UCQ-rewritings for Disjunctive Knowledge and 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, achor1@cs.ntua.gr, gstam@cs.ntua.gr

Editor: Bernardo Cuenca Grau, Oxford University, UK
Solicited reviews: Michael Morak, Institut für Angewandte Informatik, Austria; Three anonymous reviewers

Abstract. In this paper, we study the problem of query rewriting for disjunctive existential rules. Query rewriting 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 the disjunctive components of disjunctive existential rules. This process eventually generates new non-disjunctive rules, i.e., existential rules. The generated rules can then be used to produce new rewritings using existing rewriting approaches for existential rules. With the proposed technique we are able to provide complete UCQ-rewritings for union of conjunctive queries with universally quantified negation. We implemented the proposed algorithm in the COMPLETO system and performed experiments that 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 [21]; 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 in the presence of rules, which depend on the way we use the rules. The forward chaining approach [26] applies the rules on the facts in order to produce new facts. On the other hand, the backward chaining approach [20] uses the rules to translate the input query into a set of queries (called the rewriting of the initial query) that also encode answers of the initial query. Both approaches allow us to infer answers that cannot be extracted from the initial data.

Example 1.1. Let us consider a rule that defines the grand-parent relationship between two people based on the parent relationship

\[ r = \forall X \forall Y \text{parent}(X, Z) \land \text{parent}(Z, Y) \rightarrow \text{grand-parent}(X, Y) \]

and the information that

\[ \text{parent}(ana, maria) \land \text{parent}(maria, julieta). \]

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

\[ Q' = \exists Z \exists Y \text{parent}(ana, Z) \land \text{parent}(Z, Y). \]

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, this approach does not always terminate [26], and may keep

*Corresponding author. E-mail: gardero@image.ntua.gr.
generating constantly 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 size of the initial query [19] or in some cases a finite rewriting may not exist. In both approaches, the application of the rules does not always terminate. Furthermore, having no restrictions on the expressivity of the rules or having negated atoms on the query makes the query entailment problem undecidable [7].

In this paper, we focus on the entailment problem of conjunctive queries with negated atoms (CQ−) in the framework of disjunctive existential rules based on first-order logic (FOL) without function symbols.

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

\[
\forall X \exists Y \text{is-parent}(X) \rightarrow \text{father}(X,Y) \lor \\
\text{mother}(X,Y),
\]

where disjunction and existentially quantified variables can appear on the right-hand side of the implication. Conjunctive query entailment is undecidable in the case of existential rules with disjunction. However, under some restrictions the problem can become decidable [7, 15]. To the best of our knowledge, existing research on existential rules with disjunction is only based on forward chaining algorithms [13, 15] or Disjunctive Datalog rewriting [3, 9, 11, 16].

Conjunctive queries with negation let us define the counterexamples of disjunctive rules, e.g.,

\[
\exists X \forall Y \forall Z \text{person}(X) \land \neg \text{married}(X,Y) \land \\
\neg \text{parent}(X,Z)
\]
describes when the following rule does not hold

\[
\forall X \exists Y \exists Z \text{person}(X) \rightarrow \text{married}(X,Y) \lor \\
\text{parent}(X,Z).
\]

Here we use the open world assumption, where the negation is associated to the “cannot” semantics, and the example query expresses the question of whether there is a person that cannot be married and cannot be a parent. We also consider universally quantified negation [1, 27, 28], i.e., variables that are only present in negated atoms are universally quantified.

The entailment problem for CQ− is undecidable even for very simple types of rules [18, 28]. On the other hand, the use of guarded negation in queries is proven to be decidable over frontier-guarded existential rules [10]. Yet, the existing query rewriting-based approaches in the literature [14, 22, 23] 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, by using these expressive resources in a smart way we can get very interesting and useful decidable fragments.

Particularly, in this paper we are interested in solving the entailment of a union of conjunctive queries with universally quantified negation by rewriting it into a union of conjunctive queries without negation (UCQ), called a UCQ-rewriting, where each element in it is a rewriting of the initial CQ−.

Related Work Rosati studies query answering with respect to description logics (DLs) [28] and with respect to relational databases with integrity constraints (ICs) [27]. Extensions that allow safe negation and universally quantified negation are considered. The author provides a set of decidability, undecidability and complexity results for answering different types of queries with respect to various classes of DL knowledge bases and various combinations of ICs. In general, his results show that answering relational queries is unfeasible in many DLs and IC languages. The author considers unions of conjunctive queries with universally quantified negation, and shows that answering queries of this class is undecidable in every DL fragment and even in the absence of ICs.

In [20], König et al. define a generic rewriting procedure for conjunctive queries with respect to existential rules which takes as a parameter a rewriting operator, i.e., a function which takes as input a CQ and a set of existential rules and outputs a set of CQs. They also define the properties of rewriting operators that ensure the correctness and completeness of their algorithm. The authors prove that piece unification provides a rewriting operator with the desired properties. Finally, they provide an implementation of their algorithm together with some experiments. In this paper, we extend their definition of piece unification to deal also with disjunctive existential rules. We also extend their rewriting algorithm for existential rules into a sound and complete algorithm that also supports disjunctive existential rules and queries with negated atoms. However, the algorithm does not always terminate.

In [12], Bourhis et al. present a study of the complexity of query answering with respect to guarded disjunctive existential rules. The problem is $2\text{EXPTIME}$-complete.
hard even for the simplest guarded-based class of dis-
junctional 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 do not have a positive impact on the 
complexity of the problem. The only significant de-
crease in the complexity was found for atomic queries 
and linear disjunctive existential rules. The authors 
proved that for fixed atomic queries and fixed linear 
disjunctive existential rules the problem is in the AC
_0_ complexity class by establishing that the problem is 
first-order rewritable, i.e., it can be reduced to the pro-
blem of evaluating a first-order query over a database. 
The rewriting algorithm that we propose in this paper 
terminates for the fragment considered by Bourhis et 
al. The techniques and results presented in [12] can be 
used as a generic tool to study the complexity of query 
answering for several fragments of description logics. 

Some other research papers [3, 9, 11, 16] focus on 
the transformation of the query answering problem 
with respect to guarded (disjunctive) existential rules 
into a query answering problem with respect to (dis-
junctive) Datalog programs by getting rid of existen-
tial variables. In [3], Ahmetaj et al. provide a trans-
formation of the problem that yields a polynomial size 
(disjunctive) Datalog program when the maximal num-ervariables in the (disjunctive) existential rules is 
bounded by a constant. The translations proposed by 
other authors [9, 11, 16] are of exponential size.

Outline of our contributions. We introduce a restric-
ted form of FOL resolution (constraint resolution) 
that is sound and refutation complete, and where the 
subsumption theorem holds if the consequences are 
clauses without positive literals. The number of choices 
at the moment of selecting clauses to perform con-
straint resolution steps is reduced with respect to the 
number of choices we have for the case of unrestricted 
resolution steps.

Based on the constraint resolution method, we pro-
apose an algorithm to compute a UCQ rewriting for 
an input UCQ with respect to (disjunctive) existential 
rules and constraints. This algorithm can also compute 
UCQ rewritings for conjunctive queries with uni-
versally quantified negation by converting queries with 
negation into rules. The algorithm is sound, provides a 
complete UCQ rewriting, and terminates for the cases 
where there is a finite and complete UCQ rewriting of 
the input query with respect to the (disjunctive) existen-
tial rules and the negative constraints. We also present 
two theorems with sufficient conditions for the termin-
ation of the algorithm. One case requires disconnected 
disjunctive existential rules (rules where the body and 
the head do not share variables) and existential rules 
that yield a finite and complete UCQ rewriting for any 
UCQ (finite unification set). The other case is based on 
queries and knowledge bases where all the elements are 
linear (rules with at most one atom in the body and CQs 
with at most one positive atom).

Additionally, we consider unions of conjunctive 
queries with negated atoms and answer variables (de-
noted as UCQ_a) with respect to existential rules and 
constraints. The proposed algorithm is modified in or-
der to compute only the deterministic UCQ rewritings, 
i.e., the UCQ rewritings that lead to certain answers 
or that check the consistency of the knowledge base. 
We prove that the modified algorithm terminates when 
there is a finite and complete deterministic UCQ-
rewriting of the input UCQ_a with respect to the ex-
istential rules and constraints of the knowledge base. 
We also present two theorems with sufficient condi-
tions for the termination of the modified rewriting al-
gorithm. One case requires the existential rules to be 
a finite unification set and the input queries to have the 
variables in both the positive and negated atoms as part 
of the answer variables of the query. The second case 
requires the queries to have only one positive atom and 
an arbitrary number of negated atoms; also the existen-
tial rules and constraints are required to have only one 
atom in the body.

Finally, we implemented the proposed algorithm for 
rewriting conjunctive queries with negated atoms and 
answer variables with respect to existential rules in 
the COMPLETO system, and 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. Sub-
section 2.1 introduces concepts related to FOL resol-
ution and Subsection 2.2 presents the disjunctive ex-
istential rules framework. Section 3 introduces a novel 
type of resolution (Subsection 3.1) and a novel back-
ward chaining rewriting algorithm for disjunctive exist-
ential rules (Subsection 3.2). Additionally, Subsection 
3.3 explores some termination conditions for the pro-
posed rewriting algorithm and Subsection 3.4 proposes 
a modification of the algorithm that computes only the 
deterministic UCQ rewritings of UCQ_a’s. Section 4 
describes an implementation of the proposed rewriting 
algorithm. Experiments describing the performance of 
our implementation are presented in Subsection 4.1. Fi-
nally, Section 5 presents an overview of the paper with 
some conclusions.
2. Preliminaries

In this section we introduce the basic concepts related to the FOL 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. 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 FOL 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 the entailment and equivalence of formulas, as they are rarely modified in the literature. We refer the reader to [25] in case a background reading is needed.

Because in the following we will often need to modify formulas, in order to make such modifications more compact and easier to understand we introduce the definition of conjunctive (disjunctive) set formulas (CSFs and DSFs). However, the reader should be familiar with the notation because it is often used in FOL when we write rules and clauses.

