Data Management in the Existential Rule Framework : Translation of Queries and Constraints

Guillaume Pérution-Kihli

THÈSE POUR OBTENIR LE GRADE DE DOCTEUR
DE L'UNIVERSITÉ DE MONTPELLIER

En Informatique

École doctorale : Information, Structures, Systèmes (I2S)

Unité de recherche : LIRMM

Data Management in the Existential Rule Framework: Translation of Queries and Constraints

Présentée par Guillaume Pérution-Kihli
le 18 décembre 2023

Sous la direction de Marie-Laure Mugnier
et co-encadrée par Michel Leclère

Devant le jury composé de

François Goasdoué	Professeur	Université de Rennes	**Rapporteur**
Carsten Lutz	Professeur	Université de Leipzig	**Rapporteur**
Christophe Paul	Directeur de recherche	CNRS	**Examinateur**
Sophie Tison	Professeur Emérite	Université de Lille	**Présidente du jury**
Michel Leclère	Maître de conférence	Université de Montpellier	**Co-encadrant**
Marie-Laure Mugnier	Professeur	Université de Montpellier	**Directrice**

UNIVERSITÉ
DE MONTPELLIER

Remerciements

Alors que je referme ce chapitre significatif de ma vie académique, je tiens à exprimer ma profonde gratitude à toutes les personnes qui ont jalonné ce parcours de thèse.

Un merci tout particulier s'adresse à ma directrice de thèse, Marie-Laure Mugnier, et à mon co-encadrant, Michel Leclère, pour leur présence constante et leur encadrement exceptionnel. Leur expertise, leur patience, et leur capacité à inspirer et à motiver ont été des pierres angulaires de mon développement professionnel et personnel. Leur soutien indéfectible depuis mes premiers pas en 2017 a été un pilier de ma progression.

Je suis également reconnaissant envers les membres de mon comité de suivi de thèse, Stéphane Bessy et Marianne Huchard, pour leurs précieux conseils et leur soutien.

Je remercie également les membres de mon jury de thèse : les rapporteurs François Gasdoué et Carsten Lutz, Christophe Paul, et la présidente Sophie Tison, pour leur temps, leur expertise et leurs retours constructifs.

Mes remerciements s'étendent à l'ensemble de l'équipe Boréal. Votre soutien, ainsi que les discussions stimulantes et enrichissantes, ont grandement contribué à mon épanouissement scientifique et personnel.

Un merci tout spécial à Florent Tornil, mon compagnon de bureau. Ton soutien, ta patience, et ta présence lors de nos nombreuses sorties et randonnées ont été un réconfort dans les moments difficiles. Partager ces trois années avec toi a été une expérience inestimable.

Je suis également reconnaissant envers Elie Najm pour les séances de musculation après le travail, qui ont été une excellente échappatoire et un moyen de maintenir un équilibre sain.

Un grand merci à Olivier Rodriguez pour toutes nos discussions passionnées sur les langages de programmation. Ces échanges sur la supériorité de Python et C++ sur PHP resteront gravés dans ma mémoire.

Je tiens aussi à remercier chaleureusement Akira Charoensit pour ses cadeaux de soutenance, notamment le casse-tête et la belle carte, qui resteront comme des souvenirs précieux de cette période.

Et je suis également reconnaissant envers mes autres collègues de Boréal : Federico Ulliana, Jean-François Baget, Maxime Buron, David Carral et Nofar Carmeli, pour leurs perspectives enrichissantes et leurs contributions à mon développement académique.

Mes amis hors de l'équipe, Julien Rodriguez et Mattéo Delabre, méritent une mention spéciale pour leur compagnie dans nos aventures de programmation compétitive. Votre amitié a été un pilier de ma vie durant ces années.

Enfin, je tiens à remercier Julie Cailler pour son aide précieuse dans la gestion du conseil des doctorants.

En somme, merci à chacun de vous pour avoir contribué à cette aventure. Votre impact va bien au-delà de ce que les mots peuvent exprimer.

