UCQ-rewritings with Disjunctive Existential Rules and Conjunctive Queries with Negated Atoms

Enrique Matos Alfonso *, Alexandros Chortaras and Giorgos Stamou
Electrical and Computer Engineering, National Technical University of Athens, Greece
E-mails: gardero@image.ntua.gr, achort@cs.ntua.gr, gstam@cs.ntua.gr

Abstract. We focus on the problem of query rewriting for the framework of disjunctive existential rules. It is a well known approach for query answering on knowledge bases with incomplete data. We propose a rewriting technique that uses negative constraints and conjunctive queries to remove components from the disjunction in disjunctive existential rules. This process will eventually generate new existential rules. The generated rules can then be used to produce new rewritings considering the existing rewriting approaches for the existential rules framework. The algorithm is implemented in COMPLETO in order to provide complete rewritings for unions of conjunctive queries with negation. Additionally, we report some experiments to evaluate the viability of the proposed solution.

Keywords: Disjunctive Rules, Queries with Negation, Backward Chaining and Query Rewriting.

1. Introduction

Rules are very important elements in knowledge-based systems and incomplete databases [18]; they allow us to perform query answering over incomplete data and come up with complete answers. There are two main approaches to perform query answering and they define the way we use the rules in the knowledge base. Forward chaining approach [24] uses rules and facts to produce new facts. Backward chaining [17] uses rules to translate the query into other queries that also encode answers of the original query.

Example 1.1. *Let us consider a rule to define the grand-parent relationship based on the parent relationship:*

$$r = \forall X \forall Y\, parent(X, Z) \wedge parent(Z, Y) \rightarrow grand\text{-}parent(X, Y)$$

and the information that

$$parent(ana, maria) \wedge parent(maria, julieta).$$

Traditional database systems would fail to entail that the query $q = \exists Z\, grand\text{-}parent(ana, Z)$ *holds because it is not stated explicitly in the data. However, since the hypothesis of the rule holds we can then ensure (forward application of the rule) that the fact* $grand\text{-}parent(ana, julieta)$ *also holds. Adding this new information allows traditional database systems to conclude that q holds. Additionally, the rule r can also be applied to q in a backward direction to derive different queries that provide answers to the same question expressed in the original query, i.e.*

$$q' = \exists Z \exists Y\, parent(ana, Z) \wedge parent(Z, Y).$$

Forward chaining allows efficient query answering in systems where data is constant and queries change frequently. However, the size of the stored data can grow excessively and the method is not appropriate for frequently changing data. For some ontologies the approach does not always terminate [24] generating new data. Backward chaining, on the other hand, is ideal for constant queries and changing data, although the size of the rewriting can be exponential with respect to the initial size of the query or in some cases a finite rewriting does not exist. In both cases, the application of rules does not always terminate. Furthermore, hav-

*Corresponding author. E-mail: gardero@image.ntua.gr.

ing no restriction on the expressivity of the rules or having negated atoms on the query makes the entailment problem of the query undecidable [3].

In this paper we focus on query rewriting for conjunctive queries with negated atoms ($\mathsf{CQ}^\neg$) in the framework of disjunctive existential rules based on first-order logic (FOL) without functions symbols.

Disjunctive existential rules allow the representation of very expressive knowledge in FOL, e.g.

$$\forall X \exists Y \, \textit{is-parent}(X) \rightarrow \textit{father}(X, Y) \vee \textit{mother}(X, Y),$$

where disjunction and existentially quantified variables can appear on the right-hand side of the implication. Conjunctive query entailment is undecidable even in the case of existential rules without disjunction. However, under some restrictions the problem can become decidable [3]. To the best of our knowledge, the existing research for existential rules with disjunction is only based on forward chaining algorithms [11, 13].

Additionally, conjunctive queries with negation let us define the counter examples of disjunctive rules e.g.

$$\exists X \forall Y \forall Z \, \textit{person}(X) \wedge \neg \textit{married}(X, Y) \wedge \neg \textit{parent}(X, Z)$$

represents the cases where the following rule does not hold

$$\forall X \exists Y \exists Z \, \textit{person}(X) \rightarrow \textit{married}(X, Y) \vee \textit{parent}(X, Z).$$

Here we use the open world assumption, where the negation is associated to the "cannot" semantics and the example query expresses the question whether there is a person that cannot be married and cannot be a parent. The entailment problem for $\mathsf{CQ}^\neg$ is undecidable even for very simple types of rules [16]. On the other hand, the use of guarded negation in queries is proven to be decidable [7] over frontier-guarded existential rules. Yet, the existing rewriting based approaches in the literature [12, 19, 20] that propose implementations and experiments only deal with queries that introduce negation in very limited ways.

Having existential variables, disjunction and queries with negated atoms makes the entailment problem even more difficult. However, using these expressive resources in a smart way provides very interesting and useful decidable fragments.

More specifically, we are interested in rewriting a Union of conjunctive queries with negation ($\mathsf{UCQ}^\neg$) into a union of conjunctive queries (UCQ) called UCQ-rewriting and where each element in the resulting UCQ is a rewriting of the initial $\mathsf{UCQ}^\neg$.

Related Work König, M. et al. on [17] define a generic rewriting procedure for conjunctive queries with respect to existential rules which takes any rewriting operator as a parameter. They also define the properties of the rewriting operators that ensure the correctness and completeness of the algorithm. The authors prove that piece unification provides a rewriting operator with the desired properties. Finally, an implementation of their proposed algorithm is proposed together with some experiments. In this paper we extend their definition of piece unification to deal with disjunctive existential rules. We extend the existing rewriting algorithm for existential rules into a sound and complete algorithm that also supports disjunctive existential rules and queries with negated atoms. However, the proposed algorithm only terminates for some specific cases.

Pierre Bourhis et al. on [10] present a study of the complexity of query answering with respect to guarded disjunctive existential rules. The problem is a 2EXPTIME-hard even for the simplest guarded-based class of disjunctive existential rules. Different types of UCQs are also considered by the authors in order to reduce the complexity of the problem. However, the considered query languages does not have a positive impact on the complexity of the problem. The only significant decrease in the complexity was found for atomic queries and linear disjunctive existential rules. The authors proved that the problem for fixed atomic queries and fixed linear disjunctive existential rules is on the AC_0 complexity class by establishing that the problem is first-order rewritable, i.e., it can be reduced to the problem of evaluating a first-order query over a database. The rewriting algorithm that we propose in this paper stops for the fragment considered by Pierre Bourhis et al. The techniques and results presented by the authors can be used as a generic tool to study the complexity of query answering under fragments of Description Logics.

Some other research papers [2, 8, 9, 14] focus on the transformation of the problem of query answering with respect to guarded (disjunctive) existential rules into query answering with respect to (disjunctive) datalog programs. They manage to get rid of existential variables and Ahmetaj et al. on [2] provide a transformation of the problem that yields a polynomial size (disjunctive) datalog program when the maximal num-

ber of variables in the (disjunctive) existential rules is bounded by a constant. The rest of the translations proposed by other authors [8, 9, 14] are of exponential size.

Outline of our contributions. We introduce two restricted forms of FOL resolution (semi-Horn resolution and constraint resolution) that are sound and complete and where subsumption holds for restricted types of consequences. The number of choices at the moment of selecting clauses to perform resolution steps is reduced in these two types of resolution with respect to the number of choices we have in unrestricted resolution.

Based on one of the constraint resolution methods, we propose an algorithm to compute UCQ-rewritings that is capable of handling both disjunctive existential rules and conjunctive queries with negation. The algorithm is proved to be sound and complete and it terminates for the cases where there is a finite UCQ-rewriting of input query and the negative constraints with respect to the (disjunctive) existential rules. We also present two theorems with sufficient conditions for the termination of the algorithm. One case requires disconnected disjunctive existential rules (the body and the head do not share variables) and the other case is based on knowledge bases where all the elements are linear (rules with at most one atom in the body queries with at most one positive atom).

Additionally, conjunctive queries with negated atoms and answer variables are considered with respect to existential rules. We present two theorems with sufficient conditions for the termination of the rewriting process in this special case. One case requires that the variables in the positive and negated atoms of the query are also part of the answer variables of the query. The second case is based on queries with only one positive atom and an unbounded number of negated atoms.

Finally, we implemented into the system COMPLETO, the proposed algorithm for rewriting conjunctive queries with negated atoms and answer variables with respect to existential rules. We also performed experiments to evaluate the viability of the proposed solution.

Paper organization. Section 2 introduces the relevant theory needed to understand the rest of the paper. Subsection 2.1 introduces concepts related to First Order Logic resolution and Subsection 2.2 presents the disjunctive existential rules Framework. Section 3 introduces two types of resolution derivations and a backward chaining rewriting algorithm for disjunctive existential rules. Section 4 describes an implementation of the proposed rewriting algorithm in order to find UCQ-rewritings for queries with negated atoms. Experiments describing the performance of our implementation are presented in Subsection 4.1. Finally, Section 5 presents an overview of the paper with some conclusions.

2. Preliminaries

In this section we introduce the basic concepts related to the First Order Logic resolution process. Resolution is the base of all the reasoning processes we describe in this paper. All steps in high level reasoning processes can be tracked down to sequences of resolution steps that ensure the correctness. Additionally, we also describe the framework of disjunctive existential rules and present the definition of conjunctive queries with negated atoms.

2.1. First Order Logic Resolution

We assume the reader is familiar with the standard definition of First Order Logic Formulas. In this paper we focus on First Order Logic Formulas without function symbols over a finite set of predicate names and a finite set of constant symbols. We also adopt the standard definitions for substitutions and unifiers, as they are rarely modified in the literature. However, we refer the reader to [22, 23] in case a background reading is needed.

In order to make the modification of formulas more compact and easier to understand we introduce the definition of conjunctive (disjunctive) set formula (CSF and DSF).

Definition 2.1 (Conjunctive (Disjunctive) Set Formula)**.** *A Conjunctive (Disjunctive) set formula (*CSF *and* DSF *respectively) is a set of formulas* $\{F_1, \ldots, F_n\}$ *and the interpretation of this formula is equivalent to the interpretation of the conjunction (disjunction) of the formulas in the set:* $F_1 \wedge \ldots \wedge F_n$ $(F_1 \vee \ldots \vee F_n)$.

For a given set of formulas $\{F_1, \ldots, F_n\}$ *a* CSF *containing these formulas will be denoted as* $F_1, \ldots, F_n$. *On the other hand, a* DSF *will be denoted by* $[F_1, \ldots, F_n]$. *However, in case* CSF *is a subformula of another set formula we will use parenthesis to avoid ambiguity e.g.* $[(A, B), D]$ *is equivalent to* $(A \wedge B) \vee D$. *Finally, we have that an empty* CSF *is denoted by and it is equivalent to* $\top$ *and an empty* DSF *is denoted by and it is equivalent to* $\bot$.

Set operators can then be used to combine set formulas of the same type and obtain a new formula. The interpretation of the new formula is the interpretation of the resulting set formula. Additionally, formulas that are equivalent and belong to the same set can be *collapsed* into one element e.g.

$$[\neg A] \cup [A \to \bot, B] \equiv [\neg A, B].$$

We can easily prove the validity of the following axioms:

1. A DSF and a CSF of one element are equivalent:

$$[F] \equiv F.$$

2. De Morgan's Law allows to change from DSF to a CSF using negation:

$$\neg[F_1, \dots, F_n] \equiv \neg F_1, \dots, \neg F_n.$$

3. *Flattening* set formulas is possible when they are of the same type i.e for $B \in F$, two set formulas of the same type, we have that:

$$F \equiv (F \setminus \{B\}) \cup B.$$

Additionally, the entailment operator in FOL is based on a CSF $A_1, \dots, A_n$ of axioms A_i on the left-hand side of the entailment symbol:

$$A_1, \dots, A_n \models F.$$

The right-hand side of the entailment operator is defined as a DSF. For CSFs B and A (where $B \subseteq A$) and DSFs C and F (where $C \subseteq F$) we can ensure that $A \models F$ if and only if $A \setminus B \models F \cup \neg B$; likewise $A \models F$ if and only if $A \cup \neg C \models F \setminus C$.

Following, we recall some of the concepts that are more relevant to the theory presented in this paper.

An atom is a formula $a(x_1, \dots, x_n)$ where a is a *predicate* of *arity* n (denoted by a/n). The arguments x_i can either be *variables* or *constants*. A *literal* is an atom or a negated atom. The *complement* $\bar{l}$ of a literal l is $\neg a(X)$ for $l = a(\mathbf{X})$ and $a(\mathbf{X})$ in case $l = \neg a(\mathbf{X})$. A literal has a *positive* polarity if it is an atom and a *negative* polarity if it is the negation of an atom.

A formula is *ground* if it contains no variables in its atoms. Variables in a formula can be *universally quantified*, *existentially quantified* or *free*. A formula without free variables is *closed*. The set of all the variables present in a formula F is denoted by $vars(F)$.