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\}, where the set of formulas is interpreted as a conjunction (disjunction) of the formulas in the set, i.e.,

\[ F_1 \land \ldots \land F_n \ (F_1 \lor \ldots \lor F_n). \]

For a given set of formulas \{F_1, \ldots, F_n\}, a CSF containing these formulas is denoted as \{F_1, \ldots, F_n\}. On the other hand, a DSF is denoted by \{F_1, \ldots, F_n\}. Finally, an empty CSF is denoted by \{\} and is equivalent to \(\top\), and an empty DSF is denoted by and is equivalent to \(\bot\).

Note that in case a CSF is a sub-formula of another set formula we use parenthesis to avoid ambiguity, e.g., \([(A, B), D]\) is equivalent to \((A \land B) \lor D\).

Set operators can then be used to combine set formulas of the same type and obtain a new set formula. Moreover, elements of a set formula that are equivalent can be collapsed into a single element, e.g.,

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

The following axioms can be easily proven:

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

\[ [F] \equiv F. \]

2. De Morgan’s Laws allow changing a DSF to a CSF, and vice-versa, using negation:

\[ \neg[F_1, \ldots, F_n] \equiv \neg F_1, \ldots, \neg F_n \]
\[ (F_1, \ldots, F_n) \equiv [\neg F_1, \ldots, \neg F_n]. \]

3. Let B and F be two CSFs (DSFs) such that B \(\in F\). Then, \(F \equiv (F \setminus \{B\}) \cup B\). Replacing F by the equivalent formula \((F \setminus \{B\}) \cup B\) is referred to as flattening the formula F.

We model the entailment operator in FOL assuming that on the left-hand side of the entailment symbol we have a CSF \(A_1, \ldots, A_n\) of axioms \(A_i\):

\[ A_1, \ldots, A_n \models F. \]

The right-hand side of the entailment operator is assumed to be a DSF. For CSFs \(B\) and \(A\) (where \(B \subseteq A\)) and DSFs \(C\) and \(F\) (where \(C \subseteq F\)), we can prove that \(A \models F\) if and only if \(A \setminus B \models F \lor \neg B\); likewise \(A \models F\) if and only if \(A \cup \neg C \models F \lor C\).

In the following, we recall the definitions of some of the concepts that are needed for the theory presented in this paper.

A term is a constant, a variable or an expression \(f(t_1, \ldots, t_m)\) where \(f\) is a function symbol and the arguments \(t_i\) are terms. However, in this paper we focus only on simple terms, i.e., either variables or constants. We only consider Skolem function symbols internally in order to get rid of existentially quantified variables.

An atom is a formula \(a(t_1, \ldots, t_n)\) where \(a\) is a predicate of arity \(n\) (denoted by \(a/n\)). The arguments \(t_i\) of the atom are terms. A literal is an atom or a negated atom. The complement \(l\) of a literal \(l\) is \(\neg a(t_1, \ldots, t_n)\) if \(l = a(t_1, \ldots, t_n)\), and \(a(t_1, \ldots, t_n)\) if \(l = \neg a(t_1, \ldots, t_n)\). A literal is positive (or of positive polarity) if it is a non-negated atom, and negative (or of negative polarity) if it is a negated atom. Two literals are complementary if one is the complement of the other.

A formula is ground if it contains no variables. In a formula, the variables can be universally quantified, existentially quantified, or free. A formula without free variables is closed. The set of all the variables that appear in an expression \(F\) is denoted by \(\text{vars}(F)\). We denote a sequence of variables \(X_1, \ldots, X_n\) using a boldface character (e.g., \(\mathbf{X}\)).
A substitution \( \theta = \{ X_1 \leftarrow t_1, \ldots, X_n \leftarrow t_n \} \) is a finite mapping of variables \( X_i \) to terms \( t_i \). The result of applying a substitution \( \theta \) on an expression \( F \) is the expression \( F\theta \) obtained by replacing in \( F \) every occurrence of every variable \( X_i \) by the term \( t_i \).

Let \( F \) be an expression and \( \theta \) the substitution \( \{ X_1 \leftarrow Y_1, \ldots, X_n \leftarrow Y_n \} \). We say \( \theta \) is a renaming substitution for \( F \), if each \( X_i \) occurs in \( F \), and \( Y_1, \ldots, Y_n \) are distinct variables such that each \( Y_j \) is either equal to some \( X_j \) in \( \theta \), or \( Y_j \) does not occur in \( F \). The composition of two substitutions \( \theta \) and \( \sigma \) is a new substitution \( \theta \sigma \) that when applied to any expression, has the same effect as applying those substitutions in sequence (i.e., first \( \theta \) and after \( \sigma \)). A substitution \( \theta \) is more general than another substitution \( \sigma \) if there exists another substitution \( \gamma \) such that \( \sigma = \theta \gamma \).

A substitution \( \theta \) is a unifier for a set of expressions \( S = \{ F_1, \ldots, F_m \} \) iff \( F_1 \theta = \ldots = F_m \theta = F_\theta \). The expressions in \( S \) are said to be unifiable if there is a unifier for \( S \). The most general unifier (mgu) of the expressions in \( S \) is denoted by \( \text{mgu}(S) \), and it is a unifier for \( S \) that is more general than any other unifier of the expressions in \( S \). Even if it is possible for the same set \( S \) to have more than one \( \text{mgu} \), they are unique up to variable renaming.

A hypergraph is a tuple \((A, E)\), where \( A \) is a set of elements called nodes or vertices, and \( E \) is a set of non-empty subsets of \( A \) called hyperedges. We can represent a CSF of atoms as a hypergraph using the set of variables that appear in the arguments of the atoms as nodes. Each atom in the formula represents a hyperedge that connects its variables. Note that hyperedges are defined as sets of nodes and there is no notion of direction between the nodes. With this representation 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: \( \text{card}(F) = |\text{vars}(F)| \). Two variables \( u \) and \( v \) in a CSF of atoms \( F \) are connected iff they both belong to the same atom (\( \exists A \in F \mid \{v, u\} \subseteq \text{vars}(A) \)), 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 represented by an empty hyperedge. The constants in the formula play no role in their hyperedge representation.

It follows that a CSF \( F \) can be partitioned into a set \( \{ U_1, \ldots, U_m \} \) of connected CSFs such that if \( v \in \text{vars}(U_i) \) is connected to \( u \in \text{vars}(U_j) \), then \( i = j \). If \( F \) is connected, this set contains only \( F \). The connected cardinality of \( F \) is defined as the maximum cardinality of the connected CSFs in the partition of \( F \) and denoted by \( \text{card}^*(F) = \max_i (\text{card}(U_i)) \). The connected cardinality of a DSF \( \{ F_1, \ldots, F_m \} \) is the maximum connected cardinality of the formulas \( F_i \), i.e., \( \text{card}^*(\{ F_1, \ldots, F_m \}) = \max_i (\text{card}^*(F_i)) \).

**Example 2.1. The CSF**

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

is represented by the hypergraph in Figure 1. We can split \( F \) in two connected components

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

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

**Lemma 2.1.** Let \( G \) be a CSF, and let \( \{ U_1, \ldots, U_n \} \) be the partition of a given CSF \( F \) of atoms into connected CSFs. Then,

\[
G \models F \iff G \models U_i, \text{ for every } U_i.
\]

**Proof.** A detailed proof is given by Tessaris [29]. However, the reader can clearly see that since no variable is shared between the connected components \( U_i \), the assignments for the variables that make each \( U_i \) valid in \( G \) can be combined without introducing conflicts on the values that each variable gets.

**Lemma 2.2.** Let \( k \) be a natural number. There are a finite number of equivalence classes of CSFs of atoms with connected cardinality of at most \( k \) that can be constructed using a finite set of predicates and constants.

**Proof.** It is easy to check that two CSFs of atoms are equivalent iff they are unifiable by a renaming substitu-
olution. Since we have finitely many predicates and constant symbols, and at most \( k \) different variables, we can combine them in a finite number \( M \) of ways to form a connected CSF. In a CSF consisting of more than \( M \) connected CSFs we know that some of the connected components are renamings of others, and keeping only one of them is enough for the evaluation of \( F \) according to Lemma 2.1. Hence, there are at most \( 2^M \) different equivalence classes.

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

\[
F = [l^1_1, \ldots, l^n_1], \ldots, [l^m_1, \ldots, l^m_n].
\]

Every FOL formula can be transformed into an equisatisfiable CNF formula using variable standarization, Skolemization, De Morgan's laws, and the distributivity of the conjunction and disjunction logical operators.

An instance of a clause \( C \) is the result of applying a substitution \( \theta \) to the clause, i.e., \( C\theta \). If two or more literals of the same polarity in a clause \( C \) are unifiable and \( \theta \) is their most general unifier, then the clause \( C\theta \) is called a factor of \( C \), and the process of applying \( \theta \) 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 \( \sigma = mgu([l_1, l_2]) \). The binary resolvent of \( C_1 \) and \( C_2 \) with respect to the literals \( l_1 \) and \( l_2 \) is the clause:

\[
C_1 \cup C_2 = (C_1 \sigma \setminus [l_1 \sigma]) \cup (C_2 \sigma \setminus [l_2 \sigma]).
\]

\( C_1 \) and \( C_2 \) are said to be clashing clauses. A resolvent \( C_1 \cup C_2 \) of \( C_1 \) and \( C_2 \) is a binary resolvent \( C_1 \sigma_1 \cup C_2 \sigma_2 \) of factors \( C_1 \sigma_1 \) of the two clauses.

It is easy to show that resolution is sound, i.e., \( C_1, C_2 \vdash C \). Consequently, the resolution rule can be used to deduce new clauses and to prove that a formula is unsatisfiable if we are able to derive the empty clause.

**Definition 2.3 (Resolution Derivation (Refutation)).** Let \( \Sigma \) be a set of clauses and \( C \) a clause. A (resolution) derivation of \( C \) from \( \Sigma \) is a finite sequence of clauses \( R_1, \ldots, R_k \), such that each \( R_i \) is either in \( \Sigma \), or a resolvent of two clauses in \( \{ R_1, \ldots, R_{i-1} \} \). If such a derivation exists, we write \( \Sigma \vdash_r C \), and say that \( C \) can be derived from \( \Sigma \). A derivation of the empty clause \( \bot \) from \( \Sigma \) is called a refutation of \( \Sigma \). 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 to produce a resolvent \( R_i \) in a resolution derivation. In such cases, a graph or tree representation can be helpful.

