Master Thesis

Ontological Query Rewriting: Termination Criteria

Frederikos Leandrou

UNIVERSITY OF CYPRUS

DEPARTMENT OF COMPUTER SCIENCE

December 2022

ABSTRACT

Ontological queries are queries evaluated against a database and an ontology, i.e. a set of logic rules and constraints from which new knowledge can be derived from. Ontological database systems can thus be more powerful than traditional database systems. A smooth transition between the two requires their connection which comes in the form of rewriting ontological queries into equivalent ones for traditional databases, thus leading to the creation of algorithms that do that. Using the ontology is an iterative process and as such the termination of these rewrite algorithms comes into question. We focus on one such algorithm, firstly going through its basics and workings, and then exploring the cases that will lead to its termination, by applying restrictions to the form of the ontology the algorithm accepts as input. In particular, we find the size of the obtained rewriting in case of non-recursive ontology and provide proof of termination for a less restrictive case of ontology.

UNIVERSITY OF CYPRUS
DEPARTMENT OF COMPUTER SCIENCE

ONTOLOGICAL QUERY REWRITING: TERMINATION CRITERIA

Frederikos Leandrou

Supervisor
Andreas Pieris

This Master Thesis was submitted as part of the requirements needed for obtaining a Master's degree in Computer Science from the Department of Computer Science of the University of Cyprus

December 2022

APPROVAL PAGE

Master of Computer Science Thesis

ONTOLOGICAL QUERY REWRITING: TERMINATION CRITERIA

Presented by

Frederikos Leandrou

Research Supervisor _____

Research Supervisor's Name

Committee Member _____

Committee Member's Name

Committee Member _____

Committee Member's Name

University of Cyprus

December, 2022

Acknowledgements

I would like to express my heartfelt gratitude to my supervisor Andreas Pieris for the opportunity to work with him and introducing me to this subject. Without their knowledge, experience and great patience, I would not have been able to complete this thesis.

Table of Contents

LIST OF FIGURES

Chapter 1

Introduction

1.1 Motivation

Ontology, i.e. the conceptualization of a subject area showing the properties and relations between instances of these conceptualizations, has been adopted for use in data repositories and models, which can sometimes be distributed and heterogeneous. As ontologies can offer high expressive power they are starting to replace traditional data and conceptual models such as UML class diagrams and Entity Relationship schemata.

The use of ontologies in database technology created the ontological database management system. Using advanced reasoning and query processing mechanics, a database is combined with an ontology that is used to produce additional information from the database that is not explicitly contained in it. This automated production of data via reasoning provides certain flexibility to the information in the database while the ontology and the information it represent can easily be extended while also being easier for user to navigate due to thinking in terms of concepts.

As such ontological databases are a reasonable next step to database usage, if only restrained by the transition from traditional databases to ontological databases, how they can interconnect and the complexity of applying queries to every model created by the ontology. A way to introduce ontology to traditional database systems is to implement the higher levels of ontology as a façade and leave the execution of the queries to the traditional databases. This will require the translation of ontological queries into ones compatible for use in these traditional databases, thus leading to the creation of ontological query rewriting algorithms.

An ontological query rewriting algorithm will take ontological queries and rewrite them into equivalent first-order queries. This is assisted by the fact that a way to model the ontology is through the use of tuple-generating dependencies (TGDs), a type of traditional database constraints. These TGDs are of the form: $\forall X \forall Y\ \varphi(X,Y) \rightarrow \exists Z\ \psi(X,Z)$, where φ and ψ are a conjunction of atoms over a relational schema. In essence these TGDs are rules from which additional knowledge can be derived from, so a query q combined with an ontology may contain more information than what it states in itself. As shown in *Figure 1*, the rewriting algorithm will take such query q, compile it with the ontology Σ, essentially get all the information hidden in it and produce an equivalent first-order query q_Σ, that contains all that information in itself. These first-order queries can then be used in the current more widespread databases.

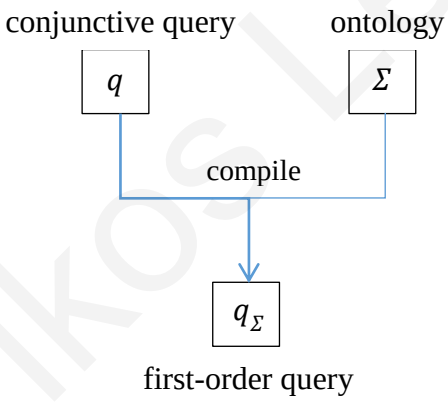

Figure 1. Ontological query rewriting process

Example 1: Consider set Σ consisting of TGD:

$$\forall X \forall Y\ related(X,Y), parent(X,Y) \rightarrow \exists Z\ created(Z,X,Y)$$

asserting that for every two individuals if they are related and one of them is the parent of the other, then there exists another individual that also created that individual. We can ask for who created Bob by posing the query q: $\exists A\ created(A,B,Bob)$ but at the same time we also have to

check for individuals that are related and a parent to Bob as per the TGD above that means they created him. So query q becomes query q_Σ :

$(\exists A\ created(A, B, Bob)) \vee (related(B, Bob) \wedge parent(B, Bob))$, with the term not included in query q being knowledge derived from the ontology.

The existence of these rewriting algorithms can be of great help as they can help with the transition to ontological database systems and provide the advantages of ontology to existing databases systems by allowing them to be slowly changed while continuing their normal operation. The main algorithm that we focus on is a rewriting algorithm called XRewrite.

1.2 Objective

Our objective in this thesis concerns an algorithm for ontological query rewriting, called XRewrite and introduced in the paper "Query Rewriting and Optimization for Ontological Databases" [1]. More specifically, we focus on its ability to terminate.

The algorithm XRewrite presented in the aforementioned paper is an algorithm that translates queries for ontological databases into equivalent ones to be used with conventional databases. Due to certain attributes in the rewriting process, there are cases in which the termination of the algorithm is not always guaranteed. We will detail the cases in which we know that the algorithm terminates but also explore additional cases in which we intuitively know that the algorithm terminates but have no concrete proof to that fact, with the intent to provide proof that they do.

1.3 Methodology and Contributions

With the introduction to the topic of query rewriting for ontological databases and the algorithm of XRewrite, firstly we go through the basic terms and ideas used in these concepts. Then we will go through an overview of the algorithm and present how it works and the problems that occur with its termination, or lack thereof. Following that we will present four syntactic classes under which the algorithm terminates along with proof of termination for each class. These

syntactic classes limit the form that the TGDs and thus the ontology can take but help with guaranteeing the termination of the algorithm. These classes are Non-Recursiveness, Linearity and a Multi-linearity Special Case, and Stickiness. Termination under these classes guarantees the translation and thus successful execution of an ontological query on any ontology that can be described using only TGDs of that class. Under the Non-Recursive class of TGDs, the successful execution of an ontological query on any non-recursive ontology is guaranteed. Linear TGDs are more expressive than the description logic DL-Lite$_R$ [2], which forms the OWL 2 QL profile of W3C's standard ontology language for the Semantic Web, as well as being useful in modelling hierarchies. With Linearity being a subclass of Multi-linearity, Multi-linear TGDs are more expressive than linear TGDs. Multi-linearity has the goal of defining a natural formalism strictly more expressive than DL-Lite$_{R,\sqcap}$, the extended version of DL-Lite$_R$ which allows for concept conjunction [3]. Stickiness allows joins to appear in rule-bodies not expressible with linear TGDs or DL(R)-Lite assertions, and can be used to encode the Cartesian product of two tables, thus being able to describe knowledge whose underlying relation structure is not treelike [1].

1.4 Document Structure

The remainder of the paper is organized as follows. In Chapter 2 we present the background of the subject of our study. At first, basic terms and terminology we need to know in order to understand the algorithm and the problem are defined. In Chapter 3 we go through an overview and explanation of how the algorithm XRewrite works. Chapters 4 to 6, are an analysis of the syntactic classes under which the algorithm terminates. Chapter 4 is for Non-Recursiveness, Chapter 5 for Linearity and the Multi-linear Special Case, and Chapter 6 for Stickiness. Lastly, Chapter 7 is the conclusion, summing up our findings and mentioning further work that can be done on the subject.

1.5 Related Work

Research done on the topic of query rewriting algorithms includes an early algorithm for DL-lite family of Description Logics [2] and implemented in the QuOnto system. This algorithm also translates the query into a union of conjunctive queries but, as a result of the redundant application of the factorization step, the resulting queries are unnecessarily large. A fix to this problem is introduced by a resolution-based rewriting for DL-lite$_R$ implemented in the Requiem system [4], solving the problem by directly handling existential quantification through proper functional terms. A more efficient algorithm, called Rapid, uses selective and stratified applications of resolution rules, taking advantage of the query's structure to reduce redundant rewritings [5]. These algorithms use specifics of DLs so they do not easily extend to TGD-based languages. A more general approach using a backward-chaining rewriting algorithm is able to deal with arbitrary TGDs, as long as the language used satisfies suitable syntactic restrictions that guarantee the algorithm's termination [6], [7], [8].

Chapter 2

Background

Our study is based on the paper "Query Rewriting and Optimization for Ontological Databases" by Georg Gottlob, Giorgio Orsi, and Andreas Pieris [1]. The paper introduces an algorithm, called XRewrite, which can translate queries for ontological databases into a union of conjunctive queries, a fragment of SQL, for their evaluation and in order to exploit the widespread existing database technology.

In order to understand the algorithm XRewrite we need to be familiar with some basics of the field of relational databases, relational queries, tuple-generating dependencies, and the chase procedure relative to such dependencies [1].