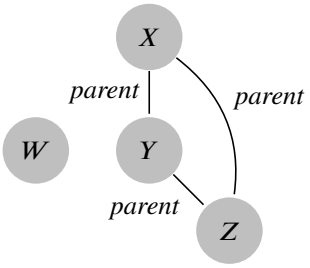

Fig. 1. Hypergraph corresponding to a CSF.

A CSF of atoms can represent a hypergraph using the set of variables in the arguments of the atoms as nodes and the predicates as labeled hyperedges. Each atom in the formula represents a hyperedge. With this notion we can define some properties for CSFs of atoms.

The *cardinality* of a CSF of atoms F is the number of variables in the formula $card(F) = |vars(F)|$. Two variables u and v in a CSF of atoms F are *connected* iff they both belong to the same atom ($\exists A \in F | \{v, u\} \subseteq vars(A)$) or if there is another variable z in F that is connected to both u and v.

A CSF of atoms F is connected if all the atoms in it contain variables and all the variables are connected to each other. An atom that has only constants in its arguments is a connected formula that is not connected to any other atom and has a cardinality of zero. It is also represented by an empty hypergraph. We can also find a partition of connected CSFs for a given CSF of atoms $\cup U_i = F$. The *connected cardinality* of F is defined as the maximum cardinality of the connected CSFs in F and denoted as $card^*(F) = \max_i (card(U_i))$.

Example 2.1. *The* CSF

$$F = parent(X, Y), parent(Y, Z), \\ grand\text{-}parent(X, Z), person(W)$$

is represented by the hypergraph in Fig. 1. We can split F in two connected components

$$\{(person(W)), \\ (parent(X, Y), parent(Y, Z), \\ grand\text{-}parent(X, Z))\}.$$

The connected cardinality of F is 3, the cardinality of the greatest connected component.

Lemma 2.1. *Let $\mathcal{K}$ be a* CSF *and $\{\dots, U_i, \dots\}$ a partition of connected* CSF*s for a given F* CSF *of atoms. Then,*

$$\mathcal{K} \models F \text{ iff } \mathcal{K} \models U_i \text{ for every } U_i.$$

Proof. A detailed proof is given by Tessaris [25]. However, the reader can clearly see that since no variable is shared between the connected components U_i, the interpretations of each component can be combined without conflicts on the assignments of the variables i.e. $\mathcal{I}_i \cup \mathcal{I}_j \models U_i, U_j$ iff $\mathcal{I}_i \models U_i$ and $\mathcal{I}_j \models U_j$. □

Lemma 2.2. *Let k be a natural number. There is a finite number of equivalence classes in the set of* CSF *of atoms that have connected cardinality at most k.*

Proof. Two CSF of atoms are equivalent iff they are unifiable by a renaming substitution, i.e. a substitution that assigns new names to the variables of the formula. Therefore, we can focus only on the number of variables present in a connected CSF of F. Because we have a finite number of predicates symbols and constant symbols and at most k different variables, we can combine them in a finite number of ways (denoted by M) to represent a connected CSF. In a CSF with more than M connected CSFs we will have that some of the connected components will be a renaming of another one, then keeping only one of them will be enough considering Lemma (2.1). Hence, there will be at most 2^M different equivalence classes. □

Example 2.2. *Let us take one predicate s/2 and the maximum cardinality $k = 2$. The different equivalence classes of* CSF*s are:*

1. $s(X, X), s(X, Y), s(Y, X), s(Y, Y)$
2. $s(X, X), s(X, Y), s(Y, X)$
3. $s(X, X), s(X, Y), s(Y, Y)$
4. $s(X, X), s(X, Y)$
5. $s(X, X), s(Y, X)$
6. $s(X, Y), s(Y, X)$
7. $s(X, Y)$
8. $s(X, X)$

Any other CSF *of cardinality less than or equal to 2 will be equivalent to one of the above formulas. Notice that in this restricted example we do not have constants.*

A *clause* C is a DSF $[l_1, \ldots, l_n]$ of literals l_i, where all the variables are universally quantified. A formula F is in conjunctive normal form (CNF) if it is a CSF of clauses i.e:

$$[l'_1, \ldots, l'_n], \ldots, [l^{(m)}_1, \ldots, l^{(m)}_{n^{(m)}}].$$

Every formula in FOL can be transformed into an equisatisfiable CNF formula using Standarization, Skolemization, De Morgan's laws and distributivity of conjunction ($\wedge$) and disjunction ($\vee$) logical operators.

An *instance* of a clause C is the result of applying a substitution θ to the clause i.e. $C\theta$. If two or more literals of the same polarity in a clause C are unifiable and θ is their most general unifier, then the clause $C\theta$ is called a *factor* of C and the application of θ is called *factorization*.

Definition 2.2 (Binary Resolution Rule)**.** *Let C_1 and C_2 be two clauses with no variables in common and let $L_1 \in C_1$ and $L_2 \in C_2$ be complementary literals with respect to a most general unifier $\sigma = mgu(\{L_1, \overline{L_2}\})$. The binary resolvent of C_1 and C_2 with respect to the literals L_1 and L_2 is the clause:*

$$C_1 \cup_r C_2 = (C_1\sigma \setminus [L_1\sigma]) \cup (C_2\sigma \setminus [L_2\sigma]).$$

C_1 and C_2 are said to be clashing clauses. The resolvent *$C_1 \cup_r C_2$ of two clauses is a binary resolvent of factors $C_i\sigma_i$ of the clauses i.e. $C_1\sigma_1 \cup_r C_2\sigma_2$.*

It is easy to show that resolution is sound i.e. $C_1, C_2 \models C_1 \cup_r C_2$. Consequently, resolution can be used to deduce new clauses and also to prove that a formula does not hold if we are able to derive the empty clause.

Definition 2.3 (Derivation (Refutation))**.** *Let Σ be a set of clauses and C a clause. A derivation of C from Σ is a finite sequence of clauses $R_1, \ldots, R_k = C$, such that each R_i is either in Σ, or a resolvent of two clauses in $\{R_1, ..., R_{i-1}\}$. If such a derivation exists, we write $\Sigma \models_r C$. We then say C can be derived from Σ. A derivation of the empty clause $\perp$ from Σ is called a refutation of Σ. The steps of a resolution derivation are the resolution operations performed to obtain the resolvents in the sequence.*

Sometimes it is useful to know which clauses were used in order to produce a resolvent R_i in a resolution derivation. In those cases a corresponding graph or tree representation can be helpful.

Definition 2.4 (Derivation (Refutation) Graph)**.** *Let Σ be a set of clauses and C a clause such that $\Sigma \models_r C$. A derivation (refutation) graph of C from Σ is a graph structure where each node represents a resolvent of two clauses or a a clause from Σ. There will be an edge from each resolvent to the clauses used in the resolution step to derive it.*

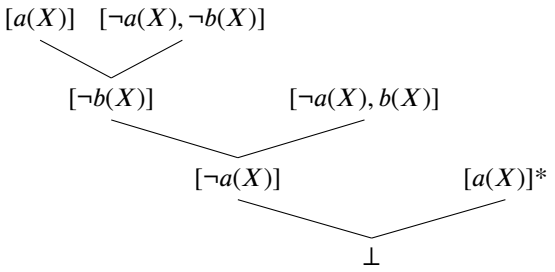

Fig. 2. Derivation tree

We focus on the important part of derivations i.e. every clause in the derivation is used in at least a resolution step with the exception of the last one. As a consequence the derivation graph can be seen as a tree-like shape where the last resolvent C is the *root* of the tree and the leaves are clauses from Σ. Cloning nodes with more than one input edge (clauses used in several resolution steps) can help us transforming the derivation graph into a *derivation tree*. Graphically we find more convenient to draw such trees or graphs upside-down (Figure 2).

Derivations and derivation graphs (trees) are closely related. They keep track of the clauses that were used to produce each resolvent. From a derivation $R_1, \ldots, R_k$ the process of building the derivation graph is straightforward. Also, finding a sequence of clauses that form a derivation is a matter of taking the nodes of the graph in pre-order traversal starting from the root and ideally without repeating clauses or factors of the same clauses.

Theorem 2.1 (Soundness of Derivation)**.** *Let Σ be a set of clauses, and C be a clause. If $\Sigma \models_r C$, then $\Sigma \models C$.*

Proof. This is straightforward consequence of the soundness of the resolution rule. □

If a clause holds, an instance of it extended with possibly more literals will also hold. This is based on the definitions of the disjunction operator and the universal quantification of the variables in the clauses.

Definition 2.5 (Subsumption)**.** *Let C and D be clauses, we say C subsumes D if there exists a substitution θ such that $C\theta \subseteq D$.*

Definition 2.6 (Deduction)**.** *Let Σ be a set of clauses and C a clause. We say that there exists a deduction of C from Σ, written as $\Sigma \models_r^d C$, if C is a tautology, or if there exists a clause D such that $\Sigma \models_r D$ and D subsumes C. If $\Sigma \models_r^d C$, we say C can be deduced from Σ.*

Resolution steps and derivations between ground instances of some clauses ensure that there are also corresponding steps or derivations using the non ground version of the clauses. This process is known as *lifting* a resolution step or a derivation.

Below we mention known theorems taken from the literature [22, 23].

Theorem 2.2 (Lifting Lemma)**.** *Let C'_1, C'_2 be ground instances of C_1, C_2, respectively. Let C' be a ground resolvent of C'_1 and C'_2. Then there is a resolvent C of C_1 and C_2 such that C' is a ground instance of C.*

Theorem 2.3 (Derivation Lifting)**.** *Let Σ be a set of clauses, and Σ' a set of ground instances of clauses from Σ. Suppose $R'_1, ..., R'_k$ is a derivation of the clause R'_k from Σ'. Then there exists a derivation $R_1, ..., R_k$ of the clause R_k from Σ, such that R'_i is an instance of R_i, for each $i \in \{1, \ldots, k\}$.*

Finally, resolution derivations allow us to infer clauses that are logical consequences of the initial knowledge in a complete way.

Theorem 2.4 (Subsumption Theorem)**.** *Let Σ be a set of clauses, and C be a clause. Then $\Sigma \models C$ iff $\Sigma \models_r^d C$*

Theorem 2.5 (Refutation Completeness of Resolution)**.** *Let Σ be a set of clauses. Then Σ is unsatisfiable iff $\Sigma \models_r \bot$.*

Performing resolution steps in a *breath-first* manner ensures we find the empty clause for unsatisfiable formulas. However, for satisfiable formulas we might never stop generating new clauses that are not subsumed by the existing clauses. In general, resolution allows us to define sound and complete algorithms but we cannot ensure termination for all formulas in the FOL fragment.

The resolution operator $\cup_r$ has very useful properties that allow us to transform derivations without affecting the consequence.

Property 2.1 (Symmetry)**.** *Let C_1 and C_2 be clashing clauses, then the resolvent can be computed symmetrically:*

$$C_1 \cup_r C_2 \equiv C_2 \cup_r C_1.$$

Property 2.2 (Distributivity)**.** *Let C_1, C_2 and C_3 be clauses such that C_3 resolves with the resolvent from C_1 and C_2 using literals from both C_1 and C_2, then we can ensure the distributivity property i.e.*

$$(C_1 \cup_r C_2) \cup_r C_3 \models (C_1 \cup_r C_3) \cup_r (C_2 \cup_r C_3).$$

Notice that the converse holds only if there is a *mgu* for the literals used in the resolution with respect to both individual clauses $(C_1 \cup_r C_3)$ and $(C_2 \cup_r C_3)$.

Property 2.3 (Commutativity)**.** *Let C_1, C_2 and C_3 be clauses such that C_3 resolves with the resolvent from C_1 and C_2 but only using literals from C_1, then we can ensure the commutativity property i.e.*

$$(C_1 \cup_r C_2) \cup_r C_3 \equiv (C_1 \cup_r C_3) \cup_r C_2.$$

Proving the above properties is straightforward if we consider them over ground instances of the clauses and we track the set operations on ground literals. As a consequence of the derivation lifting theorem 2.3, we can ensure that the properties also hold for general resolution over non ground clauses.

Example 2.3. *Let us illustrate the distributivity property with clauses $C_1 = [a(X), b(X)]$, $C_2 = [a(X), \neg b(X)]$ and $C_3 = [\neg a(X), c(X)]$. We have that:*

$$(C_1 \cup_r C_2) \cup_r C_3 = [a(X)] \cup_r [\neg a(X), c(X)] = [c(X)]$$

and

$$\begin{aligned}(C_1 \cup_r C_3) \cup_r (C_2 \cup_r C_3) &= [b(X), c(X)] \cup_r \\ &\quad [\neg b(X), c(X)] \\ &= [c(X)]\end{aligned}$$

Also, $[c(X)] \implies [c(X)]$.