Contents

Introduction

The general context of this work that integrate various data sources via a semantic layer encoded in a Knowledge Representation and Reasoning (KR) language. Such systems make it possible to build applications in which the tasks to be solved are specified at a conceptual level, using a high-level vocabulary, while leveraging multiple data sources. Hence, the knowledge representation layer acts as a mediating layer, able to integrate data sources. Importantly, this mediating level does not simply *integrate* data, as a global database schema would do, since it also provides *reasoning* capabilities to the system.

The paradigm known as *Ontology-Based Data Access (OBDA)* [Poggi et al., 2008], also called Ontology-Based Data Management [Lenzerini, 2018] is precisely aiming to integrate data and formalized knowledge in a principled way. In a nutshell, it combines concepts and techniques coming from both data management and KR, while making a fundamental distinction between the data and the conceptual levels. An OBDA system is composed of a data layer, made of one or several data sources that may have been built for independent purposes, and a conceptual layer, which describes an ontology using a vocabulary adapted to the intended application and users; declarative mappings between both levels allow one to select relevant data and to translate it at the conceptual level. The conceptual layer is formalised in a KR language, which provides reasoning capabilities. Queries to an OBDA system are expressed at the conceptual level, and answers to queries take into account inferences made by the system. More precisely, query answers are entailed by the logical encoding of the whole OBDA system. This paradigm has several advantages. First, it allows a user to formulate queries using a familiar vocabulary, without knowing how data is actually encoded and stored. Second, it provides richer answers, since not only the factual assertions directly coming from the data are considered, but also those that are inferred from both the data and the ontology. Third, the conceptual vocabulary can act as a mediating layer to integrate several data sources.

The OBDA paradigm has attracted a lot of interest from the 2010s. Several implementations are now available, from mature systems [Calvanese et al., 2011, Rodriguez-Muro et al., 2013, Calvanese et al., 2017] to research prototypes, e.g. [Sequeda et al., 2014, Buron et al., 2020]. These OBDA systems are all based on semantic web languages, namely the lightweight OWL 2 QL profile, or RDF Schema and slight extensions of it. Whereas query answering is the fundamental task to be solved, other issues related to the design and management of OBDA systems have begun to be investigated. To encompass all these issues, the name Ontology-Based Data Management (OBDM) has been coined [Lenzerini, 2018].

In this dissertation, we build on this seminal work while using more expressive languages. Indeed, we consider *existential rules* [Baget et al., 2009, Calí et al., 2009], an expressive fragment of first-order logic, which can be used to express both ontological knowledge and powerful mappings, known as GLAV mappings. More generally, existential rules can be used as a high-level declarative language to perform knowledge-

based reasoning, but also various data processing and data quality tasks. A key feature of these rules is their ability to assert the existence of individuals that may not exist in the data (or facts), which allows one to do reasoning in open domains, where the set of relevant individuals is not known in advance, or to create new strutures that make integration of heterogeneous data easier. Using existential rules for both knowledge and mappings yields a *uniform setting* for the whole OBDA specification, which facilitates its analysis and the development of techniques that need to consider both, as will be the case in this thesis.

In our setting the conceptual level is not restricted to an ontology and may include several kinds of rules used for other purposes, hence we prefer to use the more general term of *knowledge*, and call our *Knowledge-Based Data Management* (KBDM). To the best of our knowlege, the only existing system that implements the KBDM setting in the framework of existential rules is the research prototype *InteGraal* [Baget et al., 2023][1]. InteGraal is a highly modular tool specifically designed for coping with a wide range of applicative scenarios exploiting heterogeneous and federated data sources.

One of the intrinsic difficulties in designing a KBDM system is the need to get a good *understanding of a data source content*. This is a prerequisite to being able to select relevant data sources and craft the mapping to the ontology. Data sources are often provided with typical queries and integrity constraints from which valuable information about their semantics can be drawn, as long as this information is made intelligible to KBDM designers. This leads to one of our core motivating questions: when is it possible to translate pieces of information expressed in terms of the data sources into the ontological language while preserving their semantics? Recently, a line of work has led to theoretical and algorithmic tools to tackle this question for data *queries* [Lutz et al., 2018, Cima et al., 2019, Lenzerini, 2019, Cima, 2020]. In this dissertation, we revisit and extend the state-of-the-art on the translation of queries, and we also investigate the translation of integrity constraints, an issue that has been barely studied yet.

