive queries. A* cover *of $\mathcal{Q}$ is a set $\mathcal{Q}^c \subseteq \mathcal{Q}$ such that:*

1. *for any $q \in \mathcal{Q}$, there is $q' \in \mathcal{Q}^c$ such that $q' \geq q$,*
2. *elements of $\mathcal{Q}^c$ are pairwise incomparable with respect to $\geq$.*

Example 21. Let $\mathcal{Q} = \{q_1 = r(x, y) \wedge t(y, z), q_2 = r(x, y) \wedge t(y, y), q_3 = r(x, y) \wedge t(y, z) \wedge t(u, z)\}$. A cover of $\mathcal{Q}$ is $\{q_1\}$. Indeed, $q_1 \geq q_2$ and $q_1 \geq q_3$, because for $i \in \{2, 3\}$, $\pi_{1\to i}$ is a homomorphism from q_1 to q_i where:

- $\pi_{1\to 2}(x) = x, \pi_{1\to 2}(y) = \pi_{1\to 2}(z) = y$, and
- $\pi_{1\to 3}(x) = x, \pi_{1\to 3}(y) = y, \pi_{1\to 3}(z) = z$.

We now define the class of finite unification sets, which is the main focus of this section.

Definition 22 (Finite unification set). *Let $\mathcal{R}$ be a set of existential rules. $\mathcal{R}$ is a* finite unification set *if it holds for any Boolean conjunctive query q that the set of $\mathcal{R}$-rewritings of q admits a finite cover.*

Algorithm 1 is a generic breadth-first rewriting algorithm, that generates for any query q and any finite unification set $\mathcal{R}$ a sound and complete UCQ-rewriting of q with respect to $\mathcal{R}$. Generated queries are queries that belong to $\mathcal{Q}_t$ at some point; explored queries are queries that belong to $\mathcal{Q}_E$ at some point, and thus, for which all one-step rewritings are generated. At each step, a cover of explored and generated queries is computed. This means that only most general queries are kept, both in the set of explored queries and in the set of queries remaining to be explored. If two queries q_1 and q_2 are homomorphically equivalent, and only q_1 has already been explored, then q_1 is kept and q_2 is discarded. This is done in order not to explore two queries that are comparable by the most general relation – which ensures the termination of Algorithm 1.

Algorithm 1: A BREADTH-FIRST REWRITING ALGORITHM

```
Data: A fus R, a Boolean conjunctive query q
Result: A cover of the set of R-rewritings of q
Q_F := {q}; // resulting set
Q_E := {q}; // queries to be explored
while Q_E ≠ ∅ do
    Q_t := ∅; // queries generated at this rewriting step
    for q_i ∈ Q_E do
        for R ∈ R do
            for μ piece-unifier of q_i with R do
                Q_t := Q_t ∪ β(q_i, R, μ);
    Q^c := cover(Q_F ∪ Q_t);
    Q_E := Q^c \ Q_F; // select unexplored queries from the cover
    Q_F := Q^c;
return Q_F
```

Let us provide a step by step application of Algorithm 1.

Example 22. Let $\mathcal{R}_e = \{R_1, R_2, R_3, R_4, R_5\}$, defined as follows:

- $R_1 : p(x) \wedge h(x) \rightarrow s(x, y)$;
- $R_2 : f(x) \rightarrow s(x, y)$;
- $R_3 : f_1(x) \rightarrow s_1(x, y)$;
- $R_4 : t(x, y) \rightarrow t(y, x)$;
- $R_5 : s_1(x, y) \rightarrow s(x, y)$;

and q_e be the following Boolean query:

$$q_e = t(x_1, x_2) \wedge s(x_1, x_3) \wedge s(x_2, x_3)$$

Initially, $\mathcal{Q}_F = \mathcal{Q}_E = \{q_e = t(x_1, x_2) \wedge s(x_1, x_3) \wedge s(x_2, x_3)\}$. Since $\mathcal{Q}_E$ is not empty, we initialize $\mathcal{Q}_t$ to the empty set, and consider every element of $\mathcal{Q}_E$. The only element of $\mathcal{Q}_E$ is q_e, so we add to $\mathcal{Q}_t$ all possible rewritings of q_e. These are:

- $q_1 = t(x, x) \wedge p(x) \wedge h(x)$, by unifying with respect μ_1, which is defined by $(\{s(x_1, x_3), s(x_2, x_3)\}, u_1(x_1) = u_1(x_2) = x, u_1(x_3) = y)$, and is a unifier of q_e with R_1;
- $q_2 = t(x, x) \wedge f(x)$, with respect to $\mu_2 = (\{s(x_1, x_3), s(x_2, x_3)\}, u_2(x_1) = u_2(x_2) = x, u_2(x_3) = y)$, unifier of q_e with R_2;
- $q_3 = t(x_2, x_1) \wedge s(x_1, x_3) \wedge s(x_2, x_3)$ with respect to $\mu_3 = (\{t(x_1, x_2), u_3(x_1) = y, u_3(x_2) = x)$, unifier of q_e with R_4;
- $q_4 = t(x_1, x_2) \wedge s_1(x_1, x_3) \wedge s(x_2, x_3)$ with respect to $\mu_4 = (\{s(x_1, x_3)\}, u_4(x_1) = x, u_4(x_3) = y)$, unifier of q_e with R_5;
- $q_5 = t(x_1, x_2) \wedge s(x_1, x_3) \wedge s_1(x_2, x_3)$ with respect to $\mu_5 = (\{s(x_2, x_3)\}, u_5(x_2) = x, u_5(x_3) = y)$, unifier of q_e with R_5;
- $q_6 = t(x, x) \wedge s_1(x, x_3)$, with respect to $\mu_6 = (\{s(x_1, x_3), s(x_2, x_3)\}, u_6(x_1) = u_6(x_2) = x, u_6(x_3) = y)$, unifier of q_e with R_5;

Thus $\mathcal{Q}_t = \{q_1, q_2, q_3, q_4, q_5, q_6\}$. $\mathcal{Q}^c$ is set to $\{q_e, q_1, q_2, q_3, q_4, q_5, q_6\}$, since none of the generated queries are comparable. $\mathcal{Q}_E$ is set to $\{q_1, q_2, q_3, q_4, q_5, q_6\}$ and $\mathcal{Q}_F$ to $\mathcal{Q}^c$.

Algorithm 1 performs once more the while loop. Q_t is reinitialized to the empty set, and all rewritings of $\mathcal{Q}_E$ are rewritten. We thus explore every q_i for $i \leq 5$. q_1 and q_2 are not unifiable with any rule. We then explore rewritings of q_3. The following queries can be obtained by a one step rewriting of q_3:

$$q_1^3 = t(x, x) \wedge p(x) \wedge h(x),$$

$$q_2^3 = t(x, x) \wedge f(x),$$

$$q_3^3 = t(x_1, x_2) \wedge s(x_1, x_3) \wedge s(x_2, x_3),$$

$$q_4^3 = t(x, x) \wedge s_1(x, x_3),$$

$$q_5^3 = t(x_2, x_1) \wedge s_1(x_1, x_3) \wedge s(x_2, x_3),$$

$$q_6^3 = t(x_2, x_1) \wedge s(x_1, x_3) \wedge s_1(x_2, x_3).$$

As for q_4, the following rewritings are generable:

$$q_1^4 = t(x_2, x_1) \wedge s_1(x_1, x_3) \wedge s(x_2, x_3),$$
$$q_2^4 = t(x_1, x_2) \wedge s_1(x_1, x_3) \wedge s_1(x_2, x_3).$$

From q_5:

$$q_1^5 = t(x_2, x_1) \wedge s(x_1, x_3) \wedge s_1(x_2, x_3),$$
$$q_2^5 = t(x_1, x_2) \wedge s_1(x_1, x_3) \wedge s_1(x_2, x_3).$$

And from q_6:

$$q_1^6 = t(x, x) \wedge f_1(x).$$

As illustrated by Figure 4, which explicits subsumption relations among queries, a cover of the queries is $\{q_e, q_1, q_2, q_3, q_4, q_5, q_5^3, q_6^3, q_2^4, q_1^6\}$, which is the new value of $\mathcal{Q}^c$. Note that q_6 does not belong to Q^c, because the newly generated query q_2^4 is strictly more general than q_6. $\mathcal{Q}_E$ is set to $\{q_5^3, q_6^3, q_2^4, q_1^6\}$, Q_F to Q^c, and $\mathcal{Q}_E$ is explored, entering a new iteration of the while loop. Two queries are generated:

$$q' = t(x_2, x_1) \wedge s_1(x_1, x_3) \wedge s_1(x_2, x_3),$$

and

$$q'' = t(x, x) \wedge f_1(x).$$

At the end of this while loop, we have $\mathcal{Q}_E = \{q'\}$ and

$$\mathcal{Q}_F = \{q_e, q_1, q_2, q_3, q_4, q_5, q_5^3, q_6^3, q_2^4, q_1^6, q'\}.$$

Since all queries generable from q' are covered by $\mathcal{Q}_F$, the algorithm halts and outputs:

$$\{q_e, q_1, q_2, q_3, q_4, q_5, q_5^3, q_6^3, q_2^4, q_1^6, q'\}.$$

The following lemma is crucial for the completeness of Algorithm 1. It ensures that for any queries q and q' such that $q' \geq q$, any rewriting that can be obtained in one step from q is less general than a rewriting that can be obtained in one step from q'. A detailed discussion of what can happen when considering rewriting procedures where this lemma does not hold can be found in [KLMT13].