**Definition 2.4 (Derivation (Refutation) Graph).** Let \( \Sigma \) be a set of clauses and \( C \) a clause such that \( \Sigma \vdash_r C \). A derivation (refutation) graph of \( C \) from \( \Sigma \) is a directed graph where the nodes are the clauses from the derivation \( \vdash_r C \), and where there is an edge from each resolvent to the clauses used in the resolution step by which it was derived.

If we only include in the graph the last clause \( C \) and the clauses that are used in at least one resolution step of the derivation, we can see the derivation graph as a tree, in which the last resolvent \( C \) is the root of the tree and the leaves are clauses from \( \Sigma \). Indeed, by cloning nodes with more than one input edges (i.e., clauses used in several resolution steps) we can transform the derivation graph into a derivation tree. It is more convenient to draw such derivation trees upside-down (Figure 2).

**Theorem 2.1 (Soundness of Derivation).** Let \( \Sigma \) be a set of clauses, and \( C \) a clause. If \( \Sigma \vdash_r C \), then \( \Sigma \vdash C \).

**Proof.** This is a straightforward consequence of the soundness of the resolution rule.
Definition 2.5 (Subsumption). Let $C$ and $D$ be two clauses. We say that $C$ subsumes $D$ if there exists a substitution $\theta$ such that $C\theta \subseteq D$.

Definition 2.6 (Deduction). Let $\Sigma$ be a set of clauses and $C$ a clause. We say that $C$ is a deduction of $\Sigma$ from $\Sigma$, and write $\Sigma \vdash C$, if $C$ is a tautology, or if there exists a clause $D$ such that $\Sigma \vdash D$ and $D$ subsumes $C$. If $\Sigma \vdash C$, we say that the clause $C$ can be deduced from $\Sigma$.

Resolution steps or derivations involving ground instances of clauses (i.e., clause instances that do not contain variables) ensure that there are corresponding resolution steps or derivations involving the non-ground version of the clauses. This process is known as lifting a resolution step or derivation.

In the text below we recall known theorems taken from the literature [25].

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

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

Resolution derivations allow us to infer clauses that are logical consequences of an initial knowledge in a complete way.

Theorem 2.4 (Subsumption Theorem). Let $\Sigma$ be a set of clauses, and $C$ a clause. Then $\Sigma \vdash C$ iff $\Sigma \vdash C$. 

Theorem 2.5 (Refutation completeness of Resolution). Let $\Sigma$ be a set of clauses. Then $\Sigma$ is unsatisfiable iff $\Sigma \vdash \bot$.

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

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

Property 2.1 (Symmetry). If $C_1$ and $C_2$ are clashing clauses, then their resolvent clause can be computed in a symmetric way:

\[ C_1 \cup_r C_2 \equiv C_2 \cup_r C_1. \]

Property 2.2 (Distributivity). If $C_1$, $C_2$ and $C_3$ are clauses such that $C_3$ resolves with the resolvent of $C_1$ and $C_2$ using literals from both $C_1$ and $C_2$, then the following distributivity property holds:

\[ (C_1 \cup_r C_2) \cup_r C_3 \equiv (C_1 \cup_r C_3) \cup_r (C_2 \cup_r C_3). \] (1)

Note that on right-hand side of (1) the literals used in the resolution steps with respect to $C_3$ ($C_1 \cup_r C_3$ and $C_2 \cup_r C_3$) need to be the same that are used in the left-hand side.

Property 2.3 (Commutativity). If $C_1$, $C_2$ and $C_3$ are clauses such that $C_3$ resolves with the resolvent of $C_1$ and $C_2$ using only literals from $C_1$, then the following commutativity property holds:

\[ (C_1 \cup_r C_2) \cup_r C_3 \equiv (C_1 \cup_r C_3) \cup_r C_2. \]

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

Example 2.2. We will illustrate the distributivity property for $C_1 = \{a(x), b(x)\}$, $C_2 = \{a(x), \neg b(x)\}$ and $C_3 = \{\neg a(x), c(x)\}$. On the left-hand side of (1) we have the following resolution steps:

\[ (C_1 \cup_r C_2) = \{a(x)\} \]
\[ (C_1 \cup_r C_3) \cup_r (C_2 \cup_r C_3) \equiv \{a(x), \neg a(x), c(x)\} \equiv \{c(x)\} \]

and on the right-hand side

\[ \{a(x), c(x)\} \]
\[ \{\neg a(x), c(x)\} \]
\[ \{b(x), c(x)\} \]
\[ \{\neg b(x), c(x)\} = \{b(x), c(x)\} \]
\[ \{c(x)\}. \]

Clearly, $\{c(x)\} \equiv \{c(x)\}$. 

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 (which we denote by $X$) are existentially quantified, i.e., an expression of the form $\exists X l_1, \ldots, l_n$. Queries that allow negation in the literals $l_i$ are called conjunctive queries with negation (CQ$^\neg$). All the variables $X$ that appear in the positive literals of a CQ$^\neg$ are assumed to be existentially quantified. In order to avoid domain dependent queries, we use universally quantified negation [1, 27, 28, i.e., all the variables $Z$ that appear only in negative literals are assumed to be universally quantified: $\exists X Z l_1, \ldots, l_n$. Because the variable quantification rules are straightforward we omit quantifiers, e.g., instead of $\exists X Y Z person(X), married(X, Y)$ we write $person(X), married(X, Y)$. The set of variables that appear in both positive and negative literals is called the frontier of the query. Note that for now we do not introduce the concept of answer variables. Therefore, the queries we define are normally known as Boolean conjunctive queries. Consequently, throughout the paper by conjunctive query we mean Boolean conjunctive query.

A DSF of conjunctive queries (conjunctive queries with negation) is usually referred to as a union of conjunctive queries (UCQ) (union of conjunctive queries with negation (UCQ$^\neg$)). For a UCQ$^\neg$ $Q$, by $Q^{\neg\neg}$ we denote the set of CQ$^\neg$’s in $Q$ that contain exactly $k$ negated atoms, and by $Q^{\neg\neg \neg}$ the set of CQ$^\neg$’s in $Q$ that contain two or more negated atoms. We use the term query to refer to either a CQ, CQ$^\neg$, UCQ or UCQ$^\neg$.

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

A closer look at the definition of facts reveals that a fact is equivalent to a Boolean conjunctive query. However, facts are used to express existing knowledge, while queries represent questions, so they have different roles in the process of reasoning.

A rule is a closed formula of the form
\[
\forall X \exists Y Y \rightarrow H,
\]
where the body $B$ is a CSF of positive literals, and the head $H$ is a DSF in which all $H' \in H$ are CSFs of positive literals. The set $X = \text{vars}(B)$ contains the variables that appear in the body, and they are universally quantified. On the other hand, $Y = \text{vars}(H) \setminus \text{vars}(B)$ are the variables that appear only in the head. They are existentially quantified, and they are called existential variables. The frontier of a rule is the set of variables that are present in both the body and head of the rule: $\text{vars}(B) \cap \text{vars}(H)$. We omit quantifiers when writing a rule.

A disjunctive existential rule is a rule with more than one disjoint in the head, i.e., $|H| > 1$. An existential rule is a rule with exactly one disjoint in the head. For simplicity we write the head of the existential rule as a CSF of atoms. A rule with an empty disjoint in the head is a negative constraint, i.e., $B \rightarrow \bot$. If it is clear from the context that we refer to a negative constraint, we can omit the “$\rightarrow \bot$” and write only the body $B$ of the negative constraint. Sometimes we also refer to a negative constraint as a constraint.

We say that a CSF of atoms $Q$ depends on a rule $r$ iff there is a CSF of atoms $F$ such that $F \neq Q$ and $F, r \vDash Q$. A rule $r_j$ depends on a rule $r_i$, iff the body of $r_j$ depends on $r_i$. The concept of rule dependencies allows us to define the graph of rule dependencies (GRD) [4], which is a graph where nodes are rules, and a directed edge between two nodes represents the existence of a dependency between the corresponding rules.

A knowledge base ($KB$) $\mathcal{K} = (R, D)$ is composed by a CSF $R$ of rules and a CSF of facts $D$. A knowledge base $(R, D)$ is a disjunctive knowledge base (DKB) if $R \neq \emptyset$. Let us define an example DKB about family relationships.

- Facts:
  
\[
(\text{parent}(Y, \text{ana}), \text{parent}(Y, \text{jane})),
\]

\[
(\text{sibling}(\text{ana}, \text{juan}))
\]

- Existential rules:
  
\[
(\text{sibling}(X, Y) \rightarrow \text{sibling}(Y, X)),
\]

\[
(\text{sibling}(X, Y) \rightarrow \text{parent}(Z, X), \text{parent}(Z, Y))
\]

- Negative constraints:
  
\[
(\text{sibling}(X, Y), \text{parent}(X, Y)),
\]

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

\[
(\text{parent}(X, Y), \text{parent}(Y, X)),
\]

\[
(\text{parent}(X, X))
\]
Disjunctive existential rules:

\[
\text{first-deg-relative}(X,Y) \Rightarrow \begin{cases} 
\text{parent}(X,Y), \text{parent}(Y,X), \\
\text{sibling}(X,Y)
\end{cases}
\]  

(9)

Note that the existential rule (4) has an existential variable \(Z\), that refers to an anonymous entity that is a parent of both siblings, i.e., if two people are siblings, they share a parent. Rule (3) states that the predicate sibling/2 is symmetric. We could also add symmetry for same-age/2. Fact (2) states that there is an anonymous entity \(Y\) that is a parent of both jane and ana. The negative constraints state the impossibility of a person being parent of his sibling (5) and also of a person of the same age (6). Additionally, with the negative constraints we express that parent/2 is asymmetric (7) and irreflexive (8). Finally, the disjunctive existential rule (9) defines the relation that represents the first degree relative concept [24]: a parent, a child (inverse of parent), or a sibling.