Actually, both translation directions may help to bridge the gap between the data and the ontology layers, hence they can be both of interest in the design of a KBDM system. As mentioned above, translating queries or constraints expressed on a data source to the conceptual level yields a high-level view of data semantics, therefore helping the designer of a KBDM system to better understand data. On the other hand, the conceptual level allows the designer to express high-level queries or constraints in a vocabulary that makes sense to her, and translating these objects to the data level allows her to check hypotheses on the data. Finally, when several data sources are involved, the ontological level allows one to analyse the relationships between the integrity constraints possibly provided for each source, starting with identifying and resolving mismatches between sources, which is essential to ensure cohesive and coherent data integration; this also enables one to export constraints and queries of some data source to another data source.

Hence, our leading questions can be sketched as follows: When is it possible to

[1] This is an open-source sofware available at https://gitlab.inria.fr/rules/integraal

translate queries from one level to the other so that answers to queries are preserved? Similarly, when is it possible to translate constraints from one level to the other while preserving constraint satisfaction? When a perfectly faithful translation cannot be achieved, can we still provide a translation with interesting semantic properties? Can we approximate in some way perfect translations?

As will become clear in what follows, translating objects from the data level to the ontological level is much more difficult than in the other direction.

Organisation of this thesis

The remainder of this dissertation is organised into six chapters.

Chapter I is devoted to preliminary notions. Knowledge bases and query answering are introduced. The emphasis is placed on existential rules and the two main techniques for query answering in existential rule knowledge bases, called the chase and query rewriting, which are related to forward and backward chaining respectively. The chapter ends with the definition of a KBDM system and the extension of query answering techniques to this setting.

Chapter II defines a general framework for translating queries and constraints. The chapter first presents the different kinds of translations, which work in one direction (from the data to the ontology, or vice-versa) and at a given level of the KBDM specification (considering solely the mapping, or the whole specification, i.e., the mapping added with existential rules). Then, the framework for query translation is introduced. This framework is largely based on the one introduced by [Cima et al., 2019] and [Arenas et al., 2010, Pérez, 2011], which notably defines the desired faithfulness properties of a translation. In a nutshell, a translation should preserve the semantics of the translated objects; however, since a *perfect* translation may not always exist, it may be approximated by a *maximally sound* translation, which outputs a query that retrieves a maximal subset of answers to the input query, or a *minimally complete* translation, which outputs a query that retrieves all the answers to the input query with a minimal addition of other answers. To be able to translate constraints, we first need to define their semantics in a KBDM system, since such a system has to deal with both *closed-world* assumption at the data level and *open-world* assumption at the ontology level. Following this, we introduce the translation framework for constraints, which is similar to the one for query translation up to the differences between the semantics of query answering and constraint satisfaction. A mapping only transfers part of the content of a data source to the ontological level. We discuss how this part selection performed by the mapping may impact the existence of a perfect translation. Finally, we illustrate the interest of constraint translation, with practical examples of uses, notably for designing a KBDM system and ensuring the quality of the data.

Next, we investigate the translation of fundamental database queries, namely unions of conjunctive queries (UCQs), and slight extensions of them, as well as fundamental classes of integrity constraints. The emphasis is put on the direction from data sources to ontology, as it is particularly challenging and requires new tools. This

led us to investigate the issue of rewriting a UCQ (into another UCQ) with *disjunctive* existential rules. The results are then exploited for the source-to-ontology translation of UCQs. Results on the translation of UCQs are in turn used for translating integrity constraints. The next chapters present the core technical results of this thesis, with the exception of chapter IV, which reviews results from the literature.