2.2. Disjunctive Existential Rules and Conjunctive Queries with Negation Framework

A *conjunctive query* CQ is a CSF $l_1, \ldots, l_n$ of positive literals l_i where all the variables are existentially quantified i.e. $\exists \mathbf{X}\ l_1, \ldots, l_n$. Queries that allow negation in the literals l_i are called conjunctive queries with negation $\mathsf{CQ}^\neg$. In order to avoid domain dependant queries, all the variables present in positive literals $\mathbf{X}$ appear existentially quantified and the all variables that are present only in negative literals $\mathbf{Z}$ will be universally quantified i.e. $\exists \mathbf{X} \forall \mathbf{Z}\ l_1, \ldots, l_n$. Because the rules for quantifying variables are straightforward we omit quantifiers e.g. $\exists X \forall Y\, person(X), \neg married(X, Y)$ will be denoted by $person(X), \neg married(X, Y)$. The set of variables that are present in both positive and negative literals is called the *frontier* of the query. Notice that for now we do not introduce the concept of *answer variables*. Therefore, the queries we define are also known as *Boolean* conjunctive queries. A DSF of conjunctive queries (conjunctive queries with negation) is referred to as union of conjunctive queries UCQ (union of conjunctive queries with negation $\mathsf{UCQ}^\neg$)

A *fact* is a CSF $l_1, \ldots, l_n$ of positive literals l_i, where all the present variables are existentially quantified e.g. $parent(ana, Y), parent(maria, Y)$. Here existential quantifiers are also omitted.

A closer look to the definition of facts can reveal the similarities to the definition of conjunctive query. More precisely, the definition of fact is equivalent to the definition of Boolean conjunctive query. However, facts are used to express existing knowledge and queries represent questions i.e. they both have different roles in the process of reasoning.

An *existential rule* is a closed formula:

$$\forall \mathbf{X}\, \exists \mathbf{Y}\ B \rightarrow H,$$

where B (*body*) and H (*head*) are CSF of positive literals. Variables in the body $X = vars(B)$ are universally quantified and variables present only in the head $Y = vars(H) \setminus vars(B)$ are existentially quantified. The *frontier* of a rule is the set of variables that are present in both body and head of the rule $vars(B) \cap vars(H)$. The quantifiers are omitted in the notation of existential rule.

A *disjunctive existential rule* is a closed formula:

$$\forall \mathbf{X}\, \exists \mathbf{Y}\ B \rightarrow H,$$

where B (*body*) is a CSF of positive literals. The quantification of variables follows the same rules described in the definition of an existential rule. However, the head H is a DSF, where all $H' \in H$ are a CSF of positive literals. A disjunctive existential rule with an empty head is a *negative constraint*, i.e. $B \rightarrow \bot$. However, if from the context is clear that we refer to a constraint we can omit the "$\rightarrow \bot$" and write only the body B of the constraint.

We say a CSF of atoms Q *depends* on a rule r iff there is a CSF of atoms F such that $F \not\models Q$ and $F, r \models Q$. A rule r_i depends on a rule r_j iff the body of r_i depends on the rule r_j. The concept of rules dependencies allows us to define the *graph of rule dependencies* (GRD) by building a graph where the rules are the nodes and a directed edge between two nodes represents the existence of a dependency between the corresponding rules.

A *knowledge base* $(\langle \mathcal{D}, \mathcal{R}, \mathcal{R}^\vee, \mathcal{C} \rangle)$ is a CSF of a CSFs of facts $(\mathcal{D})$, a CSF of existential rules $(\mathcal{R})$, a CSF of disjunctive existential rules $(\mathcal{R}^\vee)$ and a CSF of negative constraints $(\mathcal{C})$.

Example 2.4 (Knowledge Base)**.** *Let's define some knowledge about family relationships in an example knowledge base.*

– *Facts:*

$$\mathcal{D} = (parent(Y, ana), sibling(Y, maria)), \\ sibling(ana, juan)$$

– *Existential rules:*

$$\mathcal{R} = (sibling(X, Y) \rightarrow parent(Z, X), \\ parent(Z, Y)), \\ (sibling(X, Y) \rightarrow sibling(Y, X))$$

– *Negative constraints:*

$$\mathcal{C} = (sibling(X, Y), parent(X, Y)), \\ (same\text{-}age(X, Y), parent(X, Y)) \\ (parent(X, Y), parent(Y, X)) \\ (parent(X, X))$$

– *Disjunctive existential rules :*

$$\mathcal{R}^{\vee} = first\text{-}deg\text{-}relative\ (X, Y) \rightarrow \\ [parent(X, Y), \\ parent(Y, X), \\ sibling(X, Y)] \quad (1)$$

Notice that the existential rule has an existential variable Z that refers to an anonymous entity that is the father of both siblings i.e. two people are siblings if they share a parent. We can also see a rule stating that the sibling/2 predicate is symmetric. We could also add symmetry for same-age/2. There is also a fact that states that there is an anonymous entity Y that is a parent of both maria and ana. The negative constraints states the impossibility of a person being parent of his sibling and also of a person of the same age. Additionally, with the negative constraints we express that parent/2 is asymmetric and irreflexive. Finally, the disjunctive rule defines the relation that represents the first degree relative concept [21]: a parent, a child (inverse of parent), or a sibling.

In this paper, we analyze the problem of query entailment, i.e.

$$\mathcal{R}, \mathcal{R}^{\vee}, \mathcal{C}, \mathcal{D} \models \mathcal{Q}, \quad (2)$$

where $\mathcal{Q}$ is a UCQ (UCQ$^{\neg}$). In particular we solve the entailment problem by reducing it to the entailment of UCQs with respect to the set of facts $\mathcal{D}$ i.e.

$$\mathcal{D} \models \mathcal{Q}',$$

where $\mathcal{Q}'$ is a union of conjunctive queries UCQ.

We say $\mathcal{Q}'$ is a UCQ-rewriting of $\mathcal{Q}$ with respect to $\mathcal{R}$, $\mathcal{R}^{\vee}$ and $\mathcal{C}$ if for all set of facts $\mathcal{D}$ it holds that

$$\mathcal{D} \models \mathcal{Q}' \text{ implies } \mathcal{R}, \mathcal{R}^{\vee}, \mathcal{C}, \mathcal{D} \models \mathcal{Q}.$$

If the converse

$$\mathcal{R}, \mathcal{R}^{\vee}, \mathcal{C}, \mathcal{D} \models \mathcal{Q} \text{ implies } \mathcal{D} \models \mathcal{Q}'$$

also holds for all set of facts $\mathcal{D}$, we say $\mathcal{Q}'$ is a complete UCQ-rewriting of $\mathcal{Q}$ with respect to $\mathcal{R}$, $\mathcal{R}^{\vee}$ and $\mathcal{C}$.

Additionally, we may be interested in the values that some of the variables in $\mathcal{Q}$ take. However, this does not change the semantic definition of conjunctive queries and therefore, we discuss it later in Subsection 3.4.

3. Backward Chaining with Disjunctive Knowledge

In this section we present two types of resolution (semi-Horn resolution and constraint resolution) that are sound and complete. Additionally, these types of resolution reduce the number of choices in the resolution steps. Constraint resolution is then translated into backward rewriting steps using disjunctive existential rules. Thus, allowing to define a rewriting algorithm for the framework of disjunctive existential rules that is able to solve the query entailment problem (2).

3.1. Constraint Resolution

The entailment problem (2) can be transformed into a consistency check problem:

$$\mathcal{R}, \mathcal{R}^{\vee}, \mathcal{C}, \mathcal{D}, \neg\mathcal{Q} \models \bot \quad (3)$$

in order to be solved using resolution refutation. Depending on the expressivity of the queries in $\mathcal{Q}$ the negation can yield new facts, negative constraints, existential rules or even disjunctive existential rules.

The process of resolution can be controlled in order to decrease the number of choices we have every time we perform a resolution step. This might result in bigger derivations but the algorithm to generate them is simpler.

Table 1
Different types of clauses in the disjunctive existential rules framework.

Name	Properties
Rule clause (RC)	$\|C\|^+ = 1 \wedge \|C\|^- \geq 1$
Fact clause (FC)	$\|C\|^+ = 1 \wedge \|C\|^- = 0$
Constraint clause (CC)	$\|C\|^+ = 0 \wedge \|C\|^- \geq 1$
Disjunctive RC (DRC)	$\|C\|^+ \geq 2 \wedge \|C\|^- \geq 0$

The main goal of the restrictions we introduce is to focus on generating clauses without positive literals and any number of negative literals. After, the process can continue with eliminating those remaining literals without introducing positive literals.

Definition 3.1 (Positive/Negative Charge)**.** *The positive (negative) charge $|C|^+$ ($|C|^-$) of a clause C is defined as the number of positive (negated) literals present in the clause.*

From the previous definition follows that clauses corresponding to the negation of CQs (or negative constraints) have zero positive charge. We refer to them as *constraint clauses*. A *Horn clause* C has a positive charge smaller than or equal to one i.e. $|C|^+ \leq 1$. The Skolemized version of a (disjunctive) existential rule r can produce several clauses with positive literals. A clause with only one positive literal is called *rule clause* and a clause with more than one positive literals is called *disjunctive rule clause*. Facts generate ground clauses containing only one literal and with positive polarity and we call them *fact clauses*. Table 1 shows a compact view of the properties that define the different types of clauses we will find when doing resolution based on a knowledge base coming from the disjunctive existential rules framework. Additionally, DRCs could also represent disjunctive facts with no negated atoms when the body of the rule is empty. Finally, a $\mathsf{CQ}^\neg$ can be translated into a FC, RC or a DRC depending on its positive and negative charge.

Example 3.1. *From example 2.4 the following are the corresponding clauses:*

– *Fact Clauses:*

$$[parent(f_0, ana)], [parent(f_0, ana)]$$
$$[sibling(ana, juan)]$$

– *Rule Clauses:*

$$[\neg sibling(X, Y), parent(f_2(X, Y), X)]$$
$$[\neg sibling(X, Y), parent(f_2(X, Y), Y)]$$
$$[\neg sibling(X, Y), sibling(Y, X)]$$

Table 2
Properties of the resolvent C_3 from different type of clauses C_1 and C_2.

C_1	C_2		
	CC	DRC	RC
FC	$\|C_3\|^- < \|C_1\|^-$	$\|C_3\|^- < \|C_1\|^-$	$\|C_3\|^- < \|C_1\|^-$
RC	$\|C_3\|^+ = 0$	$\|C_3\|^+ \leq \|C_2\|^+$	$\|C_3\|^+ = 1$
DRC	$\|C_3\|^+ < \|C_1\|^+$	$\|C_3\|^+ > \max(\|C_1\|^+, \|C_2\|^+)$	
CC	does not exist		

– *Constraint Clauses:*

$$[\neg sibling(X, Y), \neg parent(X, Y))],$$
$$[\neg same\text{-}age(X, Y), \neg parent(X, Y)]$$
$$[\neg parent(X, Y), \neg parent(Y, X)]$$
$$[\neg parent(X, X)]$$

– *Disjunctive Rule Clauses :*

$$[\neg first\text{-}deg\text{-}relative(X, Y), parent(X, Y),$$
$$parent(Y, X), sibling(X, Y)]$$

Table 2 shows properties of the resolvent from different types of clauses when we perform resolution steps. Any resolution refutation should involve resolution steps with respect to fact clauses, they always produce clauses with a smaller negative charge. Those resolution steps can be arranged so that they are performed in the last part of the resolution derivation. Additionally, this type of resolution steps is generally linked to Data Retrieval with respect to databases. For this reason, we mainly focus in the initial segment of the reorganized resolution derivation until it reaches a clause that has only negated atoms. This process is linked to producing conjunctive query rewritings and the resulting clause will correspond to the negation of a conjunctive query. For this reason, we want to focus on derivations that produce clauses with a non increasing positive charge. Therefore, we need to avoid resolutions steps involving two DRC, i.e. we need to use in every resolution step at least one Horn clause.

Definition 3.2 (Semi-Horn Derivation)**.** *A resolution derivation (refutation) $\Sigma \vDash_r C$ of a clause C, is semi-Horn iff all the resolution steps are based on binary resolution with at least a Horn factor of a clause. A semi-Horn derivation of a clause C is written as $\Sigma \vDash_{sH} C$. Similarly, there is a semi-Horn deduction of C from Σ, written as $\Sigma \vDash^d_{sH} C$, if C is a tautology or if there is a clause D such that $\Sigma \vDash_{sH} D$ and D subsumes C.*

Theorem 3.1 (Completeness of Semi-Horn Resolution Derivations)**.** *A set of clauses* Σ *is unsatisfiable iff* $\Sigma \models_{sH} \bot$.

Proof. Semi-Horn derivations are sound because they are also resolution derivations. The proof is based on being able to transform every resolution refutation into a semi-Horn refutation.

Horn clauses are very important in a resolution refutation. We need them in order to generate clauses with fewer positive literals and eventually end up with the empty clause.

