Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών

UCQ-rewritings for Disjunctive Knowledge and Queries with Negated Atoms

Διδακτορική διατριβή

του

ENRIQUE MATOS ALFONSO

Επιβλέπων: Γεώργιος Στάμου
Καθηγητής ΕΜΠ

Αθήνα, Μάιος 2024

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών

UCQ-rewritings for Disjunctive Knowledge and Queries with Negated Atoms

Διδακτορική διατριβή

του

ENRIQUE MATOS ALFONSO

Επιβλέπων: Γεώργιος Στάμου
Καθηγητής ΕΜΠ

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 14 Μαΐου 2024.

.....	.....	.....
Γεώργιος Στάμου	Στέφανος Κόλλιας	Ανδρέας-Γεώργιος Σταφυλοπάτης
Καθηγητής ΕΜΠ	Ομ. Καθηγητής ΕΜΠ	Καθηγητής ΕΜΠ
.....	.....	.....
Μανώλης Κουμπαράκης	Αριστείδης Παγουρτζής	Δημήτρης Φωτάκης
Καθηγητής ΕΜΠ	Καθηγητής ΕΜΠ	Καθηγητής ΕΜΠ

.....
Αθανάσιος Βουλόδημος
Επίκουρος Καθηγητής ΕΜΠ

Αθήνα, Μάιος 2024

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών

.....
Enrique Matos Alfonso

Διδάκτωρ Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

14 Μαΐου 2024

Abstract

In this thesis, the problem of query rewriting for conjunctive queries with negation with respect to disjunctive existential rules is studied. Query rewriting is a well-known approach for query answering on knowledge bases with incomplete data. Three rewriting techniques are proposed that find UCQ-rewritings for queries with negation on the (disjunctive) existential rules framework.

The first technique uses resolution with respect to constraints in order to eliminate the negated atoms from the input query. A set of conjunctive queries is produced and it can be rewritten with the existential rules in the knowledge base producing more UCQ-rewritings.

The second technique focuses on finding a UCQ-rewriting of queries with not more than one negated atom. Such queries are transformed into existential rules and the constraints or the knowledge base are rewritten. The method keeps track of the rewritings produced by the rule corresponding to the transformed query and yields a complete UCQ-rewriting.

The third rewriting technique uses 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.

The proposed techniques are implemented in the COMPLETO system (v1, v2, and v3 respectively) and some experiments were carried out to evaluate the viability of the proposed solutions.

We also address the undecidability of the existence of a finite and complete UCQ-rewriting and the identification of finite unification sets (*fus*) of rules. We introduce new rule classes, *connected linear rules* and *connected domain restricted rules*, that exhibit the *fus* property for existential rules. Additionally, we propose *disconnected disjunction* for disjunctive existential rules to achieve the *fus* property when we extend the introduced rule fragments to disjunctive existential rules.

Finally, we present ECOMPLETO, a version of our system implemented in Elixir programming language, capable of handling UCQ$^\neg$s with universally quantified negation and disjunctive existential rules. Our experiments demonstrate ECOMPLETO's consistent ability to produce finite UCQ-rewritings, and describe the performance on different ontologies and queries.

Keywords

Disjunctive Existential Rules, Queries with Universally Quantified Negation, Backward Chaining, Query Rewriting.

to all my educators

Ευχαριστίες

The work presented in this thesis explores very interesting theoretical problems related to very practical solutions. I want to thank all my colleagues at the AILS lab for their support, especially my supervisor Giorgos Stamou, whose guidance and help with bureaucratic matters made this work possible.

During this research, many colleagues kindly reviewed my drafts, helping me better communicate my findings. I want to thank Alexandros Chortaras, Stathis Delivorias, Manos Thanos, and Michael Giazitzoglou for their help.

I would also like to thank every Greek person who made me feel at home in this country.

Lastly, a big thank you to my parents for their support and belief in me, and to my wife and baby for keeping me motivated to finish this chapter.

Αθήνα, Μάιος 2024

Enrique Matos Alfonso

Περιεχόμενα

Κατάλογος Σχημάτων

Κατάλογος Πινάκων

Chapter 1

Introduction

Rules are very important elements in knowledge-based systems and incomplete databases [1]; 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 [2] applies the rules on the facts in order to produce new facts. On the other hand, the backward chaining approach [3] 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*

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

and the information that

$$\mathit{parent}(\mathit{ana}, \mathit{maria}) \wedge \mathit{parent}(\mathit{maria}, \mathit{julieta}).$$

Traditional database systems would fail to entail that the query $Q = \exists Z \, \mathit{grand\text{-}parent}(\mathit{ana}, Z)$ 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 $\mathit{grand\text{-}parent}(\mathit{ana}, \mathit{julieta})$ 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.,

$$Q' = \exists Z \exists Y \, \mathit{parent}(\mathit{ana}, Z) \wedge \mathit{parent}(Z, Y).$$

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 [2], and may keep 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 [4] 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 restriction on the expressivity of the rules or having negated atoms on the query makes the query entailment problem undecidable [5].

We focus on the entailment problem of conjunctive queries with negated atoms ($\mathsf{CQ}^{\neg}$) 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.,

$$\forall X \exists Y \mathit{is\text{-}parent}(X) \rightarrow \begin{array}{l} \mathit{father}(X, Y) \vee \\ \mathit{mother}(X, Y), \end{array}$$

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 [5, 6]. To the best of our knowledge, existing research on existential rules with disjunction is only based on forward chaining algorithms [7, 6] or Disjunctive Datalog rewriting [8, 9, 10, 11].

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

$$\exists X \forall Y \forall Z \mathit{person}(X) \wedge \begin{array}{l} \neg \mathit{married}(X, Y) \wedge \\ \neg \mathit{parent}(X, Z) \end{array}$$

describes when the following rule does not hold

$$\forall X \exists Y \exists Z \mathit{person}(X) \rightarrow \begin{array}{l} \mathit{married}(X, Y) \vee \\ \mathit{parent}(X, Z). \end{array}$$

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* [12, 13, 14], i.e., variables that are only present in negated atoms are universally quantified.

Example 1.2. *Suppose that a machine learning algorithm is able to infer with 90% accuracy that every person in a ontology is married. The rule* $\forall X, \exists Y \mathit{person}(X) \rightarrow \mathit{MarriedTo}(X, Y)$ *could be a good candidate to enrich the knowledge we have. However, to avoid inconsistencies we need to ensure that the corresponding query* $\exists X, \forall Y \mathit{person}(X), \neg \mathit{MarriedTo}(X, Y)$ *cannot be entailed by the ontology. We could easily notice that such ontology to be consistent cannot have minors, or priests because it is of common knowledge that they can not be married. Therefore, to modify the ontology in a consistent way we need to add the rule together with the restrictions that will ensure that the resulting ontology is used only in a consistent way.*

The entailment problem for $\mathsf{CQ}^{\neg}$ is undecidable even for very simple types of rules [15, 12]. On the other hand, the use of guarded negation in queries is proven to be decidable over frontier-guarded existential rules [16]. Yet, the existing query rewriting-based approaches in the literature [17, 18, 19] 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, 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 $\mathsf{UCQ}^{\neg}$.

1.1 Related Work

Rosati studies query answering with respect to *description logics* (DLs) [12] and with respect to relational databases with *integrity constraints* (ICs) [13]. 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 [3], 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 [20], Bourhis et al. present a study of the complexity of query answering with respect to guarded disjunctive existential rules. The problem is 2EXPTIME-hard even for the simplest guarded-based class of disjunctive existential rules. Different types of UCQs are also considered by the authors in order to reduce the complexity of the problem. However, the considered query languages do not have a positive impact on the complexity of the problem. The only significant decrease in the complexity was found for atomic queries and linear disjunctive existential rules. The authors proved that 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 problem 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 [20] can be used as a generic tool to study the complexity of query answering for several fragments of description logics.

Some other research papers [8, 9, 10, 11] focus on the transformation of the query answering problem with respect to guarded (disjunctive) existential rules into a query answering problem with respect to (disjunctive) Datalog programs by getting rid of existential variables. In [8], Ahmetaj

et al. provide a transformation of the problem that yields a polynomial size (disjunctive) Datalog program when the maximal number of variables in the (disjunctive) existential rules is bounded by a constant. The translations proposed by other authors [9, 10, 11] are of exponential size.

1.2 Outline of our contributions.

We present an approach that focuses on the elimination of negated atoms in queries by applying resolution of controlled length with respect to the constraints existing in the knowledge base. After this process of elimination we obtain a set of queries containing answers of the initial query and classical rewriting algorithms can then be applied. Furthermore, we prove that the method yields a set of queries containing all the answers when the expressivity of the initial query is restricted. Additionally, some experiments were conducted to compare our approach to the existing one for rewriting negated concepts.

We introduce the concept of *Horn conjunctive queries* ($\mathsf{CQ}^{\neg^h}$), a type of conjunctive queries that allows the negation of one atom and corresponds semantically to the negation of a rule on the Existential Rules framework. We reduce the entailment problem of such queries, with respect to a set of Existential Rules, to the entailment of conjunctive queries without negated atoms. Restricted forms of Existential Rules where such reduction ensures decidability of the entailment problem are also presented. Additionally, the proposed approach is implemented into COMPLETO [18] allowing us to perform query rewriting and query answering for Horn UCQs. The implementation is compared to other systems that use *tableaux* algorithms to find the answers of OWL concepts that allow the use of negation. The experimental results show that the system we implemented performs better than the other systems that were considered for the case of ontologies with a large number of ABox assertions.

We introduce a restricted 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 constraint 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 propose an algorithm to compute a UCQ-rewriting for an input UCQ with respect to (disjunctive) existential rules and constraints. The algorithm can also compute UCQ-rewritings for conjunctive queries with universally 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) existential rules and the negative constraints. We also present two theorems with sufficient conditions for the termination 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). We propose two new classes of existential rules that are generalizations of *domain restricted* rules [21] and *linear* rules [21]. Both class of rules are finite unification sets.

Additionally, we consider unions of conjunctive queries with negated atoms and answer variables (denoted as $\mathsf{UCQa}^{\neg}$) with respect to existential rules and constraints. The proposed algorithm is modified in order 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 $\mathsf{UCQa}^{\neg}$ with respect to the existential rules and constraints of the knowledge base. We also present two theorems with sufficient conditions for the termination of the modified rewriting algorithm. 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 existential 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.

Part I

Preliminaries

Chapter 2

Preliminaries

2.1 First Order Logic Resolution

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 a high-level reasoning processes can be tracked down to sequences of resolution steps that ensure its correctness. We also describe the framework of disjunctive existential rules and present the definition of conjunctive queries with negated atoms.

2.1.1 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 [22] 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.,* $F_1 \wedge \ldots \wedge F_n$ $(F_1 \vee \ldots \vee F_n)$.

For a given set of formulas $\{F_1, \ldots, F_n\}$*, a* CSF *containing these formulas 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 $(A \wedge B) \vee D$.

Set operators can then be used to combine set formulas of the same type and obtain a new set formula. Moreover, equivalent elements of a set formula 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:

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

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 $A \setminus B \models F \cup \neg B$; likewise $A \models F$ if and only if $A \cup \neg C \models F \setminus C$.

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* $\bar{l}$ 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 $vars(F)$. 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 θ 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 θ the substitution $\{X_1 \leftarrow Y_1, \ldots X_n \leftarrow Y_n\}$. We say θ 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_i is

either equal to some X_j in θ, or Y_i does not occur in F. The *composition* of two substitutions θ and σ 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 θ and after σ). A substitution θ is *more general than* another substitution σ if there exists another substitution γ such that $\sigma = \theta\gamma$.

A substitution θ is a *unifier* for a set of expressions $S = \{F_1, \ldots, F_n\}$ iff $F_1\theta = F_2\theta, \ldots, F_{n-1}\theta = F_n\theta$. 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 $mgu(S)$, 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 *mgu*, they are unique up to variable renaming.

A *hypergraph* is a tuple $\langle A, E\rangle$, 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, i.e., $card(F) = |vars(F)|$. The *width* of a CSF of atoms F (denoted by $width(F)$) is the number atoms that have variables in their arguments. Two variables u and v in a CSF of atoms F are *connected* iff they both belong to the same atom ($\exists A \in F | \{v, u\} \subseteq vars(A)$), or if there is another variable z in F that is connected to both u and v.

A CSF of atoms F is *connected* if all the atoms in it contain variables and all the variables are connected to each other. An atom that has only constants in its arguments is a connected formula that is not connected to any other atom and has a cardinality and a width of zero. It is represented by an empty hypergraph. The constants in the formula play no role in their hypergraph representation.

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

Example 2.3. *The* CSF

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

is represented by the hypergraph in Figure 2.1. We can split F in two connected components

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

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

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

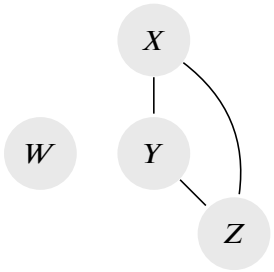


Figure 2.1: Hypergraph corresponding to a CSF.

connected CSF*s. Then,*

$$G \vDash F \;\; \textit{iff} \;\;\; G \vDash U_i \textit{ for every } \; U_i.$$

Proof. A detailed proof is given by Tessaris [23]. 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* CSF*s 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 substitution. 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. □

Lemma 2.3. *Let k be a natural number. There are a finite number of equivalence classes of* CSF*s of atoms with connected width of at most k that can be constructed using a finite set of predicates and constants.*

Proof. It is a direct consequence of Lemma 2.2 because a bound for the connected width of a CSF implies that there is also a bound for the connected cardinality. □

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* $\perp$. A formula F is in conjunctive normal form (CNF) if it is a CSF of clauses, i.e,

$$F = [l^1_1, \ldots, l^1_{n_1}], \ldots, [l^m_1, \ldots, l^m_{n_m}].$$

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

Definition 2.2 ((Binary) Resolution Rule)**.** *Let C_1 and C_2 be two clauses with no variables in common, and let $l_1 \in C_1$ and $l_2 \in C_2$ be complementary literals with respect to a most general unifier $\sigma = mgu(\{l_1, \bar{l}_2\})$. The* binary resolvent *of C_1 and C_2 with respect to the literals l_1 and l_2 is the clause:*

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

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

It is easy to show that resolution is sound, i.e., $C_1, C_2 \models C_1 \cup_r C_2$. 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 Σ be a set of clauses and C a clause. A* (resolution) derivation *of C from Σ is a finite sequence of clauses $R_1, \ldots, R_k = C$, such that each R_i is either in Σ, or a resolvent of two clauses in $\{R_1, ..., R_{i-1}\}$. If such a derivation exists, we write $\Sigma \models_r C$, and say that C can be derived from Σ. A derivation of the empty clause $\bot$ from Σ is called a* refutation *of Σ. The steps of a resolution derivation are the resolution operations performed to obtain the resolvents in the sequence.*

Sometimes it is useful to know which clauses were used 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 Σ be a set of clauses and C a clause such that $\Sigma \models_r C$. A* derivation (refutation) graph *of C from Σ is a directed graph where the nodes are the clauses from the derivation $\Sigma \models_r C$, 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 Σ. 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 2.2).

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

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

A clause logically implies any instance of it, possibly extended with more literals. This follows directly from the properties of the disjunction operator and the universal quantification.

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

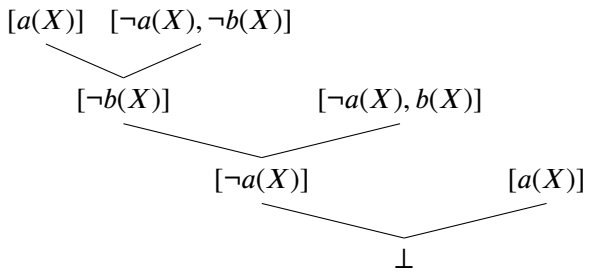


Figure 2.2: A derivation tree.

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

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 [22].

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 Σ be a set of clauses, and Σ' a set of ground instances of clauses from Σ. Suppose $R_1', ..., R_k'$ is a derivation of the clause R_k' from Σ'. Then there exists a derivation $R_1, ..., R_k$ of the clause R_k from Σ, such that R_i' is an instance of R_i, for each $i \in \{1, \ldots, k\}$.*

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 Σ be a set of clauses, and C a clause. Then $\Sigma \models C$ iff $\Sigma \models_r^d C$.*

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

Performing resolution steps in a *breath-first* manner ensures 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:*

$$(C_1 \cup_r C_2) \cup_r C_3 \equiv (C_1 \cup_r C_3) \cup_r (C_2 \cup_r C_3). \tag{2.1}$$

Note that on right-hand side of (2.1) 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 2.4. *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* (2.1) *we have the following resolution steps:*

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

and on the right-hand side

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

Clearly, $[c(X)] \equiv [c(X)]$.

2.2 Disjunctive Existential Rules 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 $\mathbf{X}$) are existentially quantified, i.e., an expression of the form $\exists \mathbf{X}\ l_1, \ldots, l_n$. Queries that allow negation in the literals l_i are called *conjunctive queries with negation* ($\mathsf{CQ}^\neg$). All the variables $\mathbf{X}$ that appear in the positive literals of a $\mathsf{CQ}^\neg$ are assumed to be existentially quantified. In order to avoid domain dependant queries we use *universally quantified negation* [12, 14, 13], i.e., all the variables $\mathbf{Z}$ that appear only in negative literals are assumed to be universally quantified:

$\exists \mathbf{X} \forall \mathbf{Z}\ l_1, \ldots, l_n$. Because the variable quantification rules are straightforward we omit quantifiers, e.g., instead of $\exists X \forall Y\, person(X), \neg married(X, Y)$ we write $person(X), \neg married(X, Y)$. 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* ($\mathsf{UCQ}^\neg$)). For a $\mathsf{UCQ}^\neg$ $\mathcal{Q}$, by $\mathcal{Q}^{\neg k}$ we denote the set of $\mathsf{CQ}^\neg$s in $\mathcal{Q}$ that contain exactly k negated atoms, and by $\mathcal{Q}^{\neg \#}$ the set of $\mathsf{CQ}^\neg$s in $\mathcal{Q}$ that contain two or more negated atoms. We use the term *query* to refer to either a CQ, $\mathsf{CQ}^\neg$, UCQ or $\mathsf{UCQ}^\neg$.

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

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

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 $\mathbf{X} = vars(B)$ contains the variables that appear in the body, and they are universally quantified. On the other hand, $\mathbf{Y} = vars(H) \setminus vars(B)$ are the variables the 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: $vars(B) \cap vars(H)$. 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 $F \not\models Q$ and $F, r \models Q$. A rule r_i depends on a rule r_j iff the body of r_i depends on r_j. The concept of rule dependencies allows us to define the *graph of rule dependencies* (GRD) [24], 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) $\mathcal{K} = \langle \mathcal{R}, \mathcal{D} \rangle$ is composed by a CSF $\mathcal{R}$ of rules and a CSF of facts $\mathcal{D}$. For a given set of rules $\mathcal{R}$, by $\mathcal{R}^\bot$ we denote the set of constraints in $\mathcal{R}$, by $\mathcal{R}^\exists$ the set of existential rules and by $\mathcal{R}^\vee$ the set of disjunctive existential rules. A knowledge base $\langle \mathcal{R}, \mathcal{D} \rangle$ is a *disjunctive* knowledge base (DKB) if $\mathcal{R}^\vee \neq \emptyset$, otherwise it is an *existential* knowledge base (EKB).