Lemma 1 ([KLMT12]). *Let q_1 and q_2 be two Boolean conjunctive queries such that $q_1 \geq q_2$. For any rewriting q_2' of q_2 such that $q_1 \not\geq q_2'$, there exists a rewriting q_1' of q_1 such that $q_1' \geq q_2'$.*

Theorem 5. *The output of Algorithm 1 is a sound and complete set of $\mathcal{R}$-rewritings of q.*

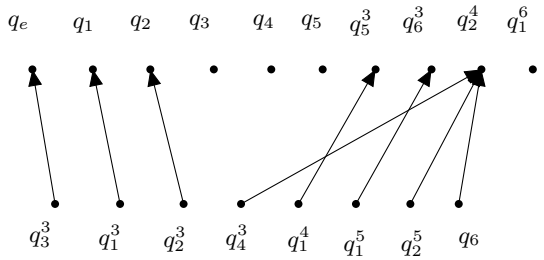

Fig. 4. There is an arrow from q to q' if and only if q' is more general than q.

Proof. We define a 1-rewriting of q as a direct rewriting, and a k-rewriting of q as a direct rewriting of a $(k-1)$-rewriting for any $k \geq 2$. We prove by induction on k that for any $i \leq k$, for any q_i that is an i-rewriting of q, there is $q_i^* \in \mathcal{Q}_{F_i}$, that is, Q_F after the i^{th} loop such that $q_i^* \geq q_i$. The only 0-rewriting of q is q, which initially belongs to $\mathcal{Q}_F$, which proves the claim for $k = 0$. Let assume that the claim is true for k, and let us show it for $k+1$. Let q_{i+1} be a $i+1$-rewriting of q. If $i < k$, q_{i+1} is covered by an element of $Q_{F_{i+1}}$ by induction assumption. Otherwise, let q_i be a k-rewriting such that q_{i+1} is a 1-rewriting of q_i. There exists $q_i^* \in Q_{F_i}$ such that $q_i^* \geq q_i$. Lemma 1 ensures that there exists q_{i+1}^* a 1-rewriting of q_i^* such that $q_{i+1}^* \geq q_{i+1}$, which ends the proof.

Backward-Shyness As for finite expansion sets, the problem of recognizing finite unification sets is undecidable. Several concrete classes of finite unification sets are known, the most famous ones being the class of linear rules ([CGK08] and also [BLMS09], under the name of atomic-hypothesis rules) and the class of sticky sets of rules [CGP10]. We present both classes of rules under the unifying concept of "backward shy"[6] class of rules. To define that notion, we need the notion of *original* and *generated* variables.

Definition 23 (Original and Generated Variables). *Let q be a Boolean conjunctive query, $\mathcal{R}$ be a set of rules, and q' be an $\mathcal{R}$-rewriting of q, obtained by a rewriting sequence $q = q_0, q_1, \ldots, q_n = q'$.* Original *variables of q' (with respect to q) are inductively defined as follows:*

- *all variables of q are original;*
- *if q_i has original variables X, and q_{i+1} is the rewriting of q_i with respect to $\mu = (q'_i, u)$, the original variables of q_{i+1} are the images of the elements of X by u.*

A variable that is not original is generated.

[6] The term "backward shy" is not standard, and inspired from shy rules [LMTV12]. However, there is no inclusion between shy and backward shy rules.

Backward shyness of a set of rules $\mathcal{R}$ ensures that any query q admits a finite set of $\mathcal{R}$-rewritings.

Definition 24 (Backward Shyness). *Let $\mathcal{R}$ be a set of rules. $\mathcal{R}$ is to be said backward shy if for any Boolean conjunctive query q, for any $\mathcal{R}$-rewriting q' of q, no generated variable of q' appears in two atoms.*

Property 3 provides an upper bound on the number of most general rewritings of a query q with respect to a backward shy set of rules.

Property 3 (Backward Shy Rules are fus*).* Let $\mathcal{R}$ be a set of backward shy rules, and q a Boolean conjunctive query. There are at most $2^{p(|terms(q)|+w)^w}$ $\mathcal{R}$-rewritings of q that are not equivalent up to isomorphism, where w is the maximum arity of a predicate and p is the number of predicates appearing in the rules.