Alphabets: We consider the following disjoint sets of symbols:

- Γ: A set of constants, the normal domain of a database, each one represents a different value.
- Γ_N: A set of labeled nulls, placeholders for unknown values and viewed as globally existentially quantified variables, different nulls may represent the same value.
- Γ_V: A set of regular variables, used in queries and dependencies.

Relational Model: A *Relational schema R* (or *schema*), is a set of relational symbols, or predicates, each with its associated arity. With r/n we denote predicate r which has arity n. By $arity(R)$ we refer to maximum arity of all predicates of R. A *position* $r[i]$ in R is identified by predicate $r \in R$ and its i-th argument. A *term* t is a constant, null or variable. An *atomic formula*, or *atom*, has the form $r(t_1, \dots, t_n)$, where r/n is a relation and $t_1, \dots, t_n$ are terms. For *atom* a, $terms(a)$ and $var(a)$ are the set of its terms and the set of its variables, respectively,

with the notations also extending to sets of atoms. Conjunctions of atoms are often identified by the sets of their atoms. An *instance I* for *schema R* is a possibly infinite set of atoms of the form $r(t)$, where $r/n \in R$ and $t \in (\Gamma \cup \Gamma_N)^n$. A *database D* is a finite instance such that $terms(D) \subset \Gamma$.

Substitutions: A *substitution* from a set of symbols S to set of symbols S' is a function $h: S \to S'$ defined as follows: $\emptyset$ is an empty substitution and, if h is a substitution, then $h \cup \{t \to t'\}$ is a substitution, where $t \in S$ and $t' \in S'$; if $t \to t' \in h$, then we write $h(t) = t'$. An assertion of the form $t \to t'$ is called *mapping*. The restriction of h to $T \subseteq S$, represented as $h|_T$, is the substitution $h' = \{t \to h(t) \mid t \in T\}$. A *homomorphism* from a set of atoms A to set of atoms A' is a substitution

$h: \Gamma \cup \Gamma_N \cup \Gamma_V \to \Gamma \cup \Gamma_N \cup \Gamma_V$ such that if $t \in \Gamma$, then $h(t) = t$ and if $r(t_1, \dots, t_n) \in A$, then $h\big(r(t_1, \dots, t_n)\big) = r(h(t_1), \dots, h(t_n)) \in A'$. A set of atoms $A = \{a_1, \dots, a_n\}$, where $n \geq 2$, *unifies* if there is a substitution γ, called *unifier* for A, such that,

$\gamma(a_1) = \cdots = \gamma(a_n)$. A *most general unifier (MGU)* for A is a unifier for A, γ_A, such that for each other unifier for A, there is a substitution γ' such that $\gamma = \gamma' \circ \gamma_A$. If a set of atoms unify, a MGU exists and the MGU for a set is always unique, up to variable renaming.

Queries: An n-ary first-order query q is an expression $\varphi(x_1, \dots, x_n)$, where $x_1, \dots, x_n$ are exactly the free variables of first-order formula φ. An answer to q over instance I is a tuple $(c_1, \dots, c_n)$ of constants such that $I \vDash \varphi(x_1/c_1, \dots, x_n/c_n)$, i.e. I satisfies $\varphi(x_1/c_1, \dots, x_n/c_n)$, where $\varphi(x_1/c_1, \dots, x_n/c_n)$ is φ with each free x_i replaced by c_i. A conjunctive query (CQ) q of arity n over schema R is an assertion of the form $p(X) \leftarrow \varphi(X, Y)$, where $X \cup Y \subset \Gamma \cup \Gamma_V$, φ is a conjunction of atoms over R, with φ also known as $body(q)$, and p is an n-ary predicate not occurring in R.

Tuple Generating Dependencies: A *tuple generating dependency (TGD)* σ over schema R is a first-order formula $\forall \boldsymbol{X} \forall \boldsymbol{Y}\, \varphi(\boldsymbol{X}, \boldsymbol{Y}) \rightarrow \exists \boldsymbol{Z}\, \psi(\boldsymbol{X}, \boldsymbol{Z})$, where $\boldsymbol{X} \cup \boldsymbol{Y} \cup \boldsymbol{Z} \subset \Gamma_V$ and where φ, ψ are

conjunctions of atoms over R, e.g. $\forall X\, human(X) \rightarrow \exists Y\, parent(Y, X)$, $\forall X \forall Y\, dog(X), owner(Y, X) \rightarrow \exists Z\, walktime(Z, Y, X)$. φ is the body of σ, $body(\sigma)$, while ψ is the head of σ, $head(\sigma)$. For brevity, universal quantifiers in front of TGDs will be omitted and commas will be used for the conjunction. σ is satisfied by instance I for R, written $I \vDash \sigma$, if the following is true: whenever there exists a homomorphism h such that $h(\varphi(\boldsymbol{X}, \mathbf{Y})) \subseteq I$, then there exists homomorphism $h' \supseteq h|_{\boldsymbol{X}}$, called extension of $h|_{\boldsymbol{X}}$ such that $h'(\psi(\boldsymbol{X}, \mathbf{Z})) \subseteq I$. Instance I satisfies set of TGDs, written $I \vDash \Sigma$, if $I \vDash \sigma$ for every $\sigma \in \Sigma$.

Conjunctive Query Answering under TGDs: Given database D for schema R and set of TGDs Σ over R, the answers we consider are those that are true in all models of D w.r.t. Σ. The models of D w.r.t. Σ, denoted $mods(D, \Sigma)$, is the set of all instances I such that $I \supseteq D$ and $I \vDash \Sigma$. The answer to an n-ary CQ q w.r.t. D and Σ, denoted $ans(q, D, \Sigma)$, is the set of n-tuples $\{\boldsymbol{t} \mid \boldsymbol{t} \in q(I),\ for\ each\ I \in mods(D, \Sigma)\}$.

The TGD Chase Procedure: The *chase procedure* or *chase* is a fundamental algorithmic tool for checking implication of dependencies [9] and checking query containment [10]. The *chase procedure* is an iterative application of the so-called *TGD chase rule*.

> *TGD chase rule*: Consider instance I for schema R, and TGD $\sigma: \varphi(\boldsymbol{X}, \boldsymbol{Y}) \rightarrow \exists \boldsymbol{Z}\, \psi(\boldsymbol{X}, \boldsymbol{Z})$ over R. σ is *applicable* to I if there exists a homomorphism h such that $h(\varphi(\boldsymbol{X}, \mathbf{Y})) \subseteq I$. The result of *applying* σ to I with h is $I' = I \cup h'(\psi(\boldsymbol{X}, \mathbf{Z}))$ and we write $I\langle\sigma, h\rangle I'$, where h' is an extension of $h|_{\boldsymbol{X}}$ such that $h'(\boldsymbol{Z})$ is a new labeled null of Γ_N not occurring in I, and following lexicographically all those in I, for each $Z \in \boldsymbol{Z}$. $I\langle\sigma, h\rangle I'$ defines a single TGD chase step.

In short, the *chase procedure* draws conclusions. A TGD σ of the form: $\varphi \rightarrow \psi$, can essentially be thought of as a rule that says: *if φ is true, then ψ is true as well*. So given a set of statements, i.e. instance I, the *chase* will check whether any conclusions can be derived from applying the rules to the statements. For example, if our statements say that φ is true, we can conclude that ψ

will be true as well and so part of our statements as well. This process of drawing conclusions will be repeated using all statements and all rules that are given. This repetition may be finite or continue infinitely; depending on the rules it is given.

Formally, a *chase sequence* of database D w.r.t. set of TGDs Σ is a sequence of chase steps $I_i\langle\sigma_i, h_i\rangle I_{i+1}$, where $i \geq 0$, $I_0 = D$ and $\sigma_i \in \Sigma$. The chase of D w.r.t. Σ, denoted $chase(D, \Sigma)$, is defines as follows:

- A finite chase of D w.r.t. Σ is a finite chase sequence $I_i\langle\sigma_i, h_i\rangle I_{i+1}$, where $0 \leq i < m$ and there is no $\sigma \in \Sigma$ applicable to I_m, where $I_m = chase(D, \Sigma)$.
- An infinite chase sequence $I_i\langle\sigma_i, h_i\rangle I_{i+1}$, where $i \geq 0$, is fair if whenever a TGD σ: $\varphi(\boldsymbol{X}, \boldsymbol{Y}) \rightarrow \exists \boldsymbol{Z}\, \psi(\boldsymbol{X}, \boldsymbol{Z})$ is applicable to I_i with homomorphism h, there exists extension h' of $h|_{\boldsymbol{X}}$ and $k > i$ such that $h'(head(\sigma)) \subseteq I_k$. An infinite chase of D w.r.t. Σ is a fair infinite chase sequence $I_i\langle\sigma_i, h_i\rangle I_{i+1}$, where $i \geq 0$; let $chase(D, \Sigma) = \bigcup_{i=0}^{\infty} I_i$.

Example 2: Consider instance $I = \{R(a, b)\}$ for schema R, and set of TGDs Σ: $\{\sigma_1: R(X, Y) \rightarrow \exists Z\, S(X, Z),\ \sigma_2: S(X, Y) \rightarrow \exists Z\, T(X, Z)$ over R. There exists homomorphism $h_1 = \{X \rightarrow a, Y \rightarrow b\}$ where $h_1\big(R(X, Y)\big) = R(a, b) \subseteq I$ and as such σ_1 is applicable to I. Applying σ_1 to I withh_1, a.k.a.$\langle\sigma_1, h_1\rangle$, gives us $I_1 = I \cup h_1(S(X, Z)) = I \cup \{S(a, z_1)\}$ where z_1 is a new labeled null. Then there exists $h_2 = \{X \rightarrow a, Y \rightarrow z_1\}$ where $h_2\big(S(X, Y)\big) = S(a, z_1) \subseteq I_1$ and thus $\langle\sigma_2, h_2\rangle$ gives us $I_2 = I_1 \cup h_2\big(T(X, Z)\big) = I_1 \cup \{T(a, z_1)\} = I \cup \{S(a, z_1), T(a, z_1)\}$. As there is no $\sigma \in \Sigma$ applicable to I_2 then $chase(I, \Sigma) = I_2 = \{R(a, b), S(a, z_1), T(a, z_1)\}$ where z_1 is a null of Γ_N.