Example 2.5 (Disjunctive Knowledge Base)**.** *Let us define an example* DKB *about family relationships.*

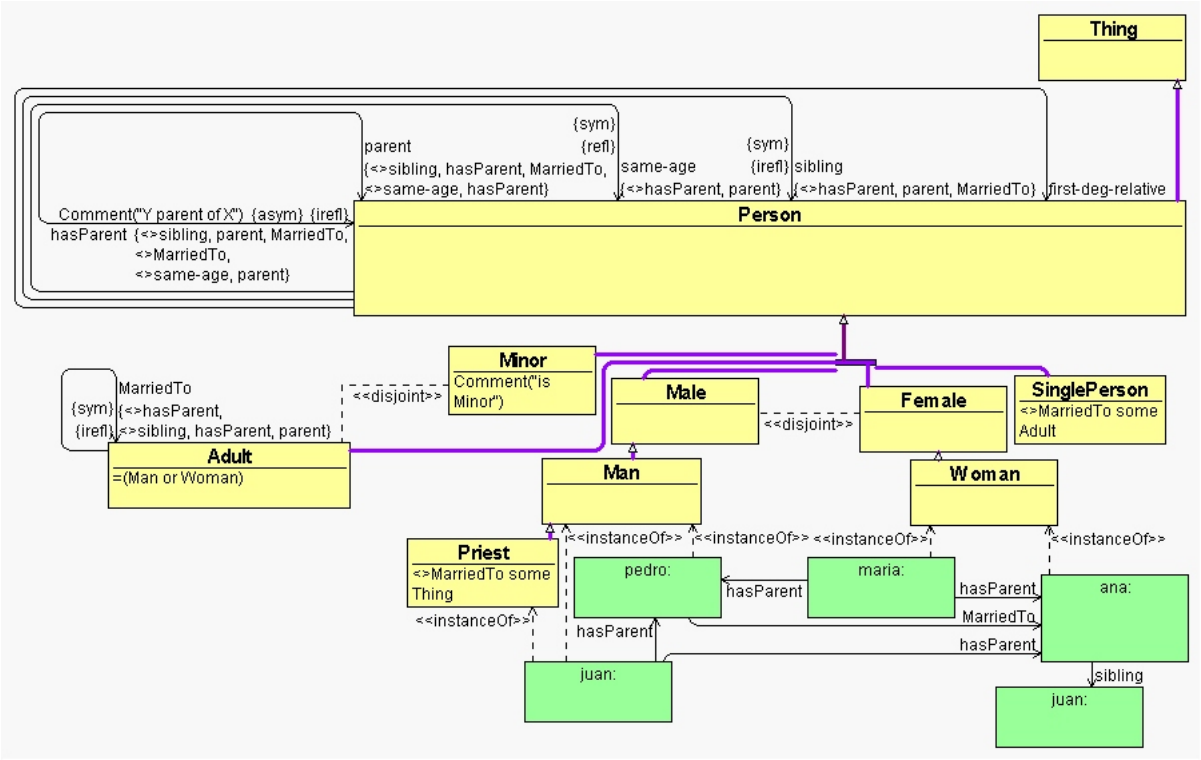


Figure 2.3: UML diagram of an ontology example generated with OWLGrEd.

- *Facts:*

$$\begin{aligned} &(parent(Y, ana), parent(Y, jane)), \\ &sibling(ana, juan) \end{aligned} \tag{2.2}$$

- *Existential rules:*

$$\begin{aligned} &(sibling(X, Y) \rightarrow \quad sibling(Y, X)), \\ &(sibling(X, Y) \rightarrow \quad parent(Z, X), parent(Z, Y)) \end{aligned} \tag{2.3}$$

- *Negative constraints:*

$$(sibling(X, Y), parent(X, Y)), \tag{2.4}$$
$$(same\text{-}age(X, Y), parent(X, Y)), \tag{2.5}$$
$$(parent(X, Y), parent(Y, X)), \tag{2.6}$$
$$(parent(X, X)) \tag{2.7}$$

- *Disjunctive existential rules :*

$$first\text{-}deg\text{-}relative(X, Y) \rightarrow [parent(X, Y), parent(Y, X), sibling(X, Y)] \tag{2.8}$$

Note that the existential rule (2.3) *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* (**??**)

states that the predicate sibling/2 is symmetric. We could also add symmetry for same-age/2. Fact (2.2) *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* (2.4) *and also of a person of the same age* (2.5)*. Additionally, with the negative constraints we express that parent/2 is asymmetric* (2.6) *and irreflexive* (2.7)*. Finally, the disjunctive existential rule* (2.8) *defines the relation that represents the first degree relative concept [25]: 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 $\langle \mathcal{R}, \mathcal{D} \rangle$:

$$\mathcal{R}, \mathcal{D} \models_? Q. \tag{2.9}$$

In particular, we solve the entailment problem (2.9) by reducing it to the entailment of a UCQ Q' with respect to the set of facts $\mathcal{D}$, i.e., to the problem

$$\mathcal{D} \models_? Q'.$$

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

$$\mathcal{D} \models Q' \text{ implies } \mathcal{R}, \mathcal{D} \models Q. \tag{2.10}$$

The CQs in Q' are called CQ-*rewritings* of Q with respect to $\mathcal{R}$. If the converse of (2.10)

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

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 [3] 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.

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

Given a knowledge base $\mathcal{K}$, a tuple $\mathbf{t}$ of constants in $\mathcal{K}$ is a certain answer of a query with answer variables Q with respect to $\mathcal{K}$ iff $\mathcal{K}, ans(\mathbf{t}) \models Q$. The set of certain answers of a query

Q with respect to a knowledge base $\mathcal{K}$ is denoted by $cert(Q, \mathcal{K})$. Computing the set $cert(Q, \mathcal{K})$ is known as the *query answering problem*.

Example 2.6. *Consider the following three* CQa*s, which have different sets of answer tuples:*

$$Q_1 = ans_1() :- sibling(X, Y),$$
$$Q_2 = ans_2(X) :- sibling(X, Y)$$
$$Q_3 = ans_3(X, Y) :- sibling(X, Y).$$

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 2.5, with a different set of facts:

$$sibling(pedro, ana)$$
$$sibling(juan, Y).$$

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:

$$\begin{aligned} cert(Q_1, \mathcal{K}) &= \{()\} \\ cert(Q_2, \mathcal{K}) &= \{pedro, ana, juan\} \\ cert(Q_3, \mathcal{K}) &= \{\langle pedro, ana\rangle, \langle ana, pedro\rangle\}. \end{aligned}$$

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

$$\mathcal{K}, ans_2(pedro) \models ans_2(X), sibling(X, Y).$$

Part II

Proposed Methods

Chapter 3

Rewriting Queries with negated Atoms

3.1 Constraint Saturation

In this section, we focus on the entailment of $\mathsf{CQ}^\neg$s with respect to existential knowledge bases. we introduce a method that removes the negated atoms from $\mathsf{CQ}^\neg$s and yields UCQ that can be rewritten using existing algorithms in order to find a complete UCQ-rewriting of the $\mathsf{CQ}^\neg$.

Example 3.7. *Lets consider the following* EKB *and the clauses C_i corresponding to the rules and constraints:*

$$\mathcal{R}^\exists = \{a(x) \to r(x,y)\} \qquad C_1 = [\neg a(x), r(x, f(x))],$$
$$\mathcal{R}^\bot = \{b(x), r(x,y) \to \bot\} \qquad C_2 = [\neg b(x), \neg r(x,y)],$$

and a query with its corresponding clause.

$$q = s(x,y), \neg a(x) \qquad C_3 = [\neg s(x,y), a(x)].$$

With the initial clauses we can do the following resolution derivation:

$$\begin{aligned} & C_1,\ C_2,\ C_3, \\ C_4 = [\neg s(x,y), r(x,f(x))] \quad & = C_1 \cup_r C_3, \\ C_5 = [\neg s(x,y), \neg b(x)] \quad & = C_4 \cup_r C_2, \end{aligned}$$

which basically means that the last clause is a consequence of the initial clauses (Correctness of Resolution). i.e.

$$\mathcal{R}, \mathcal{D} \models \exists x \exists y\ s(x,y), b(x) \to \exists x \exists y\ s(x,y), \neg a(x),$$

for any sets of facts $\mathcal{D}$.

Therefore, we can say that $s(x,y), b(x)$ is a CQ*-rewriting of our initial query. Yet, it cannot be obtained applying a classical rewriting algorithm on the initial query because it contains a negated atom. On the other hand, it is well known that resolution for FOL is semi-decidable and arbitrary length resolution derivations should be avoided. However, if we rewrite the constraint in the* KB *we get :*

$$b(x), a(x) \to \bot \qquad C_6 = [\neg b(x), \neg a(x)].$$

Now we can apply one step resolution between the clause corresponding to q (C_3) and C_6 and we

obtain again the clause corresponding to q'. Note that at this point if we have more rules in our system, the query q' could be rewritten using a classical algorithm and all the rewritings of q' will also be rewritings of q.

Generalizing the process described in the previous example, we could define *constraint saturation* as the process of eliminating negative atoms from the queries by using constraints to make resolution derivations of controlled length. We can achieve that by only allowing resolution steps using clauses corresponding to constraints.

Definition 3.7. *For a* $\mathsf{CQ}^\neg$ *q and a set of constraints $\mathcal{R}^\perp$, the* constraint saturation *$sat(q, \mathcal{R}^\perp)$ is the most general set of conjunctive queries $\{\dots, q_i, \dots\}$ with no negative atoms that can be obtained from a linear resolution derivation $C_0, C_1, \dots, C_k$, where (i) C_0 is the clause corresponding to $\neg q$, (ii) C_k is the clause corresponding to $\neg q_i$ and (iii) The set of side clauses used are the ones corresponding to the constraints $C_1, \dots, C_{|\mathcal{C}|}$ in $\mathcal{C}$ together with C_0.*

Lemma 3.4. *Let q be a* $\mathsf{CQ}^\neg$ *and $\mathcal{R}$ the rules from a* EKB*. The constraint saturation $sat(q, \mathcal{R}^\perp)$ of a query q with respect to $\mathcal{R}^\perp$ is a* UCQ*-rewriting of q with respect to R.*

Proof. Indeed, the elements $q_i \in sat(q, \mathcal{R}^\perp)$ are built using a linear resolution derivation starting with a clause corresponding to $\neg q$ and using as side clauses the clauses corresponding to the constraints $\mathcal{C}$ of our EKB. We can affirm then that $\mathcal{R}^\perp \models q_i \rightarrow q$. Therefore, for all set of facts $\mathcal{D}$ we have that $\mathcal{D} \models q_i$ implies $\mathcal{D}, \mathcal{R} \models q$. □

In Example 3.7, the shape of the initial query is very important. To ensure the termination of the resolution involving the set of clauses corresponding to the rewritten constraints and the query we need to avoid cases in which the query clause clashes with itself and produces a resolvent that could contain different answers.

Example 3.8. *Lets consider the query*

$$q = B(x), s(x, y), \neg B(y) \qquad C_0 = [\neg B(x), \neg s(x, y), B(y)],$$

and the constraint:

$$\mathcal{C} = \{B(x), W(x) \rightarrow \perp\} \qquad C_1 = [\neg B(x), \neg W(x)].$$

Here since C_0 clashes with itself one can end up in a resolution derivation of unbounded length when trying to remove the negative atoms in q. Particularly, every query of the form:

$$q_n = B(x), s(x, x_1), \dots, s(x, x_n), W(x_n)$$

is a rewriting of the initial query.

A $\mathsf{CQ}^\neg$ is *disconnected* if the corresponding clause to $C = \neg q$ does not clash with itself or if the query q' corresponding to the resolvent $\neg(C \cup_r C)$ is less general i.e. $q \succeq q'$. Disconnected queries will prevent generating new answers based on applying resolution involving the clause corresponding to $\neg q$ and itself.

Even if at the beginning our query is disconnected a rewriting of it could end up resolving with itself and such cases should also be avoided. In general, we say a $\mathsf{CQ}^{\neg}$ q is *strongly disconnected* with respect to a set of rules $\mathcal{R}$ if it is disconnected and all the queries corresponding to a clause resulting from a resolution derivation of $\neg q$ using clauses of $\mathcal{R}$ are also disconnected.

Clearly, the resolution derivations in Definition 3.7 have a bounded length for strongly disconnected queries.

Lemma 3.5. *In Definition 3.7 the length of the resolution derivation is bounded by m for a strongly disconnected query with m negative atoms.*

Indeed, every resolution step is performing resolution using a clause corresponding to a constraint, with all the atoms negated on it. Therefore, each of those steps removes one or more positive atoms on the clause corresponding to the negated query, but those positive atoms in C_0 correspond to the negated atoms in q and there are only m of them. Moreover, since the query is strongly disconnected possible steps applying resolution of the query with itself are discarded.

Theorem 3.6. *Let $\mathcal{R}$ be a set of non disjunctive rules and q a strongly disconnected query with respect to $\mathcal{R}$. If $\mathcal{R}^{\perp}$ contains all the possible rewritings of the queries corresponding to the constraints in it, then a complete* UCQ*-rewriting of the constraint saturation $sat(q, \mathcal{R}^{\perp})$ is also a complete* UCQ*-rewriting of q.*

Proof. Lemma 3.4 ensures that a complete rewriting of $sat(q, \mathcal{R}^{\perp})$ is a rewriting of q with respect to $\mathcal{R}$.

Then, we need to focus on proving that for all $\mathcal{D}$ we have that $\mathcal{R}, \mathcal{D} \models q$ implies that there is a $q_i \in \neg\mathcal{R}^{\perp}$ such that $\mathcal{R}^{\exists}, \mathcal{D} \models q_i$.

The proof is based on showing that a resolution derivation starting in $\neg q$ and ending in the empty clause can be rearranged using the commutativity property of the resolution so that the first resolution steps are applied using constraints from $\mathcal{C}$. After those steps we can affirm that the clause will correspond to a query in the constraint saturation of q, so there will also be a resolution derivation starting from a clause corresponding to a query in $sat(q, \mathcal{R}^{\perp})$ and ending in the empty clause i.e. $\mathcal{R}, \mathcal{D} \models q$ implies $\mathcal{R}, \mathcal{D} \models sat(q, \mathcal{R}^{\perp})$.

Suppose that:

$$\mathcal{R}, \mathcal{D} \models q \tag{3.1}$$

and

$$\mathcal{R}, \mathcal{D} \not\models sat(q, \mathcal{R}^{\perp}). \tag{3.2}$$

We can affirm using (3.1) that there is a linear resolution derivation starting with the clause $C_0 = \neg q$, ending in the empty clause $\perp$ $(C_0, C_1, \ldots, C_{m'}, \perp)$ that uses the clauses corresponding to $\mathcal{R}, \mathcal{D}$ and $\neg q$.

The only way to reach the empty clause is by performing resolution with respect to the clauses corresponding to the facts $\mathcal{D}$ that have the following shape: $[a(\mathbf{t}')]$. They decrease the size of the clauses in the linear resolution derivation. The initial clause has also positive atoms (corresponding to the negated atoms in the original query) and the only way to get rid of them is by doing resolution with respect to one of the literals in a clause that corresponds to a constraint. Such step will

decrease the number of positive atoms in the clause of the derivation and probably will introduce more negative atoms. Since our query has m negated atoms, in our derivation we need at least m resolution steps to get rid of the positive literals.

If we take a closer look at those resolution steps $i_1, \dots, i_{m'}$ involving constraint clauses we can try to reorganize them so that they are performed as early as possible. In case the constraint clause used at step i_j can resolve with the initial clause C_0 we can perform the step on position 1 and shift the other steps of the linear derivation. The resulting clause at step i_j will be the same.

On the other hand, if some resolution steps with clauses corresponding to rules need to be applied before we could apply a certain resolution step with a constraint clause, then based on the commutativity property we could re-arrange those resolution steps by applying resolution first to the constraint clause and then the resulting clause can be used to apply resolution to the initial clause, resulting in the same clause at step i_j. Note that this modification turns the linear derivation into a tree but the shape of the derivation is not relevant. The clause that we obtain by applying resolution between the constraint clause and the rules will have only negative literals. So it will be equivalent to a constraint C' that can be deduced by the knowledge base i.e. $\mathcal{R}, \mathcal{C} \models C'$. But since $\mathcal{C}$ contains all its rewritings, we know that there will be a constraint $C'' \in \mathcal{C}$ such that $C'' \geq C'$. In that case, the query corresponding to C' is a rewriting of the query corresponding to C''.

In case C'' clashes with C_0, instead of using C' in the resolution derivation we could use C'' which is a side clause that can resolve with the initial query and such step can again be inserted at position 1. On the other hand, if C'' does not clashes with C_0 it means that the resolution with C' yields a query that is also less general than C''. Therefore, the resulting rewriting will have answers when a constraint of the system is violated i.e. the resulting rewriting would be inconsistent.

The other possible resolution step involving a C_0 would be to resolve it with itself but our query is strongly disconnected with respect to the knowledge base as hypothesis to avoid those cases.

We will end up in a derivation with at least m initial resolution steps with respect to constraint clauses. After those steps we will have a clause that needs to correspond to a query that is less general than one of the queries in the constraints saturation $sat(q, \mathcal{R}^{\perp})$. It implies that also there is a resolution derivation that ends in the empty clause and starts with a clause corresponding to the negation of a query in $sat(q, \mathcal{R}^{\perp})$ i.e. $\mathcal{R}, \mathcal{D} \models sat(q, \mathcal{R}^{\perp})$. which contradicts (3.2) and proves our theorem.

□

Theorem (3.6) allows us to find a set of CQ that after rewriting them in a complete way, will contain all the rewritings of the original query q and then we can apply existing rewriting algorithms to them in order to find a complete UCQ rewriting of q with respect to $\mathcal{R}$.

To check whether a query is strongly disconnected or not would involve performing resolution therefore it is not an easy condition to check before starting the rewriting of the query. Furthermore, if we use a rewriting algorithm as a black box it will be impossible to check while rewriting the query. Yet, if we implement the rewriting algorithm using the method proposed in [3] we could focus on checking if the atoms introduced in every rewriting step could at some point clash with some of the negated atoms that the query had originally. At the end of the process, if we find out that the query is strongly disconnected, we will know for sure that the resulting UCQ is equivalent to the initial query.

Depending on the rules we have, there are maybe some atoms that when negated are not strongly disconnected (Example 3.9) and so any other query containing them in negated form will neither be.

Example 3.9. *Lets consider the simple query*

$$q = \neg P(x) \qquad C_0 = [P(x)],$$

and the rules from an EKB*:*

$$\begin{aligned} P(x), B(x) \rightarrow P'(x) \qquad & C_1 = [\neg P(x), \neg B(x), P'(x)], \\ P'(x), s(x,y) \rightarrow B(y) \qquad & C_2 = [\neg P'(x), \neg s(x,y), B(y)], \\ \mathcal{C} = \{B(x), W(x) \rightarrow \bot\} \qquad & C_3 = [\neg B(x), \neg W(x)]. \end{aligned}$$

After computing $((C_0 \cup_r C_1) \cup_r C_2)$ *we will reach again the clause equivalent to the query in Example 3.8:*

$$\begin{aligned} q' = B(x), \neg P'(x) \qquad & C_4 = (C_0 \cup_r C_1) = [\neg B(x), P'(x)], \\ q'' = B(x), s(x,y), \neg B(y) \qquad & (C_4 \cup_r C_2) = [\neg B(x), \neg s(x,y), B(y)], \end{aligned}$$

and we end up again in a resolution derivation of unbounded length when trying to remove the negative atoms. Therefore, no query that contains $\neg P(x)$ *will be strongly disconnected with respect to this knowledge base.*

A negated atom $\neg p(\mathbf{x})$ is *self disconnected* with respect to $\mathcal{R}$ if the query $q = \neg p(\mathbf{x})$ is strongly disconnected w.r.t. $\mathcal{R}$. Notice that the only resolution steps that would probably produce non disconnected queries are those that make resolution with a clause corresponding to a rule and the head of the rule unifies with another atom belonging to the body of another rule (or the same one) used previously in the resolution derivation. In case our rules contain only one atom in the body (known as *Atomic-hypothesis* or *linear* rules [21]), resolution steps with rules will not introduce other body atoms.