Proof. The number of distinct atoms with arguments the terms of q and at most w other terms is upper bounded by $p(|terms(q)| + w)^w$. Since a term that is not a term of q cannot appear in two different atoms, we obtain the claimed upper bound.

Linear rules [CGK08,BLMS09] are rules whose body contains only one atom. Let us observe that linear rules are backward shy.

Property 4. Any set of linear rules is backward shy.

Proof. The claim follows from the following two remarks:

- when a generated variable is introduced by a rewriting step, it appears in exactly one atom;
- if x appears in k atoms of a query q before a rewriting with respect to $\mu = (q', u)$, then $u(x)$ appears in at most k atoms in the rewriting of q with respect to μ.

We now present sticky rules, which have been introduced as a decidability criterion that may deal with non-guarded rules.

Definition 25 (Sticky Sets of Rules [CGP10]). *Let $\mathcal{R}$ be a set of rules. We iteratively mark the variables of the rule bodies of $\mathcal{R}$ according to the following marking procedure. First, for each rule $R \in \mathcal{R}$, and each variable $v \in$ body(R), we mark v if there is an atom a of* head(R) *such that v is not an argument of a. We then apply until a fixpoint is reached the following step: for each rule R, if a marked variable appears in* body(R) *at position (p, i), then we mark for each rule R' each occurrence of the variables of* body(R') *that appear in* head(R') *at position (p, i). $\mathcal{R}$ is said to be* sticky *if there is no rule R such that a marked variable appears more than once in* body(R).

Example 23 provides an example of sticky and non-sticky rules.

Example 23. Let $\mathcal{R}_1$ be a set of rules containing the following rule:

- $r(x, y) \wedge t(y, z) \rightarrow s(x, z)$

$\mathcal{R}_1$ is not sticky, since y is marked by the marking procedure, and appears twice in a rule body. On the other hand, the set containing the following two rules is sticky:

- $r(x_1, y_1) \wedge t(y_1, z_1) \rightarrow s(y_1, u_1)$
- $s(x_2, y_2) \rightarrow r(y_2, x_2)$

Indeed, x_1 and z_1 are marked at the initialization step. The propagation step marks y_2, because y_2 appears at the first position of r in the head of the second rule, as x_1 which is already marked. Finally, x_1, z_1 and y_2 are marked, and are the only marked variables. Since none of these variables appears twice in a rule body, this set of rules is sticky.

Property 5. Any sticky set of rules is backward shy.

Proof. We show Property 5 by induction on the length of the derivation. If q' is a one-step rewriting of q, then a generated variable is a variable that has been created at this rewriting step. By the initialization step of the sticky marking, such a variable appears at exactly one position, which is marked. Let assume that the induction assumption holds for any k-rewriting of q, and let q' be a $k+1$-rewriting of q. Let q_k be the k-rewriting of q from which q' has been rewritten. A generated variable of q' may appear for two different reasons: either it has been generated at the last rewriting step, and the same reasoning as before can be applied. Or it is a generated variable with respect to q_k. By induction assumption, it appears at a marked position. The stickiness property implies that it appears also only once in q', and at a marked position.

Related Work The rewriting algorithm we presented in this section is only one among several. We give here some pointers to several other rewriting algorithms or rewriting tools. Among implemented tools, we can cite Clipper [EOS+12], Kyrie [MC13], QuOnto [ACG+05], Nyaya [GOP11], Rapid [CTS11], Iqaros [VSS12], or the piece-based rewriting algorithm presented in [KLMT12].

5.3 Other Decidable Cases

Presenting algorithms that allow to deal with sets of rules ensuring neither the finiteness of the canonical model nor the existence of a sound and complete first-order rewriting is out of the scope of that short course. We give however a quick overview of topics that have been studied and relevant references.

Bounded Treewidth Sets Most of the known decidable cases that are neither finite unification sets nor finite expansion sets are *bounded treewidth sets*. This class of rules, once again not recognizable, is based on the structural property of the facts that are generated starting from any fact. Definition 26 introduces formally this notion.