Chapter III is devoted to UCQ rewriting with disjunctive existential rules, including disjunctive mappings. These rules are an extension of existential rules (and mappings) with disjunction in rule heads. On the one hand, we introduce a novel query rewriting technique, which allows to compute a UCQ-rewriting of a UCQ when there exists one. On the other hand, we provide negative results that show the difficulty of reasoning in this setting. In particular, we show that the problem of determining whether a UCQ admits a UCQ-rewriting through a disjunctive mapping is undecidable.

Chapter IV reviews results from the literature related to UCQ translation. We first present a slight extension of UCQs, namely $\text{UCQ}^{\mathbf{C},\neq}$, which has the interest of providing more faithful translations. Then, we present the state of the art concerning source-to-ontology translations of UCQs. We also present in details the notion of maximum recovery, which comes from database theory. A maximum recovery is a kind of inverse mapping, which intuitively allows to "reverse" the transformation on the data made by a mapping. This notion was not introduced in the context of query translation but in the context of data exchange. However, we found that it is a key notion for computing maximally sound source-to-ontology translations of UCQs.

Chapter V provides new results on UCQ translation, mostly in the source-to-ontology direction. Notably, it introduces several translation techniques for a KBDM system, extending the work done for OBDA on lightweight description logics ontologies, which can be seen as specific cases of existential rules. One of our main results is an algorithm that computes a minimally complete source-to-ontology translation of a $\text{UCQ}^{\mathbf{C},\neq}$ when one exists. The existence of a complete translation and the property of being minimal are both independent from any specific target query language, i.e., we show that when a complete translation of a $\text{UCQ}^{\mathbf{C},\neq}$ exists, it can be expressed as a $\text{UCQ}^{\mathbf{C},\neq}$; then a minimally-complete translation also exists and can be expressed as a $\text{UCQ}^{\mathbf{C},\neq}$. Interestingly, this result holds whether or not an ontology (i.e. a set of existential rules) is taken into account in the translation. As another main result, we show that whenever a maximally sound source-to-ontology translation of a $\text{UCQ}^{\mathbf{C},\neq}$ into a $\text{UCQ}^{\mathbf{C},\neq}$ exists, it can be obtained by rewriting the query with a maximum recovery of the mapping. A maximum recovery of the mapping has the form of a disjunctive mapping, hence we can make use of the results from Chapter III. This result holds when the mapping is considered alone, and we show that it can be extended to take into account an ontology composed of a subclass of existential rules (so-called parallelisable rules).

Chapter VI is devoted to the translation of constraints. It first presents the fundamental kinds of constraints we study, namely negative (aka denial) constraints, equality-generating dependencies (EGDs) and (disjunctive) tuple-generating dependencies ((D)TGDs). After establishing connections with related work, it leverages the

results obtained on UCQ translation (Chapter V) to compute perfect, minimally complete or maximally sound translations for these kinds of constraints. Indeed, negative constraints can be dealt with in the same way as UCQs. We show that a (slightly restricted class of) EGDs can be dealt with in the same way as $\text{UCQ}^{\mathbf{C},\neq}$. Finally, concerning (a slightly restricted class of) DTGDs, the connection is less direct but we show that we can also rely on the results obtained for $\text{UCQ}^{\mathbf{C},\neq}$.

Finally, we draw perspectives with **Chapter VII**.

I - Fundamental notions

In this chapter, we introduce fundamental notions and notations that will be used throughout the thesis. We first recall basic notions about first-order logic and homomorphisms, then we introduce knowledge bases and query answering. As knowledge representation and reasoning language we use existential rules but we also present some lightweight description logics dialects, which are in particular useful to understand related work. These dialects can be translated into specific existential rules. Concerning query answering on knowledge bases, we introduce the main techniques, based either on forward chaining (or chase) or on backward chaining (namely, query rewriting). Finally we introduce KBDM systems, which requires to add mappings to data sources, and we show how the query answering notions and techniques defined on KBs are extended to KBDM systems. Note that we briefly recall some useful complexity notions in Appendix C.