Because Σ is unsatisfiable, from the completeness theorem of resolution refutations we can ensure that $\Sigma \models_r \bot$. Let us focus on the resolution derivation tree that corresponds to the refutation of Σ. Let C_{i_1} be the closest resolvent to the root that is obtained by applying binary resolution with two non Horn factor clauses: $C_{i_1} = \left(C_{i_0} \cup_r C_{j_0}\right)$. We can assume that on the path to the root of the tree the resolution steps will then involve only Horn factors of clauses $C_{j_1}, \ldots, C_{j_k}$:

$$\left(C_{i_0} \cup_r C_{j_0}\right) \cup_r C_{j_1} \cup_r \ldots \cup_r C_{j_k} = \bot.$$

These Horn clauses clash on literals coming from C_{j_0} and/or C_{i_0}. Therefore, we can apply the distributivity (commutativity) property of the resolution operator and perform the resolution steps with respect to clauses $C_{j_1}, \ldots, C_{j_k}$ in the same order.

$$\left(C_{i_0} \cup_r^{l_1} C_{j_1} \cup_r \ldots \cup_r^{l_k} C_{j_k}\right) \cup_r \left(C_{j_0} \cup_r^{l_1} C_{j_1} \cup_r \ldots \cup_r^{l_k} C_{j_k}\right) = \bot.$$

Notice that some of the resolution steps might be skipped because the literal it resolves upon was not present in one of the resolving clauses. Those cases are based on using the commutativity property instead of distributivity.

In the end, both clauses will have factors with complementary literals:

$$\left(C_{i_0} \cup_r^{l_1} C_{j_1} \cup_r \ldots \cup_r^{l_k} C_{j_k}\right)\sigma_1 = [l_0]$$
$$\left(C_{j_0} \cup_r^{l_1} C_{j_1} \cup_r \ldots \cup_r^{l_k} C_{j_k}\right)\sigma_2 = [\overline{l_0}]$$

Therefore, the resolution between them will involve Horn factor clauses. However, we need to be careful and apply the resolution steps with respect to clauses $C_{j_1}, \ldots, C_{j_k}$ using the same complementary literals and the same Horn factors of the clauses.

The process described can continue in a top down (from the root down to the leaves) manner until all the steps involving two non Horn clauses are removed. This way, every refutation can be transformed into a semi-Horn refutation. □

In general, the subsumption theorem does not hold for semi-Horn deductions.

Example 3.2. *Let*

$$C_1 = [\neg r(X,Y), \neg r(X,Z), \neg a(Y), \neg b(Z), r(Y,Z), r(Z,Y)]$$

and

$$C_2 = [\neg r(X'',Z'), \neg a(Z''), \neg b(Z'), r(Z'',Z'), r(Z',Z''), \neg r(X'',Y''), \neg r(X'',Z''), \neg a(Y''), \neg b(Z''), r(Z'',Y'')]$$

a clause obtained from $C_1 \cup_r C_1$. *As a consequence of the correctness of resolution, for any set of Horn clauses* $\mathcal{H}$, *we have that* $\mathcal{H} \cup \{C_1\} \models C_2$. *Nevertheless,* $\mathcal{H} \cup \{C_1\} \not\models^d_{sH} C_2$. *The clause* C_2 *cannot be obtained by applying resolution with respect to Horn clauses because* $|C_2|^+ > |C_1|^+$ *and resolution with respect to Horn clauses cannot generate clauses with greater positive clause charge.*

However, if we restrict it to Horn consequences the subsumption theorem for semi-Horn deductions holds.

Theorem 3.2 (Semi-Horn Subsumption for Horn Consequences)**.** *Let* Σ *be a set of clauses and* C *a Horn clause. Then* $\Sigma \models C$ *iff* $\Sigma \models^d_{sH} C$.

Proof. Based on the subsumption theorem for general resolution derivations (2.4), the fact that $\Sigma \models C$ implies that we have a resolution deduction of C i.e. $\Sigma \models^d_r C$. Such derivation for sure involves resolution steps with respect to some Horn clauses or Horn factors clauses.

If we use the process described on the proof of Theorem (3.1), the derivation of C can also be transformed into a semi-Horn derivation.

Let C_{i_1} be the closest resolvent to the root (C) of the corresponding derivation tree that is obtained by applying binary resolution with two non Horn factor clauses: $C_{i_1} = \left(C_{i_0} \cup_r C_{j_0}\right)$. We can assume that on the path

to the root of the tree the resolution steps involve only Horn factors of clauses $C_{j_1}, \ldots, C_{j_k}$:

$$\left|\left(C_{i_0} \cup_r C_{j_0}\right) \cup_r C_{j_1} \cup_r \ldots \cup_r C_{j_k}\right|^+ \leq 1$$

implies that

$$\left|\left(C_{i_0} \cup_r C_{j_1} \cup_r \ldots \cup_r C_{j_k}\right) \cup_r \left(C_{j_0} \cup_r C_{j_1} \cup_r \ldots \cup_r C_{j_k}\right)\right|^+ \leq 1$$

Therefore, at least one of the factors of the resolvents is Horn

$$\left|\left(C_{i_0} \cup_r C_{j_1} \cup_r \ldots \cup_r C_{j_k}\right)\sigma_1\right|^+ \leq 1$$
or
$$\left|\left(C_{j_0} \cup_r C_{j_1} \cup_r \ldots \cup_r C_{j_k}\right)\sigma_2\right|^+ \leq 1$$

and this eliminates the resolution step between the two non Horn clauses. In the same way we can eliminate the rest of the resolution steps that involve two non Horn clauses and transform the initial resolution derivation into a semi-Horn derivation. □

Resolutions steps in a semi-Horn derivation are also possible between a RC and another RC or also with a DRC. Those steps do not generate clauses with a greater positive query charge. However, we should prefer steps involving CC because they always produce clauses with a smaller positive clause charge (See CC column on Table 2).

Definition 3.3 (Constraint Derivation)**.** *A resolution derivation (refutation)* $\Sigma \vDash_r C$ *is a constraint derivation iff all the resolution steps are based on binary resolution with a clause that has no positive literals. A constraint derivation is written as* $\Sigma \vDash_c C$*. Similarly, there is a constraint deduction of* C *from* Σ*, written as* $\Sigma \vDash_c^d C$*, if* C *is a tautology or if there is a clause* D *such that* $\Sigma \vDash_c D$ *and* D *subsumes* C*.*

The subsumption theorem can be formulated using constraint deductions and consequences with no positive literals.

Theorem 3.3 (Constraint Subsumption for Consequences without positive literals)**.** *Let* Σ *be a set of clauses and* C *a clause with no positive literals. Then* $\Sigma \vDash C$ *iff* $\Sigma \vDash_c^d C$*.*

Proof. Based on the subsumption theorem for semi-Horn resolution derivations (3.2), the fact that $\Sigma \vDash C$ implies that we have a semi-Horn deduction of C i.e. $\Sigma \vDash_{sH}^d C$. However, we need avoid resolution steps between two RCs or a RC and a DRC.

Such a derivation for sure involves resolution steps with respect to some constraint clauses. If we use the process described on the proof of Theorem (3.1), the semi-Horn derivation of C can also be transformed into a constraint derivation.

Let C_{i_1} be the closest resolvent to the root (C) of the corresponding derivation tree that is obtained by applying binary resolution without using a constraint clause: $C_{i_1} = \left(C_{i_0} \cup_r C_{j_0}\right)$. We can assume that on the path to the root of the tree the resolution steps involve always constraint clauses $C_{j_1}, \ldots, C_{j_k}$:

$$\left|\left(C_{i_0} \cup_r C_{j_0}\right) \cup_r C_{j_1} \cup_r \ldots \cup_r C_{j_k}\right|^+ = 0$$

implies that

$$\left|\left(C_{i_0} \cup_r C_{j_1} \cup_r \ldots \cup_r C_{j_k}\right) \cup_r \left(C_{j_0} \cup_r C_{j_1} \cup_r \ldots \cup_r C_{j_k}\right)\right|^+ = 0.$$

Therefore, at least one of the factors of the resolvents is a constraint clause:

$$\left|\left(C_{i_0} \cup_r C_{j_1} \cup_r \ldots \cup_r C_{j_k}\right)\sigma_1\right|^+ = 0$$
or
$$\left|\left(C_{j_0} \cup_r C_{j_1} \cup_r \ldots \cup_r C_{j_k}\right)\sigma_2\right|^+ = 0$$

and this eliminates the resolution step between the two non constraint clauses clauses. In the same way we can eliminate the rest of the resolution steps that involve two non constraint clauses. □

Theorem 3.4 (Completeness of Constraint Resolution Derivations)**.** *A set of clauses* Σ *is unsatisfiable iff* $\Sigma \vDash_c \bot$.

Proof. This follows from theorem 3.3 as a colorary where the consequence is the empty clause. □

3.2. Rewriting Operations and Resolution

Conjunctive query rewriting is a process that mimics the constraint derivations introduced in the previous

section. However resolution steps involving Skolem functions are performed together in order to avoid introducing literals with Skolem functions that will not be able to be removed. For existential rules, the process of query rewriting is well known [17]. However, disjunctive rules are mainly used in a forward chaining manner in most of the existing literature [11, 13].

In example 2.4 one could infer that two first-degree relatives that have the same age have to be siblings:

$$\begin{aligned} &\textit{first-deg-relative}(X, Y), \\ &\qquad \textit{same-age}(X, Y) \rightarrow \textit{sibling}(X, Y). \end{aligned} \tag{4}$$

This rule can be obtained with a constraint derivation using (1) and the clauses corresponding to:

$(\textit{same-age}(X, Y), \textit{parent}(X, Y) \rightarrow \bot)$
(Children and their parents cannot have the same age)
$(\textit{same-age}(X, Y) \rightarrow \textit{same-age}(Y, X))$
(Symmetry).

This creates the new existential rule (4) that was obtained from the initial disjunctive rule (1) and can also be used in rewriting steps defined for existential rules.

The resolution step between a RC C_r corresponding to an existential rule r and a CC C_q corresponding to the negation of a CQ q, corresponds to a rewriting step as defined in the existential rules framework [17].

Definition 3.4 (Rewriting Step). *Let $r = B \rightarrow H$ be an existential rule, q a conjunctive query and θ a unifier for $H' \subseteq H$ and $q' \subseteq q$ ($H'\theta = q'\theta$) such that:*

1. *if $v \neq v\theta$ and $v \in vars(q \setminus q')$, then $v\theta$ is a frontier variable of r or a constant.*
2. *if v is an existential variable in the rule, then $v\theta \notin (q \setminus q')$.*

Then the query $rew(r,\ q) = (B \cup (q \setminus q'))\theta$ is a rewriting of q using the rule r.

Having more than one atom in the head of existential rules corresponds to more than one RC. Nevertheless, the resolution steps using those clauses are always performed together in order to avoid unnecessary propagation of Skolemized existential variables. Thus, the resulting clause can not contain a Skolemized constant representing an existential variable in r. Hence, existential variables cannot be assigned to a variable that will be part of the result (condition 1 in definition 3.4) nor should be replaced by a variable that belongs to the result (condition 2 in definition 3.4).

For the resolution step between a DRC C_r corresponding to a disjunctive existential rule r and a CC C_q corresponding to the negation of a CQ q, we define a corresponding rewriting step.

Definition 3.5 (Disjunctive Rewriting Step). *Let $B \rightarrow H$ be a disjunctive existential rule, q a conjunctive query, $H' \subseteq H$ and θ a unifier for $q' \subseteq q$ in the query and h'_i a subset of $h_i \in H'$ ($h'_1\theta = \ldots h'_n\theta = q'\theta$) such that:*

1. *if $v \in vars(q \setminus q')$, then $v\theta$ is a frontier variable of r or a constant.*
2. *if v is an existential variable in the rule, then $v\theta \notin (q \setminus q')$.*

Then $rew(r,\ q) = (B \cup (q \setminus q') \rightarrow H \setminus H')\theta$ is a rewriting of q using the rule r.

The resulting expression $rew(r,\ q)$ can be a disjunctive rule with fewer disjunctive components, an existential rule in case $|H \setminus H'| = 1$ or a conjunctive query in case $|H \setminus H'| = 0$.

Example 3.3. *Let us consider a disjunctive rule:*

$$\begin{aligned} r_1 = \textit{diabetesRisk}(X) \rightarrow [&(\textit{diabetic}(Y), \\ &\textit{sibling}(Y, X)), \\ &(\textit{diabetic}(Z), \\ &\textit{parent}(Z, X))]. \end{aligned}$$

If we want to rewrite a query to know if there are diabetic people $q = diabetic(X_1)$, we can obtain the UCQ*-rewriting $[diabetic(X_1), diabetesRisk(X)]$, using r_1 with the unifier $\theta = \{Y \leftarrow X_1, Z \leftarrow X_1\}$.*

On the other hand, if we have a negative constraint $singleChild(X_1), sibling(Y_1, X_1)$ and a query for a diabetic parent $q' = diabetic(Y_2), parent(Y_2, X_2)$, we can derive the rule:

$$\begin{aligned} \textit{diabetesRisk}(X), \textit{singleChild}(X) \rightarrow &\textit{diabetic}(Z), \\ &\textit{parent}(Z, X) \end{aligned}$$

if we rewrite the constraint with r_1 and the unifier $\theta_2 = \{X_1 \leftarrow X, Y_1 \leftarrow Y\}$. Using the new rule we can then obtain the following UCQ*-rewriting:*

$$\begin{aligned} &[(\textit{singleChild}(X), \textit{sibling}(Y, X)), \\ &(diabetic(Y), \textit{parent}(Y, X)), \\ &(\textit{diabetesRisk}(X), \textit{singleChild}(X))]. \end{aligned}$$

Notice that the final UCQ*-rewriting contains also negated constraints which are possible reasons for which a query can be entailed i.e. inconsistent knowl-*

edge bases. However, sometimes we might want to filter out inconsistencies if we are sure that the knowledge base is consistent.