Definition 26 (Bounded treewidth set). *A set of rules $\mathcal{R}$ is called a bounded-treewidth set (*bts*) if for any fact F, there exists an integer $b = f(F, \mathcal{R})$ such that for any $\mathcal{R}$-derivation F' of F, the treewidth of core(F') is less or equal to b.*

The most prominent example of a bounded treewidth set of rules is that of a set of guarded rules. A rule is guarded if its body contains a guard, that is, an atom that contains all the body variables. The original algorithm for conjunctive query answering under guarded sets of rules performs a traditional forward chaining and stops it after a number of steps which is a function of the sizes of the rule set and the query [CGK08]. Several extensions of the notion of guarded rules have been proposed in the literature, for instance by exploiting the graph of position dependencies (Definition 17) in order to restrict the set of body variables that needs to be guarded. The interested reader can consult [CGK08,KR11,TBMR12] for more details about these restrictions and associated algorithms.

Outside of the Classification and Combinations Even if the three introduced classes cover most of the known decidable class so far, some classes are outside of this classification. It is in particular the case for *parsimonious* sets of rules [LMTV12], which is an abstract class in its own right. Another popular way of getting sets of rules that do not fall in the afore mentioned classification is to *combine* decidable classes of rules. "Raw" combination usually leads to undecidability, but several restrictions have been proposed in the literature. One can distinguish two kinds of combinations: generic combination, which relies on abstract properties of rule sets and interaction between them, and "built-in" combination, which looks at specific classes and restrict the way they interact. In the first case, the already seen graph of rule dependencies may be used to split the study of a rule set into the study of the sets that are formed by its strongly connected components. In the second category, weakly-sticky sets of rules [CGP10] and tameness [GMP13] have been proposed, allowing to combine (under some restrictions) sticky sets of rules with weakly-acyclic sets of rules for the former, and with guarded rules for the latter.

6 Ongoing Research and Open Issues

We finish this course by presenting two current research issues, that both need to be tackled in order to obtain practical systems. The first one deals with the shortcomings of current query rewriting techniques. Indeed, experiments made

with the first rewriting tools have shown the rewritings to be of enormous size, which could not even be passed to an RDMS for execution. How to circumvent that problem? This is an intricate issue, that raises both theoretical and practical work, and that we will touch upon in Section 6.1.

Then, we will consider the important topic of inconsistent data. Given the usually cited Semantic Web application area, it is highly probable that the data a user will want to use is inconsistent with respect to an ontology encompassing both existential rules and negative constraints. This raises a non-trivial problem, since the classical semantics of first-order logic does not allow to draw meaningful, or at least intuitive, results in the presence of any inconsistency. We will present some alternative semantics in Section 6.2.

6.1 Query Rewriting: Any Useful in Practice?

Behind this rather provocative title hides a serious question: how efficient can query answering systems based on query rewriting be in practice? It is widely accepted that RDMS are well optimized and efficient in practice. However, the use of ontologies in order to perform query rewriting modifies the scope of what can be called a real-world query. We first quickly exhibit what the problem is, and present some of the approaches that have been proposed in order to overcome it. Last, we touch upon benchmarking problems.

Large size of first-order rewritings. Since the evaluation of Boolean conjunctive queries is a NP-hard problem, stating that RDMS are efficient means that they are efficient on real-world queries, that is, queries that a user may try to evaluate. In particular, such queries are usually of easy structure and of small size. When adding an ontology, even when starting from a simple ontology and a simple query, one may be led to evaluate a huge union of conjunctive queries, as can be noticed with the following example.

Example 24. Let $\mathcal{R} = \{R_i\}_{1 \leq i \leq n}$, where $R_i : r_i(x, y) \rightarrow r_{i-1}(x, y)$. Let q be the following query:

$$r_0(x_1, x_2) \wedge r_0(x_2, x_3).$$

q has a UCQ-rewriting with $(n+1)^2$ conjunctive queries, which are $\{r_i(x_1, x_2) \wedge r_j(x_2, x_3)\}_{0 \leq i,j \leq n}$.

Example 24 can be generalized by taking a query of k atoms and classes/roles having n subclasses/subroles. This would yield an optimal UCQ-rewriting with $(n+1)^k$ conjunctive queries. This cannot be considered as a small query anymore, and existing systems are not able to deal with such huge queries. Moreover, it has been shown that the exponential size of the rewritings also holds for so-called pure first-order rewritings. The interested reader is invited to consult, among others, [KKPZ12].

Alternative approaches. Two approaches have been proposed in order to escape from this problem of rewriting size, with the additional constraint of not changing the data. First, changing the target language in which rewritings are expressed. Instead of unions of conjunctive queries, one may use other forms of first-order formulas [Tho13], or Datalog rewritings [RA10]. This may allow to reduce the size of the rewritings in some common cases (such as for large class hierarchies), or for a whole class of ontologies. In particular, it was shown that for a wide class of ontologies, which in particular include linear and sticky rules, there exists a polynomial non-recursive Datalog rewriting for any query [GS12]. These rewritings, however, are not claimed to be efficiently evaluable.