Property 3.1. *Negated atoms are self disconnected with respect to an Atomic-hypothesis set of rules.*

Proof. Indeed, the initial clause corresponding to $q = \neg p(\mathbf{x})$ cannot yield another clause with more than one atom by applying resolution with clauses corresponding to rules. □

3.2 Algorithm for Rewriting Conjunctive Queries with Negation

For the design of the algorithm to rewrite negative conjunctive queries we use the help of another algorithm that rewrites conjunctive queries into UCQs. We can refer to it as `rewrite-ext` : $R \times \mathsf{CQ} \rightarrow \mathsf{UCQ}$ and it can be any of the state of the art rewriters (RAPID[26], SYSNAME[27], GRAAL [28]) compatible with the theory we support for finding the constraints saturation of a conjunctive query with negation.

To illustrate better the general idea of the algorithms presented, we treat data structures in a very simple way. The method `pop` when applied to a set, returns one of its elements and also removes

it from the set. The function `cover/1` computes the set of most general conjunctive queries given a set of conjunctive queries.

3.2.1 Preprocessing

```
function computeConstraints(R)
    expand := R^⊥
    while expand.size > 0 do
       c := expand.pop
       C := C ∪ rewrite-ext(R,c)
    end for
    return cover(C)
end function
```

Algorithm 1: Function to rewrite the initial set of constraints.

For a fixed set of non-disjunctive rules $\mathcal{R}$ we initially perform the computation of the rewritings of each one of the initial constraints (Algorithm 1). Then, as long as the existential rules or constraints do not change, the same computed rewritings of the constraints can be used to perform the constraint saturation for different queries.

3.2.2 The Algorithm

```
function rewrite-completo(R, q)
    C := computeConstraints(R)
    qCsat := Csaturation(C,q)
    ucq := {}
    while qCsat.size > 0 do
       q' := qCsat.pop
       ucq := ucq ∪ rewrite-ext(R,q')
    end for
    return cover(ucq)
end function
```

Algorithm 2: Main algorithm to rewrite queries with negated atoms.

Algorithm 2 is the main algorithm for rewriting queries with negated atoms. It computes the rewriting of the constraints (if it is not already computed for that knowledge base). The function `Csaturation` (Algorithm 3) takes the clause corresponding to $\neg q$ and performs linear resolution with respect to the clauses corresponding to the constraints in $\mathcal{R}^{\perp}$ to obtain the constraint saturation $sat(q, \mathcal{R}^{\perp})$. The derivations on each step remove at least one of the negated atoms in q by performing a resolution step (`resolve/3`) with a derivation of the original query q'' and a constraint c from $\mathcal{R}^{\perp}$. For each constraint, we check all possible clashing sets with the queries in order to explore all possible linear derivations starting in $\neg q$ and ending in a clause representing a query without negative atoms. The initial query is transformed to a clause (`queryToClause/1`) by creating a set with the complement of each of the atoms in the query. In the resolution derivation, when the

clause generated has no positive atoms we can affirm that it represents a query with only positive atoms therefore we convert it back to a query (`clauseToQuery/1`), by taking the complement of the atoms in the clause.

```
function Csaturation(C, q)
    ucq := ∅
    cq := queryToClause(q)
    expand := {cq}
    while expand.size > 0 do
      q′ := expand.pop
      foreach c in C do
        foreach clashing set l in c and q′ do
          q″ := resolve(l,c,q′)
          if (q″ has no positive atom)
            ucq := ucq ∪ clauseToQuery(q″)
          else
            expand := expand ∪ {q″}
          end if
        end for
      end for
      end while
    return cover(ucq)
end function
```

Algorithm 3: Function to compute the constraints saturation of a query.

With each of the queries in the UCQ resulting from the main algorithm we can perform query answering, by trying to unify the atoms of the query with atoms in the database $\mathcal{D}$. In case the original query had some answer variables the atom $ans(\mathbf{x})$ would be part of each of the queries in the rewriting, but it will not unify with atoms in $\mathcal{D}$. As a result of the unification of the atoms in the query and the atoms in $\mathcal{D}$ (if possible and in all possible ways), the variables in the *ans* predicate will be replaced by constants and we will end up with tuples t that will be part of the answers of our query. For each query, finding an answer tuple is an NP-Complete process, yet there are potentially an exponential number of tuples that can be answers of the query.

In the process of constraints saturation, we have to avoid the queries that remove answer variables from the atoms of the query in the process of resolution to avoid ending up in domain dependent queries with answer variables that do not belong to the atoms of the query. Yet we only need to pay attention to it when we try to answer queries without safe negation.

Chapter 4

Horn Conjunctive Queries

The problem of deciding whether a query q is entailed, for a given set of non disjunctive rules $\mathcal{R}$ and data $\mathcal{D}$:

$$\mathcal{R}, \mathcal{D} \models q \tag{4.1}$$

can be transformed into an equivalent problem by changing the hypotheses and consequences of the equation:

$$\mathcal{R}^{\exists}, \mathcal{D}, \neg q \models \neg \mathcal{R}^{\bot}. \tag{4.2}$$

In Eq. (4.2), the term $\neg\mathcal{R}^{\bot}$ is a union of BCQ $\{\dots q_i \dots\}$ with Boolean queries q_i corresponding to the bodies of the constrains in $\mathcal{R}^{\bot}$, i.e.,

$$q_i = a_1(\mathbf{x_1}), \dots, a_n(\mathbf{x_n})$$

for each constraint

$$a_1(\mathbf{x_1}), \dots, a_n(\mathbf{x_n}) \rightarrow \bot \in \mathcal{R}^{\bot}.$$

On the other hand, $\neg q$ depends on the shape of q but if it contains only one negated atom it will be translated into a rule, i.e., $\neg(\exists\mathbf{X}\forall\mathbf{Y}, a_1(\mathbf{x_1}), \dots, a_n(\mathbf{x_n}), \neg p(\mathbf{y}))$ is transformed into

$$r_q = \forall\mathbf{X}\exists\mathbf{Y} a_1(\mathbf{x_1}), \dots, a_n(\mathbf{x_n}) \rightarrow p(\mathbf{y}).$$

Conjunctive queries with a negated atom are called *Horn Conjunctive Queries* ($\mathsf{CQ}^{\neg h}$).

Consequently, in order to find the answers of a Horn query we can rewrite the bodies of the constraints by using an additional rule r_q corresponding to q.

The rule r_q remains inside the framework of Existential Rules, with the variables $\mathbf{Y}$ that are not present in positive literals as existential variables. On the other hand, for Horn $\mathsf{CQ}^{\neg g}$ or even Horn $\mathsf{CQ}^{\neg s}$ we can ensure that r_q will not have any existential variables. Additionally, when we perform UCQ rewriting with the new set of rules, it is convenient to keep track of when the rule r_q was applied in order to identify the rewritings that were obtained using that rule. Thus, we can add a dummy atom $q()$ to the body of r_q. The queries with the predicate q in their atoms will be obtained by applying at least one rewriting step using r_q. Notice that the predicate q cannot be present in the knowledge base in order to avoid changing the semantics of Eq. (4.1).

The resulting set of rules $\mathcal{R} \cup \{r_q\}$ also needs to be a *fus* in order to guarantee the existence of a

UCQ-rewriting of the bodies of the constraints in the knowledge base. Therefore, by rewriting the bodies of the constraints we cannot ensure that all Horn queries can be answered. Nevertheless, depending on the rules in $\mathcal{R}$ and r_q we can check some of the sufficient conditions ensuring that the resulting set of rules is also a *fus*.

The final UCQ rewriting of the bodies of the constraints will contain (i) queries obtained using only rules from $\mathcal{R}^{\exists}$ and (ii) queries that were obtained by applying r_q at least once. The queries from (i) will allow us to express the inconsistencies of the original knowledge base and queries from (ii) will contain the rewritings of q (removing the dummy atoms).

The intuition behind Eq. (4.2) is that Eq. (4.1) will be satisfied when the application of the rule r_q introduces inconsistencies with respect to the constraints of the system.

Performing Forward Chaining with the resulting set of rules $\mathcal{R} \cup \{r_q\}$ could also be used to answer the bodies of the constraints in $\mathcal{C}$. However, there needs to be a way to tell apart the inconsistencies introduced by using r_q and those that do not depend on the presence of r_q. Basically, we have two problems to solve:

$$\mathcal{R}^{\exists}, \mathcal{D} \models \neg\mathcal{R}^{\bot} \text{ (consistency of the knowledge base)}$$

and

$$\mathcal{R}^{\exists} \cup \{r_q\}, \mathcal{D} \models \neg\mathcal{R}^{\bot} \text{ (consistency of the knowledge base if the new rule is added).}$$

Forward Chaining algorithms could also be modified in order to flag the facts that can only be obtained due to an application of r_q. We will then be interested if one of the flagged atoms is involved in triggering a constraint in $\mathcal{C}$. This would avoid computing the forward chaining saturation of both knowledge bases, considering that one of them is a subset of the other.

4.1 Union of Horn CQs

For conjunctive queries without negated atoms, the entailment of a UCQ $Q = [q_1, \dots, q_n]$ is a matter of considering the entailment problem of each query q_i separately and then taking the disjunction of the results:

$$\mathcal{K} \models [q_1, \dots, q_n] \quad \text{iff}$$

$$\mathcal{K} \models q_1 \text{ or } \dots \text{ or } \mathcal{K} \models q_i \text{ or } \dots \text{ or } \mathcal{K} \models q_n.$$

In the presence of a union of conjunctive queries with negation ($\text{UCQ}^{\neg}$), we can consider the reduction in equation (5.1) and notice that it is possible to perform resolution between two clauses corresponding to negation of queries q_i. Therefore, considering entailment of each of the queries separately would not be a complete approach.

Example 4.10. *Consider an empty set of rules with a* $\text{UCQ}^{\neg}$*:*

$$\models [a(X), (b(Y), \neg a(Y))].$$

By reasoning with each of the queries separately we obtain:

$$\models a(X)$$

and

$$\models b(Y), \neg a(Y)$$

but we would not be able to infer that $b(X)$ is a rewriting of the initial $\mathsf{UCQ}^{\neg}$, i.e.,

$$\mathcal{D} \models b(Y)$$

implies

$$\mathcal{D} \models [a(X), (b(Y), \neg a(Y))],$$

but $b(Y) \not\models a(X)$ and $b(Y) \not\models b(Y), \neg a(Y)$.

A possible approach to follow can be to convert the Horn queries into rules and perform rewriting on the remaining queries plus the constraints of the system:

$$\mathcal{R}^{\exists}, \neg Q^{\neg 1} \models \neg \mathcal{R}^{\perp}, Q^{\neg 0}. \tag{4.3}$$

Union of conjunctive queries that contain either Horn conjunctive queries or conjunctive queries are called union of Horn conjunctive queries ($\mathsf{UCQ}^{\neg h}$).

Property 4.1. *For a finite unification set $\mathcal{R}$ and a $\mathsf{UCQ}^{\neg h}$ Q, if the set of rules $\neg Q^{\neg 1}$ is also a* fus *and together they define a directed cut i.e. $\mathcal{R} \rhd \neg Q^{\neg 1}$, then the entailment problem for Q is decidable for any set of constraints and facts.*

Proof. If we consider Prop. 5.1 and the reduction of the entailment problem in Eq. (4.3) we can see that the resulting set of rules $\mathcal{R} \cup \neg Q^{\neg 1}$ is a *fus* of existential rules.

□

Property 4.1 helps us knowing beforehand if the UCQ rewriting of the initial $\mathsf{UCQ}^{\neg h}$ exists. Besides checking the *fus* property for the set of rules corresponding to the Horn queries, we also need to make sure that the rules in $\mathcal{R}$ do not depend on the set of rules corresponding to the Horn queries. Nevertheless, if our Horn queries are translated to rules that have the same property of the rules in $\mathcal{R}$ i.e. $\mathcal{R} \cup \neg Q^{\neg 1}$ are linear rules or domain restricted rules, then we don't need to check the dependencies between both set of rules to ensure decidability.

4.2 Rewriting a Union of Horn CQs

The Algorithm 4 defines the general procedure to find the rewritings of a $\mathsf{UCQ}^{\neg h}$. The queries are rewritten using an external rewriting algorithm `rewrite_ext/2`. Finally, the rewritings are filtered (`to_rewritings/2`), selecting those that have the predicate 'ans' and also transformed into answers of the original queries in Q.

```
function rewrite(R, Q)
        ucq := {}
        ucq.addAll(Q^¬0)
        ucq.addAll(R^⊥)
        ucq := rewrite_ext(ucq, R^∃ ∪ ¬Q^¬1)
        ucq := filter_rewritings(ucq, 'ans')
        return ucq
end function
```

Algorithm 4: Function to rewrite Horn UCQs.

Example 4.11. *Using the existential rules and constraints from Example 2.5 we could rewrite* $ans(X) :- Person(X), \neg MarriedTo(X,Y)$ *and obtain the following* UCQ*-rewriting:*

$$\begin{aligned} ans(X) :- [&Priest(X)), \\ &Minor(X), \\ &SinglePerson(X)]. \end{aligned}$$

Notice that r_q *has an existential variable and it cannot produce rewritings from the following constraints:*

$$\begin{aligned} &MarriedTo(X,Y), hasParent(X,Y) \rightarrow \bot \\ &MarriedTo(X,X) \rightarrow \bot. \end{aligned}$$

On the other hand, if we include Y *as answer variable, i.e.,*

$$ans(X,Y) :- Person(X), Person(Y), \neg MarriedTo(X,Y),$$

we are asking for pairs of people that cannot be married. The rewriting has 81 queries but basically the approach is to introduce the rule

$$ans(X,Y), Person(X), Person(Y) \rightarrow MarriedTo(X,Y)$$

and to rewrite the constraints:

$$\begin{aligned} &MarriedTo(X,Y), Minor(X) \rightarrow \bot \\ &MarriedTo(X,Y), Priest(X) \rightarrow \bot \\ &MarriedTo(X,Y), SinglePerson(X) \rightarrow \bot \\ &MarriedTo(X,Y), hasParent(X,Y) \rightarrow \bot \\ &MarriedTo(X,X) \rightarrow \bot. \end{aligned}$$

Chapter 5

Backward Chaining for Disjunctive Knowledge and Queries With Negated Atoms

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 (2.9).

5.1 Constraint Resolution

The entailment problem (2.9) can be transformed into a consistency check problem

$$\mathcal{R}, \mathcal{D}, \neg\mathcal{Q} \models_? \bot, \tag{5.1}$$

which can then be solved using resolution refutation. Depending on the expressivity of the queries in $\mathcal{Q}$ 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 $\neg\mathcal{Q}$ 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 5.8 (Positive/Negative Charge)**.** *The* positive (negative) charge $|C|^+$ ($|C|^-$) *of a clause* C *is the number of positive (negative) literals in the clause.*

Table 5.1: Properties of different types of clauses in the disjunctive existential rules framework.

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

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 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 5.1 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 $\mathsf{CQ}^\neg$ may produce a FC, RC or DRC, depending on its positive and negative charge.

Example 5.12. *From the* DKB *of Example 2.5 we obtain the following clauses:*

- *Fact clauses:*

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

- *Rule clauses:*

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

- *Constraint clauses:*

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

- *Disjunctive rule clauses :*

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

Table 5.2 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

Table 5.2: Properties of the resolvent C_3 for different types of clauses C_1 and C_2.

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

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 step 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 CQ-rewritings since the resulting clause corresponds to the negation of a conjunctive query.

As we can see from Table 5.2, in order to get derivations that produce clauses with a non-increasing positive charge, we need to avoid resolution 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 5.2).

Definition 5.9 (Constraint Derivation)**.** *Let Σ be a set of clauses and C a clause. A resolution derivation (refutation) $\Sigma \vDash_r C$ of a clause $C(\bot)$ from Σ is a* constraint derivation *(*refutation*) iff all its resolution steps involve resolution with a constraint clause. A constraint derivation of a clause C from Σ is written as $\Sigma \vDash_c C$. Similarly, there is a* constraint deduction *of C from Σ, written as $\Sigma \vDash_c^d C$, if C is a tautology or if there is a clause D such that $\Sigma \vDash_c D$ and D subsumes C.*

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

Theorem 5.7 (Constraint Subsumption for Constraint Clause Consequences)**.** *Let Σ be a set of clauses and C a constraint clause. Then $\Sigma \vDash C$ iff $\Sigma \vDash_c^d C$.*

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 $\Sigma \vDash_r D$ 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 $\Sigma \vDash_r D$ for sure involves resolution steps with respect to some constraint clauses. Let C_{i_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: $C_{i_1} = \left(C_{i_0} \cup_r C_{j_0}\right)$. We can assume that on the path to the root of the tree the resolution steps involve always constraint clauses $C_{j_1}, \ldots, C_{j_k}$:

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

Therefore, we can apply the distributivity property if C_{j_1} clashes on literals coming from both clauses C_{i_0} and C_{j_0}, obtaining

$$\Big((C_{i_0} \cup_r C_{j_1}) \cup_r (C_{j_0} \cup_r C_{j_1})\Big) \ldots \cup_r C_{j_k} = D.$$

On the other hand, if C_{j_1} clashes on literals coming from only one of the clauses (we assume it is C_{i_0} without loss of generality) we apply the commutativity property to obtain

$$\Big((C_{i_0} \cup_r C_{j_1}) \cup_r C_{j_0}\Big) \ldots \cup_r C_{j_k} = D.$$

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

$$\begin{array}{l} \Big(C_{i_0} \cup_r C_{j'_1} \cup_r \ldots \cup_r C_{j'_{k'}}\Big) \cup_r \\ \Big(C_{j_0} \cup_r C_{j''_1} \cup_r \ldots \cup_r C_{j''_{k''}}\Big) \end{array} = D.$$

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

$$\left|\Big(C_{i_0} \cup_r C_{j'_1} \cup_r \ldots \cup_r C_{j'_{k'}}\Big)\right|^+ = 0$$

or

$$\left|\Big(C_{j_0} \cup_r C_{j''_1} \cup_r \ldots \cup_r C_{j''_{k''}}\Big)\right|^+ = 0.$$

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 5.8 (Completeness of Constraint Resolution Derivations)**.** *A set of clauses* Σ *is unsatisfiable iff there exists a constraint refutation of* Σ*, i.e.,* $\Sigma \vDash_c \bot$.

Proof. This follows from Theorem 5.7 by taking as a consequence the empty clause that has no positive literals and is only subsumed by itself. □

5.2 Unit Resolution

A *unit* clause is a clause with only one literal. *Unit* resolution steps involve at least one unit clause. Correspondingly, we also define unit derivations and refutations.

Definition 5.10 (Unit Derivation)**.** *Let* Σ *be a set of clauses and* C *a clause. A resolution derivation (refutation)* $\Sigma \vDash_r C$ *of a clause* $C(\bot)$ *from* Σ *is a* unit derivation *(*refutation*) iff all its resolution steps involve resolution with a unit clause. A unit derivation of a clause* C *from* Σ *is written as* $\Sigma \vDash_1 C$*. Similarly, there is a* constraint deduction *of* C *from* Σ*, written as* $\Sigma \vDash_1^d C$*, if* C *is a tautology or if there is a clause* D *such that* $\Sigma \vDash_1 D$ *and* D *subsumes* C.