The chase of D w.r.t. Σ is a universal model of D w.r.t. Σ, i.e. for each $I \in mods(D, \Sigma)$ there exists homomorphism h_I such that $h_I(chase(D, \Sigma)) \subseteq I$, [11], [12]. With this property the chase becomes a formal algorithmic tool for answering queries under TGDs, as the answer to CQ q w.r.t. database D and set of TGDs Σ corresponds with the answer to q over chase of D w.r.t. Σ, that is $ans(q, D, \Sigma) = q(chase(D, \Sigma))$.

Note that the TGD chase rule given above is oblivious, i.e. it does not check whether the TGD under consideration is already satisfied and adds atoms in the instance even if not needed. There also exists a version of the rule with stricter criteria, called restricted, with the aim of adding only the atoms necessary, which is considered the standard, [11], [12].

Normal Form: A TGD σ is in *normal form* if its head has only one atom, i.e. $|head(\sigma)| = 1$, and its head contains only one occurrence of an existential quantifier variable. A set of TGDs Σ is in *normal form* if each TGD $\sigma \in \Sigma$, is in *normal form*. Every set Σ of TGDs over schema R can be transformed into a normal form set of $N(\Sigma)$ over schema $R_{N(\Sigma)}$, such that Σ and $N(\Sigma)$ are equivalent w.r.t. query answering. For a TGD $\sigma \in \Sigma$, if σ is in normal form then $N(\sigma) = \{\sigma\}$, else assuming $\{a_1, \dots, a_k\} = head(\sigma)$, $\{X_1, \dots, X_n\} = var\big(body(\sigma)\big) \cap var\big(head(\sigma)\big)$, and $Z_1, \dots, Z_m$ are the existential quantified variables of σ, let $N(\sigma)$ be the set:

$$body(\sigma) \rightarrow \exists Z_1 \, p_\sigma^1(X_1, \dots, X_n, Z_1)$$

$$p_\sigma^1(X_1, \dots, X_n, Z_1) \rightarrow \exists Z_2 \, p_\sigma^2(X_1, \dots, X_n, Z_1, Z_2)$$

...

$$p_\sigma^{m-1}(X_1, \dots, X_n, Z_1, \dots, Z_{m-1}) \rightarrow \exists Z_m \, p_\sigma^m(X_1, \dots, X_n, Z_1, \dots, Z_m)$$

$$p_\sigma^m(X_1, \dots, X_n, Z_1, \dots, Z_m) \rightarrow a_1$$

...

$$p_\sigma^m(X_1, \dots, X_n, Z_1, \dots, Z_m) \rightarrow a_k$$

where p_σ^i is an $(n+1)$-ary auxiliary predicate not occurring in R, for each $i \in [m]$. Let $N(\Sigma) = \bigcup_{\sigma \in \Sigma} N(\sigma)$ and $R_{N(\Sigma)}$ be the schema obtained by adding to R the auxiliary predicates in $N(\Sigma)$.

As such, the algorithm XRewrite assumes that the TGDs it is given are in *normal form*.

Example 3: Consider TGD: σ: $A(X,Y) \rightarrow \exists Z\ B(X,Z), C(Y,Z)$, which is not in normal form. σ can be transformed into set $N(\sigma)$ in normal form such that σ and $N(\sigma)$ are equivalent w.r.t. query answering. $N(\sigma)$ will be the set: $\big(A(X,Y) \rightarrow \exists Z\ P(X,Y,Z)\big)$, $\big(P(X,Y,Z) \rightarrow B(X,Z)\big)$ and $\big(P(X,Y,Z) \rightarrow C(Y,Z)\big)$.

Chapter 3

Rewriting Algorithm XRewrite

With the basics covered, the algorithm XRewrite will follow. As previously mentioned, the goal of this algorithm is to accept as input a CQ q over a schema R and a set Σ of TGDs over R, and rewrite this query into equivalent ones q_Σ for use in standard query language, in particular as a union of conjunctive queries, a fragment of SQL. The q_Σ produced will be called a perfect rewriting, that is evaluating q_Σ over database D yields the same result as q evaluated over ontological database $D \cup S$.

The algorithm will use two new terms, *applicable* and *factorizable*, as part of its workings. *Applicable* concerns its applicability condition, in short, whether or not the algorithm can be used or not on that particular part of the query. *Factorizable* concerns the factorizability condition, in short, whether or not a set of atoms can be reduced to their MGU. The algorithm in essence consists of two sections, the *rewriting step* and the *factorization step*. The rewriting step is used if the applicability condition is satisfied and the factorization step is used if the factorizability condition is satisfied.

3.1 Applicability Condition of XRewrite

For the algorithm we assume without loss of generality that the variable occurring in queries and the variables appearing in TGDs are two distinct disjointed sets. Also, given CQ q, a variable is called *shared* in q if it appears in more than once in q. Note that distinguished variables of q are shared since they appear in both the body and head of q. With this the Applicability Condition follows:

Definition 1 (Applicability): Considering CQ q and TGD σ and given set of atoms $S \subseteq body(q)$, it is said that σ is *applicable* to S if the following are true:

1. The set $S \cup \{head(\sigma)\}$ unifies.
2. For each $\alpha \in S$, if the term at position π in α is either a constant or a shared variable in q, then $\pi \neq \pi_{\exists}(\sigma)$.

In short, for σ to be *applicable* to S, S and head of σ *unify*, and for every atom of S, terms that are either constants or shared variables are not in the same position as the position of the existential quantified variable of σ.

Example 4: Consider TGD $\sigma: m(X) \rightarrow \exists Y\ n(X,Y)$ and query $q: \underbrace{n(A,B), n(C,B)}_{S_1}, \underbrace{n(B,E)}_{S_2}$.

For S_2 we can see that σ is applicable to it, as $S_2 \cup \{head(\sigma)\}$ unifies using $\{B \rightarrow X, E \rightarrow Y\}$, thus fulfilling the first condition above. The second condition is also fulfilled as while B appears in other atoms as well and thus is a shared variable, its position in this particular atom is not the same position as the position of the existential variable Y in σ.

For atoms S_1, σ will not be applicable. While they fulfill the first condition, B is a shared variable and appears in them in the same position as the position of existential variable Y in σ, thus violating the second condition.

3.2 Factorizability Condition of XRewrtite

Expanding on the applicability condition is the concept of factorizability, upon which the factorization step of the algorithm is based. In short, its goal is to convert some shared variables into non-shared ones for the above applicability condition to apply to them. This is done by continuously unifying all atoms that unify in the query's body. In some cases this process does not help and produces redundant queries, thus requiring a restricted version of factorization that produces only the essential queries needed. This Factorizability Condition follows:

Definition 2 (Factorizability): Considering CQ q and TGD σ and given a set of atoms $S \subseteq body(q)$, where $|S| \geq 2$, it is said that S is *factorizable* w.r.t. σ if the following are true:

1. S unifies.
2. $\pi_{\exists}(\sigma) \neq \varepsilon$.
3. There exists variable $V \notin var(body(q) \backslash S)$ that occurs in every atom of S only at position $\pi_{\exists}(\sigma)$.

Example 5: Consider TGD $\sigma: m(X), n(X,Y) \rightarrow \exists Z\ o(X,Y,Z)$ and CQs $q_1: \underbrace{t(a,A,C), t(B,a,C)}_{S_1}$, $q_2: s(C), \underbrace{t(A,B,C), t(A,E,C)}_{S_2}$ and $q_3: \underbrace{t(A,B,C), t(A,C,C)}_{S_3}$, where $a \in \Gamma$.

Checking the factorizability of this example, we see that the second condition is true as σ has an existential quantifier. So we will check the CQs for the first and third conditions.

For S_1, the first condition is true, as it unifies w.r.t. σ using the substitution $\{A \rightarrow a, B \rightarrow a\}$. The third condition is also true as C appears in all atoms of S_1 at the same position as the position of existential variable Z in σ.

For S_2 and S_3, the third condition is violated, as for S_2 the variable C also appears in q_2 but outside of S_2, and for S_3 the variable C appears in two different positions, not only at the position of existential variable Z in σ.