I.1 Basic logical notions

In this dissertation, we abstract from a specific knowledge representation formalism by considering First-Order (FO) logic. Note that we do not consider function symbols outside constants, which can be seen as nullary function symbols. We recall below some basic logical notions.

A *logical language* is of the form $\mathcal{L} = (\mathcal{P}, \mathcal{C})$, where $\mathcal{P}$ is a finite set of predicates and $\mathcal{C}$ is a (possibly infinite) set of constants. Note that in this dissertation we will also call $\mathcal{P}$ a *vocabulary*. Each predicate has a fixed number of arguments, called its *arity*. Moreover, we assume that given an infinite set of variables $\mathcal{V}$. A *term* on $\mathcal{L}$ is a constant from $\mathcal{C}$ or a variable from $\mathcal{V}$. In our examples, we denote constants by letters at the beginning of the alphabet $(a, b, c, d \ldots)$ and variables by letters at the end of the alphabet $(x, y, z, w, \ldots)$.

An *atom* on $\mathcal{L}$ has the form $p(\mathbf{t})$ where $p \in \mathcal{P}$ is a predicate of arity n and $\mathbf{t}$ is a tuple of terms on $\mathcal{L}$ with $|\mathbf{t}| = n$. An atom with predicate p is also called a *p-atom*. We consider the classical inductive definition of an FO-formula. Given a formula or a set of formulas ϕ, we denote by $\mathtt{vars}(\phi)$, $\mathtt{consts}(\phi)$, $\mathtt{terms}(\phi)$, and $\mathtt{predicates}(\phi)$ its sets of variables, constants, terms, and predicates, respectively. We furthermore note $\phi[\mathbf{x}]$ to indicate that $\mathbf{x}$ is the tuple of free variables in ϕ; we may also partition this tuple into several ones, like e.g., in $\phi[\mathbf{x}, \mathbf{y}]$. We will often see a tuple $\mathbf{x}$ of pairwise distinct variables as a set. An FO-formula is said to be *ground* if it has no variable.

We first define classical logical interpretations.

Definition I.1 (Classical interpretation)
A (classical) *interpretation* of a logical language $\mathcal{L} = (\mathcal{P}, \mathcal{C})$ is a pair $\mathcal{I} = (\Delta, .^{\mathcal{I}})$ where Δ is a (possibly infinite) non-empty set called the interpretation domain and $.^{\mathcal{I}}$ is the interpretation function of the symbols of $\mathcal{L}$ such that:

- for each $c \in \mathcal{C}, c^{\mathcal{I}} \in \Delta$;
- for each $p \in \mathcal{P}$ of arity k, $p^{\mathcal{I}} \subseteq \Delta^k$.

An interpretation $\mathcal{I}$ of $\mathcal{L}$ is a *model* of an FO-formula ϕ built on $\mathcal{L}$ if it makes ϕ true by considering the classical interpretation of connectives and quantifiers. This is denoted by $\mathcal{I} \models \phi$.

Definition I.2 (Entailment, equivalence)
Given two FO-formulas ϕ and ψ, we denote by $\phi \models \psi$ the classical logical *entailment*, *i.e.*, $\phi \models \psi$ means that all the models of ϕ are models of ψ. If we have $\phi \models \psi$ and $\psi \models \phi$, we say that ϕ and ψ are (logically) *equivalent*, which is denoted by $\phi \equiv \psi$.

Entailment can also be defined between a possibly infinite set of FO-formulas Φ and a FO-formula ψ, denoted $\Phi \models \psi$, to mean that any interpretation $\mathcal{I}$ that is a model of all the formulas in Φ is a model of ψ.

In this work, we make the Unique Name Assumption (UNA), which states that distinct constants denote distinct individuals, hence are interpreted by distinct elements of a domain. It follows that any domain in an interpretation of $\mathcal{L} = (\mathcal{P}, \mathcal{C})$ contains a subset in bijection with $\mathcal{C}$. To simplify, we consider that each constant is interpreted by itself, which leads to the following definition.