Unit resolution is not refutation complete. A very simple propositional example suffices to show the incompleteness of unit refutations.

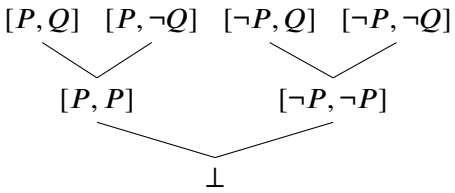


Figure 5.1: A refutation of S.

Example 5.13. *Let* $S = \{[P, Q], [P, \neg Q], [\neg P, Q], [\neg P, \neg Q]\}$. *Figure 5.1 shows a refutation of* S. *However,* S *contains no unit clause, i.e., there is no unit refutation of* S.

On the other hand, if Σ contains only one clause that is not a unit clause, the completeness of unit refutations can be easily shown.

Theorem 5.9 (Completeness of Unit Resolution Derivations)**.** *A set of clauses* Σ*, that contains only one clause that is not a unit clause, is unsatisfiable iff there exists a unit refutation of* Σ*, i.e.,* $\Sigma \vDash_1 \bot$.

Proof. Similar to the proof of Theorem 5.7. □

5.2.1 Query Containment

Given two $\mathsf{CQ}^\neg$s Q_1 and Q_2, the *query containment* problem is the problem of knowing if all the certain answers of Q_1 are included in the certain answers Q_2 ($Q_1 \subseteq Q_2$). Query containment can also be expressed as an entailment problem, e.g.,

$$Q_1 \vDash_? Q_2. \tag{5.2}$$

In Equation (5.2) all the clauses that correspond to Q_1 are unit clauses and Q_2 is represented by only one clause, i.e., the Skolemized version of $\neg Q_2$. Therefore, the containment of two queries Q_1 and Q_2 is equivalent to the problem of finding a unit refutation for the clauses corresponding to $Q_1 \wedge \neg Q_2$.

5.3 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 [3]. However, in most of the existing literature disjunctive rules are mainly used in a forward chaining manner [7, 6] or to perform Disjunctive Datalog rewritings [8, 9, 10, 11].

In Example 2.5, one could infer that two first-degree relatives that have the same age have to be siblings:

$$\begin{aligned} &\mathit{first\text{-}deg\text{-}relative}(X, Y), \\ &\qquad \mathit{same\text{-}age}(X, Y) \quad \rightarrow \mathit{sibling}(X, Y). \end{aligned} \tag{5.3}$$

This rule can be obtained by a constraint derivation using (2.8) and the clauses corresponding to:

$$(\textit{same-age}(X, Y), \textit{parent}(X, Y) \rightarrow \bot)$$
$$(\textit{same-age}(X, Y) \rightarrow \textit{same-age}(Y, X))$$

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 (5.3) can then be used in rewriting steps defined for existential rules.

A rewriting step as defined in the existential rules framework [3] corresponds to the resolution steps between RCs corresponding to an existential rule and a CC corresponding to the negation of a CQ.

Definition 5.11 (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 θ (i.e., $H'\theta = Q'\theta$) such that*

1. *if $v \in vars(Q \setminus Q')$ and $v \neq v\theta$, 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 $v\theta \notin vars(Q \setminus Q')$,*

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 RCs. Nevertheless, the resolution steps with such RCs are always performed together in order to avoid unnecessary propagation of Skolemized existential variables. Thus, the resulting clause cannot contain a Skolem term representing an existential variable of r. Hence, existential variables cannot be assigned to a variable that will be part of the result (condition 1 in Definition 5.11) nor should be replaced by a variable that belongs to the result (condition 2 in Definition 5.11).

Definition 5.11 is an adaptation to our framework of the *piece-based* rewriting step proposed in [3].

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 5.11 with the goal to support both existential rules and disjunctive existential rules.

Definition 5.12 (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 θ (i.e., $h'_1\theta = \ldots h'_n\theta = Q'\theta$) such that*

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

then $(B \cup (Q \setminus Q') \rightarrow H \setminus H')\theta$ 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 5.14. *Consider a disjunctive existential rule:*

$$r_1 = diabetesRisk(X) \rightarrow \begin{array}{l}[(diabetic(Y), \\ \quad sibling(Y, X)), \\ (diabetic(Z), \\ \quad parent(Z, X))].\end{array}$$

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

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

$$diabetesRisk(X), singleChild(X) \rightarrow \begin{array}{l}diabetic(Z), \\ parent(Z, X),\end{array}$$

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:*

$$\begin{array}{l}[(singleChild(X), sibling(Y, X)), \\ (diabetic(Y), parent(Y, X)), \\ (diabetesRisk(X), singleChild(X))].\end{array}$$

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

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

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.,

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

In a similar way, if $\mathcal{Q}$ is a $\text{UCQ}^{\neg}$, the entailment problem can be reduced to the entailment of a UCQ:

$$\begin{array}{l}\mathcal{R}^{\exists}, \mathcal{R}^{\vee}, \mathcal{R}^{\perp}, \mathcal{D} \models \mathcal{Q} \quad \text{iff} \\ (\mathcal{R}^{\exists}, \neg \mathcal{Q}^{\neg 1}), (\mathcal{R}^{\vee}, \neg \mathcal{Q}^{\neg \#}), \mathcal{D} \models \neg \mathcal{R}^{\perp}, \mathcal{Q}^{\neg 0},\end{array} \tag{5.4}$$

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

Theorem 5.10 (Soundness and Completeness of Rewritings)**.** *Let $\langle\mathcal{R},\mathcal{D}\rangle$ be a* DKB *and $\mathcal{Q}$ a* UCQ*. Then $\mathcal{R},\mathcal{D}\models\mathcal{Q}$ iff there is a rewriting $\langle\mathcal{R}',\mathcal{Q}'\rangle$ of $\left\langle(\mathcal{R},\mathcal{R}^{\vee}),(\mathcal{Q},\neg\mathcal{R}^{\perp})\right\rangle$ such that $\mathcal{D}\models\mathcal{Q}_i$ for some conjunctive query $\mathcal{Q}_i$ in $\mathcal{Q}'$.*

Proof. The k-step rewriting of $\left\langle(\mathcal{R},\mathcal{R}^{\vee}),(\mathcal{Q},\neg\mathcal{R}^{\perp})\right\rangle$ 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 5.8), this theorem also holds. □

```
function rewrite_k(R, Q)
  Q := Q ∪ ¬R^⊥
  R := R \ R^⊥
  do
    R_old := R
    Q_old := Q
    Q := rewrite_k^∃(R^∃, Q)
    R := rewrite^∨(R, Q)
  while (Q ≠ Q_old or R ≠ R_old)
  return Q
end function
```

Algorithm 5: Function to rewrite UCQs with respect to existential rules and disjunctive existential rules.

Given a set of rules $\mathcal{R}$ and a UCQ Q, function $\texttt{rewrite}_k/2$ presented in Algorithm 5 computes all the rewritings of $\left\langle(\mathcal{R},\mathcal{R}^{\vee}),(\mathcal{Q},\neg\mathcal{R}^{\perp})\right\rangle$. The algorithm alternates between computing the rewritings of CQs using existential rules ($\texttt{rewrite}_k^{\exists}/2$ presented in Algorithm 6) and computing the rewritings using disjunctive existential rules ($\texttt{rewrite}^{\vee}/2$ presented in Algorithm 7). New CQs are used to generate more rules, and new existential rules are used to generate more CQs until a fixed point is reached, i.e., until no new rule or query is produced.

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

Algorithm 6: Function to rewrite UCQs using existential rules.

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

Algorithm 7: Function to rewrite UCQs using disjunctive existential rules.

Function `rew`/2 (in Algorithm 6) computes the set of the results of all possible rewriting steps for all the combinations of existential rules and CQs in its arguments. This step is known as the *expansion* of a query, and it generates more conjunctive queries. On the other hand, function $\texttt{rew}^{\vee}$/2 (in Algorithm 7) 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 CQs in its arguments that do not yield a conjunctive query. This restriction does not affect completeness because a CQ Q' that can be generated from a CQ 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 r' ($r' \in \texttt{rew}^{\vee}(\{Q\}, \{r\})$), and then a rewriting step using r' generates the query Q' ($Q' \in \texttt{rew}(\{Q\}, \{r'\})$).

The `cover`/1 function (in Algorithm 6) computes, for a given UCQ Q, the minimal subset $Q' \subseteq Q$ such that for all $q \in Q$ there is a $q' \in Q'$ such that q' subsumes q, i.e., the corresponding clause to q' subsumes the corresponding clause to q.

The `cover`/1 function allows us to keep always the minimal set of CQs that can yield the same results. In [3], the authors perform a deeper analysis showing that using the cover computation on the rewriting algorithm they propose ensures that the resulting UCQ-rewriting will be of minimal size (cardinality).

In both $\texttt{rewrite}_k^{\exists}$/2 and $\texttt{rewrite}^{\vee}$/2, all newly generated CQs and rules are also expanded, unless some CQs are removed when computing the cover. The process stops when a fixed point is reached.

All CQs generated by Algorithm 5 are computed according to our definition of a rewriting; this ensures the correctness, i.e., every CQ that is generated is a CQ-rewriting of the input query with respect to the input sets of rules and constraints.

The rewriting function for disjunctive existential rules ($\texttt{rewrite}^{\vee}$/2) generates all the possible rules using an input UCQ rewriting. The fact that new rules have less disjunctive components in the head ensures that the output is always finite. Therefore, the completeness of the result of Algorithm 5 relies totally on the completeness of the result provided by Algorithm 6.

Algorithm 6 describes the rewriting function for existential rules ($\texttt{rewrite}_k^{\exists}$/2), and is based on the general rewriting algorithm proposed on [3]. It implements a breath-first expansion process where each iteration of the loop expands a new level of conjunctive queries. We have introduced the parameter k that allows us to control how many levels of CQs will be expanded and ensures

termination of each individual call to Algorithm 6 for $k \neq \infty$. However, the loop in Algorithm 5 will keep on calling Algorithm 6 as long as new CQs are generated, without affecting the completeness of the whole rewriting process. The parameter k defines a *pause* on the existential rules rewriting process in order to generate more rules from the disjunctive existential rules.

König et al. study the completeness of the UCQ-rewriting computed by the rewriting algorithm for existential rules based on different definitions of the query expansion function [3]. Our expansion function (`rew`/2) ensures that the computed UCQ-rewriting is complete because it corresponds to their piece-based rewriting operator that ensures the completeness. Moreover, if there is a finite and complete UCQ-rewriting of the input UCQ, the function $\texttt{rewrite}^{\exists}_{k}/2$ will find it after a finite number of calls to it.

5.4 Rewritable Queries and Disjunctive Knowledge Bases

The termination of Algorithm 5 depends on the termination of Algorithms 6 and 7. Algorithm 7 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 6 is denoted by $\texttt{rewrite}^{\exists}/2$ and corresponds to the classical rewriting algorithm for existential rules proposed in [3], whose termination is studied in [5]. 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 [5]. A set of existential rules that ensures the existence of a finite UCQ-rewriting for any UCQ is called a finite unification set (*fus*) [21]. There are some classes of existential rules that have the *fus* property:

1. *Linear* existential rules [21]: existential rules with one atom in the body.

2. *Disconnected* existential rules [29]: existential rules that do not share variables between the body and the head.

3. *Domain restricted* rules [21]: existential rules that each atom in the head contains none or all of the variables in the body.

4. *Acyclic graph of rule dependencies* (*aGRD*) [24]: existential rules that do not contain cycles in the *graph of rule dependencies*.

5. *Sticky* rules [30]: Each marked variable occurs at most once in a rule body. The marked variable set is built from a rule set using the following marking procedure: (i) for each rule r_i and for each variable v occurring in the body of r_i, if v does not occur in all atoms of the head of r_i, mark (each occurrence of) v in the body of r_i; (ii) apply until a fixpoint is reached: for each rule r_i, if a marked variable v appears at position $p[k]$ in the body of r_i, then for each rule r_j (including $i = j$) and for each variable x appearing at position $p[k]$ in the head of r_j, mark each occurrence of x in the body of r_j.

If a set $\mathcal{R}$ of existential rules is a *fus* and the new existential rules generated by Algorithm 7 are also a *fus*, combining them could yield a new set of existential rules that is not a *fus* [5]. Therefore,

we need stronger conditions to ensure that we always call Algorithm 6 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* ($\mathcal{R}_1 \rhd \mathcal{R}_2$) if none of rules in $\mathcal{R}_1$ depends on a rule of $\mathcal{R}_2$.

Property 5.1. *Let $\mathcal{R}$ be a set of existential rules with a directed cut $\mathcal{R}_1 \rhd \mathcal{R}_2$. For any* CQ *$Q$ and any set of facts $\mathcal{D}$ we have that*

$$\mathcal{D}, \mathcal{R} \models Q \text{ if there is a CQ } Q' \text{ such that } \mathcal{D}, \mathcal{R}_1 \models Q' \text{ and } Q', \mathcal{R}_2 \models Q.$$

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 [5]. □

Property 5.1 [5] allows us to study the decidability of entailment when we combine two sets of rules for which the entailment problem is decidable.

In Algorithm 7, 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 6. Therefore, we need ways to ensure that the total number of existential rules generated by Algorithm 7 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 5.11. *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 5 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 $\mathcal{R}^{\exists}$ 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 5 stops because at some point no new rules and no new queries can be generated. □

Theorem 5.11 is closely related to Theorem 7.13 and Lemma 7.12 proposed by Bourhis et al. in [20]. 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., $vars(B) \cap vars(H) = \emptyset$. 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 5.12. *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 $B \to H \in \mathcal{R}_2$ is bounded as follows:

$$\begin{aligned} card^*(\texttt{rew}(B \to H, \quad & Q)) \leq \\ & \max(card^*(B), card^*(Q)). \end{aligned}$$

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.,

$$\begin{aligned} card^*(\texttt{rewrite}^{\exists} \quad & (\mathcal{R}_1 \cup \mathcal{R}_2, Q)) \leq \\ \max(\quad & card^*(\texttt{rewrite}^{\exists}(\mathcal{R}_1, B)), \\ & card^*(\texttt{rewrite}^{\exists}(\mathcal{R}_1, Q))). \end{aligned}$$

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 5.13. *Let $\mathcal{R}$ be a set of rules and Q a* UCQ. *If $\mathcal{R}^{\exists}$ is a fus, and $\mathcal{R}^{\vee}$ a set of disconnected disjunctive existential rules, then Algorithm 5 stops for any set of constraints $\mathcal{R}^{\perp}$, any* UCQ *Q and for any value of k.*

Proof. The new existential rules produced by the function $\texttt{rewrite}^{\vee}$ are disconnected rules and they can be combined with the rules in $\mathcal{R}^{\exists}$ and yield a *fus* (follows from Theorem 5.12).

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

$$\bigcup_j B'_j \cup \bigcup_j Q'_j \to H',$$

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

Because $\mathcal{R}^{\exists}$ is a *fus* we have a finite number of rewritings B'_j and Q'_j. This ensures that there is only a finite number of different bodies B' for the generated existential rules. The number of different heads, H' is obviously finite too. Therefore, there is only a finite number of different existential rules that will eventually be generated by $\texttt{rewrite}^{\vee}$. Thus, Algorithm 5 stops for any value of k. □

5.4.1 Expanding the Existing Fragments

Domain restricted (*dr*) rules [21] are existential rules where all the atoms in the head contain none or all of the variables in the body of the rule. However, if we consider rules where the bodies can have more than one connected component, then the definition of *dr* rules can be generalized.

Definition 5.14 (Connected domain restricted rule)**.** *A rule is called* connected domain restricted *(cdr) rule if for every connected component $\mathbf{C}$ in the body of the rule and for every atom $\mathbf{h}$ in the head, $\mathbf{h}$ contains none or all the variables of $\mathbf{C}$.*

Example 5.15 (Common ancestor and six degrees of separation rules)**.** *In biology and genealogy, the* most recent common ancestor *(MRCA),* last common ancestor *(LCA), or* concestor *of a set of organisms is the most recent individual from which all the organisms of the set are descended. We could express a simpler rule stating that for every two organisms there exists a common ancestor:*

$$organism(X), organism(Y) \rightarrow organism(Z), ancestor(Z, X), ancestor(Z, Y)$$

The rule is obviously not domain restricted but it is connected domain restricted.

Another example of cdr rule that is not a dr is the six degrees of separation *rule. It describes the idea that all people are six, or fewer, social connections away from each other.*

$$\begin{aligned} person(X), person(Y) \rightarrow\ & knows(X, X_1), knows(X_1, X_2), knows(X_2, X_3), \\ & knows(X_3, X_4), knows(X_4, X_5), knows(X_5, Y) \end{aligned}$$

In the example rules we assume that the predicate ancestor/2 is irreflexive and antisymmetric, and knows/2 is reflexive and symmetric.

Atoms in the head of a *cdr* rule r contain all the variables of some (possibly none) connected components in the body of the rule. We can be sure that the rewritings of a CQ q with respect to r will not introduce new variables that are connected to the variables in the part of q that is not modified by the rewriting. Some new variables might be introduced but they will be in isolated connected components. Hence, the connected cardinality of the rewritings with respect to *cdr* rules is not increasing. Consequently, the class of *cdr* rules also has the *fus* property.

Theorem 5.14. *A set of cdr existential rules is a fus.*

Proof. A UCQ-rewriting q' that is generated using a *cdr* rule r and a CQ q has new connected components C'_i that are either (i) not connected to the rest of the query or (ii) that all their variables were already present in an atom of q'. Therefore, the only new variables (w.r.t. the variables in q) that are introduced in the rewritings are part of disconnected components that come from the body of the set of rules (case i). For case (ii), we can ensure that in q there was a connected component C_j that had an atom with all the variables in the newly introduced connected component C'_i, thus $card^*(C'_i) \leq card^*(C_j)$. We can then ensure that the generated UCQ-rewritings have a bounded connected cardinality. Therefore, *cdr* rules can only produce a finite number UCQ-rewritings (Lemma 2.2). □

Definition 5.14 also applies to disjunctive existential rules. However, a rule generated by a disjunctive rewriting step involving a *cdr* rule might not be a *cdr* rule. Therefore, the *fus* property cannot be extended to connected domain restricted disjunctive rules.

Example 5.16. *Consider the rule* $a(X), b(Y) \rightarrow [r(X,Y), c(X), c(Y)]$ *and the* CQ $r(X,Y), s(X,Y)$. *They both generate a new disjunctive rule* $a(X), b(Y), s(X,Y) \rightarrow [c(X), c(Y)]$ *that is not a cdr rule. If another* CQ $c(X), s(X,Z)$ *is used instead, then a disjunctive rule that is not a cdr rule is again generated, i.e.,* $a(X), b(Y), s(X,Z) \rightarrow [r(X,Y), c(Y)]$.

We use a similar approach to define a new rule class based on linear rules.

Definition 5.15 (Connected linear rule)**.** *A rule is called* connected linear *rule (clr) if every atom in the head either does not contain variables from the body or contains variables from only one connected component in the body and this connected component has only one atom.*

Both rules of Example 5.15 are also connected linear rules.

Example 5.17. *The following rule is not a cdr but it is clearly a connected linear rule.*

$$graduated(X,Z), graduated(Y,W) \rightarrow exam(V), passed(X,V), passed(Y,V)$$

Theorem 5.15. *A set of connected linear existential rules is a fus.*

Proof. A UCQ-rewriting q' that is generated using a *clr* rule r and a CQ q has new atoms a'_i that are either (i) not connected to the rest of the query or (ii) only connected to variables which were already present in an atom of q.