3.2.1 Algorithm XRewrite

Following the above definitions, we can now present the algorithm:

Algorithm 1 The algorithm XRewrite

Input: a CQ q over a schema R and a set Σ of TGDs over R
Output: the perfect rewriting of q w.r.t. Σ

$i := 0$;
$\boldsymbol{Q}_{\text{REW}} := \{\langle q, r, u\rangle\}$;
repeat
  $\boldsymbol{Q}_{\text{TEMP}} := \boldsymbol{Q}_{\text{REW}}$;
  foreach $\langle q, x, u\rangle \in \boldsymbol{Q}_{\text{TEMP}}$, where $x \in \{r, f\}$ **do**
    foreach $\sigma \in \Sigma$ **do**
      // rewriting step
      foreach $S \subseteq body(q)$ such that σ is applicable to S **do**
        $i := i + 1$;
        $q' := \gamma_{S,\sigma^i}(q[S \,/\, body(\sigma^i)])$;
        if there is no $\langle q'', r, *\rangle$ such that $q' \simeq q''$ **then**
          $\boldsymbol{Q}_{REW} := \boldsymbol{Q}_{REW} \cup \{\langle q', r, u\rangle\}$;
        end
      end
      // factorization step
      foreach $S \subseteq body(q)$ which is factorizable w.r.t σ **do**
        $q' := \gamma_S(q)$;
        if there is no $\langle q'', *, *\rangle \in \boldsymbol{Q}_{REW}$ such that $q' \simeq q''$ **then**
          $\boldsymbol{Q}_{REW} := \boldsymbol{Q}_{REW} \cup \{\langle q', f, u\rangle\}$;
        end
      end
    end
    // query q is now explored
    $\boldsymbol{Q}_{REW} := (\boldsymbol{Q}_{REW} \backslash \{\langle q, x, u\rangle\}) \cup \{\langle q, x, e\rangle\}$;
  end
until $\boldsymbol{Q}_{TEMP} = \boldsymbol{Q}_{REW}$;
$\boldsymbol{Q}_{FIN} := \{q \mid \langle q, r, e\rangle \in \boldsymbol{Q}_{REW}\}$;
return $\boldsymbol{Q}_{FIN}$;

The algorithm consists of a number of loops but in short is an iterative application of two central steps, the *rewriting step* and the *factorization step*. These steps will be applied to every atom of every query in set of queries $\boldsymbol{Q}_{REW}$. $\boldsymbol{Q}_{REW}$ consists of both the initial CQ q and the rewritten queries produced by the two steps, noted as r and f for the queries produced by the *rewriting step* and *factorization step* respectively. This process is repeated exhaustively, until no further

changes can be observed in $\boldsymbol{Q}_{REW}$. This is done by comparing the $\boldsymbol{Q}_{REW}$ of the previous loop, now called $\boldsymbol{Q}_{TEMP}$, with the $\boldsymbol{Q}_{REW}$ produced in the current loop. To avoid redundancy, e and u are used to note which queries have already been explored or unexplored respectively. Once no further changes are observed in $\boldsymbol{Q}_{REW}$, and with all its queries being marked as explored, this means that the initial CQ q has been fully explored, with $\boldsymbol{Q}_{REW}$ being a set of all queries produced by this exploration. This set will be q_{Σ}, the result of running XRewrite with CQ q and Σ set of TGDs.

The essence of the algorithm can be thought of as running the *chase procedure* in reverse. Given a conclusion and a set of rules that arrive at said conclusion, we try to find what set of statements can be used with these rules to arrive at the given conclusion. This can be seen in the *rewriting step*, which looks like it runs the implication of the TGDs in reverse.

3.2.2 Rewriting Step

In the rewriting step of the algorithm, in simple terms, for every atom(s) of the query, if that atom(s) exists in the head of a TGD, it substitutes it with the body of that TGD. To do that however, the atom(s) must satisfy the applicability condition.

In the case of ***Example 4*** above, S_1 does not satisfy the condition and will not be rewritten. S_2 does satisfy the condition and so it goes through the rewriting step giving us q': $n(A,B), n(C,B), m(B)$. The correlation with the chase procedure in reverse can be seen as given atoms that unify with the head of a TGD, i.e. the conclusion, we substitute them with the body of a TGD, i.e. the statements that are used to arrive at that conclusion.

Formally, for each $S \subseteq body(q)$ where σ applicable to S, the i-th application of the rewriting step creates query $q' = \gamma_{S,\sigma^i}(q[S/body(\sigma^i)])$, with σ^i being the TGD obtained from σ by replacing every variable X with X^i, γ_{S,σ^i} being the MGU for set $S \cup \{head(\sigma^i)\}$, and $q[S/body(\sigma^i)]$ being obtained from q by replacing S with $body(\sigma^i)$. Using integer i, such in σ^i, we can rename the variables of σ, with the renaming avoiding produced clutter. In the end of the

rewriting step there exists an if condition that checks whether the query produced is an isomorphism, i.e. equivalently the same, of one that already exists in our set of queries $\boldsymbol{Q}_{REW}$. If there is no equivalent query then that means there has been a change and so this new query is added to our set, as it is new data.

3.2.3 Factorization Step

In the factorization step, the algorithm, in essence, checks every query in the set of queries and replaces the atoms in each query with its most general unifier, in essence getting rid of redundant atoms and reducing the size of the queries. To do so however the atoms must satisfy the factorizability condition.

In the case of ***Example 5***, S_1 satisfies the factorizability condition and goes through the factorization step giving us q_1': $t(a, a, C)$.

Formally, for each $S \subseteq body(q)$ that is factorizable w.r.t. σ, the factorization step creates query $q' = \gamma_S(q)$, with γ_S being the MGU for S. In the end, similarly with the rewriting step, there is an if condition that checks whether the query produced has an isomorphic one and if not, stores it in the set of queries before terminating the factorization loop.

3.3 Termination of XRewrite

For the algorithm XRewrite to terminate it is necessary to apply some restrictions on our TGDs and divide them into syntactic classes. This is due to the fact that the algorithm must be *database independent*, i.e. it must apply to every database possible and not depend on certain characteristics a database may have. As it stands, with no restrictions, there are cases in which the algorithm will not terminate.

Example 6: Consider TGDs $\sigma: P(x), R(x, y) \rightarrow P(y)$, CQ $q: P(C_n)$ and database $D: \{P(C_1), R(C_1, C_2), R(C_2, C_3), \dots, R(C_{n-1}, C_n)\}$, where $\{C_1, \dots, C_n\}$ are constants.

In this example, intuitively, the *chase procedure* will start with $P(C_1)$ and $R(C_1, C_2)$, and using σ will produce $P(C_2)$. Then similarly, using $P(C_2)$ with $R(C_2, C_3)$ will produce $P(C_3)$, et cetera until it produces $P(C_n)$.

The algorithm however will work in reverse. Assuming $P(C_n)$, using σ, the algorithm will produce the statement $\exists x\, P(x) \wedge R(x, C_n)$. Then similarly, using $P(x)$ and σ, will produce $\exists x \exists x'\, P(x') \wedge R(x', x)$, that using $P(x')$ produces $\exists x' \exists x''\, P(x'') \wedge R(x'', x')$, ad infinitum. In this case, the algorithm does not use information from database D. This makes it *database independent* but at the same time making it unable to know when to stop.

While a rewrite algorithm that is *database independent* and has no restrictions can exist, it will require the use of a recursive query language. However our aim is to rewrite to a simple form of a union of CQs, which is a fragment of SQL. And since SQL cannot support recursiveness this necessitates the use of syntactic classes.

Chapter 4

Non-Recursiveness

The first syntactic class of TGDs we use to guarantee the termination of XRewrite is non-recursiveness, that is, TGDs whose use will not lead to the production of the same information repeatedly. Proving that the algorithm terminates under the Non-Recursive class of TGDs means the successful execution of an ontological query on any non-recursive ontology.

Definition 3 (Non-Recursiveness): A set of TGDs Σ is non-recursive when the dependency graph of Σ is acyclic.

Recursive TGDs are the TGDs whose dependency graph does have cycles, i.e. a predicate can lead to itself. The problem with recursiveness is that it creates a loop that the reasoning of our ontology can get caught in and never terminate. In non-recursive TGDs, such a loop does not exist and as both our ontology and database are finite, intuitively our reasoning terminates.

Example 7: Consider set of TGDs Σ_1: $\{(A(x) \rightarrow B(x)), (B(x) \rightarrow C(x)), (C(x) \rightarrow A(x))\}$ and Σ_2: $\{(D(x) \rightarrow E(x)), (E(x) \rightarrow F(x)), (F(x) \rightarrow G(x))\}$. As shown in *Figure 2*, Σ_1 is recursive as A is replaced by B, B by C and C by A, creating a loop. Σ_2 has no such loop and thus is non-recursive.

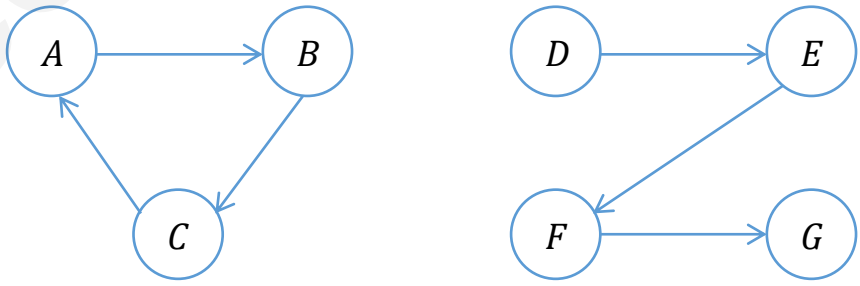

Figure 2. Dependency graph of recursive and non-recursive TGD sets

While termination under non-recursive TGDs is widely accepted as true due to the nature of the class, no detailed analysis has been presented, which we will give.

In simple terms, we can prove that the algorithm terminates by finding an upper bound to the number of different CQs that can be constructed. By finding a finite number of resulting CQs we can conclude that the algorithm terminates. In our proof of termination we will use the stratification of the set of TGDs, a way to characterize non-recursive TGDs.