Definition I.3 (Interpretation with UNA)
A *UNA-interpretation* of a logical language $\mathcal{L} = (\mathcal{P}, \mathcal{C})$ is an interpretation $\mathcal{I} = (\Delta, .^{\mathcal{I}})$ (in the sense of Definition I.1) such that:

- $\mathcal{C} \subseteq \Delta$;
- for each $c \in \mathcal{C}, c^{\mathcal{I}} = c$;

In the following, we will consider UNA-interpretations and simply call them interpretations. Note that taking UNA-interpretations instead of general interpretations does not make any difference in our framework, except for equality constraints (see Section VI.4).

I.2 Atomsets and homomorphisms

Among the FO-formulas, we will often consider existentially quantified conjunctions of atoms (to represent an instance, a conjunctive query, the body of a rule, etc.). Such formulas can be put in so-called prenex form, where all the quantifiers occur in front of the conjunction of atoms and apply to all the atoms. It is common to think of these formulas as *sets of atoms* (with possibly distinguished variables corresponding to the free variables of the formula), since their semantics depend only on the atoms that occur in them. In the following, we will use the term *atomset* to refer to a (possibly infinite) set of atoms. We will often denote by A an atomset, and by I a finite atomset (in particular, we will later define instances as atomsets).

Example I.1: Atomset

Let us consider the existentially closed formula $\exists x.\ (p(a,b) \wedge q(a,x,b) \wedge r(x))$. The associated atomset is $\{p(a,b), q(a,x,b), r(x)\}$.

Given an atomset A, we can define an interpretation that is "isomorphic" to A: the idea is that the atoms are interpreted by themselves.

Definition I.4 (Isomorphic interpretation)
Given an atomset A on $\mathcal{L} = (\mathcal{P}, \mathcal{C})$, its *isomorphic interpretation*, denoted by $\mathcal{I}(A)$, is the interpretation $(\Delta, .^{\mathcal{I}})$ defined as follows:

- Δ is in bijection with $\texttt{terms}(A) \cup \mathcal{C}$;
- The constants are interpreted by themselves, that is, for all $c \in \mathcal{C}, c^{\mathcal{I}} = c$;
- For each $p \in \mathcal{P}$, $p^{\mathcal{I}} = \{(t_1, \ldots, t_n)\} \mid p(t_1, \ldots, t_n) \in A\}$.

Note that the second point is satisfied by definition of a UNA-interpretation. We state it for the sake of clarity.

To a finite atomset I, we assign the *closed* formula that is the existential closure of the conjunction of its atoms. We call it *the formula associated with* I. Importantly, the isomorphic interpretation of I is a model of this formula (that is, $\mathcal{I}(I) \models I$).

In addition, we will sometimes have to say that the isomorphic interpretation of an atomset A is a model of a formula ϕ, that is, $\mathcal{I}(A) \models \phi$. To simplify the notation while avoiding confusion with $A \models \phi$ (which means that every model of the formula associated with A is a model of ϕ), we also denote this relation between an atomset and an FO-formula by $A \models_1 \phi$.

So, when we consider a *finite* atomset I, we can state two relations between I and an FO-formula ϕ: the fact that the formula associated with I entails ϕ, denoted $I \models \phi$, and the fact that the isomorphic interpretation of I is a model of ϕ, denoted $I \models_1 \phi$. Of course, $I \models \phi$ entails $I \models_1 \phi$ for any I and ϕ but the converse is not true.

Definition I.5 (Substitution)
Let X be a set of variables and T be a set of terms. A *substitution* s of X by T is a mapping from X to T. Given an atom $\alpha = p(t_1, \ldots, t_n)$, $s(\alpha)$ is the atom obtained by substituting each occurrence of the variable $x \in \texttt{vars}(\alpha) \cap X$ by $s(\alpha)$, that is, $s(\alpha) = p(s(t_1), \ldots, s(t_n))$. Given an atomset A, $s(A)$ is the atomset obtained by applying s to each atom of A, that is, $s(A) = \{s(\alpha) \mid \alpha \in A\}$.