A *clr* prevents an atom in the head of the rule from containing variables from two different atoms (or connected components) in the body. Therefore, the rewritten atoms in q are never replaced by more than one corresponding atom that is connected to the rest of the query. This ensures that the newly formed connected component in q' will not have more atoms than those existing in q. The rewriting q' can have other atoms that are not a "replacement" of atoms in q but those atoms are not connected to the atoms that existed in q. They come from other connected components that were present in the body of the rules. Thus, the UCQ-rewritings which are introduced using connected linear rules have a bound on the number of atoms in their connected components. Therefore, connected linear rules may only produce a finite number UCQ-rewritings (Lemma 2.3). □

The definition 5.15 may also be extended to disjunctive existential rules. However, a rule generated by a disjunctive rewriting step involving a *clr* rule might not be a *clr* rule. Therefore, the *fus* property cannot be extended to connected linear disjunctive rules.

Example 5.18. *Consider a connected linear rule* $a(X), b(Y) \rightarrow [r(X,W), c(X), c(Y)]$ *and a* CQ $c(X), s(X,Z)$. *We can generate new disjunctive rule (i.e.,* $a(X), b(Y), s(X,Z) \rightarrow [r(X,W), c(Y)]$*) that is not a connected linear rule.*

A disjunctive existential rule can be restricted to have disconnected disjoints.

Definition 5.16 (disconnected disjunction)**.** *A disjunctive existential rule has* disconnected disjunction *if the disjoint components in the head of the rule never share variables with the same*

connected component in the body of the rule. A disjunctive existential rule that has disconnected disjunction is called a D-disjunctive existential rule *or* DDER.

Theorem 5.16. *The rewritings of* DDER*s are also* DDER*s.*

Proof. Let $C_1, \ldots, C_n \rightarrow [D_1, \ldots, D_m]$ be a DDER r_1. Without loss of generality, we define a rewriting r_2 that removes D_1 and introduces new atoms B in the body, i.e., $B, C_1, \ldots, C_n \rightarrow [D_2, \ldots, D_m]$. The atoms in B may possibly merge some connected components C_i of the body of the rule. In particular, those that were connected to D_1. However, those components cannot be connected to any of the remaining disjoints $[D_2, \ldots, D_m]$ due to the fact that r_1 is a DDER. Thus, r_2 is also a DDER. □

Using similar reasoning, we can also affirm that *cdr* (*clr*) that are also DDER, generate rewritings that are also *cdr* (*clr*).

Theorem 5.17. *Let $\mathcal{R}$ be a* DDER *that is also a cdr (clr). Then, $\mathcal{R}$ is also a fus.*

Proof. We state that the Algorithm 5 cannot generate infinitely many rewritings if the rules are DDER and *cdr* (*clr*).

Let Q be a UCQ and M be the maximum cardinality (width) of the bodies in the rules of $\mathcal{R}$ and the CQs in Q. Given that all the rules in $\mathcal{R}$ are *cdr* (*clr*), a rewriting step will only produce queries with a cardinality bounded by M. Additionally, the cardinality of the bodies of rules produced as rewritings of disjunctive rules in $\mathcal{R}^{\vee}$ will be bounded by M because they are DDER. The newly generated existential rules will have the same *fus* property of $\mathcal{R}^{\exists}$, i.e., *cdr* (*clr*) and this ensures that at every step of the algorithm $\mathcal{R}^{\exists}$ is a *fus*.

Thus, the rewriting Algorithm 5 will stop due to the fact that it can only produce finitely many rewritings of the initial arguments. □

For other types of queries and knowledge bases there is no certainty that the rewriting algorithm will stop. However, we can still try to compute the rewritings up to a certain depth. Nevertheless, we should point that our rewriting algorithm stops if there is a finite and complete UCQ-rewriting of the input query with respect to the rules and the constraints in the knowledge base.

Theorem 5.18. *Let $\mathcal{R}$ be a set of rules and Q a* UCQ*. If a* UCQ *Q has a finite and complete* UCQ*-rewriting with respect to $\mathcal{R}$, then Algorithm 5 stops for any finite value of k.*

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 5 (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 $Q = Q_{old}$ 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 5 where $\mathcal{R} = \mathcal{R}_{old}$. Therefore, Algorithm 5 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 5.18 could potentially produce an infinite number of rewritings.

Example 5.19. *To illustrate this we can consider the following* DKB*:*

- *Existential rules:*

$$\begin{aligned}(r(X,W), r(W,Y) \rightarrow \quad & r(X,Y)),\\ (b(X) \rightarrow \quad & a(X)),\\ (c(X) \rightarrow \quad & a(X)).\end{aligned}$$

- *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* $\mathcal{R}^{\exists}$ *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* `cover/1` *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 6 needs to have a finite depth (i.e., $k \neq \infty$) in order to avoid infinite loops.

Theorem 5.18 ensures that Algorithm 5 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 *fus* property for the set of existential rules in the knowledge base. On the other hand, Theorems 5.11 and 5.17 ensure that Algorithm 5 will always terminate if the required conditions are met.

5.5 On Queries with Answer Variables and Linear Queries

While Theorems 5.11 and 5.17 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.

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 $\mathbf{X}$ 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 $\mathcal{R}$ and then transform the entailment problem, i.e.,

$$\mathcal{R}, \mathcal{D}, ans(\mathbf{t}) \models Q \text{ iff } \mathcal{D}, ans(\mathbf{t}) \models Q'. \tag{5.5}$$

The answer atoms in the elements of Q' will be affected by the *mgus* 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 UCQa$^\neg$ can be transformed into the entailment of a UCQ (5.4). However, the presence of answer atoms in CQa$^\neg$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 $ans(\mathbf{X_1}), \ldots, ans(\mathbf{X_n}), B'$ can only be entailed in case there is an *mgu* for $\{ans(\mathbf{X_1}), \ldots, ans(\mathbf{X_n})\}$. 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 $\mathcal{D}$ 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 5 needs to be modified in order to avoid unnecessary rewritings of queries with answer variables. In Algorithms 6 and 7 we need to modify the functions *rew*/2 and *rew*$^\vee$/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, $\{ans(\mathbf{X_1}), \ldots, ans(\mathbf{X_n})\} \subseteq B$) are replaced by the CQ $B\theta$ (rule $B\theta \rightarrow H\theta$) where $\theta = mgu(\{ans(\mathbf{X_1}), \ldots, ans(\mathbf{X_n})\})$. We call these modified functions *deterministic one-step rewriting* functions. Algorithm 5 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 5.20. *Consider a knowledge base without rules and the following* UCQa$^\neg$

$$Q = [(ans(X, Z), r(X, Y), r(Y, Z), \neg r(X, Z)),\\ (ans(X, a), r(X, a))].$$

The query Q has infinitely many CQ*-rewritings of the following form:*

$$ans(X, a), ans(X, X_1), \ldots, ans(X, X_{n-1}),\\ r(X, X_n), r(X_n, X_{n-1}), \ldots, r(X_1, a).$$

However, there is a finite and complete deterministic UCQ*-rewriting of Q:*

$$[(ans(X, a), r(X, X_2), r(X_2, a), r(a, a)),\\ (ans(X, a), r(X, X_1), r(X_1, a)),\\ (ans(X, a), r(X, a))].$$

Answer variables play a different role when splitting CSFs on connected components. The *super cardinality* of a CSF of atoms F with answer variables $\mathbf{X}$ is the number of non-answer variables in it: $card_+(F) = |vars(F) \setminus vars(\mathbf{X})|$. 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 of atoms 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 $v \in vars(U_i)$ is super connected to $u \in vars(U_j)$, 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 $card^*_+(F) = \max_i(card_+(U_i))$. 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., $card^*_+([F_1, \ldots, F_m]) = \max_i (card^*_+(F_i))$.

Lemma 5.6. *Let $\mathcal{K}$ be a knowledge base and F a* CSF *of atoms with answer atom $ans(\mathbf{X})$ partitioned into the super connected components $\{U_1, \ldots, U_n\}$. Then,*

$$\mathcal{K}, ans(\mathbf{t}) \models F \quad \textit{iff} \quad \mathcal{K}, ans(\mathbf{t}) \models ans(\mathbf{X}), U_i \quad \textit{for every } U_i. \tag{5.6}$$

Proof. It follows directly from Lemma 2.1 after transforming (5.6) into:

$$\mathcal{K} \models F\theta \quad \text{iff} \quad \mathcal{K} \models U_i\theta \quad \text{for every } U_i \neq ans(\mathbf{X}), \tag{5.7}$$

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

Lemma 5.7. *Let k be a natural number. There are a finite number of equivalence classes of* CSF*s 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 5.6 and similar arguments to the ones used in the proof of Lemma 2.2. □

Given a set of rules $\mathcal{R}$ without disjunctive existential rules (i.e., $\mathcal{R}^\vee = \emptyset$), using reduction (5.4) we can focus on the disjunctive rules that are obtained from the negation of the CQa$^\neg$s:

$$\mathcal{R} \models \mathcal{Q} \quad \text{iff} \quad (\mathcal{R}^\exists, \neg\mathcal{Q}^{\neg 1}), \neg\mathcal{Q}^{\neg\#} \models \neg\mathcal{R}^\perp, \mathcal{Q}^{\neg 0}, \tag{5.8}$$

and study when Algorithm 5 with deterministic one-step rewriting functions terminates.

If all the variables in the frontier of the CQa$^\neg$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 5.

Theorem 5.19. *Let $\mathcal{R}$ be a set of rules without disjunctive existential rules and $\mathcal{Q}$ a* UCQa$^\neg$*. If all the variables in the frontier of the* CQa$^\neg$*s in $\mathcal{Q}$ are also answer variables and $\mathcal{R}^\exists$ is a fus, then Algorithm 5 with deterministic one-step rewriting functions, applied on the rules $\mathcal{R} \cup \mathcal{Q}^{\neg 1} \cup \neg\mathcal{Q}^{\neg\#}$, and* UCQ *$\mathcal{Q}^{\neg 0}$, terminates for any value of k.*

Proof. The variables that appear in negated atoms of the $\mathsf{CQ}^{\neg}$s will end up being the variables in the head of the corresponding rules in $(\neg\mathcal{Q}^{\neg i})_{i>0}$. 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 ($\mathcal{R}_a$) 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',

$$\begin{aligned} card^*_+(\texttt{rewrite}^{\exists} \quad & (\mathcal{R}^{\exists} \cup \mathcal{R}_a, Q')) \leq \\ \max(\quad & card^*_+(\texttt{rewrite}^{\exists}(\mathcal{R}^{\exists}, B)), \\ & card^*_+(\texttt{rewrite}^{\exists}(\mathcal{R}^{\exists}, Q'))), \end{aligned}$$

where B is the body of rules in $\mathcal{R}_a$.

Using Lemma 5.7 we can also affirm that there are finitely many rewritings that can be obtained. Therefore, every existential rule that is generated from the negated $\mathsf{CQ}^{\neg}$s in $(\mathcal{Q}^{\neg i})_{i>0}$ together with the rules in $\mathcal{R}^{\exists}$ will yield a finite and complete deterministic UCQ-rewriting. Thus, we can ensure the termination of every iteration of the loop in Algorithm 5 for any value of k.

Likewise, every existential rule that is generated from the negated $\mathsf{CQ}^{\neg}$s in $(\mathcal{Q}^{\neg i})_{i>0}$ will have a body with a bounded super connected cardinality. Thus, we cannot generate infinitely many existential rules from the $\mathsf{CQ}^{\neg}$s in $(\mathcal{Q}^{\neg i})_{i>0}$ (Lemma 5.7), which ensures the termination of Algorithm 5 with deterministic one-step rewriting functions for any value of k. □

Based on Theorem 5.11, we can also define other restrictions on $\mathsf{UCQa}^{\neg}$s that ensure that the rewriting algorithm stops. As in the case of existential rules, we say that a CQ ($\mathsf{CQ}^{\neg}$) is *linear* if it contains only one positive literal. Similarly, a UCQ ($\mathsf{UCQ}^{\neg}$) is linear if all the CQs ($\mathsf{UCQ}^{\neg}$s) in it are also linear. A CQa ($\mathsf{CQa}^{\neg}$) is linear if it contains only one positive literal in the body. Likewise, a UCQa ($\mathsf{UCQa}^{\neg}$) is linear if all the CQas ($\mathsf{UCQa}^{\neg}$s) in it are also linear.

Theorem 5.20. *Let $\mathcal{R}$ be a set of rules without disjunctive existential rules and Q a $\mathsf{UCQa}^{\neg}$. If $\mathcal{R}$ is a set of linear rules and Q a linear $\mathsf{UCQa}^{\neg}$, then Algorithm 5 with deterministic one-step rewriting functions, applied on the rules $\mathcal{R} \cup \mathcal{Q}^{\neg 1} \cup \neg\mathcal{Q}^{\neg\#}$, and UCQ $\mathcal{Q}^{\neg 0}$, stops for any value of k.*

Proof. Because Q is linear the corresponding disjunctive rules $(\neg\mathcal{Q}^{\neg i})_{i>1}$ will have two atoms in the body and one of them will be an answer atom. The CQas 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 $\mathcal{R}$ and the existential rules generated from $(\neg\mathcal{Q}^{\neg i})_{i>0}$ will have a maximum of two atoms. Thus, there are finitely many deterministic CQ-rewritings that can be generated.

We conclude that Algorithm 6 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 5 with deterministic one-step rewriting functions stops for any value of k. □

If there is a finite and complete deterministic UCQ-rewriting of a UCQa$^{\neg}$ $\mathcal{Q}$ with respect to a set of rules $\mathcal{R}$ that does not contain disjunctive existential rules, we can ensure that Algorithm 5 with deterministic one-step rewriting functions stops.

Theorem 5.21. *Let $\mathcal{R}$ be a set of rules without disjunctive existential rules and $\mathcal{Q}$ a* UCQa$^{\neg}$. *If there is a finite and complete deterministic* UCQ*-rewriting of $\mathcal{Q}$ with respect to $\mathcal{R}$, then Algorithm 5 with deterministic one-step rewriting functions, applied on the rules $\mathcal{R} \cup \mathcal{Q}^{\neg 1} \cup \neg\mathcal{Q}^{\neg\#}$, and* UCQ *$\mathcal{Q}^{\neg 0}$, 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 $\mathcal{Q}_f$, then after a finite number of iterations (with a finite value of k) a rewriting equivalent to $\mathcal{Q}_f$ will be generated. The completeness of $\mathcal{Q}_f$ ensures us that any further rewriting of $\mathcal{Q}_f$ using existential rules and deterministic one-step rewriting functions will not produce new deterministic CQs, i.e., the condition $\mathcal{Q} = \mathcal{Q}_{old}$ holds for the rest of the iterations in the loop. Consequently, after finitely many iterations we also reach the condition $\mathcal{R} = \mathcal{R}_{old}$. Therefore, Algorithm 5 terminates because after a finite number of iterations the condition to continue iterating on the loop will not hold. □

The concepts of connected domain restricted rules, connected linear rules, and disconnected disjunction can be modified to represent queries with negated atoms using the concept of supper connection. We basically define the properties that would ensure that the corresponding disjunctive existential rule $r_q = \neg q$ holds the desired property, considering that answer variables can be interpreted as constants when we analyze the connections.

Definition 5.17. *A* CQa *is a connected domain restricted query (cdrq) if, for every super connected component C in the positive liters of the query and for every negated atom h, h contains none or all the nonanswer variables of C.*

Definition 5.18. *A* CQa *is a connected linear query (clq) if every negated atom is either not supper connected to positive atoms or it is supper connected to only one positive atom.*

Additionally, a concept corresponding to disconnected disjunction can be defined for CQas.

Definition 5.19. *A* CQa *has disconnected negation, if every negated atom is not super connected to another negated atom.*

Theorem 5.22. *Let $\mathcal{R}$ be a set of existential rules and $\mathcal{Q}$ a* UCQa$^{\neg}$ *with disconnected negation. If $\mathcal{R}$ is a set of cdr(clr) rules and $\mathcal{Q}$ a cdrq(clq)* UCQa$^{\neg}$*, then Algorithm 5 with deterministic one-step rewriting functions, applied on the rules $\mathcal{R} \cup \mathcal{Q}^{\neg 1} \cup \neg\mathcal{Q}^{\neg\#}$, and* UCQ *$\mathcal{Q}^{\neg 0}$, stops for any value of k.*

Proof. Algorithm 6 will be called with $\mathcal{R}' = \mathcal{R} \cup \mathcal{Q}^{\neg 1} \cup \neg\mathcal{Q}^{\neg\#}$, a *cdr* (*clr*) that have disconnected disjuntion. Hence, there is a finite UCQ rewriting of $\mathcal{Q}^{\neg 0}$ with respect to $\mathcal{R}'$. Using Theorem 5.21 we can ensure that Algorithm 6 stops for $\mathcal{R}'$ and $\mathcal{Q}^{\neg 0}$ using deterministic one-step rewriting functions and a finite value of k. □

Part III

Implementations and Experimental Evaluation

Chapter 6

Constraint Saturation Evaluation

Negated concepts are strongly disconnected queries with respect to a $\mathsf{DL-Lite}$ ontology because $\mathsf{DL-Lite}$ axioms are a set of linear of rules (Property 3.1). Therefore, the constraint saturation for negated concepts will contain all the possible answers. In this paper, we focus on comparing the performance of the proposed approach with another approach by Jianfeng Du and Jeff Z. Pan that is able to rewrite negated concepts [17].

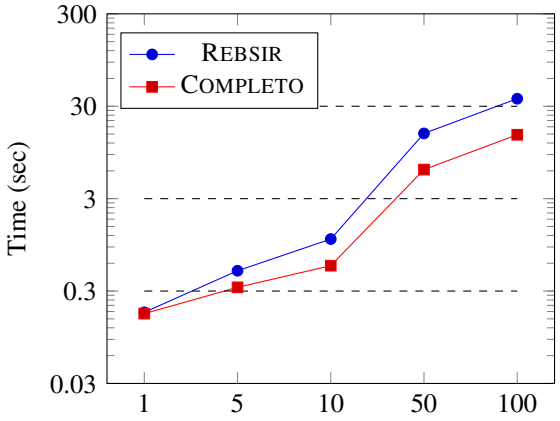
COMPLETO v1 was implemented using RAPID as an external rewriter and a connection to a MySQL database for efficient instance retrieval. A TBox is used to obtain a rewriting of the initial constraints in the system. Then, the assertions that could be encoded initially in OWL format are translated to a MySQL database. Finally, the rewriting and instance retrieval processes can be carried out by using the constraints rewriting, the database of assertions and the queries that need to be rewritten and answered. Appendix A shows a description of the system and how it can be installed and used that is also available online[1].

The experiments were carried out on an Intel® Core™ i7-3612QM CPU @ 2.10GHz x 8, with 8 Gb of RAM memory and a SSD running Ubuntu 17.04 64-bit. The benchmark used consists of two groups of ontologies used in [17]. One of the groups was from the Lehigh University Benchmark (LUBM) [31] and the other from DBPedia (version 2014) [32]. Some axioms were removed from the original versions of the ontologies in order to make them compatible with the RAPID system. Also, constraints stating that sibling atomic concepts are disjoint were added to the LUMB ontologies. The group of LUMB ontologies consists of the same set of axioms and different numbers of assertions associated with different numbers of universities (1, 5, 10, 50, and 100) given as a parameter to the LUMB generator [31]. The second group of DBPedia ontologies was built with basic assertions about atomic concepts and abstract roles from DBPedia-as-Tables[2] to construct the ABox. Each version of the ontology uses the same axioms and a percentage of the assertions (1, 5, 10, 50 and 100%). The queries to rewrite and to answer using the assertions were built by negating each of the concepts present in the TBox. For the LUBM ontologies, they were 43 concepts and for the DBPedia ontologies 783.