Definition 4 (Stratification): A stratification of a set of existential rules Σ is a sequence of $\Sigma_1, \dots, \Sigma_n$such that for some function $f: sch(\Sigma) \rightarrow \{1, \dots, n\}$:

- $\{\Sigma_1, \dots, \Sigma_n\}$ is a partition of Σ
- For each predicate $P \in sch(\Sigma)$, all the rules with P in the head are in $\Sigma_{\mu(P)}$, i.e. in the same set of the partition
- If $\forall X \forall Y\ (\dots \wedge P(X,Y) \wedge \dots \rightarrow \exists Z(\dots \wedge R(X,Z) \wedge \dots)) \in \Sigma$, then $\mu(P) < \mu(R)$

In other words, stratification will create a number of numbered partitions of our set of TGDs Σ, these partitions are also called strata levels. The partition a TGD will belong to depends on the predicate P that appears in its head, with all the TGDs that have P in their head belonging to the same partition. If predicate P appears in body of a TGD that has predicate R in its head, then the number assigned to the partition of TGDs that have P in their head, will be lower than the number assigned to the partition of TGDs that have R in their head. For our rewriting, this means that an atom with predicate R can be substituted by atom with predicate P, iff R is assigned a higher number than P. Consequently, atoms with predicate L in body of TGD of the lowest number cannot be substituted at all.

Example 8: Consider set of TGDs $\Sigma: \{\sigma_1: A(x) \rightarrow B(x), \sigma_2: B(x) \rightarrow C(x), \sigma_3: C(x) \rightarrow D(x)\}$. As B appears in $head(\sigma_1)$ and body(σ_2), σ_1 is given a lower strata number than σ_2. Similarly, C appears in $head(\sigma_2)$ and $body(\sigma_3)$ and thus σ_2 is given a lower strata number than σ_3. This results in creating a stratification of $\Sigma: \{\Sigma_1: \{\sigma_1\}, \Sigma_2: \{\sigma_2\}, \Sigma_3: \{\sigma_3\}\}$. Note that the predicate in the body of the TGD in the lowest strata level, i.e. A, does not appear in the head of a TGD and thus will never be replaced by another predicate.

Theorem 1: *Consider CQ q over a schema R and set of TGDs Σ over R. If $\Sigma \in Non-Recursive$, then $XRewrite(q, \Sigma)$ terminates.*

Proof: If set of TGDs $\Sigma \in Non-Recursive$ then there exists stratification of Σ, $\{\Sigma_1, \Sigma_2, \ldots, \Sigma_n\}$. Then consider atom a that unifies with a TGD in strata Σ_n, e.g. $b(x) \rightarrow a$(x). This substitution is one step of the algorithm.

Due to the stratification, the predicate in the body of a TGD in strata Σ_n will appear in the head of a TGD in strata Σ_{n-1}, e.g. $c(x) \rightarrow b(x)$. This means that b can and will be substituted by c, counting as another step. Similarly, the predicate in the body of a TGD in strata Σ_{n-1} will appear in the head of a TGD in strata Σ_{n-2}, leading to another substitution. This phenomenon will be repeated up to and until the TGD in the lowest strata level and atom a is fully explored.

A graphical representation of an example of the above substitutions follows, as *Figure 3*, with nodes being the atoms and edges being the substitutions, with each level representing a strata level:

Figure 3. Representation example of substitution process

However, to fully explore an atom we need to also explore all of its substitution options. In the above case, atom a was substituted by atom b as per TGD $(x) \rightarrow a(x)$, but what happens if there was also TGD $d(x) \rightarrow a(x)$, thus giving an option to a to be substituted by d.

This is where strata cardinality, i.e. the cardinality of a strata level, comes in. The cardinality of each strata level is the number of TGDs that belong to that level, as multiple TGDs can belong to the same strata level. This means that a predicate of the body of TGD in strata Σ_n may not

only appear in the head of a single TGD but a number of TGDs in strata Σ_{n-1}, and in the worst case scenario it may appear in all of them.

When this is true, the substitution representation takes a tree-like form, as depicted in *Figure 4*.

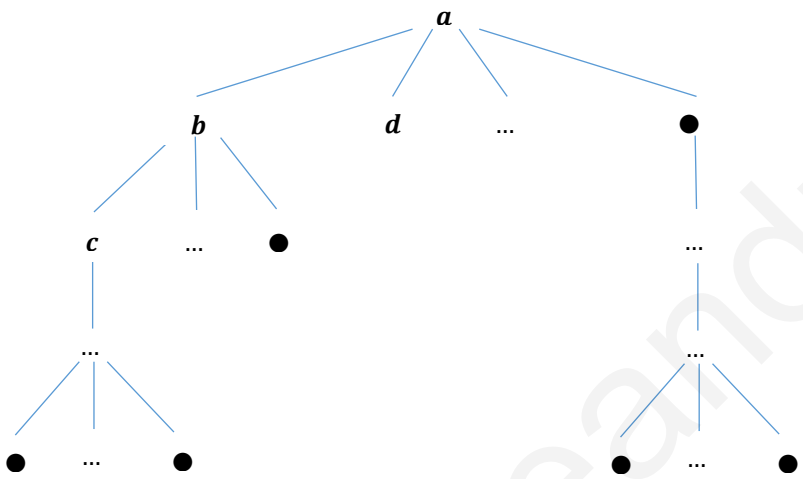

Figure 4. Representation example of substitution process with multiple substitution options

Like *Figure 3*, the nodes are atoms, the edges are substitutions and the levels are the strata levels. In this case though, an atom can branch out by selecting different substitution options.

The number of possible substitutions for each atom is the cardinality of each strata level below it. More clearly, the number of substitution choices for an atom x whose predicate appears in the head of a TGD in strata Σ_n, is equal to the cardinality of the strata level directly below it, $|\Sigma_{n-1}|$. In the worst case scenario, the cardinality of each strata level is the maximum that it can be, that is, equal to the cardinality of the whole set Σ.

Lastly, there is the fact that an atom can be substituted by a number of atoms, e.g. $b(x), c(x) \rightarrow a(x)$. This will result in more steps to fully explore the atom but the logic does not change.

Finally, let us combine all of the above to find a formula for the upper bound:

Consider CQ q: $\alpha_1 \wedge \alpha_2 \wedge \ldots \wedge \alpha_k$ over schema R and a set of TGDs $\Sigma \in Non-Recursive$ over R.

Since $\Sigma \in Non - Recursive$ there exists stratification of Σ, $\{\Sigma_1, \Sigma_2, \ldots, \Sigma_n\}$. As such we use the logic shown above to rewrite q. The representation will take the form shown in *Figure 5* after a single step:

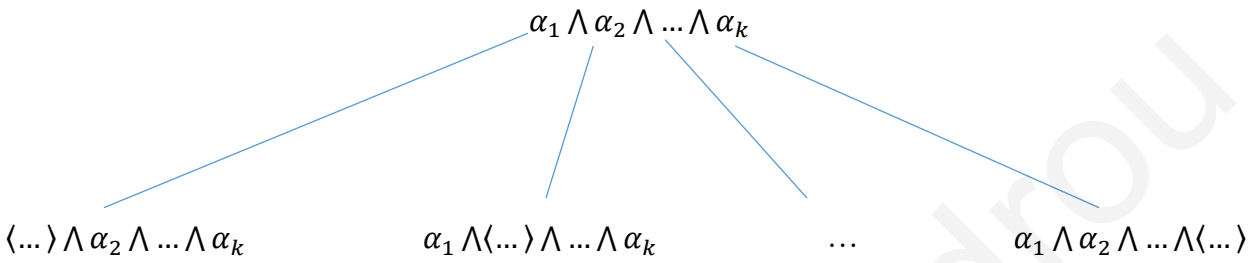

Figure 5. Representation of the substitution process for a CQ after one step

Note that each $\langle \ldots \rangle$ of the leaf nodes in the above figure may either be either a single atom or a conjunction of atoms. In the worst case scenario, as to maximize the amount of substitutions, $\langle \ldots \rangle$ will be a conjunction of atoms, with their predicates being in the head of a TGD in the strata level directly below the strata level of the TGD whose head was the predicate of the atom that was substituted. So an atom a would be substituted by a conjunction of atoms, $\alpha'_1 \wedge \alpha'_2 \wedge \ldots \wedge \alpha'_x$.

As such, in the case of CQ q above, when an atom is substituted, e.g. α_1, it will be substituted by a conjunction of atoms as above, giving a query q' of the following form: $(a'^1_1 \wedge a'^1_2 \wedge \ldots \wedge a'^1_x) \wedge \alpha_2 \wedge \ldots \wedge \alpha_k$.

To calculate the size of query q' we have to follow the substitutions made. Starting with an initial length of k atoms, the 0th level so to speak, an atom was removed due to the substitution, giving q' a length of $k - 1$. However, the atom that was removed was substituted in by a conjunction of atoms with length x, creating a query with length $k - 1 + x$.

Do note that x is not a constant as it is the length of the conjunction of atoms that substitute an atom and thus varies based on the size of the body of the TGD that leads to the substitution of that particular atom. In the worst case scenario, in order to maximize the number of atoms in the

query and thus the amount of possible substitutions and steps in the algorithm, x will be equal to the length of the body of the TGD with the biggest body, $x = maxbody$. With x defined, each substitution will now produce a query of length $k - 1 + maxbody$. So the leaf nodes in *Figure 5* have a length of $k - 1 + maxbody$.