Another approach is to reduce the scope of the rewritings: that is, instead of providing sound and complete answers on any database, the rewriting technique will provide sound and complete answers only on databases that fulfill some additional constraints. In particular, it may be possible in some settings to assume that the database is already complete with respect to some database dependencies. In the case of a rewriting taking the form of a set of conjunctive queries, this would imply that some of the conjunctive queries that are part of this set are not required, and one could ignore them. The interested reader is invited to consult [Ros12,RMC12] for more details.

Benchmarking problems. An additional problem when one wants to evaluate or compare different approaches is the current lack of benchmarks. Ideally, a test case would contain an ontology, some data, and a conjunctive query. Unfortunately, such trios are not widely available. The benchmark classically used since [PUHM09] to evaluate query rewriting algorithms is composed of five ontologies, with five queries each, and no associated data. The most used ontology is LUBM [7], and several adaptations of it have been proposed [RMC12,LSTW12], together with data generators. The small number of queries in the benchmark is already a serious weakness. This has already been noticed, and a recent paper proposed an automatic generation of relevant queries in order to test soundness and completeness of algorithms [ISG12]. Recently, more expressive real-world ontologies (expressed thanks to Description Logics) have been used for the evaluation of query rewriting [TSCS13]. However, once again queries are hand-crafted and no data are available.

6.2 Dealing with Inconsistent Data

In a setting where data come from several heterogeneous and possibly unchecked sources, it is highly probable for the knowledge base to be inconsistent. An example of inconsistent knowledge base is presented in Example 25.

Example 25. Let us consider the following fact:

$$F = \{cat(Tom), barks(Tom)\},$$

and the following rules:

[7] http://swat.cse.lehigh.edu/projects/lubm/

- $barks(x) \rightarrow dog(x)$
- $cat(x) \rightarrow animal(x)$
- $dog(x) \rightarrow animal(x)$

Last, let us specify that the classes *dog* and *cat* are disjoint, thanks to the following negative constraint:

$$dog(x) \wedge cat(x) \rightarrow \bot.$$

According to the classical first-order semantics, the answer to any Boolean query is yes, since the given knowledge base is inconsistent. This is somehow counterintuitive, and alternative semantics have been proposed in order to provide a more intuitive behavior [LLR+10,LMS12]. We present two of them in the following, which are based on the notion of *repair*.

Definition 27 (Repair). *Let $\mathcal{R}$ be a set of rules, $\mathcal{C}$ be a set of constraints, and F be a fact. A* repair *of F (with respect to $\mathcal{R}$ and $\mathcal{C}$) is a maximal subset F' of F such that $F', \mathcal{R}, \mathcal{C} \not\models \bot$.*

Given this definition of a repair, one can define a semantics for consistent query answering as follows: the query should be entailed by any repair together with the set of rules.

Definition 28 (AR semantics). *Let $\mathcal{R}$ be a set of rules, $\mathcal{C}$ be a set of constraints, F be a fact and q be a Boolean conjunctive query. $\mathcal{K}$ entails q for the AR semantics if for every repair F' of F with respect to $\mathcal{R}$ and $\mathcal{C}$, it holds that $F', \mathcal{R} \models q$.*

A more conservative semantics requires for the query to be entailed by the *intersection* of all the repairs together with the set of rules. Intuitively, this means that any atom that may lead to some contradiction is ignored during the reasoning.

Definition 29 (IAR semantics). *Let $\mathcal{R}$ be a set of rules, $\mathcal{C}$ be a set of constraints, F be a fact and q be a Boolean conjunctive query. $\mathcal{K}$ entails q for the AR semantics if it holds that $F_\cap, \mathcal{R} \models q$, where $F_\cap$ is the intersection of all the repairs of F with respect to $\mathcal{R}$ and $\mathcal{C}$.*

Let us exhibit the difference in the behavior of the two semantics on an example.

Example 26. Let us consider again the knowledge base of Example 25. There are two repairs: $F_1 = \{cat(Tom)\}$ and $F_2 = \{barks(Tom)\}$. Let us consider the query $q = animal(Tom)$. According to the AR semantics, q is entailed by the knowledge base, since q is entailed by F_1 and $\mathcal{R}$ as well as by F_2 and $\mathcal{R}$. However, since the intersection between F_1 and F_2 is empty, q is not entailed by $\mathcal{K}$ according to the IAR semantics.