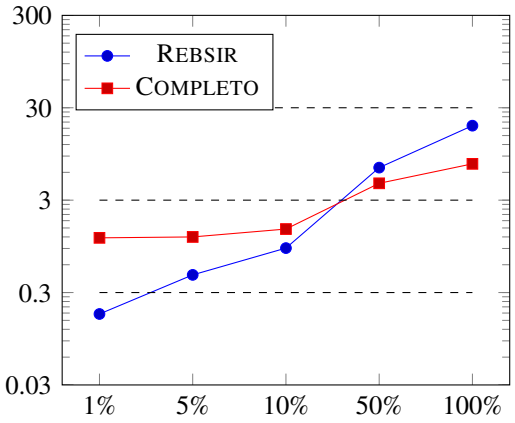
Figure 6.1 shows the comparison of the average runtime taken to answer each query of the dataset. The performance of COMPLETO v1 is better than the performance of REBSIR for the LUBM group of ontologies. On the other hand, for the DBPedia datasets COMPLETO v1 on average

[1] http://image.ntua.gr/completo-rr2017/Appendixes.pdf
[2] http://web.informatik.uni-mannheim.de/DBpediaAsTables/

takes longer than REBSIR to answer a query for small ontologies. Yet, when the ontologies grow in size the performance of COMPLETO v1 gets closer to the performance of REBSIR eventually becoming better than it.



(a) LUBM dataset group.



(b) DBPedia dataset group.

Figure 6.1: Comparison of the Average Runtime per query for the benchmarks.

The relative runtime can be seen in Fig. 6.2 where the coordinates of the dots plotted represent the average times taken by each system in each of the datasets. Both axes of the graph are in logarithmic scale and points over the $y = x$ line represent datasets where the COMPLETO v1 system takes on average less time than REBSIR to answer a query. We can clearly see that for the LUBM group COMPLETO v1 is always faster and it takes on average 59% of the time taken by REBSIR to answer a query. For the DBPedia group of ontologies, COMPLETO v1 on average takes 238% of the time that REBSIR takes to answer a query. Yet, for both cases, we can see that the COMPLETO v1 system is more scalable i.e. with the increase of the size of the ABox the COMPLETO v1 system improves the relative runtime difference.

Figure 6.3 shows a comparison of the maximum *resident set size* (RSS) of the systems during the process of answering all the queries of the benchmark. REBSIR uses in all the cases less memory than COMPLETO v1. Considering the relative memory difference for each case, on average, REBSIR uses 54% of the memory used by COMPLETO v1.

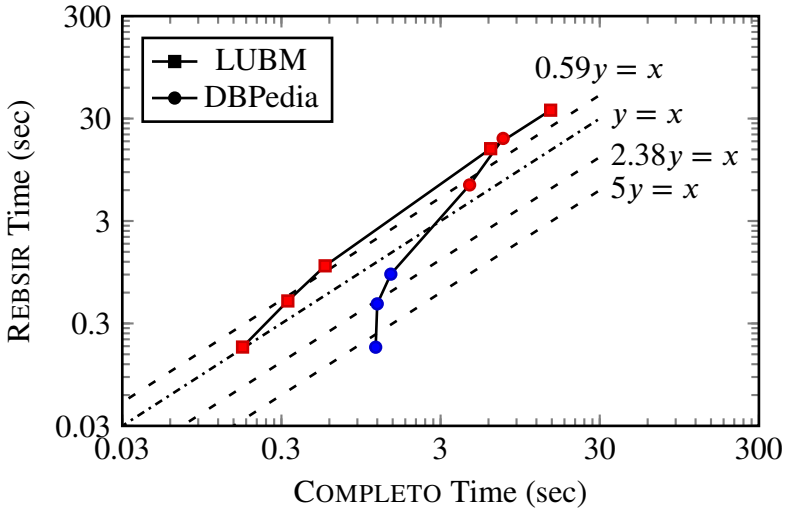
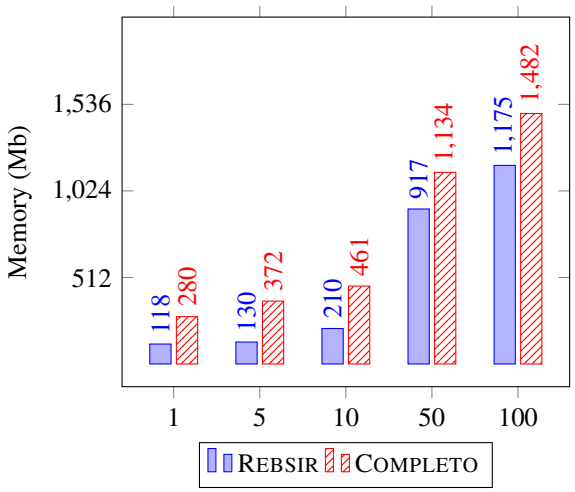
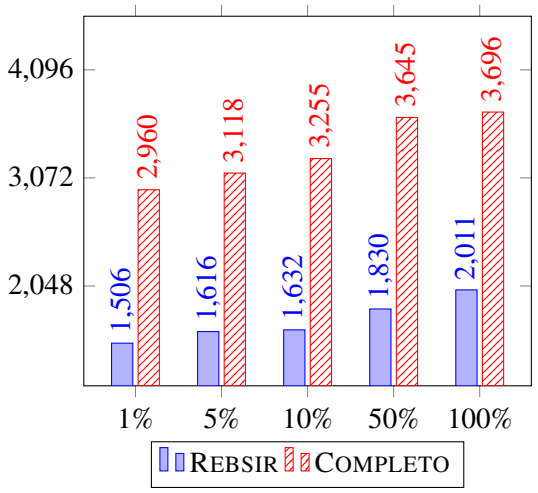


Figure 6.2: Relative Average Runtime per query for the benchmark.



(a) LUBM dataset group.



(b) DBPedia dataset group.

Figure 6.3: Comparison of the maximum RSS used to answer all the queries of the benchmark.

Chapter 7

Horn Conjunctive Queries Evaluation

In COMPLETO v2, our system was upgraded with respect to the version presented in [18]. We now use the GRAAL [28] rewriting system that is able to rewrite CQs using existential rules. We also changed the MySQL database to H2 database for representing the ABox assertions in the ontology. The OWL 2 ER fragment of the TBox and the rules corresponding to the Horn queries are used to rewrite the constraints of the system. Then, the ABox assertions that could be encoded initially in OWL format are translated to a H2 database. Finally, the consistency check and instance retrieval processes can be carried out by using the constraints rewriting, the database of assertions and the queries that need to be rewritten and answered. A description of the system and how it can be installed and used is available online[1] and also in Appendix A. Additionally, we used the interfaces provided in the OWL API for the FACT++ [33] and HERMIT [34] solvers. Both systems perform reasoning using tableau techniques. However, they are designed to deal with ontology languages that are more expressive than OWL 2 ER.

The experiments were carried out on an Intel® Core™ i7-5930K CPU @ 3.50GHz x 6, with 32 Gb of RAM memory and a SSD running Ubuntu 16.04 64-bit. We used ontologies that have both constraints and assertions. Additionally, they also belong to the OWL 2 ER fragment. The benchmark is composed by one of the groups of ontologies used in [17], the one from the Lehigh University Benchmark (LUBM) [31]. Additionally, constraints stating that sibling atomic concepts are disjoint were added. The group of LUBM ontologies consists of the same set of axioms and different number of assertions associated to different number of universities (1, 5, 10, and 50) given as a parameter to the LUBM generator [31]. Additionally we tested the system using two small ontologies travel[2] and films[3]. Figure 7.1 shows the axiom counts of the ABox and ABox+TBox for the ontologies of the benchmark.

The queries for the experiments were obtained by applying Association Rules techniques implemented using the Weka software [35]. Rules that describe hidden relations between the concepts of the ontology were obtained and we rewrote the counter examples expressions corresponding to the rules in order to check when it is consistent to add the new rules to the ontologies. A total of 86 queries were used for the travel ontology, 14 for the films ontology and 7 for the LUBM ontologies group.

[1] `http://image.ntua.gr/~gardero/completo2.0/usermanual.pdf`
[2] `http://www.owl-ontologies.com/travel.owl`
[3] `https://www.irit.fr/recherches/MELODI/ontologies/FilmographieV1.owl`

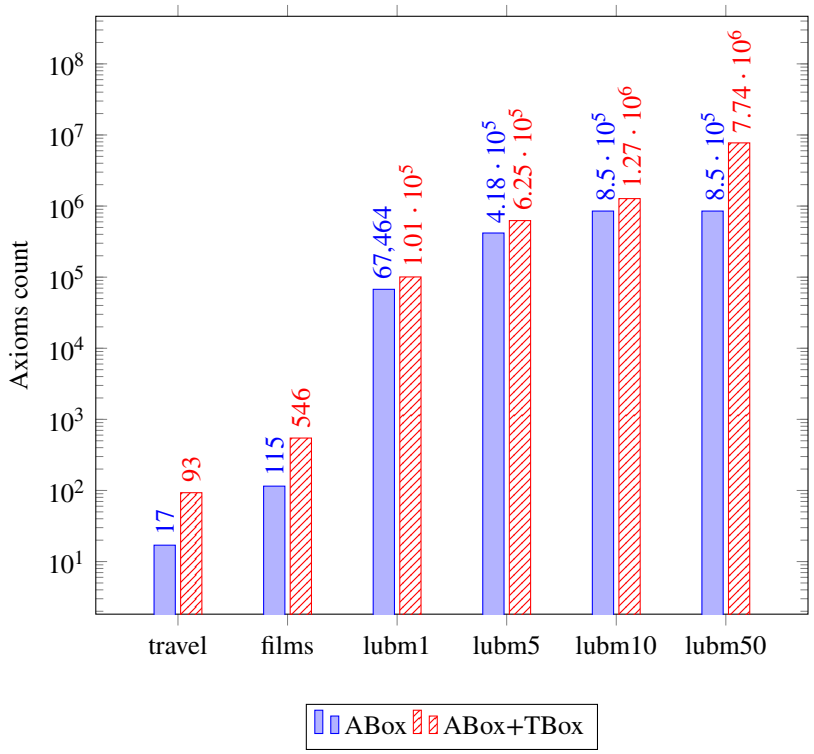


Figure 7.1: Axiom counts of the ontologies of the benchmark.

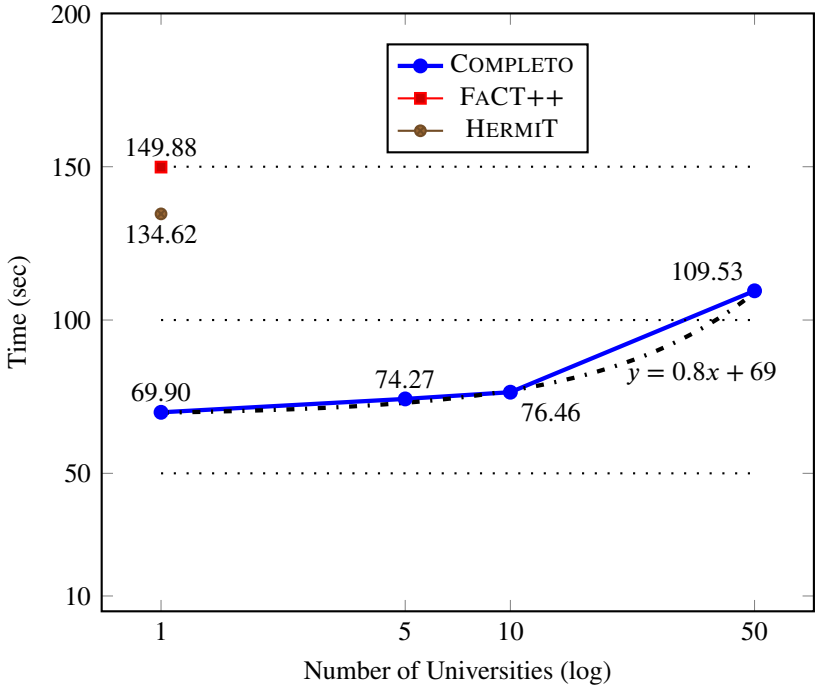


Figure 7.2: Comparison of the Average Runtime per query for the LUBM group of ontologies.

Figure 7.2 shows the average runtime per query for the LUBM group of ontologies. The runtime for COMPLETO increases with the size of the ABox and one can notice that it describes a linear relation ($y = 0.8x + 69$) to the number of universities in the ABox. On the other hand, FACT++ and HERMIT were only able to provide answers for the ABox containing only one university and they both took on average approximately twice as longer than COMPLETO for answering each query. For the rest of the LUBM ontologies, the systems reached the timeout of one hour per query defined for the experiments.

For the films ontology, COMPLETO takes 28.9 sec per query while HERMIT and FACT++ take less than 0.2 sec per query. In the case of the travel ontology, HERMIT and FACT++ take both 0.03 sec per query while COMPLETO takes 0.48 sec per query. Both Films and travel ontology have a very small number of assertions compared to the group of LUBM ontologies.

The RAM memory used by COMPLETO is not strongly affected by the size of the ABox and the average is 2,480 MB. The other systems used more than 5,800 MB for the only case where they were able to give answers.

Chapter 8

General Conjunctive Queries with Negated Atoms Implementation and Evaluation

COMPLETO[1] is a query rewriting system that focuses on answering $\mathsf{UCQ}^{\neg}$s in the framework of disjunctive existential rules. The system is implemented in java. The first version of COMPLETO (COMPLETO v1) [18] answers $\mathsf{CQ}^{\neg}$ using a resolution-based approach to eliminate negated atoms. The proposed algorithm is complete only for a restricted type of queries.

In the second version of the system [19] (COMPLETO v2), only queries with one negated atom are answered by being transformed into rules. The approach is complete but termination is guaranteed only when the resulting set of rules is a *fus*.

The 3rd version of COMPLETO (COMPLETO v3) [36] implements Algorithm 5 with deterministic one-step rewriting functions and answers queries with answer variables that have an arbitrary number of negated atoms. Algorithm 5 can be seen as a generalization of both algorithms proposed in [18, 19]. Indeed, queries with one negated atom are transformed into rules, while the rewriting defined for disjunctive rules is similar to what was presented in [18] as constraint resolution. Furthermore, COMPLETO v3 takes advantage of the termination results for knowledge bases consisting of a *fus* and $\mathsf{UCQ}^{\neg}$s whose frontier is part of the answer variables of the query (Theorem 5.19), as well as for knowledge bases consisting only of linear elements (Theorem 5.20). 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.

The current version of the system is called ECOMPLETO[2] and it is implemented in the Elixir programming language. The system answers queries with answer variables that contain an arbitrary number of negated atoms with respect to disjunctive existential rules. Ontologies are provided in DLGP+ format, a proposed extension of DLGP[3] v2.0 that allows the specification of disjunctive existential rules and negated atoms in queries.

Disjunction in DLGP+ is specified in the head of a rule by writing a list of the disjoints en-

[1] `http://image.ntua.gr/~gardero/completo3.0/`
[2] `https://github.com/gardero/ecompleto`
[3] `https://graphik-team.github.io/graal/papers/datalog+_v2.0_en.pdf`

Table 8.1: Rewriting experiments results for the CQa$^\neg$s from LUBM and TRAVEL ontologies.

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

closed in squared brackets. The disjoint elements can be a single atom or several atoms enclosed in brackets, e.g.,

```
[disj. rule] [leaf(X), (inner_node(X), edge(X,Y))] :- node(X).
```

Negation in queries with negated atoms is specified with the minus symbol before an atom, e.g.,

```
[q neg] ? :- person(X), -marriedTo(X,Y).
```

8.1 Experiments

To the best of our knowledge, there is no other system that produces UCQ-rewritings for UCQ$^\neg$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 [18, 19]. The first is the Lehigh University Benchmark (LUBM) ontology [31], 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[4] that has 10 disjoint class axioms. The OWL 2 ER [37] 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 CQa$^\neg$s for each ontology which we used to let COMPLETO produce finite UCQ-rewritings of the UCQa$^\neg$ that contain all the queries in the file and also for each separated CQa$^\neg$. 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 [38] on a dataset obtained from the assertions of the ontologies. The queries and the ontologies we used are publicly available [5].

Table 8.1 shows the size of the UCQ-rewriting (rew) for the UCQa$^\neg$ containing all the CQa$^\neg$s in the file and the minimum (min), mean and maximum (max) statistics for the rewriting of each individual CQa$^\neg$ in the file. The table also shows the time (time) in seconds and the RAM memory

[4] https://protege.stanford.edu/ontologies/travel.owl

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

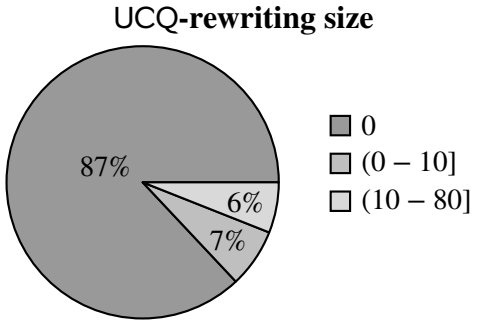


Figure 8.1: Size of the UCQ-rewritings for the TRAVEL ontology.

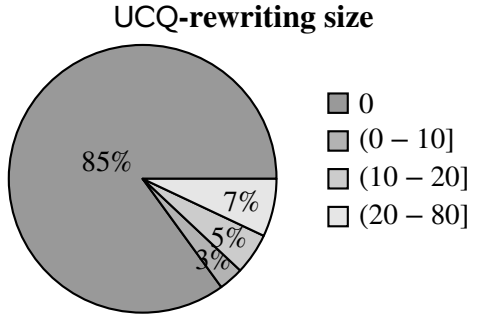


Figure 8.2: Size of the UCQ-rewritings for the LUBM ontology.

(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 $\mathsf{UCQa}^{\neg}$ or individual $\mathsf{CQa}^{\neg}$.

For the TRAVEL ontology, the size of the UCQ-rewriting of the $\mathsf{UCQa}^{\neg}$ is smaller than the biggest UCQ-rewriting for an individual $\mathsf{CQa}^{\neg}$. The time that it took to compute the rewriting of the $\mathsf{UCQa}^{\neg}$ is the time that it takes on average to rewrite 125 individual $\mathsf{CQa}^{\neg}$s (5 min). The RAM memory used to rewrite the $\mathsf{UCQa}^{\neg}$ is approximately double of the RAM used for rewriting the individual $\mathsf{CQa}^{\neg}$ that consumed the most RAM memory.

8.1.1 COMPLETO v3 Experiments

For the LUBM ontology, the size of the rewriting of the $\mathsf{UCQa}^{\neg}$ has 11 more queries than the biggest rewriting for an individual $\mathsf{CQa}^{\neg}$. The time that it took to compute the rewriting of the $\mathsf{UCQa}^{\neg}$ is the time that it takes on average to rewrite 30 individual $\mathsf{CQa}^{\neg}$s (less than 2 hours). The RAM memory used to write the $\mathsf{UCQa}^{\neg}$ is less than the RAM memory that was used for rewriting the individual $\mathsf{CQa}^{\neg}$ that consumed the most RAM memory.

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

Figures 8.1 and 8.2 show information about the UCQ-rewriting size. In both ontologies, at least 85 % of the queries have zero rewritings. In this case, the CQ-rewritings of the CQas are subsumed by the CQ-rewritings of the negative constraints of the DKB.