In this paper, we study the query entailment problem for disjunctive knowledge bases, i.e., the problem of knowing whether a query \(Q\) can be entailed from a disjunctive knowledge base \((\mathcal{R}, \mathcal{D})\):

\[
\mathcal{R}, \mathcal{D} \vDash Q.
\]  

(10)

In particular, we solve the entailment problem (10) by reducing it to the entailment of a UCQ \(Q'\) with respect to the set of facts \(\mathcal{D}\), i.e., to the problem

\[
\mathcal{D} \vDash Q'.
\]  

(11)

We say that \(Q'\) is a UCQ-rewriting of \(Q\) with respect to \(\mathcal{R}\) if for all set of facts \(\mathcal{D}\) it holds that

\[
\mathcal{D} \vDash Q' \implies \mathcal{R}, \mathcal{D} \vDash Q.
\]  

(12)

The UCQs in \(Q'\) are called UCQ-rewritings of \(Q\) with respect to \(\mathcal{R}\). If the converse of (11)

\[
\mathcal{R}, \mathcal{D} \vDash Q \implies \mathcal{D} \vDash Q'
\]

also holds for all set of facts \(\mathcal{D}\), we say that \(Q'\) is a complete UCQ-rewriting of \(Q\) with respect to \(\mathcal{R}\). Note that according to our definition, a UCQ-rewriting may not be complete. In this respect, our definition follows the definition of UCQ-rewriting from [20] because we extend many of the concepts and algorithms proposed by the authors.

Finally, we may be interested in the values that some of the variables in \(Q\) 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 first present constraint resolution, a novel type of resolution that is sound and refutation complete. Constraint resolution reduces the number of available choices in the resolution process by focusing on producing resolvents with a smaller number of positive literals. Constraint resolution is then translated into backward rewriting steps, allowing the definition of a rewriting algorithm for the framework of disjunctive existential rules that is able to solve the query entailment problem (10).

3.1. Constraint Resolution

The entailment problem (10) can be transformed into a consistency check problem

\[
\mathcal{R}, \mathcal{D}, \neg Q \not\vdash \bot,
\]  

(12)

which can then be solved using resolution refutation. Depending on the expressivity of the queries in \(Q\) their negation can yield new facts, negative constraints, existential rules or even disjunctive existential rules.

To apply resolution, we need to convert first the facts \(\mathcal{D}\) and rules \(\mathcal{R}\) of the DKB, as well as the negated query \(\neg Q\) to a CNF. In what follows, our purpose is to define a restricted resolution strategy in order to control the process of resolution among these clauses, so as to decrease the number of available choices every time we perform a resolution step. This might result in longer derivations, but the algorithm to generate them is simpler. The main goal of the restrictions we introduce is to make the resolution process focus on eventually generating resolvents without positive literals and any number of negative literals. The process can then continue by eliminating those remaining negative literals without introducing again positive literals.

Definition 3.1 (Positive/Negative Charge). The positive (negative) charge \(|C|^+ (|C|^-)\) of a clause \(C\) is the number of positive (negative) literals in the clause.
Properties of the different types of clauses in the disjunctive existential rules framework.

<table>
<thead>
<tr>
<th>Name</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rule clause (RC)</td>
<td>$</td>
</tr>
<tr>
<td>Fact clause (FC)</td>
<td>$</td>
</tr>
<tr>
<td>Constraint clause (CC)</td>
<td>$</td>
</tr>
<tr>
<td>Disjunctive RC (DRC)</td>
<td>$</td>
</tr>
</tbody>
</table>

According to the above definition, a Horn clause $C$ is a clause with positive charge smaller than or equal to one, i.e., $|C^+| \leq 1$. The clause we obtain by converting to CNF a negative constraint or the negation of a $\text{CQ}$ has zero positive charge. We call a clause with no positive literals a constraint clause (CC). The Skolemized version of an existential rule can produce several clauses with one positive literal, while a disjunctive existential rule can give rise to several clauses with more than one positive literal. We call a clause with only one positive literal a rule clause (RC), and a clause with more than one positive literal a disjunctive rule clause (DRC). Facts generate ground clauses containing only one literal, which has positive polarity, and we call such clauses fact clauses (FCs). Note that existential variables in the facts are replaced by Skolem terms. Table 1 summarizes the properties that define the different types of clauses we may encounter when doing resolution on a DKB. As we can see, RCs, FCs and CCs are Horn clauses. Finally, a $\text{CQ}^+$ may produce a FC, RC or DRC, depending on its positive and negative charge.

**Example 3.1.** From the DKB of Example 2.3 we obtain the following clauses:

- Fact clauses:
  
  $[\text{parent}(f_0, \text{ana})]$
  $[\text{parent}(f_0, \text{jane})]$
  $[\text{sibling}(\text{ana}, \text{juan})]$.

- Rule clauses:
  
  $[\text{sibling}(X, Y), \text{sibling}(Y, X)]$
  $[\neg \text{sibling}(X, Y), \text{parent}(f_2(X, Y), X)]$
  $[\neg \text{sibling}(X, Y), \text{parent}(f_2(X, Y), Y)]$.

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

<table>
<thead>
<tr>
<th>$C_1$</th>
<th>$C_2$</th>
<th>$\text{DRC}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>C_1^+</td>
<td>&lt;</td>
</tr>
<tr>
<td>$</td>
<td>C_1^+</td>
<td>= 0$</td>
</tr>
<tr>
<td>$\text{RC}$</td>
<td>$\text{RC}$</td>
<td>$\text{DRC}$</td>
</tr>
</tbody>
</table>

$\text{- Horn clause}$:

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

Table 2 shows the properties of the resolvent for different types of clauses when performing a resolution step. From these properties follows that a resolution refutation should involve resolution steps with respect to fact clauses, because they always produce clauses with a smaller negative charge. Such 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 on the initial part of a rearranged resolution derivation, until it reaches a clause that has only negated atoms. Such a process is linked to producing $\text{CQ}$-rewritings, since the resulting clause corresponds to the negation of a conjunctive query.

As we can see from Table 2, in order to get derivations that produce clauses with a non-increasing positive charge, we need to avoid resolutions steps involving two DRCs, i.e., we need to use in every resolution step at least one Horn clause. However, if we focus on resolution steps where one of the clauses used is a CC, the resolvent will always have a smaller positive charge (See CC column on Table 2).

**Definition 3.2 (Constraint Derivation).** Let $\Sigma$ be a set of clauses and $C$ a clause. A resolution derivation (refutation) $\Sigma \not\vdash C$ of a clause $C \in \Sigma$ from $\Sigma$ is a constraint derivation (refutation) iff all its resolution steps involve resolution with a constraint clause. A constraint derivation of a clause $C$ from $\Sigma$ is written as $\Sigma \not\vdash^\circ C$. Similarly, there is a constraint deduction of $C$ from $\Sigma$, written as $\Sigma \not\vdash^\Delta C$, if $C$ is a tautology or if there is a clause $D$ such that $\Sigma \not\vdash D$ and $D$ subsumes $C$.

The subsumption theorem can be formulated using constraint deductions and consequences with no positive literals.
Theorem 3.1 (Constraint Subsumption for Constraint Clause Consequences). Let \( \Sigma \) be a set of clauses and \( C \) a constraint clause. Then \( \Sigma \vdash C \) iff \( \Sigma \vdash^d C \).

Proof. Based on the subsumption theorem for resolution derivations (Theorem 2.4), the fact that \( \Sigma \vdash C \) implies that we have a deduction of \( C \), i.e., there is a derivation \( \Sigma \vdash^d D \) of a clause \( D \) that subsumes \( C \). Note that \( D \) needs to be a CC in order to subsume another CC. This proof is based on being able to transform every resolution derivation of a CC \( D \) into a constraint derivation of \( D \).

The resolution derivation \( \Sigma \vdash^d D \) for sure involves resolution steps with respect to some constraint clauses. Let \( C_{j_1} \) be the closest resolvent to the root (\( D \)) of the corresponding derivation tree that is obtained by applying binary resolution without using a constraint clause. We can assume that on the path to the root of the tree the resolution steps involve always constraint clauses \( C_{j_1}, \ldots, C_{j_k} : \)

\[
\left( C_{j_0} \cup C_{j_0} \right) \cup C_{j_1} \cup C_{j_2} \cup \ldots \cup C_{j_k} = D.
\]

Therefore, we can apply the distributivity property if \( C_{j_1} \) clashes on literals coming from both clauses \( C_{j_0} \) and \( C_{j_0} \), obtaining

\[
\left( \left( C_{j_0} \cup C_{j_0} \right) \cup C_{j_1} \right) \cup C_{j_2} \cup \ldots \cup C_{j_k} = D.
\]

On the other hand, if \( C_{j_1} \) clashes on literals coming from only one of the clauses (we assume it is \( C_{j_0} \) without loss of generality) we apply the commutativity property to obtain

\[
\left( \left( C_{j_0} \cup C_{j_1} \right) \cup C_{j_0} \right) \cup C_{j_2} \cup \ldots \cup C_{j_k} = D.
\]

If we continue the same process for all the clauses \( C_{j_1}, \ldots, C_{j_k} \) in the same order, we obtain

\[
\left( C_{j_0} \cup C_{j_1} \cup C_{j_2} \cup \ldots \cup C_{j_k} \right) \cup C_{j_0} = D.
\]

Because \( D \) is a constraint clause at least one of the clauses used to obtain it is also a constraint clause:

\[
\left| \left( C_{j_0} \cup C_{j_1} \cup C_{j_2} \cup \ldots \cup C_{j_k} \right) \right|^+ = 0
\]

or

\[
\left| \left( C_{j_0} \cup C_{j_1} \cup C_{j_2} \cup \ldots \cup C_{j_k} \right) \right|^+ = 0.
\]

This eliminates the resolution step between the two non-constraint clauses. In the same way, we can eliminate the rest of the resolution steps that involve two non-constraint clauses. Thus, transforming the existing deduction of \( C \) into a constraint deduction of \( C \).

Theorem 3.2 (Completeness of Constraint Resolution Derivations). A set of clauses \( \Sigma \) is unsatisfiable iff there exists a constraint refutation of \( \Sigma \), i.e., \( \Sigma \vdash^c \bot \).