Let us note that the problem of consistent conjunctive query answering under AR semantics is intractable (i.e., not polynomial) with respect to data complexity [LLR+10,Bie12], even for ontology languages as inexpressive as DL-Lite$_{\text{core}}$. An idea to overcome this hardness result has been to develop "approximation schemes" for AR-semantics, that is, to provide two families of efficiently computable semantics that upper and lower bound AR-semantics, while converging towards it [BR13]. However, the design and implementation of efficient consistent query answering algorithms remains an open issue.

References

[ACG+05] A. Acciarri, D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, M. Palmieri, and R. Rosati. Quonto: Querying ontologies. In *AAAI*, pages 1670–1671, 2005.

[ACKZ09] A. Artale, D. Calvanese, R. Kontchakov, and M. Zakharyaschev. The DL-Lite family and relations. *J. Artif. Intell. Res. (JAIR)*, 36:1–69, 2009.

[AHV95] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison Wesley, 1995.

[Bag04] J.-F. Baget. Improving the forward chaining algorithm for conceptual graphs rules. In *KR'04*, pages 407–414. AAAI Press, 2004.

[BBL05] F. Baader, S. Brandt, and C. Lutz. Pushing the $\mathcal{EL}$ envelope. In *IJCAI*, pages 364–369, 2005.

[BCM+07] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, second edition, 2007.

[Bie12] Meghyn Bienvenu. On the complexity of consistent query answering in the presence of simple ontologies. In *AAAI*, 2012.

[BLMS09] J.-F. Baget, M. Leclère, M.-L. Mugnier, and E. Salvat. Extending Decidable Cases for Rules with Existential Variables. In *IJCAI*, pages 677–682, 2009.

[BLMS11] J.-F. Baget, M. Leclère, M.-L. Mugnier, and E. Salvat. On Rules with Existential Variables: Walking the Decidability Line. *Artif. Intell.*, 175(9-10):1620–1654, 2011.

[BM02] J.-F. Baget and M.-L. Mugnier. The Complexity of Rules and Constraints. *J. Artif. Intell. Res. (JAIR)*, 16:425–465, 2002.

[BMT11] J.-F. Baget, M.-L. Mugnier, and M. Thomazo. Towards farsighted dependencies for existential rules. In *RR*, pages 30–45, 2011.

[BR13] Meghyn Bienvenu and Riccardo Rosati. Tractable approximations of consistent query answering for robust ontology-based data access. In *IJCAI*, 2013.

[BV81] C. Beeri and M. Vardi. The implication problem for data dependencies. In *ICALP'81*, volume 115 of *LNCS*, pages 73–85, 1981.

[CGK08] A. Calì, G. Gottlob, and M. Kifer. Taming the Infinite Chase: Query Answering under Expressive Relational Constraints. In *KR*, pages 70–80, 2008.

[CGL+05] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Dl-lite: Tractable description logics for ontologies. In *AAAI*, pages 602–607, 2005.

[CGL+07] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Tractable reasoning and efficient query answering in description logics: The DL-lite family. *J. Autom. Reasoning*, 39(3):385–429, 2007.

[CGL09] A. Calì, G. Gottlob, and T. Lukasiewicz. A General Datalog-Based Framework for Tractable Query Answering over Ontologies. In *PODS*, pages 77–86. ACM, 2009.

[CGP10] A. Calì, G. Gottlob, and A. Pieris. Query rewriting under non-guarded rules. In *AMW*, 2010.

[CLM81] A. K. Chandra, H. R. Lewis, and J. A. Makowsky. Embedded implicational dependencies and their inference problem. In *STOC*, pages 342–354, 1981.

[CM09] M. Chein and M.-L. Mugnier. *Graph-based Knowledge Representation: Computational Foundations of Conceptual Graphs*. Springer Publishing Company, Incorporated, 1 edition, 2009.

[CTS11] A. Chortaras, D. Trivela, and G. B. Stamou. Optimized query rewriting for OWL 2 QL. In *CADE*, pages 192–206, 2011.

[EOS+12] Thomas Eiter, Magdalena Ortiz, Mantas Simkus, Trung-Kien Tran, and Guohui Xiao. Query rewriting for horn-shiq plus rules. In *AAAI*, 2012.

[FKMP05] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data Exchange: Semantics and Query Answering. *Theor. Comput. Sci.*, 336(1):89–124, 2005.

[GHK+12] B. Cuenca Grau, I. Horrocks, M. Krötzsch, C. Kupke, D. Magka, B. Motik, and Z. Wang. Acyclicity conditions and their application to query answering in description logics. In *KR*, 2012.