Figures 8.3 and 8.4 show the cumulative distribution of the rewriting runtime. Dashed horizontal lines represent the mean runtime. Each bar represents the number of queries that were rewritten

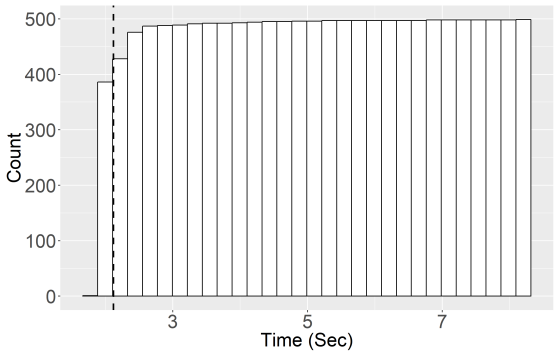


Figure 8.3: Cumulative distribution of the time needed to compute the UCQ-rewriting for the TRAVEL ontology.

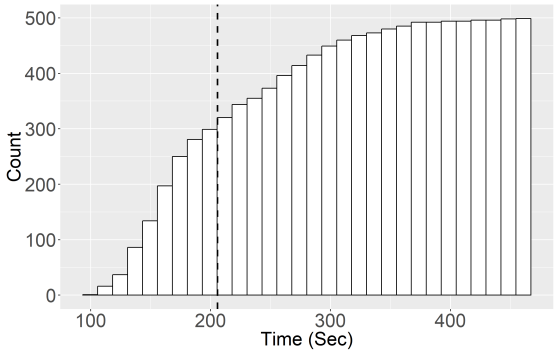


Figure 8.4: Cumulative distribution of the time needed to compute the UCQ-rewriting for the LUBM ontology.

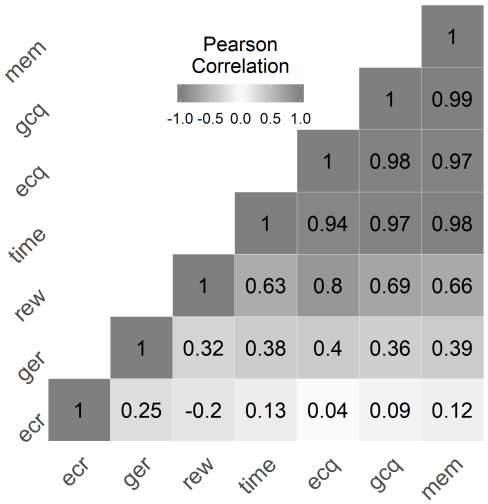


Figure 8.5: Correlation matrix with different performance parameters for the TRAVEL ontology.

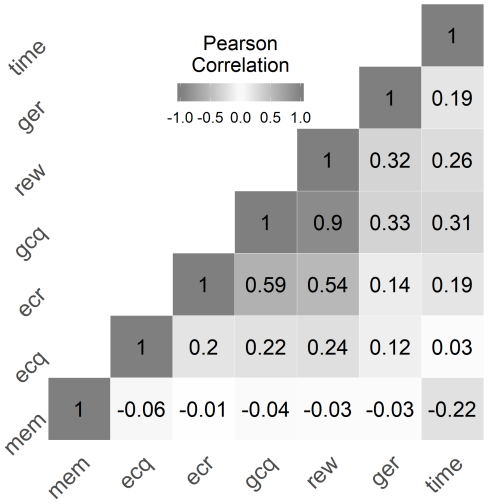


Figure 8.6: Correlation matrix with different performance parameters for the LUBM ontology.

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 8.5 and 8.6 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 8.21. *One of the queries for the* TRAVEL *ontology was:*

$$ans(X) :- \neg Capital(X), \neg Town(X), Destination(X).$$

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

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

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.

8.1.2 ECOMPLETO Experiments

Table 8.2: Distribution metrics computed on the query rewriting runtime and the memory used for both ontologies.

	LUBM		Travel	
Metric	Runtime (m)	Memory (Mb)	Runtime (m)	Memory (Mb)
mean	18.59	370.46	0.035	104.54
std	1.67	37.00	0.150	16.70
min	15.33	328.00	0.020	93.00
25%	17.62	350.00	0.022	101.00
50%	18.43	359.00	0.023	103.00
75%	19.10	371.00	0.024	105.00
max	24.76	526.00	2.454	337.00

Table 8.2 shows the mean, std, min, max, and the 25th, 50th, and 75th percentiles of the UCQ rewriting runtime and the used RAM memory for both ontologies. The UCQ rewriting runtime is on average 500 times faster for the TRAVEL ontology than for the LUBM ontology. The rewriting process for the TRAVEL ontology uses on average one third of the RAM memory used to rewrite queries compared to the LUBM ontology.

Figure 8.7 shows the runtime vs RAM memory of the rewriting process for each query of the LUBM ontology. There are 3 clusters that group the points according to their standardized coordinates. Figure 8.8 shows also the size of the UCQ rewriting using colors. The darker the datapoint is, the larger the size of the corresponding rewriting is. The cluster grouping shows some correlation with the size of the rewriting.

Table 8.3 shows the count, mean, std, min, max, and the 25th, 50th, and 75th percentiles of the UCQ rewriting runtime and the used RAM memory for each of the clusters of LUBM queries.

Figure 8.9 shows the distribution of the query rewriting runtime for the LUBM ontology. The distribution is multimodal and it is split according to the cluster group.

The distribution of the RAM memory used in the rewriting process is shown in Figure 8.10. We can notice a bimodal shape, despite having 3 clusters that group the queries.

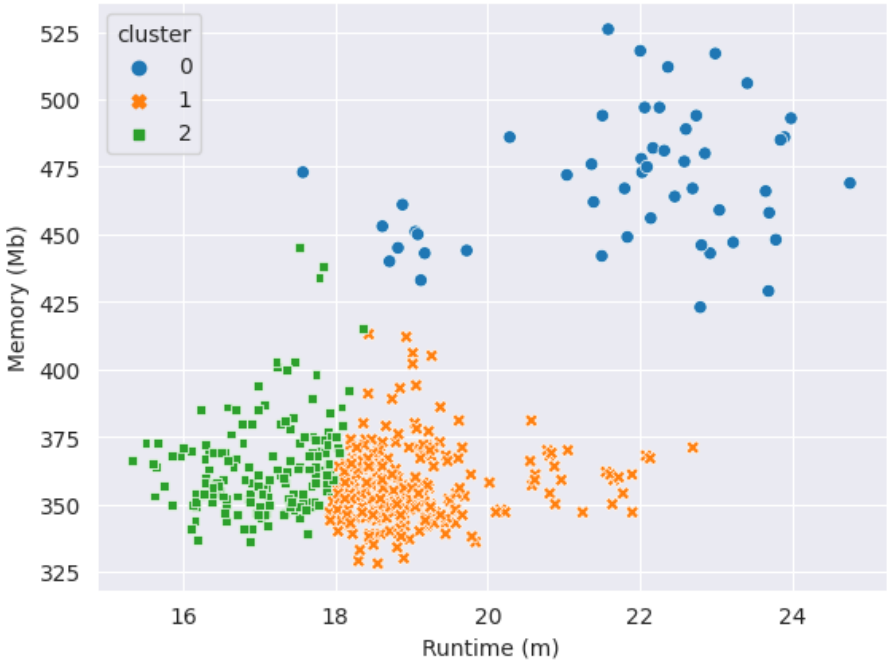


Figure 8.7: Clustering of query rewriting runtime vs memory usage for LUBM.

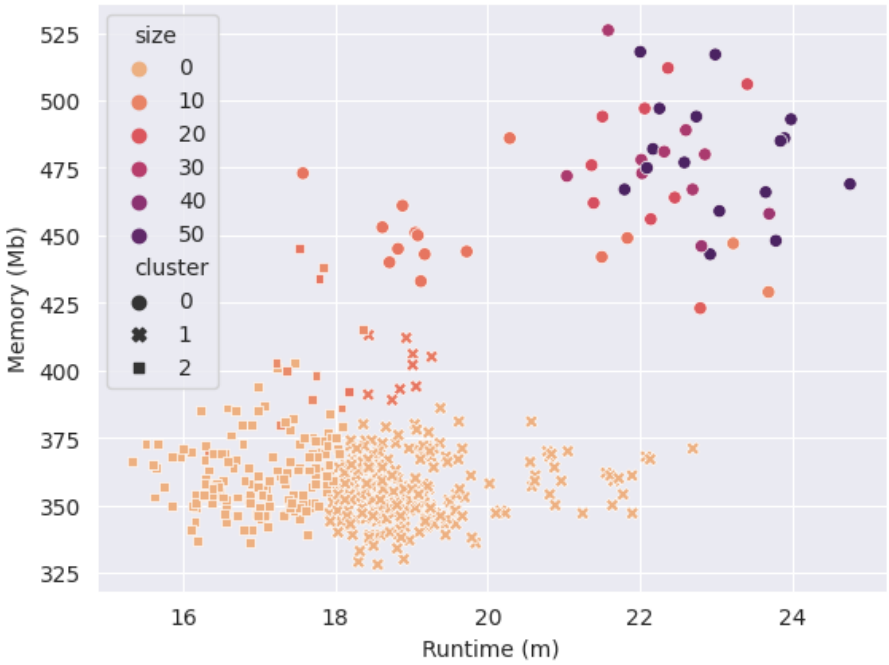


Figure 8.8: Clustering of query rewriting runtime vs memory usage and size of the rewriting for LUBM.

Table 8.3: Distribution metrics computed on the query rewriting runtime and the memory used for both clusters in LUBM ontology.

	Runtime (m)			Memory (Mb)	
Metric	Cluster 0	Cluster 1	Cluster 2	Cluster 0	Cluster 1,2
count	50	280	170	50	450
mean	21.82	18.95	17.06	469.64	359.44
std	1.72	0.88	0.68	24.31	15.49
min	17.57	17.92	15.33	423.00	328.00
25%	21.37	18.40	16.53	449.25	349.00
50%	22.15	18.68	17.10	468.00	357.00
75%	22.90	19.20	17.65	485.75	367.00
max	24.76	22.69	18.36	526.00	445.00

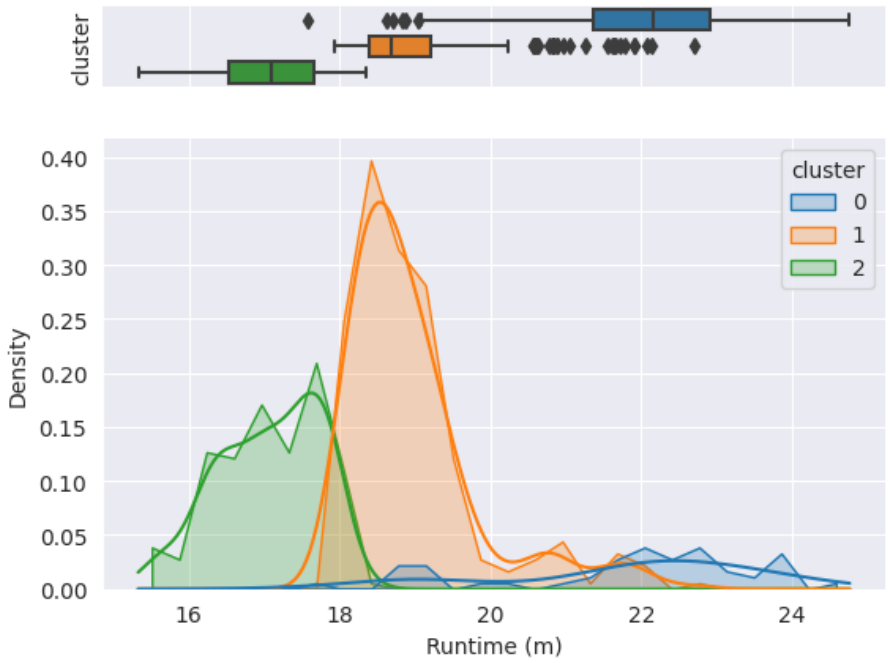


Figure 8.9: Histogram of query rewriting runtime for LUBM.

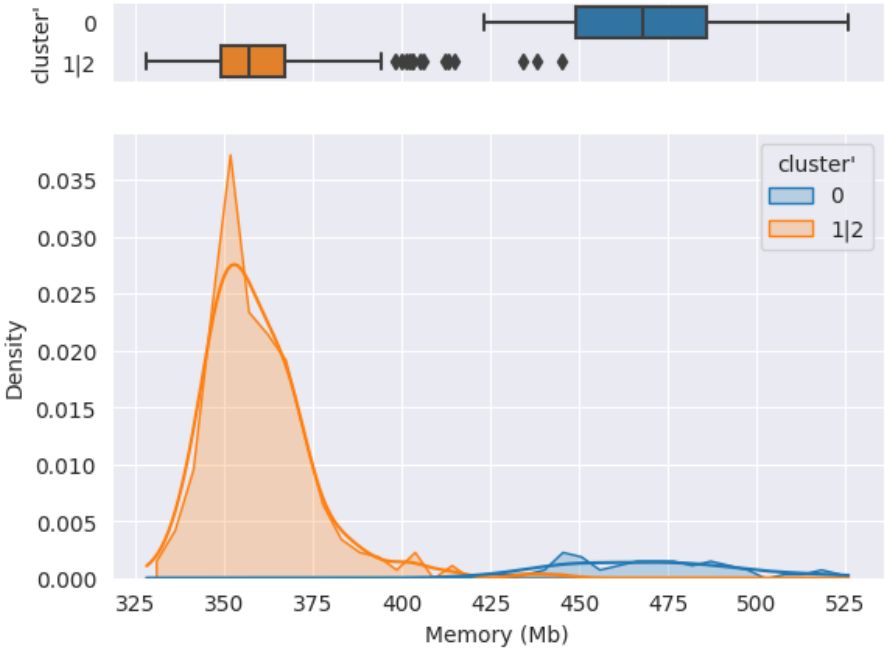


Figure 8.10: Memory usage histogram for LUBM.

Chapter 9

Conclusions

Firstly, we proposed a method to rewrite conjunctive queries with negated atoms based on resolution in order to eliminate the negated atoms. The constraints in the knowledge base are rewritten using the rules in order to express the inconsistencies without the need of the rules. The expanded set of constraints is used to build a constraint saturation of the initial query by eliminating the negated atoms using resolution. Finally, conventional rewriting algorithms are used in the resulting union of conjunctive queries.

The method was implemented in the first version of COMPLETO. The RAPID system was used as an external rewriter and a connection to a MySQL database allowed efficient instance retrieval for the obtained rewriting.

COMPLETO v1 was compared to REBSIR for rewriting negated concepts. The experimental results showed that COMPLETO v1 is generally faster than REBSIR, especially when the number of assertions in the knowledge base grows. The relative performance of COMPLETO v1 with respect to REBSIR is always improved when the number of assertions grows. On the other hand, REBSIR used less memory resources.

Despite the satisfactory performance of COMPLETO v1, we believe that the principal result is the first definition of query answering for queries with negated atoms based on the classical rewriting algorithms.

Secondly, we proposed the definition of Horn queries and union of Horn conjunctive queries. We also provide a way to check the decidability of the entailment problem for these queries, by using a reduction to the entailment problem of conjunctive queries. The reduction also allows to answer Horn conjunctive queries using classic query answering methods developed on conjunctive queries.

We propose the use of a conjunctive query rewriting approach to provide a UCQ rewriting for a union of Horn conjunctive queries. The method is implemented in version 2.0 of the system COMPLETO. The GRAAL system was used as an external rewriter and a connection to an H2 database allowed efficient instance retrieval for the obtained rewriting.

COMPLETO v2 was compared to FACT++ and HERMIT for rewriting Horn queries. The experimental results showed that COMPLETO v2 is faster than the other systems for big ontologies. For ontologies with more than half a million axioms FACT++ and HERMIT were not able to find the answers to the queries on time. For very small ontologies COMPLETO v2 is still fast, yet FACT++

and HERMIT outperform it.

We are very glad with the satisfactory performance of COMPLETO v2 but we believe that the principal result is the definition of Horn queries and the way we can answer them. We can use the system to improve the consistency check for the ontologies.

Moreover, 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 fewer 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 linear $\mathsf{CQ}^\neg$s 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 $\mathsf{UCQa}^\neg$s with at most one positive atom (with respect to linear existential rules and linear constraints) and for $\mathsf{UCQa}^\neg$s that include the frontier in the answer variables (with respect to a *fus*). Both types of $\mathsf{UCQa}^\neg$s are very expressive. We also proposed two new classes of existential rules that have the *fus* property and that are generalizations of *domain restricted* rules and *linear* rules.

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 specializes 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 a reasonable time and using a reasonable amount of RAM memory. Also, rewriting $\mathsf{UCQ}^\neg$s with a large number of queries takes considerably less time than the time required to rewrite all the $\mathsf{CQ}^\neg$s individually.

Finally, we took on the significant computational challenge involved in determining the existence of finite and complete UCQ-rewritings, as well as the identification of finite unification sets (*fus*) of rules. Our contributions encompass the introduction of two novel rule classes: *connected*

linear rules and *connected domain restricted rules*, which have the *fus* property and surpass the expressiveness of their antecedent rule classes, namely linear rules and domain-restricted rules.

Furthermore, we introduced the concept of *disconnected disjunction* tailored to disjunctive existential rules. This novel conceptualization makes it easier to achieve the *fus* property, even in the context of disjunctive existential rules. In terms of practical implementation, we have elaborated upon the architecture of our system, ECOMPLETO, specifically engineered for the task of query rewriting within the framework of disjunctive existential rules. The system handles $\mathsf{UCQ}^{\neg}$ queries that include universally quantified negation. In addition, it offers an augmented version of DLGP+, thereby facilitating the specification of disjunctive existential rules and the inclusion of negated atoms within queries.

Empirical evaluation of our system attested to the consistent performance of ECOMPLETO in generating finite UCQ-rewritings for a diverse array of queries. Remarkably, the system exhibited enhanced efficiency during the rewriting process, notably in the case of the TRAVEL ontology, where it demonstrated a fast performance together with diminished memory consumption, contrasted against the LUBM ontology.

Finally, the experiments provide valuable insights into ECOMPLETO's performance when producing UCQ-rewritings for $\mathsf{UCQ}^{\neg}$s involving universally quantified negation and disjunctive existential rules. The significant differences in runtime and memory efficiency between the LUBM and TRAVEL ontologies emphasize the importance of considering ontology complexity when assessing ECOMPLETO's performance. Clustering and distribution patterns offer additional insights into the behavior of ECOMPLETO under different query scenarios. Overall, these findings contribute to our understanding of the system's capabilities and provide valuable information for researchers and practitioners working with complex queries and ontologies in the context of disjunctive existential rules.

Appendices

Appendix A

COMPLETO's user manual

Figure A.1 shows a description of the COMPLETO system and its main workflow.

Installing the Software

The software is contained in a jar[1] file and it needs to be executed by a java[2] compiler:

```
java -jar completo.jar \
   [other options]
```

The option `-Xmx` can help us increase the memory assigned to the process.

Running the Experiments

For running the experiments we need to know how to execute the main actions provided by the COMPLETO system. The TBoxes[3] and the ontologies with assertions[4] used in the experiments can be downloaded to provide the necessary inputs to the system.

Constraints Rewriting.