However, do not forget that there are substitution options for each atom, equal to the strata cardinality of the strata level below it. In the worst case scenario, each atom has $|\Sigma|$ options to choose from. Thus the representation will look like *Figure 6*:

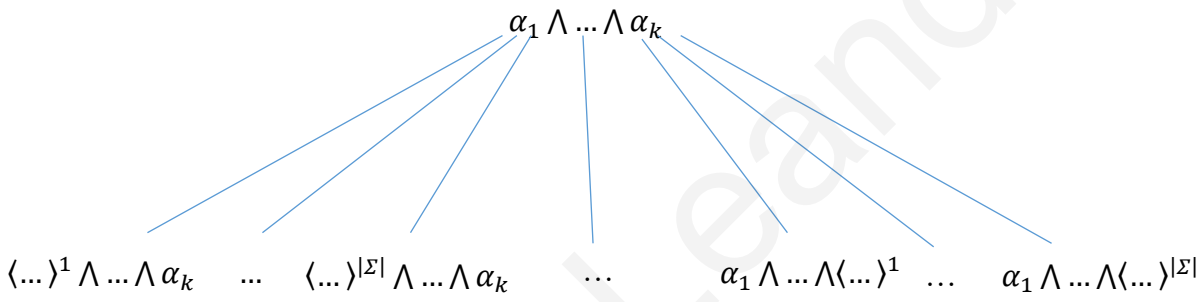

Figure 6. Representation of the substitution process with substitution options for a CQ after one step

Each $\langle \ldots \rangle$ is still a conjunction of atoms but now an atom has $|\Sigma|$ options to choose from, e.g. $\{\langle \ldots \rangle^1, \ldots, \langle \ldots \rangle^{|\Sigma|}\}$. So each atom of our initial CQ q produces $|\Sigma|$ queries q', and since the number of atoms is equal to $length(q) = k$, a total of $k|\Sigma|$ queries are produced of length $k - 1 + maxbody$.

Similarly at the next level, the 1st, each atom of q' will produce $|\Sigma|$ queries q'', for a total of $(k - 1 + maxbody)|\Sigma|$ queries q'' produced by one q'. With the number of q' being $k|\Sigma|$, a grand total of $k|\Sigma| * (k - 1 + maxbody)|\Sigma|$ queries q'' will be produced. The length of these q'' will follow the formula: $k - 1 + maxbody$. The initial length of k however will now be equal to their own length of $k - 1 + maxbody$, giving a length of $(k - 1 + maxbody) - 1 + maxbody = k - 2 + 2maxbody$.

As above, at the next level, the 2nd, each atom of q'' will produce $|\Sigma|$ queries q''' for a total of $(k-2+2maxbody)|\Sigma|$ per q''. With $(k-1+maxbody)|\Sigma| * k|\Sigma|$ number of q'', that makes a total of $k|\Sigma| * (k-1+maxbody)|\Sigma| * (k-2+2maxbody)|\Sigma|$ queries q'''. The length of q''' will be $((k-1+maxbody)-1+maxbody)-1+maxbody = k-3+3maxbody$.

So at level i, the queries will have a length of $k-i(1+maxbody)$ and each query produces an amount of queries equal to $(k-i(1+maxbody)) * |\Sigma|$. With the total number queries produced being at that level being $\prod_0^i (k-i(1+maxbody)) * |\Sigma|$.

Calculating the total amount of queries produced will be the sum of queries produced at each level. With n levels, i.e. the total number of strata levels, we have a total amount of $\sum_0^n \prod_0^i (k-i(1+maxbody)) * |\Sigma|$.

So for a CQ q with k atoms over R, and set of TGDs Σ over R, if $\Sigma \in Non-Recursive$ then the maximum number of queries produced, i.e. the size of rewriting, will be $\sum_0^n \prod_0^i (k-i(1+maxbody)) * |\Sigma|$. Since this is a finite number and the algorithm does not drop queries it has generated then the algorithm terminates and the claim follows.

Chapter 5

(Multi)Linearity

In this chapter we will take a look at two syntactic classes that are closely related, Linearity and Multi-linearity, with Linearity itself being a subclass of Multi-linearity. As for Multi-linearity, we will take a look at a special case of it, using multi-linear TGDs along with an additional restriction.

5.1 Linearity

Linearity is a basic syntactic class of TGDs. A TGD σ is linear when its body consists of a single atom, i.e. $|body(\sigma)| = 1$. A set of TGDs Σ is linear when all of the TGDs $\sigma \in \Sigma$ are linear. Linearity is incredibly simple yet forms a robust language, more expressive than DL-Lite$_R$ [2], and with a variety of advantages and applications, such as modeling hierarchies. It is known that the algorithm XRewrite terminates under linear TGDs, as it takes advantage of the simple format of linearity. The fact that the algorithm assumes TGDs to be in normal form does not affect us as the normalization procedure preserves linearity.

The proof of termination of the algorithm under linear TGDs is based on the following statement:

Lemma 1: *Consider CQ q over schema R, and set of TGDs Σ over R. For each $q' \in q_\Sigma$, if $\Sigma \in linear$ then $|q| \geq |q'|$.*

Proof: We can see how this statement is true because of how linear TGDs are. Since each linear TGD has only one body atom, during the rewriting step each body atom of CQ q is replaced by a single atom. During the factorization step atoms are unified, and thus reduced, resulting in CQ q' with a fewer number of atoms than q, i.e. $|q| \geq |q'|$.

Theorem 2: *Consider CQ q over schema R, and set of TGDs Σ over R. If $\Sigma \in linear$ then $XRewrite(q, \Sigma)$ terminates.*

Proof: Considering CQ q over schema R, and set of TGDs Σ over R if $\Sigma \in linear$ then we get that $|q| \geq |q'|$, as per *Lemma 1*. The statement $|q| \geq |q'|$ implies that each $q' \in q_\Sigma$ can be rewritten into an equivalent CQ with at most $k = |q| * arity(R)$ variables. Thus q_Σ contains (modulo variable renaming) at most k variables. With the number of CQs that can be constructed using k variables and $|R|$ predicates being finite, and since the algorithm does not drop generated queries, the algorithm will terminate.

As for the time needed for the algorithm to terminate in case of linear TGDs, in case of non-recursive linear TGDs we can derive it from the formulae presented in the chapter above.

5.2 Multi-linearity Special Case

Related to the class of linear TGDs is the class of multi-linear TGDs. Multi-linearity is a generalization of linearity, with linear TGDs being a particular instance of multi-linear TGDs. As such Multi-linearity is more expressive than Linearity and thus DL-Lite$_R$, and DL-Lite$_{R,\sqcap}$, the extended version of DL-Lite$_R$, [3].

Definition 5 (Multi-linearity): A TGD σ is multi-linear if and only if all the variables that appear in $body(\sigma)$ appear in every atom of $body(\sigma)$, [13].

With linear TGDs only having one atom in their body, they satisfy the condition for multi-linearity.

As multi-linearity is a more general case than linearity, multi-linear TGDs are more expressive than linear ones. While, in the case of non-recursive TGDs, that may be insignificant because of guaranteed termination of the algorithm, in case of recursive TGDs where the termination is not always guaranteed, a more expressive class of TGDs than linearity is more useful and a step forward to guaranteeing termination in case of recursiveness, with no restrictions for our TGDs.

However, in order to guarantee termination we use a special case of multi-linear TGDs, the case where our TGDs are multi-linear and the length of the body of each TGD is the same.

To prove the termination of the algorithm, we use a property derived from the definition of multi-linearity. If a TGD σ is multi-linear then all the variables that appear in $body(\sigma)$ appear in every atom of $body(\sigma)$. This means that when a TGD is multi-linear then all variables appearing in the body of a TGD are shared variables in it. Using this fact we can take advantage of the factorization process of the algorithm and limit the size of the produced query.

At first, we will apply the above reasoning to an atomic query which results in formulating the following theorem, with its proof following:

Definition 6 (Initial rewrite): Considering CQ q over schema R and set of TGDs Σ over R, q' is the query produced by an initial rewrite, by substituting all atoms of initial query q once.

Theorem 3: *Consider an atomic query q over a schema R, and a set Σ of multi-linear TGDs over R, where* $|body|$ *of all TGDs in Σ is the same. For each* $q'' \in q'$, $|q''| \leq |q'|$.

Proof: Consider atomic query q. After one step of the algorithm, query q will be rewritten into a query q' with a number of atoms, n (for the minimum case $n = 2$), as shown in *Figure 7*.

Figure 7. Atomic query q after initial rewrite into q' with two atoms

This initial rewrite gives an upper limit that the length of the query must not exceed, the $|q'|$ that appears in *Theorem 3*, which is equal to n. If in every query produced afterwards the length stays the same, we will have successfully limited the size of the query.

At first we will try to limit the length of the query produced by the rewrite step. This is done by replacing our TGDs σ, whose $body(\sigma) > 1$, with ones that have a shorter body of the same length. To limit the length of the rewrite step as much as possible every new TGD σ' will only have two atoms in its body, i.e. $|body(\sigma')| = 2$.

Example 9: Consider query $j: A$, and TGD $r: B, C, D, E \rightarrow A$, that leads to atom A being substituted by atoms B, C, D and E. Using r to rewrite j will result in j': B, C, D, E. Now let us replace TGD r by a set of TGDs r': $\{(B, X \rightarrow A), (C, Y \rightarrow X), (D, E \rightarrow Y)\}$. Using the set of TGDs r' to rewrite j will result in j': B, X, j'': B, C, Y and finally j''': B, C, D, E, thus making the result of these two rewrites the same. The second rewrite will take more steps to do but each length of each substitution is limited to n.

Figure 8 shows a representation of the substitution when limiting the length of each substitution to n.

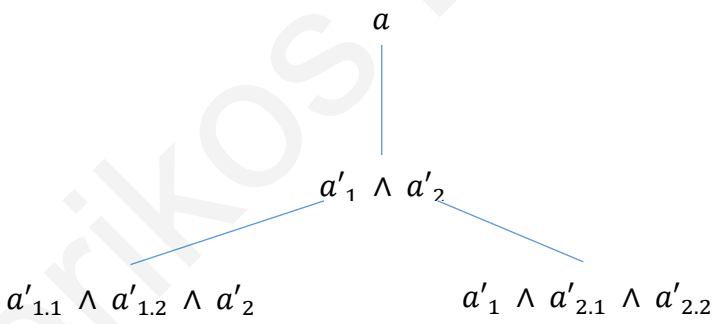

Figure 8. Every atom of query q' is substituted by a conjunction of atoms with length equal to n, where n=2.