A rewriting can then be defined for knowledge bases using rewriting steps.

Definition 3.6 (Rewriting)**.** *Let $\langle \mathcal{R}, \mathcal{R}^\vee, \mathcal{Q} \rangle$ be set of existential rules, disjunctive existential rules and a* UCQ*. A one step rewriting $\langle \mathcal{R}', \mathcal{R}'^\vee, \mathcal{Q}' \rangle$ of $\langle \mathcal{R}, \mathcal{R}^\vee, \mathcal{Q} \rangle$ can be obtained as follows:*

1. *by adding to $\mathcal{Q}$ the result q' of a rewriting step that uses one of the queries in $\mathcal{Q}$ and a rule in $\mathcal{R}$, where $\mathcal{Q}' = \mathcal{Q} \cup [q']$.*
2. *by adding to $\mathcal{R}^\vee$ (to $\mathcal{R}$ or to $\mathcal{Q}$) the result f' of a disjunctive rewriting step that uses one of the queries in $\mathcal{Q}$ and a rule in $\mathcal{R}'^\vee$, where $\mathcal{R}^\vee = \mathcal{R}^\vee \cup (f')$ if f' is a disjunctive existential rule, $\mathcal{R}' = \mathcal{R} \cup (f')$ if f' is an existential rule, otherwise $\mathcal{Q}' = \mathcal{Q} \cup (f')$.*

A k-steps rewriting of $\langle \mathcal{R}, \mathcal{R}^\vee, \mathcal{Q} \rangle$ is obtained applying a one step rewriting to a $(k-1)$-steps rewriting of $\langle \mathcal{R}, \mathcal{R}^\vee, \mathcal{Q} \rangle$. For any k, a k-steps rewriting of $\langle \mathcal{R}, \mathcal{R}^\vee, \mathcal{Q} \rangle$ is a rewriting of $\langle \mathcal{R}, \mathcal{R}^\vee, \mathcal{Q} \rangle$.

So far we can deal with existential rules, disjunctive existential rules and conjunctive queries. However, constraints and conjunctive queries with negated atoms need to be considered also. Negative constraints are transformed into queries in the rewriting process i.e.

$$\mathcal{D}, \mathcal{C}, \mathcal{R}, \mathcal{R}^\vee \models \mathcal{Q} \quad \text{iff} \quad \mathcal{D}, \mathcal{R}, \mathcal{R}^\vee \models \neg\mathcal{C} \cup \mathcal{Q}.$$

In a similar way if we have a UCQ$^\neg$ $\mathcal{Q} \cup \mathcal{Q}^\neg$, where $\mathcal{Q}$ is a UCQ and $\mathcal{Q}^\neg$ CSF that contains only CQ$^\neg$s, the entailment problem can be reduced to the entailment of a UCQ:

$$\begin{array}{l} \mathcal{D}, \mathcal{C}, \mathcal{R}, \mathcal{R}^\vee \models \mathcal{Q} \cup \mathcal{Q}^\neg \quad \text{iff} \\ \mathcal{D}, \mathcal{R}, \mathcal{R}^\vee, \neg\mathcal{Q}^\neg \models \neg\mathcal{C} \cup \mathcal{Q}, \end{array} \tag{5}$$

where $\neg\mathcal{Q}^\neg$ can contain some existential rules (negations of CQ$^\neg$s with one negated atom), some disjunctive existential rules (negations of CQ$^\neg$s with more than one negated atom) and some facts (negations of CQ$^\neg$s with no positive literal).

Theorem 3.5 (Soundness and Completeness of Rewritings)**.** *Let $\langle \mathcal{D}, \mathcal{C}, \mathcal{R}, \mathcal{R}^\vee \rangle$ be a knowledge base and $\mathcal{Q}$ a* UCQ*. Then $\mathcal{D}, \mathcal{C}, \mathcal{R}, \mathcal{R}^\vee \models \mathcal{Q}$ iff there is a query q', that belongs to the* UCQ *component of $\langle \mathcal{R}', \mathcal{R}'^\vee, \mathcal{Q}' \rangle$ a rewriting of $\langle \mathcal{R}, \mathcal{R}^\vee, \mathcal{Q} \cup \neg\mathcal{C} \rangle$ such that $\mathcal{D} \models q'$.*

Proof. The k-steps rewriting of $\langle \mathcal{R}, \mathcal{R}^\vee, \mathcal{Q} \cup \neg\mathcal{C} \rangle$ is based on a constraint derivation. Moreover, such rewriting can be mapped to a constraint derivation. Since constraint derivations are sound and complete (Theorem 3.4), this theorem also holds. □

Algorithm 1 Function to rewrite UCQs with respect to existential rules and disjunctive existential rules.

```
function rewrite_k(R, R^∨, C, Q)
  Q'' := Q ∪ ¬C
  R'' := R
  R''^∨ := R^∨
  do
    R' := R''
    R'^∨ := R''^∨
    Q' := Q''
    Q'' := rewrite_k^∃(R', Q')
    R'', R''^∨ := rewrite^∨(R', R'^∨, Q'')
  while (Q'' ≠ Q' or R''^∨ ≠ R'^∨
                  or R'' ≠ R')
  return Q''
end function
```

Algorithm 1 alternates the rewriting of CQs using existential rules (rewrite$_k^\exists$/2) and the rewriting using disjunctive existential rules (rewrite$^\vee$/3). New CQs are used to generate more rules and new rules are used to generate more CQs until a fix point is reached i.e. no new rule or query is produced.

The function rew/2 computes the set of all one step rewritings with all the combinations of existential rules and CQs in the arguments. This step is known as the expansion of a query and it generates more conjunctive queries, existential rules or disjunctive existential rules. On the other hand, the function rew$^\vee$/2 defines the expansion of a disjunctive existential rule and computes the set of all one step rewritings with all the combinations of disjunctive existential rules and CQs in the arguments that does not yield a conjunctive query. This restriction does not affect completeness because those CQ that can be generated with a disjunctive rewriting step ($rew(r,\ q) = q'$) can also be generated in two steps, i.e. a disjunctive rewriting step generates an existential rule ($rew(r,\ q) = r'$) and then a rewriting step using that existential rule ($rew(r',\ q) = q'$).

The cover/1 function computes for a given CSF F the minimal subset $F' \subseteq F$ such that for all $c \in F$ there

Algorithm 2 Function to rewrite UCQs using existential rules.

```
function rewrite^∃_k(R, Q)
  Q' := Q
  Q'' := Q
  level := 0
  do
    Q''' :=cover(Q' ∪ rew(Q'', R))
    Q'' := Q''' \ Q'
    Q' := Q'''
    level := level + 1
  while Q'' ≠ ∅ and level < k
  return Q'
end function
```

Algorithm 3 Function to rewrite UCQs using disjunctive existential rules.

```
function rewrite^∨(R, R^∨, Q)
  R'^∨ := R^∨
  R''^∨ := R^∨
  do
    R' := R ∪ R'^∨ ∪ rew^∨(Q, R''^∨)
    R'''^∨, R :=cover(R')
    R''^∨ := R'''^∨ \ R'^∨
    R'^∨ := R'''^∨
  while R''^∨ ≠ ∅
  return R'^∨, R
end function
```

is a $c' \in F'$ such that c' subsumes c i.e. the corresponding clause c'_c to c' subsumes c_c, the corresponding clause to c. However, the definition of subsumption needs to be extended for the case of (disjunctive) existential rules where there are more than one clause in their CNF representation. In a similar way for c and c' CNFs, we say c' subsumes c if all the clauses in c are subsumed by a clause in c'. In Algorithm 3 the result of `cover`/1 is a set of rules however we abuse the notation to split it into a set of disjunctive rules (first component) and of existential rules (second component).

The `cover`/1 function allow us to keep always the minimal set of CQs that can yield the same results. In [17] the authors perform a deeper analysis showing that using the cover computation on the rewriting algorithm they propose ensures that the resulting UCQ-rewriting will be of minimal size (cardinality). In our case, the minimal rewriting property is also present despite of the fact that the extension we propose for subsumption in CNFs does not ensure that the cover of a set of rules will be minimal.

Example 3.4. *For $\mathcal{R} = (a \to b), (b \to c), (a \to c)$ we have that cover($\mathcal{R}$) = $\mathcal{R}$. However, there is a subset $\mathcal{R}' = (a \to b), (b \to c)$ of $\mathcal{R}$ and $\mathcal{R}' \models \mathcal{R}$. Thus, the cover of a set of rules does not always return a set of minimal size.*

Redundant existential rules will generate redundant queries but they will be removed when the cover of the UCQ is computed in $\texttt{rewrite}^{\exists}_k$/2 conserving the minimal size property of the UCQ-rewriting.

The generated elements are also expanded unless they are removed when computing the cover. The process stops when a fix point is reached. The one-step rewriting function (`rew`/2) can keep track of the pairs of conjunctive queries and rules that were used in order not to repeat their rewritings and focus only on rewriting using the new rules and new conjunctive queries.

The CQ elements generated by the algorithm are based on our definition of rewriting and this ensures the correctness i.e. every CQ generated is a CQ-rewiting of the input query with respect of the input knowledge base.

The rewriting function for disjunctive existential rules ($\texttt{rewrite}^{\vee}$/3) generates all the possible rules using an input UCQ rewriting. The fact that new rules have less disjoint components in the head ensures that the output is always finite. Therefore, the completeness of Algorithm 1 relies totally on the completeness of Algorithm 2 ($\texttt{rewrite}^{\exists}$/2). Algorithm 2 describes the rewriting function for existential rules ($\texttt{rewrite}^{\vee}_k$/3) that is based on the general rewriting algorithm proposed on [17]. It is a breath-first expansion process where each iteration of the loop expands a new level of conjunctive queries. We have introduced the parameter k that allows to control how many levels of CQs will be expanded and ensures termination of each individual call to Algorithm 2 for $k \neq \infty$. However, the loop on Algorithm 1 will keep on calling Algorithm 2 as long as new CQs are generated without affecting the completeness of the whole rewriting process. The parameter k defines a *pause* on the existential rules rewriting process in order to generate more rules from the disjunctive existential rules. Mélanie König et al. study the completeness of the rewriting algorithm for existential rules based on different definitions of the

query expansion function [17]. Our expansion function (rew/2) ensures the completeness of the existential rules rewriting algorithm because it is based on their *piece-based* rewriting operator that ensures the completeness. Moreover, if there is a finite rewriting of input UCQ, the function ($\texttt{rewrite}_k^{\exists}$/2) will be able to find it after a finite number of calls to it.

3.3. Rewritable Queries and Knowledge Bases

The termination of Algorithm 1 depends on the termination of algorithms 2 and 3. Algorithm 3 always terminates because the produced rewritings contain less disjunctive components in the head. On the other hand, setting $k = \infty$ or executing a possibly infinite number of calls to Algorithm 2 corresponds to the classical rewriting algorithm for existential rules proposed in [17] and its termination is studied in [3]. In general, it is undecidable to know if there exists a finite UCQ-rewriting for every UCQ with respect to an arbitrary set of existential rules. A set of existential rules that ensures the existence of a UCQ-rewriting for any UCQ is called a finite unification set (*fus*) [4]. There are some classes of rules that ensure the *fus* property:

1. *Linear* rules [4]: rules with one atom in the body.
2. *Disconnected* rules [6]: rules that do not share variables between the body and the head.
3. *Acyclic graph of rule dependencies* (*aGRD*) [5]: rules that do not contain cycles in the *graph of rule dependencies*.

If $\mathcal{R}$ is a *fus* and the new rules generated from algorithm 3 are also a *fus*, combining them could yield a new set of rules that is not a *fus* [3].

A *cut* $\{\mathcal{R}_1, \mathcal{R}_2\}$ is a partition of the set of rules $\mathcal{R}$ and it is a *directed cut* $\mathcal{R}_1 \triangleright \mathcal{R}_2$ if none of rules in $\mathcal{R}_1$ depends on a rule of $\mathcal{R}_2$.

Property 3.1. *Given a set of rules $\mathcal{R}$ with a directed cut $\mathcal{R}_1 \triangleright \mathcal{R}_2$, for all queries q and set of facts $\mathcal{D}$:*

$$\mathcal{D}, \mathcal{R} \models q \text{ if there is a query } P \text{ such that } \mathcal{D}, \mathcal{R}_1 \models P \text{ and } P, \mathcal{R}_2 \models q.$$

Proof. The proof is based on being able to organize the application of rules from $\mathcal{R}$. The existing dependencies ensure that rules of $\mathcal{R}_1$ are never depending on rules from $\mathcal{R}_2$. For a detailed proof check [3]. □

Property 3.1[3] allows us to study the decidability of entailment when we combine two sets of rules for which the entailment problem is decidable.