Proof. This follows from Theorem 3.1 by taking as consequence the empty clause that has no positive literals and is only subsumed by itself.

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 [20]. However, in most of the existing literature disjunctive rules are mainly used in a forward chaining manner [13, 15] or to perform Disjunctive Datalog rewritings [3, 9, 11, 16].

In Example 2.3, one could infer that two first-degree relatives that have the same age have to be siblings:

\[
\text{first-deg-relative}(X, Y),
\]

\[
\text{same-age}(X, Y) \rightarrow \text{sibling}(X, Y).
\]

This rule can be obtained by a constraint derivation using (9) and the clauses corresponding to:

\[
\left( \text{same-age}(X, Y), \text{parent}(X, Y) \rightarrow \bot \right)
\]

\[
\left( \text{same-age}(X, Y) \rightarrow \text{same-age}(Y, X) \right)
\]

expressing respectively that children and their parents cannot have the same age and that the same-age/2 predicate is symmetric.

The new existential rule (13) can then be used in rewriting steps defined for existential rules.

A rewriting step as defined in the existential rules framework [20] corresponds to the resolution steps between RCs corresponding to an existential rule and a CC corresponding to the negation of a CQ.

Definition 3.3 (Rewriting Step). Let \( r = B \rightarrow H \) be an existential rule, and \( Q \) a conjunctive query. If there is a subset \( H' \subseteq H \) that unifies with some \( Q' \subseteq Q \) through a mgu \( \theta \) (i.e., \( H' \theta = Q' \theta \)) such that
1. If \( v \in \text{vars}(Q \setminus Q') \) and \( v \neq \theta \), then \( \theta \) is a frontier variable of \( r \) or a constant, and

2. If \( v \) is an existential variable of the rule \( r \), then \( \theta \notin \text{vars}(Q \setminus Q') \), then the query \((B \cup (Q \setminus Q'))\theta\) is a rewriting of \( Q \) using the existential rule \( r \).

If an existential rule \( r \) has more than one atoms in its head, it gives rise to more than one disjunctive rewriting step proposed in \([20]\).

For the resolution steps between some DRCs obtained from a disjunctive existential rule and a CC obtained from the negation of a CQ, we define a corresponding rewriting step generalizing Definition 3.3 with the goal to support both existential rules and disjunctive existential rules.

**Definition 3.4 (General (Disjunctive) Rewriting Step).** Let \( r = B \rightarrow H \) be a rule, and \( Q \) a conjunctive query. If there is a subset \( H' \subseteq H \), and for each \( h_i \in H' \) there is a subset \( h'_i \subseteq h_i \) that unifies with some \( Q' \subseteq Q \) through a mgu \( \theta \) (i.e., \( h'_1\theta = \ldots = h'_{n}\theta = Q'\theta \) such that

1. If \( v \in \text{vars}(Q \setminus Q') \), then \( v \theta \) is a frontier variable of \( r \) or a constant, and

2. If \( v \) is an existential variable of the rule \( r \), then \( v \theta \notin \text{vars}(Q \setminus Q') \),

then \((B \cup (Q \setminus Q'))\theta\) is a rewriting of \( Q \) using the rule \( r \). A rewriting step is a disjunctive rewriting step if the rule used is a disjunctive existential rule.

A disjunctive rewriting step can yield a disjunctive rule with fewer disjunctive components, an existential rule in case \(|(H \setminus H')\theta| = 1\) or a negative constraint (the negation of a conjunctive query) in case \( H = H' \).

**Example 3.2.** Consider a disjunctive existential rule:

\[ r_1 = \text{diabetesRisk}(X) \rightarrow [\text{diabetic}(Y), \text{parent}(Z, X)] \]

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

On the other hand, if we have the negative constraint \( \text{singleChild}(X_1), \text{sibling}(Y_1, X_1) \) and the query \( Q' = \text{diabetic}(Y_2), \text{parent}(Y_2, X_2) \) asking if there is a diabetic parent, we can derive the existential rule

\[ \text{diabetesRisk}(X), \text{singleChild}(X) \rightarrow \text{diabetic}(Z), \text{parent}(Z, X), \]

by rewriting the constraint using the rule \( r_1 \) and the unifier \( \theta_2 = \{ X_1 \leftarrow X, Y_1 \leftarrow Y \} \). Using the new existential rule we obtain the following UCQ-rewriting:

\[ [\text{singleChild}(X), \text{sibling}(Y, X)), \text{diabetic}(Y), \text{parent}(Y, X)), \text{diabetesRisk}(X), \text{singleChild}(X)] \]

Note that the final UCQ-rewriting contains also negated constraints which are possible reasons for which a query can be entailed, i.e., inconsistent knowledge bases. However, sometimes we might want to filter out the negated constraints if we are sure that the knowledge base is consistent.

Using the above-defined rewriting steps, we can now define rewriting for DKBs.

**Definition 3.5 (Rewriting).** Let \( (R, Q) \) be a tuple consisting of a set of \( R \) of rules and a UCQ \( Q \). A one-step rewriting \((R', Q')\) of \((R, Q)\) can be obtained by adding to \( R \) or to \( Q \), as appropriate, the result \( f' \) of a general rewriting step that uses one of the conjunctive queries in \( Q \) and a rule in \( R \), i.e., \( Q' = Q \cup \lnot f' \) if \( f' \) is a negative constraint, otherwise \( R' = R \cup \{ f' \} \).

A k-step rewriting of \((R, Q)\) is obtained by applying a one-step rewriting to a \( (k-1) \)-step rewriting of \((R, Q)\). For any \( k \), a k-step rewriting of \((R, Q)\) is a rewriting of \((R, Q)\).

So far we have dealt with rewritings of conjunctive queries with respect to existential rules and disjunctive existential rules. However, we have not considered negative constraints and conjunctive queries with negated atoms. Negative constraints are transformed into queries in the rewriting process, i.e.,

\[ R^\exists, R^\forall, R^\bot, D \vdash Q \quad \text{iff} \quad R, R^\forall, D \vdash \lnot R^\bot, Q. \]
In a similar way, if $Q$ is a UCQ, the entailment problem can be reduced to the entailment of a UCQ:

$$R^3, R^\forall, R^\bot, D \models Q \text{ iff } (R^3, \neg Q^{-1}), (R^\forall, \neg Q^{-\#}), D \models \neg R^\bot, Q^{-0}, \tag{14}$$

where $\neg Q^{-1}$ contains existential rules (negations of $Q^-$s with one negated atom) and $\neg Q^{-\#}$ disjunctive existential rules (negations of $Q^-$s with more than one negated atom).

Theorem 3.3 (Soundness and Completeness of Rewritings). Let $\langle R, D \rangle$ be a DKB and $Q$ a UCQ. Then $R, D \models Q$ iff there is a rewriting $\langle R', Q' \rangle$ of $\langle (R, R^\forall), (Q, \neg R^\bot) \rangle$ such that $D \models Q$ for some conjunctive query $Q^\prime$ in $Q'$. 

Proof. The $k$-step rewriting of $\langle (R, R^\forall), (Q, \neg R^\bot) \rangle$ is based on a constraint derivation. Moreover, such a rewriting can be mapped to a constraint derivation. Since constraint derivations are sound and complete (Theorem 3.2), this theorem also holds. \hfill \square

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

```
function rewrite_k(R, Q)
    Q := Q U \neg R^\bot
    R := R \setminus R^\bot
    do
        R_old := R
        Q_old := Q
        Q := rewrite_k^3(R^3, Q)
        R := rewrite^\forall(R, Q)
    while (Q \neq Q_old or R \neq R_old)
    return Q
end function
```

Given a set of rules $R$ and a UCQ $Q$, function $\text{rewrite}_{k/2}$ presented in Algorithm 1 computes all the rewritings of $\langle (R, R^\forall), (Q, \neg R^\bot) \rangle$. The algorithm alternates between computing the rewritings of $Q$ using existential rules ($\text{rewrite}_k^3$ presented in Algorithm 2) and computing the rewritings using disjunctive existential rules ($\text{rewrite}^\forall$ presented in Algorithm 3). New $Q$'s are used to generate more rules, and new existential rules are used to generate more $Q$'s until a fixed point is reached, i.e., until no new rule or query is produced.

Algorithm 2 Function to rewrite UCQs using existential rules.

```
function rewrite^\forall_k(R, Q)
    Q_old := Q
    Q_exp := Q
    level := 0
    do
        Q := cover(Q U rew(Q_exp, R))
        Q_exp := Q \setminus Q_old
        Q_old := Q
        level := level + 1
    while Q_exp \neq \emptyset and level < k
    return Q
end function
```

Algorithm 3 Function to rewrite UCQs using disjunctive existential rules.

```
function rewrite^\forall(R, Q)
    R_old := R
    R_exp := R^\forall
    do
        R := R U rew(Q, R_exp)
        R_exp := R \setminus R_old
        R_old := R
    while R_exp \neq \emptyset
    return R
end function
```

Function $\text{rew}_{k/2}$ (in Algorithm 2) computes the set of the results of all possible rewriting steps for all the combinations of existential rules and $Q$s in its arguments. This step is known as the expansion of a query, and it generates more conjunctive queries. On the other hand, function $\text{rew}^\forall_{k/2}$ (in Algorithm 3) computes the expansion of disjunctive existential rules by computing the set of the results of all disjunctive rewriting steps for all the combinations of disjunctive existential rules and $Q$s in its arguments that do not yield a conjunctive query. This restriction does not affect completeness because a $Q^\prime$ that can be generated from a $Q$ in a disjunctive rewriting step using rule $r$ can also be generated in two steps. In particular, a disjunctive rewriting step generates first an existential rule $\forall^\prime$. 