This may seem counterintuitive as each atom would turn into n atoms and thus still end up increasing the length of the query to greater than n. This is where the factorization step of the algorithm will come in. Turning our TGDs into sets of equivalent ones with same shorter length will help us generalize the following procedure. Remember that, in short, the factorization step will take a set of atoms of the query that have a shared variable, and are essentially redundant information, and replace then by a single atom. Because of multi-linearity, every variable that

appears in the body of TGD σ will appear in every atom of $body(\sigma)$. That means that once we use σ to rewrite atom a, all the atoms that substitute it, a conjunction of atoms A': $a'_{.1}, a'_{.2}, \dots a'_{.n}$, will have all their variables be shared. The matter of number and position of variables in an atom does not matter as we can substitute it with an equivalent atom so that the position and number of variables match in the atoms produced. As the variable are shared, that means that the result of the rewrite step is factorizable, with the atoms being merged and replaced into a single atom of A'. The only case in which the factorization step does not work is when $|body(\sigma)| < 2$, i.e. $body(\sigma) = 1$. This means that if an atom is rewritten, it is either substituted by a single atom, or by a conjunction of atoms that are factorizable and thus merged into a single atom. So no matter what, the end result will be that it is substituted by a single atom.

In the end, an atom has two choices, either it cannot be rewritten and stays a single atom, or it can be rewritten and is ultimately substituted by a single atom, as shown in *Figure 9*.

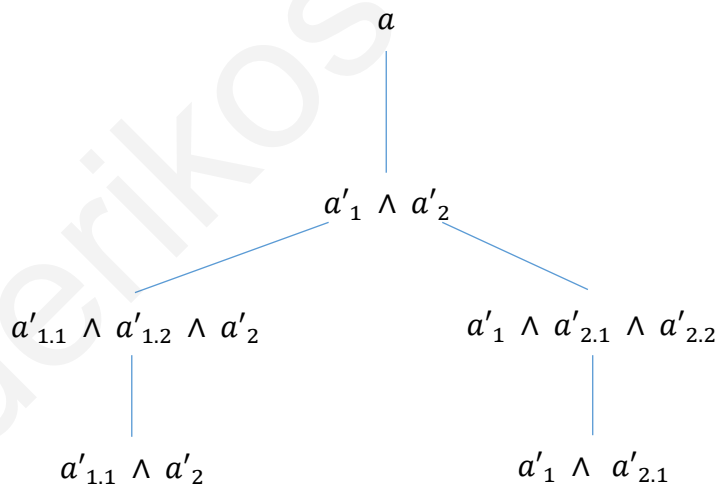

Figure 9. Due to the above property, the product of rewritten atoms is factorizable and merges into a single atom.

That means than after our initial rewrite that results in q', the length of the query will not increase in size, i.e. $|q''| \leq |q'|$, and thus satisfying the assumption presented in *Theorem 3*. In fact, the length of the query may decrease as the atoms produced by different substitutions but

in the same level of substitutions may be factorizable with each other, reducing the number of atoms even further.

So, considering an atomic query q over a schema R, and a set Σ of multi-linear TGDs over R, where the body of all TGDs in Σ is the same size, after an initial rewrite that creates query q', for each $q'' \in q'_{\Sigma}$, $|q''| \leq |q'|$.

In fact, we can also refine the above conclusion even further by defining the value of $|q'|$. As $|q'|$ becomes equal to the body of a TGD in Σ, in the worst case scenario it will become equal to $maxbody(\Sigma)$, the size of the largest body of TGD in Σ. Adding this to *Theorem 3* proves the following theorem:

Theorem 4: *Consider an atomic query q over a schema R, and a set Σ of multi-linear TGDs over R, where $|body|$ of all TGDs in Σ is the same. For each $q'' \in q'_{\Sigma}$, $|q''| \leq maxbody(\Sigma)$.*

Theorem 3 will finally give us the proof of termination. It follows the same logic as the logic used for the proof of termination for linearity. After an initial rewrite that produces q', all following rewrites will be of an equal or smaller length than q'.

$|q''| \leq maxbody(\Sigma)$ implies that each $q'' \in q'_{\Sigma}$ can be rewritten into an equivalent conjunctive query with at most $k = maxbody(\Sigma) * arity(R)$ variables. Thus q'_{Σ} contains (modulo variable renaming) at most k variables. With the number of conjunctive queries that can be constructed using k variables and $|R|$ predicates being finite, and since the algorithm does not drop generated queries, the algorithm will terminate.

This proof of termination of the algorithm for multi-linear TGDs and atomic queries will form the basis for proof of termination and in the more general case of non-atomic queries and multi-linear TGDs.

Theorem 5: *Consider a conjunctive query q over a schema R, and a set Σ of multi-linear TGDs over R, where $|body|$ of all TGDs in Σ is the same. For each $q'' \in q'_{\Sigma}$, $|q''| \leq |q'|$.*

Proof: Consider conjunctive query q that is made up from a number of atoms $a_1, \dots, a_n$. It can also be said that q is made up from a number of atomic queries. The idea is that if *Theorem 3* applies for each atom in q then it will also similarly apply collectively to q.

As per *Definition 6*, q' is the query that results after each atom in q is substituted once. In more detail, each atom of q will go through the process described in the section above, with each atom a_i undergoing an initial rewrite that will result in a conjunction of atoms A_i'. This results in q', a conjunctive query made up of $A_1', \dots, A_n'$.

Per the description above, each A_i' will have a limit, giving us a bound that its size cannot exceed as the results of any further substitutions to it, i.e. A_i'', will never be of a larger size than that bound of $|A_i'|$. As $|A_i''| \leq |A_i'|$, then also $\sum_{i=1}^{n}|A_i''| \leq \sum_{i=1}^{n}|A_i'|$. As $|q''| = \sum_{i=1}^{n}|A_i''|$ and $|q'| = \sum_{i=1}^{n}|A_i'|$, the above gives us $|q''| \leq |q'|$.

We can also refine the above by defining the value of $|q'|$. In the worst case scenario, $|A_i'| = maxbody(\Sigma)$ and as $|q'| = \sum_{i=1}^{n}|A_i'|$, this results in $|q'| = \sum_{i=1}^{n} maxbody(\Sigma) = n * maxbody(\Sigma)$. With n being the number of atoms in q this results in $|q'| = maxbody(\Sigma) * |q|$ and proves the following theorem:

Theorem 6: *Consider a conjunctive query q over a schema R, and a set Σ of multi-linear TGDs over R, where $|body|$ of all TGDs in Σ is the same. For each $q'' \in q'_{\Sigma}$, $|q''| \leq maxbody(\Sigma) * |q|$.*

So now we have given a bound and limited the size of the query which leads to the following theorem of the termination of the algorithm XRewrite:

Theorem 7: *For every conjunctive query q over a schema R, and a set Σ of multi-linear TGDs over R, where $|body|$ of all TGDs in Σ is the same, then $XRewrite(q, \Sigma)$ terminates.*

Proof: By *Theorem 4.1* we get that after an initial rewrite for every initial atom of q, that creates query q', for each $q'' \in q'_{\Sigma}$, $|q''| \leq maxbody(\Sigma) * |q|$. That initial rewrite for every initial atom of q consists of a finite amount of substitutions, equal to the number of atoms of q, with substitution options also being finite. Additionally, $|q''| \leq maxbody(\Sigma) * |q|$ for each $q'' \in q'_{\Sigma}$, implies that each $q'' \in q'_{\Sigma}$ can be equivalently rewritten as a conjunctive query with at most $k = maxbody(\Sigma) * |q| * arity(R)$ variables. Therefore q'_{Σ} contains (modulo variables remaining) at most k variables. With the maximum number of conjunctive queries that can be constructed using k variables and $|R|$ predicates being finite and since the algorithm does not drop queries it has generated, the claim of the above theorem holds true.

Chapter 6

Stickiness

Another syntactic class of TGDs is called stickiness. The idea of stickiness is to create a class that allows for meaningful joins in rule bodies [14]. Stickiness allows joins to appear in rule-bodies not expressible with linear TGDs or DL(R)-Lite assertions [1]. This is done by ensuring that during the chase procedure, terms associated with body variables appearing more than once are always propagated. To do that we use a procedure called SMarking. With Σ, a set of TGDs as input, SMarking returns the same set after marking some of its body variables.

In more detail, the SMarking procedure consists of two steps, the marking step and the propagation step. In the marking step, every variable of a TGD, which appears in its body but not in its head, is marked. After all variables that can be marked by the marking step are marked, the propagation step is exhaustively applied. The propagation step checks pairs of TGDs in the following way. If an atom α in the head of a TGD σ has the same predicate as an atom α' in the body of TGD σ', if α' has a marked variable in some position, the variables of α in the same position will be marked in all its occurrences in the body of a. Repeating this until no further changes are observed will give us the SMarking of Σ.

Formally, consider Σ set of TGDs, TGD $\sigma \in \Sigma$ and variable $x \in body(\sigma)$. We recursively define when x is marked in Σ:

1. If x in not in $head(\sigma)$, then x is marked.
2. Assuming that $head(\sigma) \in R(\bar{t})$ and $x \in \bar{t}$, if there is $\sigma' \in \Sigma$ with $R(\bar{u})$ in $body(\sigma')$, and each variable in $R(\bar{u})$ at a position $pos(R(\bar{t}), x)$ is marked in σ', then x is marked in $body(\sigma)$.