Even if the union of $\mathcal{R}$ and the set of existential rules produced by Algorithm 3 yields a *fus*, the process of generating new rules could potentially continue forever after we obtain new CQs from Algorithm 2. Therefore, we need ways to ensure that the total number of existential rules generated by Algorithm 3 is a finite number i.e. at some point the algorithm will not produce new rules.

Existential rules with one atom in the body ensure that the rewritings will never grow in size. Indeed, a rewriting operation will replace one or more atoms for the atomic body.

An (a disjunctive) existential rule $B \rightarrow H$ is called linear rule if it has only one atom in the body i.e. $|B| = 1$. Similarly, a constraint $B \rightarrow \bot$ is called *linear constraint* if it has only one atom i.e. $|B| = 1$.

Theorem 3.6. *The rewriting algorithm 1 stops for atomic queries Q and a knowledge base composed by linear existential rules $\mathcal{R}$, linear disjunctive existential rules $\mathcal{R}^{\vee}$, linear constraints C and any value of k.*

Proof. Linear existential rules are a *fus* and rewriting queries with them stops even when $k = \infty$. The new rules generated with the linear disjunctive existential rules and the atomic queries will also be linear rules and adding them to $\mathcal{R}$ will produce a *fus*. Additionally, the number of single atoms that can be built using predicates, variables and constants is bounded. Therefore, the number of rules that we can derive from disjunctive rules is finite. □

Theorem 3.6 is closely related to Theorem 7.13 and Lemma 7.12 proposed by Bourhis et al. in [10]. However, it is still interesting to prove it considering the algorithm we have proposed so that the technique can be extended to the study of other fragments.

Rules that do not share variables between the head and the body produce rewritings where the introduced body of the rule is not connected to the rest of the the query.

An (a disjunctive) existential rule $B \rightarrow H$ is called *disconnected* rule (disc-rule) if no variable from the body is present in the head of the rule i.e. $vars(B) \cap vars(H) = \emptyset$.

Disconnected rules might still share constants between the body and the head and this allows to express knowledge about specific individuals.

Theorem 3.7. *Let $\mathcal{R}_1$ be a* fus *and $\mathcal{R}_2$ a set of disconnected existential rules. The union of both sets $\mathcal{R}_1 \cup \mathcal{R}_2$ is also a* fus.

Proof. The rewritings produced by disconnected rules add atoms that do not share variables with the query. Therefore, we can say that the connected cardinality in rewritings produced with rules $B \rightarrow H \in \mathcal{R}_2$ is bounded:

$$card^*(\texttt{rew}(B \rightarrow H, q)) \leq \max(card^*(B), card^*(q)).$$

The rewritings obtained from rules in $\mathcal{R}_1$ could increase the connected cardinality of the queries, but they only produce a finite number of rewritings because $\mathcal{R}_1$ is a *fus*. Thus, we can affirm that the connected cardinality of the rewritings of an initial query q will be bounded i.e.

$$card^*(\texttt{rewrite}^{\exists}(\mathcal{R}_1 \cup \mathcal{R}_2, q)) \leq \max(card^*(\texttt{rewrite}^{\exists}(\mathcal{R}_1, B)), card^*(\texttt{rewrite}^{\exists}(\mathcal{R}_1, q))).$$

Using lemma 2.2 we can ensure that the number of rewritings produced by $\mathcal{R}_1 \cup \mathcal{R}_2$ cannot be infinite. Thus, $\mathcal{R}_1 \cup \mathcal{R}_2$ is also a *fus*. □

Theorem 3.8. *The rewriting algorithm 1 stops for knowledge base composed by a* fus $\mathcal{R}$*, a set of disconnected disjunctive existential rules* $\mathcal{R}^\vee$*, any set of constraints* $\mathcal{C}$ *and any* UCQ $\mathcal{Q}$*.*

Proof. The new existential rules produced by $\texttt{rewrite}^\vee$ are disconnected rules and they can be combined with $\mathcal{R}$ and produce another *fus* (Theorem 3.7).

Rewritings of disjunctive rules $B_i \rightarrow H_i$ will have the following form:

$$\bigcup_j B'_j \cup \bigcup_j q'_j \rightarrow H',$$

where $H' \subseteq H_i\sigma$ is a subset of an instance $H_i\sigma$ of the original head of the rule H_i, $B'_j \subseteq \texttt{rewrite}^{\exists}(B_j, \mathcal{R})\sigma'$, $q'_j \subseteq \texttt{rewrite}^{\exists}(q_j, \mathcal{R})\sigma''$ and none of the B'_j or q'_j share variables between them because they are introduced using an atom in the head of the disjunctive rule that do not shares variables with the body. Because $\mathcal{R}$ is a *fus* we have a finite number of rewritings B'_j and q'_j. This ensures there is only a finite number of different bodies B' for the generated existential rules. The number of different heads, H' is obviously finite too. Therefore, there is only a finite number of different existential rules that will eventually be generated by $\texttt{rewrite}^\vee$. Thus, algorithm 1 stops. □

For other types of queries and knowledge bases there is no certainty that the algorithm will stop. However, we can still try to compute the rewritings up to a certain depth. Nevertheless, we should point that our algorithm stops for queries and knowledge bases that have a finite rewriting.

Theorem 3.9. *The rewriting algorithm 1 stops if there is a finite* UCQ*-rewriting of of a* UCQ $\mathcal{Q}$ *and a set of negated constraints constraints* $\neg\mathcal{C}$ *with respect to a knowledge base composed by a set o existential rules* $\mathcal{R}$*, a set of disjunctive existential rules* $\mathcal{R}^\vee$ *and a finite value of k, i.e.* $k \neq \infty$*.*

Proof. The completeness of the definition of rewritings and the fact that we produce rewritings using all possible one-step rewritings ensures that if there is a finite UCQ rewriting $\mathcal{Q}_f$, then after a finite number of steps Algorithm 1 will produce a rewriting equivalent to $\mathcal{Q}_f$. However, a subset of all the existential rules needed to generate the finite and complete UCQ-rewriting of the initial query could potentially produce an infinite number of rewritings. To illustrate this we can consider the following knowledge base:

- Existential rules:

$$\mathcal{R} = (r(X, W), r(W, Y) \rightarrow r(X, Y)), (b(X) \rightarrow a(X)) (c(X) \rightarrow a(X))$$

- Disjunctive existential rules :

$$\mathcal{R}^\vee = s(X) \rightarrow [b(X), c(X)]$$

If we try to rewrite the UCQ $[a(X), (s(X), r(X, f))]$ with respect to $\mathcal{R}$ using $k = \infty$ it would produce an infinite set of rewritings of the form:

$$s(X), r(X, W_1), r(W_1, W_2), \ldots, r(W_n, f).$$

However, if $k = 1$ (or any other finite value) this could allow us to produce new rules with the disjunctive existential rule that would eventually generate a query $s(X)$ and together with the application of the $\texttt{cover}/1$ function it would remove the queries that can produce the infinite set of rewritings and yield the finite and complete UCQ-rewriting $[a(X), s(X), b(X), c(X)]$ of the initial query.

Therefore, the expansion process in Algorithm 2 needs to have a finite depth (i.e. $k \neq \infty$) to avoid infinite loops.

Any further rewriting of $\mathcal{Q}_f$ using existential rules will not produce new conjunctive queries ($\mathcal{Q}'' = \mathcal{Q}'$ for the rest of the iterations in the loop). The algorithm can still produce new (disjunctive) existential rules but since no new CQs are generated the number of new rules that can be produced is finite due to the fact that new rules are produced with strictly less disjoint components. Therefore, in a finite number of steps the algorithm will not be able to produce new CQs or new rules i.e., it will meet the stop condition defined in the loop.

□

3.4. On Queries with Answer Variables and Linear Queries

While Theorems 3.6 and 3.8 impose rather strong restrictions on the disjunctive framework, they also ensure the existence of finite UCQ-rewritings for very expressive types of queries with negated atoms and answer variables.

A CQ q with answer variables is denoted by the expression $q(\mathbf{X}) \text{ :- } B$, where $var(\mathbf{X}) \subseteq var(B)$ are free variables, $\mathbf{X}$ is called the answer tuple and B is a CSF of atoms where the variables $var(B) \setminus var(\mathbf{X})$ are existentially quantified. Queries without answer variables are called *Boolean* queries and for them the answer tuple is the empty tuple $\mathbf{X} = ()$. A CQ$^\neg$ with answer variables is defined in the same way, however we do not allow variables that are only present in negated atoms to be part of the answer variables in $\mathbf{X}$. A UCQ (UCQ$^\neg$) with answer variables $q(\mathbf{X}) \text{ :- } B$ is defined for a UCQ (UCQ$^\neg$) B, where each of the answer variables $var(\mathbf{X})$ belongs as a free variable to each query in B.

If we replace variables in $\mathbf{X}$ by constants in a knowledge base $\mathcal{K}$ and the resulting closed formula $B\theta$ is entailed $\mathcal{K} \models B\theta$, we say that the tuple $\mathbf{X}\theta$ is a *certain answer* of q with respect to $\mathcal{K}$. The set of certain answers of a query q with respect to a knowledge base $\mathcal{K}$ is denoted by $cert(q, \mathcal{K})$. Notice that we are only interested in answers containing constants in $\mathcal{K}$.

Example 3.5. *Let us consider three different set of answer variables:*

$q_1\ () \text{ :- } sibling(X, Y)$
// Empty tuple $\{()\}$ *if someone has a sibling.*
$q_2\ (X) \text{ :- } sibling(X, Y)$
// The set of people that have siblings.
$q_3\ (X, Y) \text{ :- } sibling(X, Y)$
// The set of pairs of people that are siblings.

Also, let us consider a knowledge base $\mathcal{K}$ based on example 2.4 together with the following facts:

$sibling(perdo, ana)$ *// pedro and ana are siblings.*
$sibling(juan, Y)$ *// juan has a sibling.*

The certain answers of the queries with respect to $\mathcal{K}$ are the following:

$cert(q_1, \mathcal{K}) = \{()\}$
// There are some siblings in $\mathcal{K}$.
$cert(q_2, \mathcal{K}) = \{pedro, ana, juan\}$
// Notice that sibling/2 is symetric.
$cert(q_3, \mathcal{K}) = \{\langle pedro, ana\rangle, \langle ana, pedro\rangle\}$
// No identity for the sibling of juan.

The answers can be verified easily by applying the corresponding substitutions to the query e.g. $\mathcal{K} \models \exists Y\, sibling(pedro, Y)$ ensures pedro is a certain answer of q_2.

For CQs (CQ$^\neg$s) with answer variables we focus on the computation of the certain answers instead of just solving the entailment problem. For theoretical purposes we can try with all possible assignments of constants in $\mathcal{K}$ to variables in $\mathbf{X}$. Nevertheless, in the implementation of algorithm 1, instead of trying with every possible ground instance of $\mathbf{X}$ we can use variables in the rewriting process. However, answer variables are not standardized apart and they can only be assigned to constants from $\mathcal{K}$ and not to Skolem functions. This requires some minor modifications in algorithm 1 (see Section 4).

Sometimes the disjtunctive existential rules are not disconnected but if they eventually will produce a disconnected existential rule, the rewriting algorithm will also stop.

Example 3.6. *Let us consider the following knowledge base:*

$B(X), s(X, Y) \rightarrow [B(Y), W(Y)]$
$B(c) \rightarrow \bot$
$B(a), s(a, b), s(b, c)$

in order to answer queries with different answer variables: $Q_1() \text{ :- } W(X)$ and $Q_2(X) \text{ :- } W(X)$.

To prove Q_1 we can use a simple constraint refutation:

$$
\begin{aligned}
&[\neg B(X), \neg s(X,Y), B(Y), W(Y)] \cup_r [\neg W(X)] \\
&= [\neg B(X), \neg s(X,Y), B(Y)] \\
&[\neg B(X), \neg s(X,Y), B(Y)] \cup_r [\neg B(c)] \\
&= [\neg B(X), \neg s(X,c)] \\
&[\neg B(X), \neg s(X,Y), B(Y)] \cup_r [\neg B(X), \neg s(X,c)] \\
&= [\neg B(X'), \neg s(X',X), \neg s(X,c)] \\
&(([\neg B(X'), \neg s(X',X), \neg s(X,c)] \cup_r \\
&[s(b,c)]) \cup_r [s(a,b)]) \cup_r [B(a)] = \bot.
\end{aligned}
$$

On the other hand, for the case of Q_2 we add $[\neg W(t)]$ where $t \in \{a,b,c\}$. However, the empty clause cannot be reached.

$$
\begin{aligned}
&[\neg B(X), \neg s(X,Y), B(Y), W(Y)] \cup_r [\neg W(t)] \\
&= [\neg B(X), \neg s(X,t), B(t)] \\
&[\neg B(X), \neg s(X,Y), B(Y)] \cup_r [\neg B(c)] \\
&= [\neg B(X), \neg s(X,c)] \\
&\textit{// notice that this forces } t = c \\
&[\neg B(X), \neg s(X,c), B(c)] \cup_r [\neg B(X), \neg s(X,c)] \\
&= [\neg B(X'), \neg s(X',c), \neg s(c,c)].
\end{aligned}
$$

We end up in a clause that is subsumed by a previously generated clause $[\neg B(X), \neg s(X,c)]$. Therefore, in this case appliying resolution with respect to the resulting rule $[\neg B(X), \neg s(X,t), B(t)]$ is not necessary. This prevent us from performing an infinite number of resolution steps in order to find a constraint refutation.