3.3. Rewritable Queries and Disjunctive Knowledge Bases

The termination of Algorithm 1 depends on the termination of Algorithms 2 and 3. Algorithm 3 always terminates because the produced rules 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 is denoted by rewrite\( ^{\infty}2 \) and corresponds to the classical rewriting algorithm for existential rules proposed in [20], whose termination is studied in [7]. In general, the problem of knowing if there exists a finite UCQ-rewriting for any UCQ with respect to an arbitrary set of existential rules is undecidable [7]. A set of existential rules that ensures the existence of a finite UCQ-rewriting for any UCQ is called a finite unification set (fus) [6]. There are some classes of existential rules that have the fus property:

1. Linear existential rules [6]: existential rules with only one atom in the body.
2. Disconnected existential rules [8]: existential rules that do not share variables between the body and the head.
3. Acyclic graph of rule dependencies (aGRD) [4]: existential rules that do not contain cycles in the graph of rule dependencies.

If a set \( \mathcal{R} \) of existential rules is a fus and the new existential rules generated by Algorithm 3 are also a fus, combining them could yield a new set of existential rules that is not a fus [7]. Therefore, we need stronger conditions to ensure that we always call Algorithm 2 with a set of existential rules that is a fus.

For a set of existential rules \( \mathcal{R} \), a cut is a partition \( \{ \mathcal{R}_1, \mathcal{R}_2 \} \) of \( \mathcal{R} \), 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.** Let \( \mathcal{R} \) be a set of existential rules with a directed cut \( \mathcal{R}_1 \triangleright \mathcal{R}_2 \). For any UCQ \( Q \) and any set of facts \( D \) we have that

\[
D, \mathcal{R} \vDash Q \quad \text{if and only if} \quad D, \mathcal{R}_1 \vDash Q' \quad \text{and} \quad Q', \mathcal{R}_2 \vDash Q.
\]

**Proof.** The proof is based on being able to organize the application of the rules of \( \mathcal{R} \). The existing dependencies ensure that the rules of \( \mathcal{R}_1 \) are never depending on the rules of \( \mathcal{R}_2 \). For a detailed proof check [7].
Property 3.1 [7] allows us to study the decidability of entailment when we combine two sets of rules for which the entailment problem is decidable.

In Algorithm 3, even if the resulting set of existential rules \( \mathcal{R} \) is 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 bounded, i.e., at some point the algorithm will not produce new rules.

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

A rule \( B \rightarrow H \) is linear if it has only one atom in the body, i.e., \(|B| = 1\).

**Theorem 3.4.** Let \( \mathcal{R} \) be a set of rules and \( Q \) a UCQ. If \( Q \) contains only atomic queries and \( \mathcal{R} \) only linear rules, then Algorithm 1 stops for 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 by the linear disjunctive existential rules and the atomic queries will also be linear rules, and combining them with \( \mathcal{R}^3 \) will also produce a fus. Additionally, the number of single atoms that can be built using a finite number of predicates, variables and constants is bounded. Therefore, the number of rules that we can derive from the linear disjunctive existential rules is finite. Consequently, Algorithm 1 stops because at some point no new rules and no new queries can be generated.

Theorem 3.4 is closely related to Theorem 7.13 and Lemma 7.12 proposed by Bourhis et al. in [12]. 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 remaining part of the query.

A rule \( B \rightarrow H \) is disconnected if no variable from the body is present in the head of the rule, i.e., \( \text{vars}(B) \cap \text{vars}(H) = \emptyset \). Disconnected rules can still share constants between the body and the head of the rule and this allows us to express knowledge about specific individuals.

**Theorem 3.5.** 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.** Disconnected rules add atoms to the rewritings that do not share variables with the remaining part of the query. It follows that the connected cardinality of CQ-rewritings produced by rules \( B \rightarrow H \in \mathcal{R}_2 \) is bounded as follows:

\[
\text{card}^*(\text{rewrite}^3(B \rightarrow H)) \leq \max(\text{card}^*(B), \text{card}^*(Q)).
\]

The rules in \( \mathcal{R}_1 \) may produce CQ-rewritings with a larger connected cardinality, but they only produce a finite number of CQ-rewritings because \( \mathcal{R}_1 \) is a fus. It follows that the connected cardinality of the CQ-rewritings of an initial query \( Q \) is bounded, i.e.,

\[
\text{card}^*(\text{rewrite}^3(\mathcal{R}_1 \cup \mathcal{R}_2), Q) \leq \max(\text{card}^*(\text{rewrite}^3(\mathcal{R}_1), B), \text{card}^*(\text{rewrite}^3(\mathcal{R}_1, Q))).
\]

Using Lemma 2.2 we conclude 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 3.6.** Let \( \mathcal{R} \) be a set of rules and \( Q \) a UCQ. If \( \mathcal{R}^3 \) is a fus, and \( \mathcal{R}' \) a set of disconnected disjunctive existential rules, then Algorithm 1 stops for any set of constraints \( \mathcal{R}^1 \), any UCQ \( Q \) and for any value of \( k \).

**Proof.** The new existential rules produced by the function \( \text{rewrite}^3 \) are disconnected rules and they can be combined with the rules in \( \mathcal{R}^3 \) and yield a fus (follows from Theorem 3.5).

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

\[
\bigcup_j B'_i \cup \bigcup_j Q'_j \rightarrow H',
\]

where \( H' \subseteq H_i \sigma \) is a subset of an instance \( H_i \sigma \) of the original head of the rule \( H_i \), \( B'_i \subseteq B'' \sigma' \) is a subset of an instance \( B'' \sigma' \) of a rewriting of the body \( B_j \) of a disjunctive existential rule, i.e., \( B'' \in \text{rewrite}^3(B_j, \mathcal{R}^3) \), and \( Q'_j \subseteq Q'' \sigma'' \) is a subset of an instance \( Q'' \sigma'' \) of a rewriting of an input UCQ or a negated constraint \( Q_j \), i.e., \( Q'' \in \text{rewrite}^3(Q_j, \mathcal{R}^3) \). The substitutions \( \sigma' \) and \( \sigma'' \) are compositions of the音乐会 applied in the rewriting steps. None of the \( B'_i \) or \( Q'_j \) share variables between them because they are introduced using an atom in the head of the disjunctive rule that does not share variables with the body.

Because \( \mathcal{R}^3 \) is a fus we have a finite number of rewritings \( B'_i \) and \( Q'_j \). This ensures that there is only a fi-
The completeness of the definition of rewritings

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 $Q_f$, then after a finite number of steps Algorithm 1 (with a finite value of $k$) will produce a rewriting equivalent to $Q_f$. Because $Q_f$ is complete, any further rewriting of $Q_f$ using existential rules will not produce new conjunctive queries, i.e., the condition $Q = Q_{\text{det}}$ holds 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 in the head. Consequently, we will reach an iteration of the loop in Algorithm 1 where $R = R_{\text{det}}$. Therefore, Algorithm 1 terminates because after a finite number of iterations the condition to continue iterating on the loop will no hold.

Note that a subset of the existential rules needed to generate the finite and complete UCQ-rewriting of the initial query in Theorem 3.7 could potentially produce an infinite number of rewritings.

Example 3.3. To illustrate this we can consider the following DKB:

- Existential rules:

$(r(X, W), r(W, Y) \rightarrow r(X, Y)),$
$(b(X) \rightarrow a(X)),$
$(c(X) \rightarrow a(X)).$

- Disjunctive existential rules:

$s(X) \rightarrow [b(X), c(X)].$

If we try to rewrite the UCQ $[a(X), (s(X), r(X, f))]$ with respect to $R^3$ using $k = \infty$ it would produce an infinite set of CQ-rewritings of the form:

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

However, for $k = 1$ (or any other finite value) the new rules produced by the disjunctive existential rule would eventually generate the conjunctive query $s(X)$. The application of the cover1 function would then 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., $k \neq \infty$) in order to avoid infinite loops.

Theorem 3.7 ensures that Algorithm 1 stops only if there is a finite and complete UCQ-rewriting for the input query otherwise the algorithm may never stop. However, it does not require the $\text{fin}$ property for the set of existential rules in the knowledge base. On the other hand, Theorems 3.4 and 3.6 ensure that Algorithm 1 will always terminate if the required conditions are met.

3.4. On Queries with Answer Variables and Linear Queries

While Theorems 3.4 and 3.6 impose rather strong restrictions on the disjunctive framework, they also suggest the existence of finite UCQ-rewritings for very expressive types of queries with negated atoms and answer variables.

A CQ $Q$ with answer variables (CQA) is a CQ of the form $\text{ans}(X), B$, where $B$ is a CSF of atoms, called the body of the query, and $\text{ans}(X)$ is the answer atom of the query. The fresh predicate $\text{ans}n$ is called the answer predicate and $X$ a tuple of variables or constants such that $\text{var}(X) \subseteq \text{var}(B)$, is called the answer tuple of the query. A CQA $Q$ is often written as $\text{ans}(X) \leftarrow B$. Conjunctive queries without answer variables are called Boolean CQs and for them the answer tuple is the empty tuple $X = ()$. A CQ with answer variables (CQA) is defined in the same way, but variables that appear only in negated atoms are not allowed to be part of the answer tuple $X$. A UCQ (UCQA) with answer variables (UCQA or UCQA) respectively
is a DSF of CQ as (CQa’s) with the same answer predicate. In general, a query with answer variables refers to CQa, CQa’s, UCQ a or to a CQ a.

Given a knowledge base $\mathcal{K}$, a tuple $t$ of constants in $\mathcal{K}$ is a certain answer of a query with answer variables $Q$ with respect to $\mathcal{K}$ if $\mathcal{K}, \text{ans}(t) \models Q$. The set of certain answers of a query $Q$ with respect to a knowledge base $\mathcal{K}$ is denoted by $\text{cert}(Q, \mathcal{K})$. Computing the set $\text{cert}(Q, \mathcal{K})$ is known as the query answering problem.

Example 3.4. Consider the following three CQ as, which have different sets of answer tuples:

$Q_1 = \text{ans}_1() : - \text{sibling}(X, Y),$

$Q_2 = \text{ans}_2(X) : - \text{sibling}(X, Y),$

$Q_3 = \text{ans}_3(X, Y) : - \text{sibling}(X, Y).$

We expect the certain answers of $Q_1$ to contain the empty tuple if someone has a sibling or otherwise be the empty set, the certain answers of $Q_2$ to be the set of people that have siblings, and the certain answers of $Q_3$ to be set of pairs of people that are siblings.