As for *stickiness*, a set of TGDs Σ is called *sticky* if, for every $\sigma \in \text{SMarking}(\Sigma)$, each marked variable appears only once. *Stickiness* guarantees the first-order rewritability of CQ answering [13]. Additionally, the normalization procedure also preserves stickiness so the algorithm assuming normal form TGDs does not affect the stickiness.

Example 10: Consider Σ set of TGDs: $\sigma_1: r(X,Y) \rightarrow \exists Z\, r(Y,Z)$, $\sigma_2: r(X,Y) \rightarrow s(X)$, $\sigma_3: s(X), s(Y) \rightarrow p(X,Y)$ and $\sigma_4: r(X,Y), r(Z,X) \rightarrow s(X)$. The SMarking of Σ will start with the above first condition, the initial marking step. This will mark variables with a cap(^), resulting in Σ looking like this:

$\sigma_1: r(\hat{X}, Y) \rightarrow \exists Z\, r(Y,Z)$, $\sigma_2: r(X,\hat{Y}) \rightarrow s(X)$, $\sigma_3: s(X), s(Y) \rightarrow p(X,Y)$ and $\sigma_4: r(X,\hat{Y}), r(\hat{Z},X) \rightarrow s(X)$.

Following the initial step, the second condition will be used, the propagation step. This will mark variables with a double cap, resulting in Σ looking like this:

$\sigma_1: r(\hat{X}, \hat{\hat{Y}}) \rightarrow \exists Z\, r(Y,Z)$, $\sigma_2: r(X,\hat{Y}) \rightarrow s(X)$, $\sigma_3: s(X), s(Y) \rightarrow p(X,Y)$ and $\sigma_4: r(X,\hat{Y}), r(\hat{Z},X) \rightarrow s(X)$.

One of the reasons that Y is marked in σ_1 is because the atom in $head(\sigma_1)$ appears in $body(\sigma_4)$ with a marked variable in it. The marked variable in the atom of $body(\sigma_4)$ is the first one, so the first variable in the atom of $head(\sigma_1)$, i.e. Y, will be marked in all of its occurrences in $body(\sigma_1)$.

The termination of XRewrite hinges on the above property that each marked variable appears only once. That property will lead to the following statement:

Lemma 2: *Consider CQ q over schema R and set of TGDs Σ over R, for each q' ∈ q_Σ, if $\Sigma \in sticky$, then every variable of $var(q') \backslash var(q)$ occurs only once in q'.*

The above statement can be proven true following an induction on the rewriting and factorization steps of the algorithm [1].

Theorem 8: *Consider CQ q over a schema R, and a set Σ of multi-linear TGDs over R. If $\Sigma \in Sticky$, then $XRewrite(q, \Sigma)$ terminates.*

Proof: Assume that $\Sigma \in sticky$, and given CQ $p \in q_\Sigma$, let $p^\star$ be the query obtained from p by replacing every variable of $var(p) \backslash var(q)$ with symbol $\star$.

The set $var(p) \backslash var(q)$ will contain all variables that appear in p, except all the variables that appear in the initial query q. So all variables except those of the initial query are now replaced by $\star$. Normally, this would be a problem but per the statement above, every variable of $var(p) \backslash var(q)$ appears only once in p as $\Sigma \in sticky$. Since these variables appear only once, that means that they are not used for any joins, so their name can be freely changed with the meaning of the query staying the same.

Example 11: Let there be TGD σ: $R(x, y), R(y, z) \rightarrow R(x, z)$ and CQ q: $R(A, B)$.

Consider $p_1 : R(A, N_1), R(N_2, B)$ and $p_2 : R(A, N_1'), R(N_2', B)$ where $p_1, p_2 \in q_\Sigma$. Normally, N_1, N_2, N_1', N_2' all refer to a different variable but since they appear only once, they can freely be renamed without changing the meaning of p_1 or p_2. With $var(p_1) \backslash var(q) = \{N_1, N_2\}$ and $var(p_2) \backslash var(q) = \{N_1', N_2'\}$ we get that ${p_1}^\star$: $R(A, \star), R(\star, B)$ and ${p_2}^\star$: $R(A, \star), R(\star, B)$. Following the $\star$ renaming, we can see that ${p_1}^\star = {p_2}^\star$, and thus $p_1 \simeq p_2$.

So, for each pair of CQs $p_1, p_2 \in q_\Sigma$, if ${p_1}^\star = {p_2}^\star$ then p_1 and p_2 are the same modulo bijective variable renaming, aka $p_1 \simeq p_2$. This check of isomorphism is implemented in the algorithm, meaning that it will not explore queries when it has already explored a query with the same meaning. Since the algorithm will not do redundant explorations of queries, we only have to calculate the amount of non-isomorphic queries that the algorithm can explore. Following the $\star$ renaming, the terms that can used in queries will be the terms used in the initial query q and the $\star$ variable, i.e. $terms(q) \cup \{\star\}$. These terms are a finite amount. With the number of predicates in R also being finite, the amount of unique queries that can be produced is finite. Since the

amount of queries the algorithm can explore is finite and it does not drop queries it has explored, the algorithm will terminate.

Chapter 7

Conclusion

This thesis presents a brief introduction to the field of ontological databases and query rewriting for ontological databases, centered on the algorithm XRewrite. With the results of query rewriting and the algorithm being in the form used in widespread databases, this can be an easier point of introduction to the above fields. Presented as an analysis of XRewrite we go through the basics of ontological databases and commonly used terminology and procedures.

Beyond an introduction, XRewrite is a quite powerful solution to the problem of query rewriting for ontological databases. Our analysis goes through a simple presentation of the logic behind it, how and why it works, and the problem of its termination. With the termination not being guaranteed we present various syntactic classes of TGDs in which the algorithm is proven to terminate: non-recursive, linear, a multi-linear special case, and sticky. Despite termination in the case of non-recursive TGDs being intuitively known as a fact, we provide new information in how exactly the algorithm works in these cases and provide the size of the rewriting. Additional new results is the proof of termination under a special case of multi-linear TGDs, especially in the case of recursive TGDs. Being more general than linear TGDs, this expands the scope of cases in which XRewrite terminates. As for the efficiency of XRewrite, the algorithm has a low data complexity as it uses only the query and ontology which are typically of a size that can be productively assumed to be fixed and is usually much smaller than the relational database, with an implementation and evaluation of the algorithm being available [1].

Further research to be done on XRewrite includes finding proof of its termination under multi-linearity with no further restrictions applied as well as trying to find a more general and expressive class of TGDs under which the algorithm will terminate, like bounded dependencies.

Ontological databases are an area with active and ongoing research, with properties we have not discussed on this paper. Any contribution is welcome as ontological databases are aimed to replace present conventional databases in the near future and ontologies see more use.

Bibliography

[1] Giorgio Orsi, and Andreas Pieris Georg Gottlob, "Query Rewriting and Optimization for Ontological Databases," *ACM Trans. Datab. Syst. 39, 3, Article 25*, p. 46, 2014.

[2] Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati Diego Calvanese, "Tractable reasoning and efficient query answering in description logics: The DL-Lite family," *J. Autom. Reason*, vol. 39, no. 3, pp. 385-429, 2007.

[3] Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati Diego Calvanese, "Data Complexity of Query Answering in Description Logics," *Artificial Intelligence*, vol. 195, pp. 335–360, 2013.

[4] Boris Motik, and Ian Horrocks Hector Perez-Urbina, "Tractable query answering and rewriting under description logic constraints," *J. Appl. Logic*, vol. 8, no. 2, pp. 186-209, 2010.

[5] Despoina Trivela, and Giorgos B. Stamou Alexandros Chortaras, "Optimized query rewriting for owl 2 ql," in *Proceedings of the 23rd International Conference on Automated Deduction*, 2011, pp. 192-206.

[6] Georg Gottlob, Thomas Lukasiewicz, and Andreas Pieris Andrea Cali, "A logical toolbox for ontological reasoning," *SIGMOD Rec*, vol. 40, no. 3, pp. 5-14, 2011.

[7] Michel LeClere, Marie-Laure Mugnier, and Michael Thomazo Melanie Konig, "A sound and complete backward chaining algorithm for existential rules," in *Proceedings of the 6th International Conference on Web Reasoning and Rule Systems*, 2012, pp. 122-138.

[8] Michel LeClere, Marie-Laure Mugnier, and Michael Thomazo Melanie Konig, "On the exploration of the query rewriting space with existential rules," in *Proceedings of the 7th International Conference on Web Reasoning and Rule Systems*, 2013, pp. 123-137.

[9] Alberto O. Mendelzon, and Yehoshua Sagiv David Maier, "Testing implications of data dependencies," *ACM Trans. Database Syst.*, vol. 4, no. 4, pp. 455-469, 1979.

[10] David S. Johnson and Anthony C. Klug, "Testing containment of conjunctive queries under functional," *J. Comput. Syst. Sci*, vol. 28, no. 1, pp. 167–189, 1984.

[11] Phokion G. Kolaitis, Renee J. Miller, and Lucian Popa Ronald Fagin, "Data exchange: Semantics and query answering," *Theor. Comput. Sci*, vol. 336, no. 1, pp. 89-124, 2005.

[12] Alan Nash, and Jeff B. Remmel Alin Deutsch, "The chase revisisted," in *Proceedings of the 27th ACM Symposium on Principles of Database Systems*, pp. 149-158.

[13] Georg Gottlob and Thomas Lukasiewicz Andrea Cali, "A GENERAL DATALOG-BASED FRAMEWORK FOR TRACTABLE," Oxford, 2010.

[14] Georg Gottlob, and Andreas Pieris Andrea Cal`ı, "Towards More Expressive Ontology Languages: The Query Answering Problem," *Artificial Intelligence*, vol. 193, pp. 87-128, 2012.