If we take a closer look at Q_2 in example 3.6, we notice that the RC produced in the first resolution step had no variables in the positive literal and this corresponds to a disconnected existential rule. In general, if the disjunctive rules only unify with grounded atoms all the resulting existential rules will be disconnected rules. Therefore, for a knowledge base $\langle \mathcal{R}, \mathcal{R}^\vee, \mathcal{C} \rangle$ with $\mathcal{R}$ being a *fus* and a UCQ Q possibly with answer variables, the rewriting algorithm 1 stops if the rules in $\mathcal{R}^\vee$ always generate disconnected existential rules. There are many combination of properties that can help in the the generation of disconnected rules. However, they seem to be very restrictive and algorithm 1 could check all the new rules that are generated and abort if one of them is not discontented. This would be an incomplete approach but such algorithm would stop and give a complete answer if indeed all the new rules end up being disconnected rules.

Using the reduction in equation (5) we can focus on the disjunctive rules that come from the negation of a $\mathsf{CQ}^\neg$. In this case, we can ensure that if all the variables in the frontier of the query are also answer variables, then the corresponding disjunctive rules will be disconnected.

Theorem 3.10. *The rewriting algorithm 1 stops for a knowledge base composed by a* fus $\mathcal{R}$*, a set of constraints $\mathcal{C}$, a* $\mathsf{UCQ}^\neg$ $Q \cup Q^\neg$ *where all the variables in the frontier of the* $\mathsf{CQ}^\neg$*s in $Q^\neg$ are part of the answer variables and any value of k.*

Proof. When $\mathcal{R}^\vee = \emptyset$, the entailment of $\mathsf{UCQ}^\neg$s can be transformed into the entailment of a UCQ (5):

$$\mathcal{R}, \mathcal{C} \models Q \cup Q^\neg \quad \text{iff} \quad \mathcal{R}, \neg Q^\neg \models Q \cup \neg \mathcal{C}.$$

The variables present in negated atoms of the $\mathsf{CQ}^\neg$s will end up being the variables in the head of the corresponding disjunctive rules. However, the variables that are only present in the negated atoms will be translated to existential variables and answer variables are replaced by constants in the entailment of queries with answer variables. Therefore, the resulting disjunctive rules will have only constants and existential variables in the head i.e. they will be disconnected and Theorem 3.8 ensures that algorithm 1 stops for disconnected disjunctive rules combined with a *fus* and any finite or infinite value of k. □

Similarly, we can define other restrictions in $\mathsf{UCQ}^\neg$s that allow us to ensure that the rewriting algorithm stop based on Theorem 3.6. A *linear* CQ ($\mathsf{CQ}^\neg$) is a CQ ($\mathsf{CQ}^\neg$) with only one positive literal. Similarly, a UCQ ($\mathsf{UCQ}^\neg$) is linear if all the queries in it are also linear.

Theorem 3.11. *The rewriting algorithm 1 stops for a knowledge base composed by a linear set of existential rules $\mathcal{R}$, a linear set of constraints $\mathcal{C}$, a linear* $\mathsf{UCQ}^\neg$ $Q \cup Q^\neg$ *and any value of k.*

Proof. Similarly, the entailment of $\mathsf{UCQ}^\neg$s can be transformed into the entailment of a UCQ (5):

$$\mathcal{R}, \mathcal{C} \models Q \cup Q^\neg \quad \text{iff} \quad \mathcal{R}, \neg Q^\neg \models Q \cup \neg \mathcal{C}.$$

Because $Q^\neg$ is linear we have that the corresponding disjunctive rules $\neg Q^\neg$ will also be linear. Therefore, Theorem 3.6 ensures that algorithm 1 stops for for our system composed by linear elements and for any value of k. □

Finally, if there is a finite UCQ rewriting of a $\mathsf{UCQ}^\neg$ $Q \cup Q^\neg$ with respect to a knowledge base with existential rules $\mathcal{R}$ and negative constraints $\mathcal{C}$, we can ensure that algorithm 1 will stop based on Theorem 3.9.

Theorem 3.12. *The rewriting algorithm 1 stops if there is a finite* UCQ *rewriting of a* $\mathsf{UCQ}^\neg$ $\mathcal{Q} \cup \mathcal{Q}^\neg$ *with respect to a knowledge base with existential rules* $\mathcal{R}$*, negative constraints* $\mathcal{C}$ *and a finite value of* k*.*

Proof. A straightforward consequence of and Theorem 3.9 and the following transformation of the entailment problem:

$$\mathcal{R}, \mathcal{C} \models \mathcal{Q} \cup \mathcal{Q}^\neg \quad \text{iff} \quad \mathcal{R}, \neg\mathcal{Q}^\neg \models \mathcal{Q} \cup \neg\mathcal{C}.$$

□

4. Implementation and Experiments

COMPLETO[1] is a query rewriting system that focuses on answering $\mathsf{UCQ}^\neg$s in the framework of existential rules. The first version of COMPLETO[20] answers $\mathsf{CQ}^\neg$ using a resolution based approach to eliminate the negated atoms.The algorithm proposed is complete only for a very restricted type of queries.

In the second version of the system [19], queries with only one negated atom are answered by transforming them into rules. The approach is complete but we can ensure the termination only when the resulting set of rules is a *fus*.

The current version of COMPLETO answers queries with an arbitrary number of atoms. Algorithm 1 can be seen as a generalization of both algorithms proposed previously. Indeed, we transform queries with one negated atom into rules and the rewriting defined for disjunctive rules is similar to what was presented in [20] as constraint resolution. Furthermore, we take advantage of the termination results for a knowledge base composed by a *fus* and $\mathsf{UCQ}^\neg$ where the frontier is part of the answer variables of the query (Theorem 3.10) and for knowledge bases where all the elements are linear (Theorem 3.11). Choosing k lim inf allows the rewriting with respect to existential rules to be performed by an external rewriter.

The algorithms defined in previous sections were implemented in order to answer $\mathsf{UCQ}^\neg$s with answer variables in COMPLETO system. Instead of replacing the answer variables with constants we define them as *global* variables, a special type of variables that keep the same value in each CQ-rewriting. In order to differentiate those variables we include a fresh predicate q in the queries and/or rules that contain those global variables e.g. a CQ with answer variables $q(\mathbf{X}) :- B$ will be represented by a CQ $q(\mathbf{X}), B$. The rewriting steps applied to B will apply the same substitutions to $\mathbf{X}$. The atoms $q(\mathbf{X})$ are known as *answer* atoms.

[1] http://image.ntua.gr/~gardero/completo3.0/

Example 4.1. *The query*

$$q(X, Y) :- person(X), person(Y), \neg married(X, Y)$$

is represented by the query

$$q(X, Y), person(X), person(Y), \neg married(X, Y)$$

and translated into the rule

$$q(X, Y), person(X), person(Y) \rightarrow married(X, Y).$$

If we use the constraint $married(X, X) \rightarrow \bot$ *the following rewriting is obtained* $q(X, Y), person(X)$*. Thus, when we query a database with people, the tupples of the form* X, X *will be answers of the initial query. This ensures that no person can be married to itself.*

Each CQ-rewriting that is entailed by the data $\mathcal{D}$ represents a different way to entail the initial CQ. The answer atoms identify the answer tupple that is produced by each CQ-rewriting. The variables in answer atoms are global and in the derivation of a CQ we cannot produce two different occurrences of answer atoms. Having existential rules and UCQs ensures that all the rewriting produced are CQs and it is not possible to apply resolution between them. On the other hand, introducing disjunctive rules or allowing negation for $\mathsf{UCQ}^\neg$s produces rewritings that could correspond to existential rules or disjunctive existential rules that also contain answer atoms in their body. Therefore, we have to consider the interaction between the rewritings and the possibility to introduce several occurrences of answer atoms in a CQ.

Example 4.2. *Let us Consider the elements in example 4.1 with a new query*

$$q(X, Y) :- parent(X_1, X), parent(Y_1, Y), married(X_1, Y_1)$$

represented by

$$q(X, Y), parent(X_1, X), parent(Y_1, Y), married(X_1, Y_1).$$

When we use the rule corresponding to the query from example 4.1 we can generate a rewriting with two occurrences of answer atoms:

$$q(X,Y), parent(X_1,X), parent(Y_1,Y), \\ q(X_1,Y_1), person(X_1), person(Y_1).$$

In this case the query yields certain answers only when $X = X_1$ *and* $Y = Y_1$.

To approach this issue, we unify the arguments of the multiple answer atoms after each call to Algorithm 2 with $k = 1$ i.e. if a CQ q (rule r) contains n occurrences of answer atoms $q(\mathbf{X_i})$ we then replace the query q (the rule r) by $q\theta$ ($r\theta$), where θ is the unifier $mgu(q(\mathbf{X_1}), \dots, q(\mathbf{X_n}))$ of all the answer atoms $q(\mathbf{X_i})$ in q (or in the rule r). This unification step is important for the termination of the rewriting process and it needs to be performed after each 1-step rewriting in the algorithm. Consequently, this prevents us from using external rewriters if we want to terminate for the cases where answer variables allow us to terminate. Furthermore, having chosen $k = 1$ for Algorithm 1 ensure that our implementation will also stop when there is a finite UCQ-rewriting.

Despite the limitation that the current version of COMPLETO is focused only in the existential rules fragment, disjunctive existential rules could also be encoded as CQ$^\neg$s and added as input to the system, i.e.

$$\mathcal{R}, \mathcal{C}, \mathcal{R}^\vee \models \mathcal{Q} \cup \mathcal{Q}^\neg \quad \text{iff} \quad \mathcal{R}, \mathcal{C} \models \mathcal{Q} \cup \mathcal{Q}^\neg \cup \neg\mathcal{R}^\vee.$$

4.1. Experiments

To the best of our knowledge there is no other system that produces UCQ-rewritings for UCQ$^\neg$s. Therefore, the experiments were preformed in order to get a general idea of the performance of COMPLETO producing UCQ-rewritings. We used a Intel(R) Core(TM) i5-7300HQ CPU at 2.50 GHz with 8 GB of RAM running 64-bit Windows 10.

For the experiments we used two ontologies that contain negative constraints used in previous research papers based on queries with negation [19, 20]. The Lehigh University Benchmark (LUBM) ontology [26] with an additional 70 disjoint classes assertions added for the atomic *siblings* concepts. Also, some rules were removed in order to have a finite unification set of rules. Additionally, we used the TRAVEL ontology[2] that has 10 disjoint class axioms. The OWL 2 ER fragment of both ontologies was translated into existential rules.

We also prepared queries files with 500 CQ$^\neg$s for both ontologies and the system produced the rewritings of the UCQ$^\neg$ that contains all the queries in the file and also for each separated CQ$^\neg$. The queries contain 3 atoms and 2 of them are negated. We generated the queries by performing Association Rules Mining [1] on a dataset obtained from the assertions of the ontologies. The queries and the ontologies used are available in the URL of the system COMPLETO[3].

Table 3 shows the size of the UCQ rewriting (rew) for the UCQ$^\neg$ containing all the queries in the file and the minimum (min), mean and maximum (max) statistics for the rewriting of each individual query in the file. Additionally we show the time (time) in seconds and the memory used (men) in Mb used by the rewriting process in each of the cases. The results give an idea of the performance of the system with respect to each UCQ$^\neg$ or individual CQ$^\neg$.

For the TRAVEL ontology, the size of the rewriting of the UCQ$^\neg$ is smaller than the biggest rewriting for a individual CQ$^\neg$. The time that it took to compute the rewriting of the UCQ$^\neg$ is the time that takes in average to rewrite 125 individual queries (5 min). The RAM memory used to write the UCQ$^\neg$ is approximately the double of the ram used for rewriting the individual query that consumed the most memory.

For the LUBM ontology, the size of the rewriting of the UCQ$^\neg$ has 11 more queries than the biggest rewriting for an individual CQ$^\neg$. The time that it took to compute the rewriting of the UCQ$^\neg$ is the time that takes in average to rewrite 30 individual queries (less than 2 hours). The RAM memory used to write the UCQ$^\neg$ is less than the ram memory used for rewriting the individual query that consumed the most memory.

For both ontologies the memory consumed to compute the rewritings was approximately 2 GB.