Consider also a knowledge base $\mathcal{K}$ based on Example 2.3, with a different set of facts:

$sibling(\text{pedro}, \text{ana}),$

$sibling(\text{juan}, Y).$

The first fact states that pedro and ana are siblings, while the second fact that juan has a sibling.

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

$\text{cert}(Q_1, \mathcal{K}) = \{()\},$

$\text{cert}(Q_2, \mathcal{K}) = \{\text{pedro, ana, juan}\},$

$\text{cert}(Q_3, \mathcal{K}) = \{(\text{pedro, ana}), (\text{ana, pedro})\}. $

Thus, there are some siblings in $\mathcal{K}$ (by $Q_1$), pedro, ana, and juan have siblings (by $Q_2$) and ana is a sibling of pedro and pedro a sibling of ana since sibling/2 is symmetric (by $Q_3$). Note that the sibling of juan has no identity so it is not included in the certain answers of $Q_3$. The answers can be easily verified by solving the corresponding entailment problems, e.g., pedro is a certain answer of $Q_2$ because

$\mathcal{K}, \text{ans}_2(\text{pedro}) \models \text{ans}_2(X), \text{sibling}(X, Y).$

For queries with answer variables we focus on the query answering problem instead of the entailment problem. In theory we could try all possible assignments of constants in $\mathcal{K}$ to variables in the answer tuple $X$ and check whether the resulting query is entailed. However, computing a UCQ-rewriting for each possible assignment of constants would not be very efficient. We can compute the UCQ-rewritings $Q'$ of a query with answer variables $Q$ with respect to the rules in $\mathcal{K}$ and then transform the entailment problem, i.e.,

$R, D, \text{ans}(t) \models Q \iff R, D, \text{ans}(t) \models Q'.$

(15)

The answer atoms in the elements of $Q'$ will be affected by the mgu's of the rewriting process but the answer variables will never be replaced by an existential variable because of condition 1 in the definition of general (disjunctive) rewriting step.

The entailment of a UCQ a can be transformed into the entailment of a UCQ (14). However, the presence of answer atoms in CQa’s will create rules with answer atoms in their body that may produce rewritings with more than one occurrence of answer atoms. Because the set of facts can only contain one answer atom, a rewriting with more than one occurrence of answer atoms $\text{ans}(X_1), \ldots, \text{ans}(X_n), B'$ can only be entailed in case there is an mgu for $\{\text{ans}(X_1), \ldots, \text{ans}(X_n)\}$. A UCQ-rewriting of a query with answer variables is deterministic if the CQs in it do not contain more than one occurrence of answer atoms. Rewritings without answer atoms correspond to rewritings of the negated constraints. They allow us to check the consistency of the data $D$ with respect to the rules in our knowledge base. However, the query answering problem does not make much sense when one of these rewritings is entailed by the data.

Algorithm 1 needs to be modified in order to avoid unnecessary rewritings of queries with answer variables. In Algorithms 2 and 3 we need to modify the functions $\text{rew}/2$ and $\text{rew}^\lor/2$ that compute one-step rewritings so that they only give rewritings with no more than one answer atom. More specifically, the resulting CQs $B$ (rules $B \rightarrow H$) with more than one answer atom (i.e., $\{\text{ans}(X_1), \ldots, \text{ans}(X_n)\} \subseteq B$) are replaced by the CQ $B\theta$ (rules $B\theta \rightarrow H\theta$) where $\theta = \text{mgu}(\{\text{ans}(X_1), \ldots, \text{ans}(X_n)\})$. We call these modified functions deterministic one-step rewriting functions. Algorithm 1 with deterministic one-step rewriting functions computes a deterministic UCQ-rewriting that is complete based on the fact that the answer predicate is fresh, i.e., it is not used in the knowledge base. Using deterministic one-step rewriting functions helps with the termination of the rewriting process.
Example 3.5. Consider a knowledge base without rules and the following UCQA:

\[ Q = \left[ (\text{ans}(X, Z), r(X, Y), r(Y, Z), \neg r(X, Z)), (\text{ans}(X, a), r(X, a)) \right] \]

The query \( Q \) has infinitely many CQ-rewritings of the following form:

\[ \text{ans}(X, a), \text{ans}(X, X_1), \ldots, \text{ans}(X, X_{n-1}), r(X, X_a), r(X_n, X_{n-1}), \ldots, r(X_1, a) \]

However, there is a finite and complete deterministic UCQ-rewriting of \( Q \):

\[ \left[ (\text{ans}(X, a), r(X, X_2), r(X_3, a), r(a, a)), (\text{ans}(X, a), r(X, X_1), r(X_1, a)), (\text{ans}(X, a), r(X, a)) \right] \]

Answer variables play a different role when splitting CSFs on connected components. The super cardinality of a CSF of atoms \( F \) with answer variables \( X \) is the number of non-answer variables in it: \( \text{card}_s(F) = |\text{vars}(F) \setminus \text{vars}(X)| \). We say that two non-answer variables \( u \) and \( v \) in a CSF of atoms \( F \) are super connected if they belong to the same atom, or if there is another non-answer variable \( z \) in \( F \) that is super connected to both \( u \) and \( v \).

A CSF \( F \) is super connected if all the atoms in it contain non-answer variables super connected to each other. Atoms containing only constants or answer variables in their arguments are super connected formulas with a super cardinality of zero.

A CSF \( F \) can be partitioned into a set \( \{ U_1, \ldots, U_n \} \) of super connected components such that if \( u \in \text{vars}(U_i) \) is super connected to \( u \in \text{vars}(U_j) \), then \( i = j \). The super connected cardinality of \( F \) is defined as the maximum super cardinality of the super connected components in the partition of \( F \) and denoted as \( \text{card}_s^*(F) = \max_i(\text{card}_s(U_i)) \). The super connected cardinality of a DSF \( [F_1, \ldots, F_m] \) is the maximum super connected cardinality of the formulas \( F_i \), i.e., \( \text{card}_s^*(F_1, \ldots, F_m) = \max_i(\text{card}_s^*(F_i)) \).

Lemma 3.1. Let \( K \) be a knowledge base and \( F \) a CSF of atoms with answer atom \( \text{ans}(X) \) partitioned into the super connected components \( \{ U_1, \ldots, U_n \} \). Then,

\[ K, \text{ans}(t) \vdash F \iff K, \text{ans}(t) \vdash \text{ans}(X), U_i \quad \text{for every } U_i \]  \hspace{1cm} (16)

Proof. It follows directly from Lemma 2.1 after transforming (16) into:

\[ K \vdash F\theta \iff K \vdash U_i\theta \quad \text{for every } U_i \neq \text{ans}(X), \]  \hspace{1cm} (17)

where \( \theta = \text{mgu}(\text{ans}(t), \text{ans}(X)) \). Note that \( \theta \) replaces the answer variables by constants and the resulting connected components will be the same as the super connected components.

Lemma 3.2. Let \( k \) be a natural number. There are a finite number of equivalence classes of CSFs of atoms with super connected cardinality of at most \( k \) that can be constructed using a finite set of predicates and a finite set of constants.

Proof. Straightforward using Lemma 3.1 and similar arguments to the ones used in the proof of Lemma 2.2.

Given a set of rules \( R \) without disjunctive existential rules (i.e., \( R^\exists = \emptyset \)), using reduction (14) we can focus on the disjunctive rules that are obtained from the negation of the CQa-s:

\[ R \vdash Q \iff (R^\exists, \neg Q^\exists, \neg Q^\exists^0) \vdash \neg R^\exists, Q^\exists^0, \]  \hspace{1cm} (18)

and study when Algorithm 1 with deterministic one-step rewriting functions terminates.

If all the variables in the frontier of the CQa-s are also answer variables, then the corresponding disjunctive existential rules will act similarly to disconnected rules if we use deterministic one-step rewriting functions on Algorithm 1.

Theorem 3.8. Let \( R \) be a set of rules without disjunctive existential rules and \( Q \) a UCQA. If all the variables in the frontier of the CQa-s in \( Q \) are also answer variables and \( R^\exists \) is a fus, then Algorithm 1 with deterministic one-step rewriting functions, applied on the rules \( R \cup Q^{-1} \cup \neg Q^{-\#} \) and UCQ \( Q^{-0} \), terminates for any value of \( k \).

Proof. The variables that appear in negated atoms of the CQ-s will end up being the variables in the head of the corresponding rules in \( (\neg Q^{-\#})_{>0} \). However, the variables that only appear in the negated atoms will be translated to existential variables and only answer variables are going to be frontier variables of the corresponding rules. Therefore, every new existential rule will have the frontier variables included in the set of answer variables.
In the presence of deterministic one-step rewriting functions, existential rules with all the frontier included in the set of answer variables (R_a) add atoms to the rewritings that do not share non-answer variables with the remaining part of the query. This ensures that when they are combined with a fus, the super connected cardinality of the UCQ-rewritings will be bounded. More specifically for any UCQ Q',

\[
\text{card}_s^a(\text{rewrite}_3^a (R^3 \cup R_a, Q')) \leq \max (\text{card}_s^a(\text{rewrite}_3^a(R^3, B)), \text{card}_s^a(\text{rewrite}_3^a(R^3, Q'))),
\]

where B is the body of rules in R_a.

Using Lemma 3.2 we can also affirm that there are finitely many rewritings that can be obtained. Therefore, every existential rule that is generated from the negated CQ's in (Q^\neg)_i>0 together with the rules in R^3 will yield a finite and complete deterministic UCQ-rewriting. Thus, we can ensure the termination of every iteration of the loop in Algorithm 1 for any value of k.

Likewise, every existential rule that is generated from the negated CQ's in (Q^\neg)_i>0 will have a body with a bounded super connected cardinality. Thus, we cannot generate infinitely many existential rules from the CQ's in (Q^\neg)_i>0 (Lemma 3.2), which ensures the termination of Algorithm 1 with deterministic one-step rewriting functions for any value of k.

Based on Theorem 3.4, we can also define other restrictions on UCQa’s that ensure that the rewriting algorithm stops. As in the case of existential rules, we say that a CQ (CQ^\neg) is linear if it contains only one positive literal. Similarly, a UCQ (UCQ^\neg) is linear if all the CQs (UCQ^\neg)'s in it are also linear. A CQa (UCQa)” is linear if it contains only one positive literal in the body. Likewise, a UCQa (UCQa)” is linear if all the UCQas (UCQa)”s in it are also linear.