[GHK+13] B. Cuenca Grau, I. Horrocks, M. Krötzsch, C. Kupke, D. Magka, B. Motik, and Z. Wang. Acyclicity notions for existential rules and their application to query answering in ontologies. *J. Artif. Intell. Res. (JAIR)*, 47:741–808, 2013.

[GMP13] Georg Gottlob, Marco Manna, and Andreas Pieris. Combining decidability paradigms for existential rules. *TPLP*, 13(4-5):877–892, 2013.

[GOP11] G. Gottlob, G. Orsi, and A. Pieris. Ontological queries: Rewriting and optimization. In *ICDE*, pages 2–13, 2011.

[GS12] G. Gottlob and T. Schwentick. Rewriting ontological queries into small nonrecursive datalog programs. In *KR*, 2012.

[ISG12] M. Imprialou, G. Stoilos, and B. Cuenca Grau. Benchmarking ontology-based query rewriting systems. In *AAAI*, 2012.

[KKPZ12] Stanislav Kikot, Roman Kontchakov, Vladimir V. Podolskii, and Michael Zakharyaschev. Long rewritings, short rewritings. In *Description Logics*, 2012.

[KLMT12] M. König, M. Leclère, M.-L. Mugnier, and M. Thomazo. A sound and complete backward chaining algorithm for existential rules. In *RR*, pages 122–138, 2012.

[KLMT13] M. König, M. Leclère, M.-L. Mugnier, and M. Thomazo. On the exploration of the query rewriting space with existential rules. In *RR*, 2013.

[KR11] M. Krötzsch and S. Rudolph. Extending decidable existential rules by joining acyclicity and guardedness. In *IJCAI*, pages 963–968, 2011.

[LLR+10] D. Lembo, M. Lenzerini, R. Rosati, M. Ruzzi, and D. F. Savo. Inconsistency-tolerant semantics for description logics. In *RR*, pages 103–117, 2010.

[LMS12] T. Lukasiewicz, M. V. Martinez, and G. I. Simari. Inconsistency handling in datalog+/- ontologies. In *ECAI*, 2012.

[LMTV12] N. Leone, M. Manna, G. Terracina, and P. Veltri. Efficiently computable datalog; programs. In *KR*, 2012.

[LSTW12] C. Lutz, I. Seylan, D. Toman, and F. Wolter. The combined approach to OBDA: Taming role hierarchies using filters. In *SSWS+HPCSW*, 2012.

[LTW09] C. Lutz, D. Toman, and F. Wolter. Conjunctive Query Answering in the Description Logic $\mathcal{EL}$ Using a Relational Database System. In *IJCAI*, pages 2070–2075, 2009.

[Mar09] B. Marnette. Generalized schema-mappings: from termination to tractability. In *PODS*, pages 13–22, 2009.

[MC13] José Mora and Óscar Corcho. Engineering optimisations in query rewriting for obda. In *I-SEMANTICS*, pages 41–48, 2013.

[OWL09] W3C OWL Working Group. *OWL 2 Web Ontology Language: Document Overview*. W3C Recommendation, 2009. Available at http://www.w3.org/TR/owl2-overview/.

[PUHM09] H. Pérez-Urbina, I. Horrocks, and B. Motik. Efficient query answering for OWL 2. In *International Semantic Web Conference*, pages 489–504, 2009.

[RA10] R. Rosati and A. Almatelli. Improving query answering over dl-lite ontologies. In *KR*, 2010.

[RMC12] M. Rodriguez-Muro and D. Calvanese. High performance query answering over DL-lite ontologies. In *KR*, 2012.

[Ros12] Riccardo Rosati. Prexto: Query rewriting under extensional constraints in dl - lite. In *ESWC*, pages 360–374, 2012.

[SM96] E. Salvat and M.-L. Mugnier. Sound and complete forward and backward chainingd of graph rules. In *ICCS*, pages 248–262, 1996.

[TBMR12] M. Thomazo, J.-F. Baget, M.-L. Mugnier, and S. Rudolph. A generic querying algorithm for greedy sets of existential rules. In *KR*, 2012.

[Tho13] M. Thomazo. Compact rewriting for existential rules. In *IJCAI*, 2013.

[TSCS13] Despoina Trivela, Giorgos Stoilos, Alexandros Chortaras, and Giorgos B. Stamou. Optimising resolution-based rewriting algorithms for dl ontologies. In *Description Logics*, pages 464–476, 2013.

[VSS12] T. Venetis, G. Stoilos, and G. B. Stamou. Incremental query rewriting for OWL 2 QL. In *Description Logics*, 2012.