Figures 3 and 4 show information about the rewriting size. In both ontologies at least 85 % of the queries have zero rewritings. In this case, they are subsumed by the rewritings of the constraints of the system.

Figures 5 and 6 show cummulative distribution of the rewriting runtime. Dashed horizontal lines represent the mean runtime. Each bar represents the number of queries that were rewritten under the corresponding time. Note that in both cases the runtime for more than 60 % of the queries was smaller than the mean runtime.

[2] http://www.owl-ontologies.com/travel.owl

[3] http://image.ntua.gr/~gardero/completo3.0/

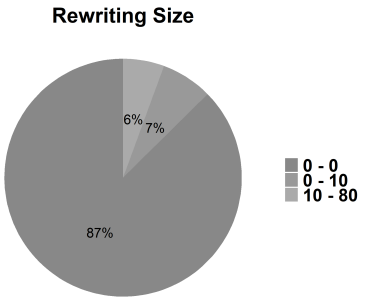

Fig. 3. Size of the rewritings for the TRAVEL ontology.

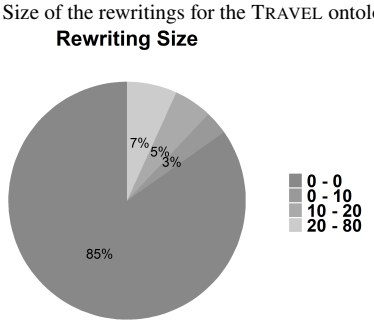

Fig. 4. Size of the rewritings for the LUBM ontology.

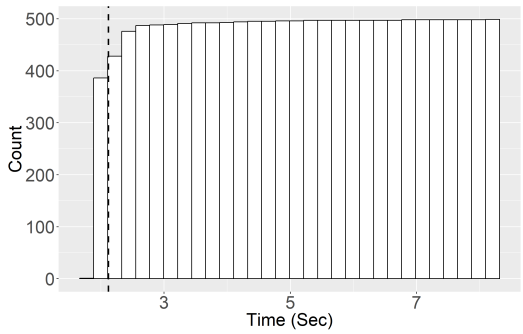

Fig. 5. Cumulative distribution of the time that takes the computation of the rewritings for the TRAVEL ontology.

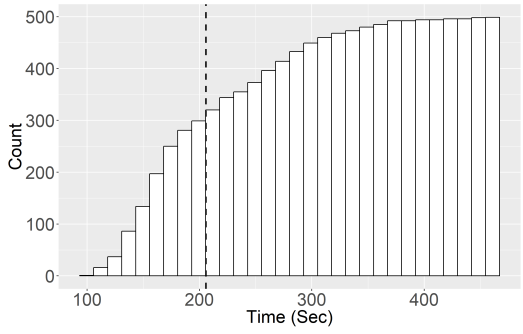

Fig. 6. Cumulative distribution of the time that takes the computation of the rewritings for the LUBM ontology.

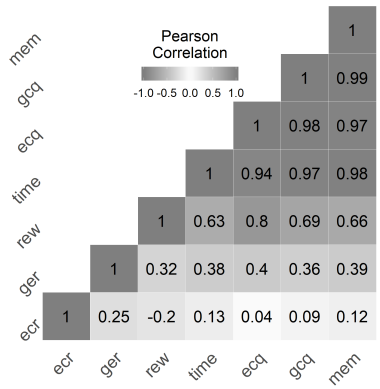

Fig. 7. Correlation matrix with different performance parameters for the TRAVEL ontology.

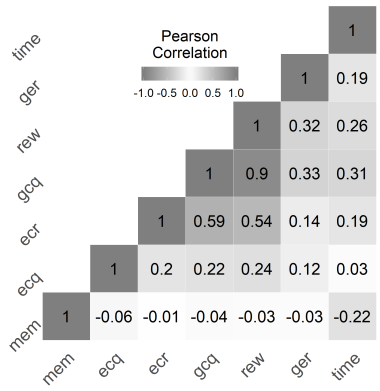

Fig. 8. Correlation matrix with different performance parameters for the LUBM ontology.

Figures 7 and 8 show the correlation matrix with different performance parameters for the TRAVEL and LUBM ontology. In order to get an idea of the rewriting process we computed the memory used by the system (mem), the time that takes the commputation of the rewriting (time), the size of the rewriting (rew), the number of generated existential rules in the rewriting process (ger), the number of rewritten (expanded) disjunctive existential rules (ecr) and also the number of generated (gcq) and rewritten (ecq) conjunctive queries. For the LUBM ontology we can notice that the number of generated CQs and the number of rewritings have a correlation coefficient of 0.9. For the TRAVEL ontology we can see that time, ecq, gcq and mem are all correlated with coefficients higher than 0.9.

Table 3
Rewriting experiments for UCQ¬s from LUBM and TRAVEL ontologies.

Ontology	Info	rew	time	mem
LUBM	UCQ¬	77	6193.12	2138
	min	0	104	1129
	mean	4	205.58	2069
	max	55	466	2237
TRAVEL	UCQ¬	18	264.96	2043
	min	0	1	123
	mean	2	2.12	143
	max	76	8	920

Example 4.3. *One of the queries for the* TRAVEL *ontology was:*

$$Q(X) :- \neg Capital(X), \neg Town(X), \\ Destination(\mathbf{X}).$$

It focuses on destinations that are can not be capitals or towns. The UCQ*-rewriting produced by* COMPLETO *was the following:*

$$Q(X) :- [\, Farmland(X), \\ NationalPark(X), \\ RuralArea(X)].$$

Considering the above interpretation of the query, the answer tells us that other only farmlands, national parks and rural areas cannot be tows or capitals.

5. Conclusions

In this paper we focused on applying the query rewriting approach to the framework of disjunctive existential rules in order to produce complete UCQ-rewritings that encode the answers of an initial query.

Two special cases of First Order Logic resolution were introduced in order to ensure the completeness our rewriting approach. Semi-Horn resolution uses in all the steps at least a Horn clause and the subsumption theorem holds for Horn consequences. We also introduced constraint resolution, where every resolution step involves one clause without positive literals and the subsumption theorem holds when the consequence is a clause without positive literals. The completeness theorem holds for both types of resolution, allowing them to be used in refutation procedures for First Order Logic formulas.

Based mainly in the definition of constraint resolution we propose an extension of the rewriting approach for existential rules in order to deal with disjunctive existential rules. The rewriting of a disjunctive existential rule produces disjunctive rules with less disjunctions in the head and eventually will produce an existential rule or a conjunctive query. The rules generated from disjunctive rules are then used in order to find additional rewritings of the conjunctive query rewriting. The proposed algorithm can be used for general knowledge bases with disjunctive existential rules; however, there are rather strong conditions that ensure the existence of a finite UCQ-rewriting. In the paper we study some of those sufficient conditions that ensure that the proposed algorithm stops and yields a finite UCQ-rewriting.

Using the proposed algorithm and taking advantage of the stopping criteria, we implemented a sound and complete rewriting approach for union of conjunctive queries with negated atoms and answer variables in the COMPLETO system that specialises in query answering for conjunctive queries with negation. The implementation was evaluated on two ontologies in order to get an idea of the performance.

The experimental results showed that the implementation is able to provide UCQ-rewritings in reasonable time and using using a reasonable amount of RAM memory. Also, rewriting UCQ¬s with a large amount of queries takes considerably less time than the time required to rewrite all the CQ¬s individually.

In the future, we would like to focus on implementing the proposed algorithm in a more efficient way and also on being able to use disjunctive rules as part of the knowledge base and not only the ones corresponding to the queries with negated atoms.

Acknowledgments

We would like to thank Lida Petrou for providing the queries for the experiments. Also, we thank Stathis Delivorias and Michael Giazitzoglou for their support and comments in the writing process of this paper.

References

[1] Agrawal, R.; Imieliński, T.; Swami, A.: Mining association rules between sets of items in large databases. Proceedings of the 1993 ACM SIGMOD international conference on Management of data - SIGMOD '93. p. 207.

[2] Ahmetaj S., Ortiz M., Simkus M.: Rewriting Guarded Existential Rules into Small Datalog Programs. International Conference on Database Theory, ICDT 2018: 4:1-4:24 2017.

[3] Baget, J.F., Leclère, M., Mugnier, M.L., Salvat, E.: On rules with existential variables: Walking the decidability line. Artificial Intelligence **175**(9), 1620–1654 (2011).
[4] Baget, J.-F., Leclère, M., Mugnier, M.L., Salvat, E.:Extending decidable cases for rules with existential variables. In International Joint Conferences on Artificial Intelligence (IJCAI), 677–682, 2009.
[5] Baget, J.-F.:Improving the forward chaining algorithm for conceptual graphs rules. In KR, pages 407–414. AAAI Press, 2004.
[6] Baget, J.-F., Leclère, M., Mugnier.:The Complexity of Rules and Constraints. Journal of Artificial Intelligence Research (JAIR), 16:425–465, 2002.
[7] Bárány, V., ten Cate, B., Otto, M.: Queries with Guarded Negation. InProc. International Conference on Very Large Data Bases (VLDB), Endow. 5, 11 (July 2012), 1328-1339.
[8] Bárány V., Benedikt M., and ten Cate B.: Rewriting guarded negation queries. InProc. of the International Symposium on Mathematical Foundations of Computer Science (MFCS' 13), pages 98–110. ACM, 2013.
[9] Bienvenu, M., ten Cate, B., Lutz, C., and Wolter, F.: Ontology-based data access: A study through disjunctive datalog, CSP, and MMSNP. ACM Transactions on Database Systems (TODS) 39(4):33:1–33:44, 2014.
[10] Bourhis P., Manna M., Morak M., and Pieris A. 2016. Guarded-Based Disjunctive Tuple-Generating Dependencies. ACM Transactions on Database Systems (TODS) 41, 4, Article 27 (November 2016).
[11] Carral, D., Dragoste I., Krötzsch, M. : Tractable Query Answering for Expressive Ontologies and Existential Rules. Proceedings of the 16th International Semantic Web Conference (ISWC'17), LNCS 10587, Springer 2017.
[12] Du, J., Pan, J.Z.: Rewriting-Based Instance Retrieval for Negated Concepts in Description Logic Ontologies, pp. 339–355. Springer International Publishing, Cham (2015).
[13] Gottlob, G., Manna, M., Morak, M., Pieris, A.:On the Complexity of Ontological Reasoning under Disjunctive Existential Rules. 1–18. Mathematical Foundations of Computer Science (MFCS 2012), 2012.
[14] Gottlob, G., Rudolph, S., and Simkus, M.: Expressiveness of guarded existential rule languages. Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS '14, 2014, pages 27–38. ACM, 2014.
[15] Guo, Y., Pan, Z., Heflin, J.: LUBM: A Benchmark for OWL Knowledge Base Systems. Web Semant. **3**(2-3), 158–182 (Oct 2005).
[16] Gutiérrez-Basulto, V., Ibañez-García, Y., Kontchakov, R., Kostylev, E.V.: Conjunctive Queries with Negation over DL-Lite: A Closer Look, pp. 109–122. Springer Berlin Heidelberg, Berlin, Heidelberg (2013).
[17] König, M., Leclère, M., Mugnier, M., Thomazo, M.: Sound, complete and minimal UCQ-rewriting for existential rules. Semantic Web **6**(5), 451–475 (2015).
[18] Libkin, L.: Incomplete information and certain answers in general data models. Proceedings of the thirtieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, pages 59–70, 2011.
[19] Matos Alfonso, E., Stamou, G.: On Horn Conjunctive Queries, pp 115-130: Rules and Reasoning - Second International Joint Conference, RuleML+RR 2018, Luxembourg, Luxembourg, September 18–21, 2018, Proceedings.
[20] Matos Alfonso, E., Stamou, G.: Rewriting Queries with Negated Atoms, pp. 151–167. Springer International Publishing, Cham (2017).
[21] Definition for first-degree relative, NCI Dictionary of Cancer Terms https://www.cancer.gov/publications/dictionaries/cancer-terms/def/first-degree-relative.
[22] Nienhuys-Cheng, Shan-Hwei, de Wolf, Roland. : First-order logic: Foundations of Inductive Logic Programming pp. 17–34 Springer Berlin Heidelberg, CY - Berlin, Heidelberg (1997).
[23] Nienhuys-Cheng, Shan-Hwei, de Wolf, Roland. : Resolution: Foundations of Inductive Logic Programming Springer Berlin Heidelberg, CY - Berlin, Heidelberg (1997) pp. 55–74.
[24] Onet, A.:The Chase Procedure and its Applications in Data Exchange. Data Exchange, Information, and Streams, 2013.
[25] Tessaris, S.:Questions and answers: reasoning and querying in Description Logic.PhD thesis, University of Manchester. PhD thesis, University of Manchester (2001).
[26] Yuanbo Guo, Zhengxiang Pan, and Jeff Heflin. Lubm: A benchmark for owl knowledge base systems. *Web Semant.*, 3(2-3):158–182, October 2005.