Given a owl ontology with constraints in it, we can generate a file with the rewriting of the constraints. The command to execute is the following:

```
java -jar completo.jar \
   -o [ontology path] \
   (-q [queries path] -nconcepts) \
   -c [constraints path]
```

where the path for the constraints should refer to a non existing file where all the rewriting of the constraints will be written as queries. Also, if we want to generate a queries file with the negation of all the concepts in the ontology we then use `-q [queries path] -nconcepts`. Files with queries have the following format:

[1] `http://image.ntua.gr/completo/completo.jar`
[2] version 1.8 or more recent.
[3] `http://image.ntua.gr/completo/tbox.zip`
[4] `http://image.ntua.gr/completo/ontofile.zip`

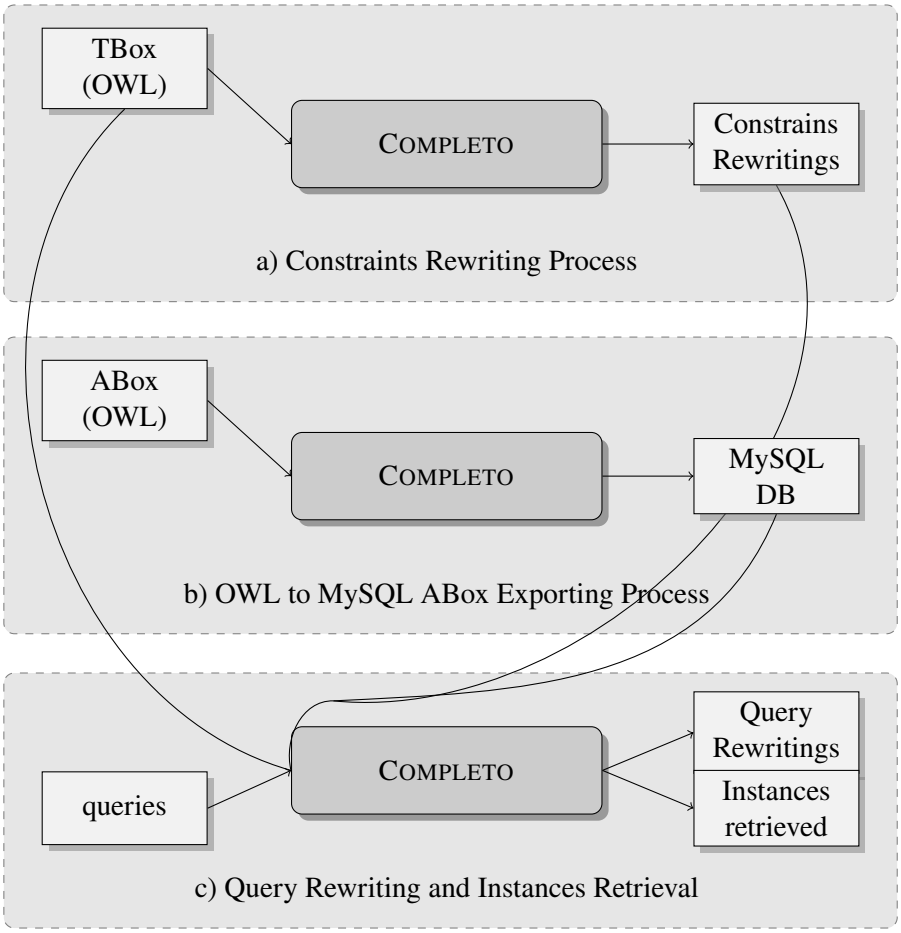


Figure A.1: Diagram of the COMPLETO system.

```
Q(?X) <- -AdministrativeStaff(?X).
Q(?X) <- -Article(?X).
Q(?X) <- -AssistantProfessor(?X).
Q(?X) <- -AssociateProfessor(?X).
```

and files with constraints have boolean queries describing the constraints:

```
_answer <- College(?X), ResearchGroup(?X).
_answer <- College(?X), researchProject(?X, ?_u0).
_answer <- AdministrativeStaff(?X), Article(?X).
_answer <- AdministrativeStaff(?X), Book(?X).
```

where negated atoms come with a minus "-" sign, variables with a "?" sign before the identifier. Queries files can also be built manually.

OWL to MySQL ABox Exporting Process.

In order to generate a SQL database with the assertions contained in an owl ontology we need to run:

```
java -jar completo.jar \
   -o [ontology path]  -eabox \
   -aboxname [abox SQL name]
```

The command will create a SQL database using MySQL software. The database will be used in the instance retrieval process for the queries. Additionally, we need a configuration file (`sqlconfig.properties`) containing some important data used to establish the connection to MySQL:

```
username=[username]
password=[password]
url=[url:port of the installed MySQL instance]
```

Also, for big databases we will need to increase the limits of the `thread_stack` and `max_allowed_packet` on the configurations files of MySQL.

Query Rewriting and Instance Retrieval.

The main task in the experiments is related to rewriting queries and finding the instances of the result in SQL databases. In order to execute the task we need to use the following command:

```
java -jar completo.jar \
   -o [ontology path] -q [queries path]  \
   (-qi [index of the query to be rewritten])  \
   -c [constraints path] -aboxname [abox SQL name] \
   (-apath [path to output the answers of the queries])
```

where all the queries in the file will be rewritten one by one unless an index is specified with the `-qi` option is specified and in such a case only the query in the corresponding index will be rewritten.

In case we are only interested in the rewritings of the queries we should provide the file to output the rewritings and remove the information about the Abox:

```
java -jar completo.jar \
   -o [ontology path] -q [queries path]  \
   (-qi [index of the query to be rewritten])  \
   -c [constraints path] \
   (-qrewritings [path to output the rewritings])
```

Appendix B

ECOMPLETO's user manual

ECOMPLETO is a query rewriting system that supports queries with universally quantified negation and ontologies with disjunctive existential rules. It Provides a UCQ rewriting that can be used to find all the answers of the input query with respect to the rules in the ontology.

Running

- From bash:

```
time mix run -e "ECompleto.Experiments.rewrite(
        'ontologies/travel.dlgp',
        'ontologies/travel.queries2.txt'
        ) |> Enum.map(&(\"#{&1}\"))"
```

- From iex -S mix

```
iex(1)> ECompleto.Experiments.rewrite(
    'ontologies/travel.dlgp',
    'ontologies/travel.queries.txt',
    0)
```

Usage Examples

- To get the rewritings of a query

```
ECompleto.Experiments.rewrite(
    'experiments/AGOSUV-bench/A/A.dlp',
    'experiments/AGOSUV-bench/A/A_queries.dlp',
    0)
    |>  ECompleto.Program.to_file(
    'experiments/AGOSUV-bench/A/A_rewritings_0_ecompleto.dlgp'
    )
```

- To get the answers of a query

```
ECompleto.Experiments.answer(
    'experiments/AGOSUV-bench/A/A.dlp',
    'experiments/AGOSUV-bench/A/A_queries.dlp',
    0)
    |>  ECompleto.Program.to_file(
    'experiments/AGOSUV-bench/A/A_answers_0_ecompleto.dlgp'
    )
```

DLGP+ Notation

DLGP+ is an extension of the existing DLGP v2.0 notation that allows the specification of disjunctive existential rules and negated atoms in queries.

Disjunction is specified in the head of a rule by writing a list in squared brackets. The disjoint elements can be a single atom or several atoms enclosed in brackets, e.g.,

```
[disj] [leaf(X), (inner_node(X), edge(X,Y))] :- node(X).
```

Negation in queries with negated atoms is specified with the minus symbol, e.g.,

```
[q] ? :- person(X), -marriedTo(X,Y).
```

Notes

Consider that currently, Skolem functions use the variable name as the functor of the Skolem term. This is okay if we do rewriting with respect to constraint clauses and never propagate those Skolem terms. However, the Chase algorithms need to ensure that the nulls generated have a globally unique ID. Implement some sort of substitution for the prefixes in the OWL notation inside the dlgp programs.

Bibliography

[1] Leonid Libkin. *Incomplete information and certain answers in general data models*. *Proceedings of the 30th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2011, June 12-16, 2011, Athens, Greece*, Maurizio Lenzerini και Thomas Schwentick, επιμελητές, σελίδες 59–70. ACM, 2011.

[2] Adrian Onet. *The Chase Procedure and its Applications in Data Exchange*. *Data Exchange, Integration, and Streams*, Phokion G. Kolaitis, Maurizio Lenzerini και Nicole Schweikardt, επιμελητές, τόμος 5 στο *Dagstuhl Follow-Ups*, σελίδες 1–37. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2013.

[3] Mélanie König, Michel Leclère, Marie-Laure Mugnier και Michaël Thomazo. *Sound, complete and minimal UCQ-rewriting for existential rules*. *Semantic Web*, 6(5):451–475, 2015.

[4] Stanislav Kikot, Roman Kontchakov, Vladimir V. Podolskii και Michael Zakharyaschev. *Long Rewritings, Short Rewritings*. *Proceedings of the 2012 International Workshop on Description Logics, DL-2012, Rome, Italy, June 7-10, 2012*, Yevgeny Kazakov, Domenico Lembo και Frank Wolter, επιμελητές, τόμος 846 στο *CEUR Workshop Proceedings*. CEUR-WS.org, 2012.

[5] Jean-François Baget, Michel Leclère, Marie-Laure Mugnier και Eric Salvat. *On rules with existential variables: Walking the decidability line*. *Artif. Intell.*, 175(9-10):1620–1654, 2011.

[6] Georg Gottlob, Marco Manna, Michael Morak και Andreas Pieris. *On the Complexity of Ontological Reasoning under Disjunctive Existential Rules*. *Mathematical Foundations of Computer Science 2012 - 37th International Symposium, MFCS 2012, Bratislava, Slovakia, August 27-31, 2012. Proceedings*, Branislav Rovan, Vladimiro Sassone και Peter Widmayer, επιμελητές, τόμος 7464 στο *Lecture Notes in Computer Science*, σελίδες 1–18. Springer, 2012.

[7] David Carral, Irina Dragoste και Markus Krötzsch. *Tractable Query Answering for Expressive Ontologies and Existential Rules*. *The Semantic Web - ISWC 2017 - 16th International Semantic Web Conference, Vienna, Austria, October 21-25, 2017, Proceedings, Part I*, Claudia d'Amato, Miriam Fernández, Valentina A. M. Tamma, Freddy Lécué, Philippe Cudré-Mauroux, Juan F. Sequeda, Christoph Lange και Jeff Heflin, επιμελητές, τόμος 10587 στο *Lecture Notes in Computer Science*, σελίδες 156–172. Springer, 2017.

[8] Shqiponja Ahmetaj, Magdalena Ortiz και Mantas Simkus. *Rewriting Guarded Existential Rules into Small Datalog Programs*. *21st International Conference on Database Theory, ICDT 2018, March 26-29, 2018, Vienna, Austria*, Benny Kimelfeld και Yael Amsterdamer, επιμελητές, τόμος 98 στο *LIPIcs*, σελίδες 4:1–4:24. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

[9] Vince Bárány, Michael Benedikt και Balderten Cate. *Rewriting Guarded Negation Queries*. *Mathematical Foundations of Computer Science 2013 - 38th International Symposium, MFCS 2013, Klosterneuburg, Austria, August 26-30, 2013. Proceedings*, Krishnendu Chatterjee και Jirí Sgall, επιμελητές, τόμος 8087 στο *Lecture Notes in Computer Science*, σελίδες 98–110. Springer, 2013.

[10] Georg Gottlob, Sebastian Rudolph και Mantas Simkus. *Expressiveness of guarded existential rule languages*. *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS'14, Snowbird, UT, USA, June 22-27, 2014*, Richard Hull και Martin Grohe, επιμελητές, σελίδες 27–38. ACM, 2014.

[11] Meghyn Bienvenu, Balderten Cate, Carsten Lutz και Frank Wolter. *Ontology-Based Data Access: A Study through Disjunctive Datalog, CSP, and MMSNP*. *ACM Trans. Database Syst.*, 39(4):33:1–33:44, 2014.

[12] Riccardo Rosati. *The Limits of Querying Ontologies*. *Database Theory - ICDT 2007, 11th International Conference, Barcelona, Spain, January 10-12, 2007, Proceedings*, Thomas Schwentick και Dan Suciu, επιμελητές, τόμος 4353 στο *Lecture Notes in Computer Science*, σελίδες 164–178. Springer, 2007.

[13] Riccardo Rosati. *On the decidability and finite controllability of query processing in databases with incomplete information*. *Proceedings of the Twenty-Fifth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 26-28, 2006, Chicago, Illinois, USA*, Stijn Vansummeren, επιμελητής, σελίδες 356–365. ACM, 2006.

[14] Serge Abiteboul, Richard Hull και Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[15] Víctor Gutiérrez-Basulto, Yazmin Angélica Ibáñez-García, Roman Kontchakov και Egor V. Kostylev. *Conjunctive Queries with Negation over DL-Lite: A Closer Look*. *Web Reasoning and Rule Systems - 7th International Conference, RR 2013, Mannheim, Germany, July 27-29, 2013. Proceedings*, Wolfgang Faber και Domenico Lembo, επιμελητές, τόμος 7994 στο *Lecture Notes in Computer Science*, σελίδες 109–122. Springer, 2013.

[16] Vince Bárány, Balderten Cate και Martin Otto. *Queries with Guarded Negation*. *Proc. VLDB Endow.*, 5(11):1328–1339, 2012.

[17] Jianfeng Du και Jeff Z. Pan. *Rewriting-Based Instance Retrieval for Negated Concepts in Description Logic Ontologies*. *The Semantic Web - ISWC 2015 - 14th International Semantic Web Conference, Bethlehem, PA, USA, October 11-15, 2015, Proceedings, Part I*, Marcelo

Arenas, Óscar Corcho, Elena Simperl, Markus Strohmaier, Mathieu d'Aquin, Kavitha Srinivas, Paul T. Groth, Michel Dumontier, Jeff Heflin, Krishnaprasad Thirunarayan και Steffen Staab, επιμελητές, τόμος 9366 στο *Lecture Notes in Computer Science*, σελίδες 339–355. Springer, 2015.

[18] Enrique Matos Alfonso και Giorgos Stamou. *Rewriting Queries with Negated Atoms*. *Rules and Reasoning - International Joint Conference, RuleML+RR 2017, London, UK, July 12-15, 2017, Proceedings*, Stefania Costantini, Enrico Franconi, William Van Woensel, Roman Kontchakov, Fariba Sadri και Dumitru Roman, επιμελητές, τόμος 10364 στο *Lecture Notes in Computer Science*, σελίδες 151–167. Springer, 2017.

[19] Enrique Matos Alfonso και Giorgos Stamou. *On Horn Conjunctive Queries*. *Rules and Reasoning - Second International Joint Conference, RuleML+RR 2018, Luxembourg, September 18-21, 2018, Proceedings*, Christoph Benzmüller, Francesco Ricca, Xavier Parent και Dumitru Roman, επιμελητές, τόμος 11092 στο *Lecture Notes in Computer Science*, σελίδες 115–130. Springer, 2018.

[20] Pierre Bourhis, Marco Manna, Michael Morak και Andreas Pieris. *Guarded-Based Disjunctive Tuple-Generating Dependencies*. *ACM Trans. Database Syst.*, 41(4):27:1–27:45, 2016.

[21] Jean-François Baget, Michel Leclère, Marie-Laure Mugnier και Eric Salvat. *Extending Decidable Cases for Rules with Existential Variables*. *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*, Craig Boutilier, επιμελητής, σελίδες 677–682, 2009.

[22] , Shan-Hwei Nienhuys-Cheng και Ronaldde Wolf, επιμελητές. *Foundations of Inductive Logic Programming*, τόμος 1228 στο *Lecture Notes in Computer Science*. Springer, 1997.

[23] Sergio Tessaris. *Questions and answers: reasoning and querying in Description Logic*. Διδακτορική Διατριβή, University of Manchester, 2001.

[24] Jean-François Baget. *Improving the Forward Chaining Algorithm for Conceptual Graphs Rules*. *Principles of Knowledge Representation and Reasoning: Proceedings of the Ninth International Conference (KR2004), Whistler, Canada, June 2-5, 2004*, Didier Dubois, Christopher A. Welty και Mary-Anne Williams, επιμελητές, σελίδες 407–414. AAAI Press, 2004.

[25] *NCI Dictionary of Cancer Terms: first-degree relative*. `https://www.cancer.gov/publications/dictionaries/cancer-terms/def/first-degree-relative`.

[26] D. Trivela, G. Stoilos, A. Chortaras και G. Stamou. *Optimising Resolution-Based Rewriting Algorithms for OWL Ontologies*. 2015.

[27] Georg Gottlob, Giorgio Orsi και Andreas Pieris. *Query Rewriting and Optimization for Ontological Databases*. *ACM Trans. Database Syst.*, 39(3):25:1–25:46, 2014.

[28] Jean François Baget, Michel Leclère, Marie Laure Mugnier, Swan Rocher και Clément Sipieter. *Graal: A Toolkit for Query Answering with Existential Rules*, σελίδες 328–344. Springer International Publishing, Cham, 2015.

[29] Jean-François Baget και Marie-Laure Mugnier. *Extensions of Simple Conceptual Graphs: the Complexity of Rules and Constraints*. *J. Artif. Intell. Res.*, 16:425–465, 2002.

[30] Andrea Calì, Georg Gottlob και Andreas Pieris. *Advanced Processing for Ontological Queries. PVLDB*, 3:554–565, 2010.

[31] Yuanbo Guo, Zhengxiang Pan και Jeff Heflin. *LUBM: A benchmark for OWL knowledge base systems*. *J. Web Semant.*, 3(2-3):158–182, 2005.

[32] Christian Bizer, Jens Lehmann, Georgi Kobilarov, Sören Auer, Christian Becker, Richard Cyganiak και Sebastian Hellmann. *DBpedia - A Crystallization Point for the Web of Data*. *Web Semant.*, 7(3):154–165, 2009.

[33] Dmitry Tsarkov και Ian Horrocks. *FaCT++ Description Logic Reasoner: System Description*. *Proceedings of the Third International Joint Conference on Automated Reasoning*, IJCAR'06, σελίδες 292–297, Berlin, Heidelberg, 2006. Springer-Verlag.

[34] Birte Glimm, Ian Horrocks, Boris Motik, Giorgos Stoilos και Zhe Wang. *HermiT: An OWL 2 Reasoner*. *Journal of Automated Reasoning*, 53(3):245–269, 2014.

[35] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann και Ian H. Witten. *The WEKA Data Mining Software: An Update*. *SIGKDD Explor. Newsl.*, 11(1):10–18, 2009.

[36] Enrique Matos Alfonso, Alexandros Chortaras και Giorgos Stamou. *UCQ-Rewritings for disjunctive knowledge and queries with negated atoms*. *Semantic Web*, 12(4):685–709, 2021.

[37] Jean-François Baget, Alain Gutierrez, Michel Leclère, Marie-Laure Mugnier, Swan Rocher και Clément Sipieter. *Datalog+, RuleML and OWL 2: Formats and Translations for Existential Rules*. *Proceedings of the RuleML 2015 Challenge, the Special Track on Rule-based Recommender Systems for the Web of Data, the Special Industry Track and the RuleML 2015 Doctoral Consortium hosted by the 9th International Web Rule Symposium (RuleML 2015), Berlin, Germany, August 2-5, 2015*, Nick Bassiliades, Paul Fodor, Adrian Giurca, Georg Gottlob, Tomás Kliegr, Grzegorz J. Nalepa, Monica Palmirani, Adrian Paschke, Mark Proctor, Dumitru Roman, Fariba Sadri και Nenad Stojanovic, επιμελητές, τόμος 1417 στο *CEUR Workshop Proceedings*. CEUR-WS.org, 2015.

[38] Rakesh Agrawal, Tomasz Imielinski και Arun N. Swami. *Mining Association Rules between Sets of Items in Large Databases*. *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, Washington, DC, USA, May 26-28, 1993*, Peter Buneman και Sushil Jajodia, επιμελητές, σελίδες 207–216. ACM Press, 1993.