**Theorem 3.9.** Let R be a set rules without disjunctive existential rules and Q a UCQa”. If R is a set of linear rules and Q a linear UCQa”, then Algorithm 1 with deterministic one-step rewriting functions, applied on the rules R \cup Q^{-1} \cup \neg Q^{-\#}, and UCQ Q^{-0}, stops for any value of k.

**Proof.** Because Q is linear the corresponding disjunctive rules (\neg Q^{-\#})_{i>1} will have two atoms in the body and one of them will be an answer atom. The UCQs will have one atom in the body and the negated constraints will only have one atom. Therefore, the deterministic one-step rewriting functions ensure that the new rules generated from disjunctive rules will contain only two atoms in the body and one of them will be an answer atom. There is a finite number of rules that can be generated with two atoms in the body and a decreasing number of disjoints in the head.

The deterministic CQ-rewritings produced by linear existential rules in R and the existential rules generated from (\neg Q^{-\#})_{i>0} will have a maximum of two atoms. Thus, there are finitely many deterministic CQ-rewritings that can be generated.

We conclude that Algorithm 2 terminates every time it is called in the main loop and also that the condition to continue executing the loop at some point will not hold because there are a finite number of rules and CQs rewritings that can be generated. Consequently, Algorithm 1 with deterministic one-step rewriting functions stops.

Finally, if there is a finite and complete deterministic UCQ-rewriting of a UCQa” Q with respect to a set of rules R that does not contain disjunctive existential rules, we can ensure that Algorithm 1 with deterministic one-step rewriting functions stops.

**Theorem 3.10.** Let R be a set rules without disjunctive existential rules and Q a UCQa”. If there is a finite and complete deterministic UCQ-rewriting of Q with respect to R, then Algorithm 1 with deterministic one-step rewriting functions, applied on the rules R \cup Q^{-1} \cup \neg Q^{-\#}, and UCQ Q^{-0}, terminates for any finite value of k.

**Proof.** Deterministic one-step rewriting functions will only discard rewritings that are not deterministic. The algorithm produces rewritings using all possible deterministic one-step rewritings and this ensures that if there is a finite deterministic UCQ rewriting Q_f, then after a finite number of iterations (with a finite value of k) a rewriting equivalent to Q_f will be generated. The completeness of Q_f ensures us that any further rewriting of Q_f using existential rules and deterministic one-step rewriting functions will not produce new deterministic CQs, i.e., the condition Q = Q_{old} holds for the rest of the iterations in the loop. Consequently, after finitely many iterations we also reach the condition R = R_{old}. Therefore, Algorithm 1 terminates because after a finite number of iterations the condition to continue iterating on the loop will not hold.
4. Implementation and Experiments

Completo\(^1\) is a query rewriting system that focuses on answering UCQ’s in the framework of existential rules. The first version of Completo [22] answers CQ\(^-\) using a resolution-based approach to eliminate negated atoms. The algorithm proposed is complete only for a very restricted type of queries.

In the second version of the system [23], queries with only one negated atom are answered by being transformed into rules. The approach is complete but terminated is guaranteed only when the resulting set of rules is a fus.

The current version of Completo implements Algorithm 1 with deterministic one-step rewriting functions and answers queries with answer variables that have an arbitrary number of negated atoms. Algorithm 1 can be seen as a generalization of both algorithms proposed in [22, 23]. Indeed, queries with one negated atom are transformed into rules, while the rewriting defined for disjunctive rules is similar to what was presented in [22] as constraint resolution. Furthermore, we take advantage of the termination results for knowledge bases consisting of a fus and CQ’s whose frontier is part of the answer variables of the query (Theorem 3.8), as well as for knowledge bases consisting only of linear elements (Theorem 3.9). Choosing \(k = \infty\) allows the rewriting with respect to existential rules to be performed by an external rewriter if the are no answer variables in the queries.

4.1. Experiments

To the best of our knowledge there is no other system that produces UCQ-rewritings for UCQ’s with universally quantified negation. Therefore, the experiments were performed in order to get a general idea of the performance of Completo producing UCQ-rewritings. We used an 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 and have been used in previous research papers based on queries with negation [22, 23]. The first is the Lehigh University Benchmark (LUBM) ontology [17], enriched with 70 additional disjoint classes axioms added for the atomic sibling classes, i.e., for classes asserted to share the same super-class. Secondly, we used the TRAVEL ontology\(^2\) that has 10 disjoint class axioms. The OWL 2 ER [5] fragment of both ontologies was translated into existential rules. We were not able to prove the fus property for the set of existential rules obtained from neither of the two ontologies we used.

We also prepared a query file with 500 CQ’s for each ontology which we used to let Completo produce finite UCQ-rewritings of the UCQ’s that contains all the queries in the file and also for each separated CQ’s. The queries contain 3 atoms and 2 of them are negated. The queries have one variable in the frontier which is also the answer variable of the query. We generated the queries by performing Association Rule Mining [2] on a dataset obtained from the assertions of the ontologies. The queries and the ontologies we used are publicly available \(^3\).

Table 3 shows the size of the UCQ-rewriting (rew) for the UCQ’s containing all the CQ’s in the file and the minimum (min), mean and maximum (max) statistics for the rewriting of each individual CQ’s in the file. The table also shows the time (time) in seconds and the RAM memory (mem) 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’s or individual CQ’s.

<table>
<thead>
<tr>
<th>Ontology</th>
<th>Info</th>
<th>rew</th>
<th>time</th>
<th>mem</th>
</tr>
</thead>
<tbody>
<tr>
<td>LUBM</td>
<td>UCQ(^-)</td>
<td>77</td>
<td>6193.12</td>
<td>2138</td>
</tr>
<tr>
<td></td>
<td>min</td>
<td>0</td>
<td>104</td>
<td>1129</td>
</tr>
<tr>
<td></td>
<td>mean</td>
<td>4</td>
<td>205.58</td>
<td>2069</td>
</tr>
<tr>
<td></td>
<td>max</td>
<td>55</td>
<td>466</td>
<td>2237</td>
</tr>
<tr>
<td>TRAVEL</td>
<td>UCQ(^-)</td>
<td>18</td>
<td>264.96</td>
<td>2043</td>
</tr>
<tr>
<td></td>
<td>min</td>
<td>0</td>
<td>1</td>
<td>123</td>
</tr>
<tr>
<td></td>
<td>mean</td>
<td>2</td>
<td>2.12</td>
<td>143</td>
</tr>
<tr>
<td></td>
<td>max</td>
<td>76</td>
<td>8</td>
<td>920</td>
</tr>
</tbody>
</table>

\(^1\)http://image.ntua.gr/~gardero/completo3.0/
\(^2\)https://protege.stanford.edu/ontologies/travel.owl
\(^3\)http://image.ntua.gr/~gardero/completo3.0/ontologies/
writing for an individual CQs\(^{-}\). The time that it took to compute the rewriting of the UCQa\(^{-}\) is the time that it takes on average to rewrite 30 individual CQa\(^{-}\)s (less than 2 hours). The RAM memory used to write the UCQa\(^{-}\) is less than the RAM memory that was used for rewriting the individual CQa\(^{-}\) that consumed the most RAM memory.

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

Figures 3 and 4 show information about the UCQ-rewriting size. In both ontologies at least 85% of the queries have zero rewritings. In this cases, the CQ-rewritings of the CQa\(^{-}\)s are subsumed by the CQ-rewritings of the negative constraints of the DKB.

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

Figures 7 and 8 show the correlation matrix with different performance parameters for the TRAVEL and LUBM ontologies. In order to get an idea of the rewriting process we computed the RAM memory used by the system (mem), the time that it takes to compute the UCQ-rewriting (time), the size of the UCQ-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 size of the UCQ-rewriting 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 greater than or equal to 0.94.

**Example 4.1.** One of the queries for the TRAVEL ontology was:

\[
\text{ans}(X) : = \neg \text{Capital}(X), \neg \text{Town}(X), \text{Destination}(X).
\]

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

\[
\begin{align*}
\text{ans}(X) &:= \text{Farmland}(X), \\
\text{ans}(X) &:= \text{NationalPark}(X), \\
\text{ans}(X) &:= \text{RuralArea}(X) 
\end{align*}
\]

Considering the above interpretation of the query, the answer tells us that only farmlands, national parks and rural areas cannot be town or capital destinations.

5. Conclusions

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

To ensure the completeness of our rewriting approach, we introduced a special case of first-order logic resolution (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 resolution completeness theorem also holds, allowing constraint resolution to be used in refutation procedures for FOL formulas.

Based on the definition of constraint resolution we proposed 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 produces an existential rule or a conjunctive query. The rules generated from disjunctive rules are then used in order to find additional rewritings of the initial conjunctive query rewriting. The proposed algorithm can be used for general knowledge bases with disjunctive existential rules; it terminates for the cases where there is a finite and complete UCQ-rewriting of the input queries with respect to the (disjunctive) existential rules and the negative constraints. However, there are rather strong conditions that are able to ensure the existence of a finite and complete UCQ-rewriting.

Moreover, we studied some of the sufficient conditions that ensure that the proposed algorithm terminates. One case requires atomic CQs and all elements of the knowledge base to be linear. The other case, requires a fus and disconnected disjunctive existential rules without imposing restrictions on the input CQs or the constraints. Both cases impose very strong conditions on the disjunctive existential rules. However, for knowledge bases without disjunctive existential rules, we were able to provide finite and complete deterministic UCQ-rewritings for UCQa’s with at most one positive atom (with respect to linear existential rules and linear constraints) and for UCQa’s that include the frontier in the answer variables (with respect to a fus). Both types of UCQa’s are very expressive.

Using the proposed algorithm and taking advantage of the stopping criteria, we implemented a sound and complete rewriting approach for unions 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.

The experimental results showed that the implementation is able to provide UCQ-rewritings in reasonable time and using a reasonable amount of RAM memory. Also, rewriting UCQ’s with a large number 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 studying other sufficient conditions that ensure the termination of our rewriting algorithm.

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