Definition I.6 (Homomorphisms)
Given two atomsets A_1 and A_2, a *homomorphism* h from A_1 to A_2 is a substitution of $\texttt{vars}(A_1)$ by $\texttt{terms}(A_2)$ such that $h(A_1) \subseteq A_2$ (we say that A_1 *maps* to A_2 (by h)). A homomorphism h from A_1 to A_2 is an *isomorphism* if h^{-1} is a homomorphism from A_2 to A_1. An *endomorphism* (respectively *automorphism*) of an atomset A_1 is a homomorphism (respectively isomorphism) from A_1 to A_1.

When A_1 maps to A_2 and A_2 maps to A_1, we say that A_1 and A_2 are *homomorphically equivalent*, which is denoted by $\equiv$.

Example I.2: Equivalence and Isomorphism

Let us consider the atomsets A_1 and A_2 defined as follows:
$A_1 = \{p(a,b), p(b,b), p(a,x), p(x,y), p(y,b)\}$
$A_2 = \{p(a,b), p(b,b)\}$
Since $A_2 \subseteq A_1$, A_2 maps to A_1 by the identity. We also have that A_1 maps to A_2 by the homomorphism $h = \{x \mapsto b, y \mapsto b\}$. Hence, $A_1 \equiv A_2$. Note, however, that they are not isomorphic (indeed, h^{-1} is not a homomorphism). Note also that since $A_2 \subseteq A_1$, h is actually an endomorphism.

It is well known that for finite atomsets I_1 and I_2, the following holds: I_1 maps to I_2 iff $I_2 \models I_1$ (and iff $I_2 \models_1 I_1$). Hence, (the formulas assigned to) I_1 and I_2 are logically equivalent if and only if I_1 and I_2 are homomorphically equivalent.

The concept of homomorphism could be extended to sets of atoms with equality. Since we will use equality in very specific cases (see Section I.6 about backward chaining) we prefer to avoid adding new definitions. Instead, we assume that when dealing with atomsets that may contain equalities, we implicitly perform the substitutions associated with equalities before seeking a homomorphism.

When there is a homomorphism from an atomset A to an atomset B, we also say that A is *more general* than B or, conversely, that B is *more specific* than A.

The notion of homomorphism can also be defined between interpretations:

Definition I.7 (Homomorphism between interpretations)
Let $\mathcal{L} = (\mathcal{P}, \mathcal{C})$ be a logical language and $\mathcal{I}_1 = (\Delta_1, .^{\mathcal{I}_1})$ and $\mathcal{I}_2 = (\Delta_2, .^{\mathcal{I}_2})$ be two interpretations of $\mathcal{L}$. A *homomorphism* h from $\mathcal{I}_1$ to $\mathcal{I}_2$ is a mapping from Δ_1 to Δ_2 such that:

- for each $p \in \mathcal{P}$ and each $(t_1, \ldots, t_n) \in p^{\mathcal{I}_1}$, $(h(t_1), \ldots, h(t_n)) \in p^{\mathcal{I}_2}$;
- for each $c \in \mathcal{C}$, $h(c^{\mathcal{I}_1}) = c^{\mathcal{I}_2}$;

Then, we can define the notion of closure by homomorphism: a class C of (closed) formulas is *closed by homomorphism* if for any formula ϕ in C, for any interpretations $\mathcal{I}_1$ and $\mathcal{I}_2$, if $\mathcal{I}_1 \models \phi$ and $\mathcal{I}_1$ maps to $\mathcal{I}_2$, then $\mathcal{I}_2 \models \phi$.

The following properties are immediate but worth mentioning:

- Given two atomsets A_1 and A_2, there is a homomorphism from A_1 to A_2 if and only if there is a homomorphism from $\mathcal{I}(A_1)$ to $\mathcal{I}(A_2)$.
- The formulas associated with finite atomsets I are closed under homomorphism.

Finally, note that we can also define the *atomset isomorphic to an interpretation* in a natural way: in this translation, the domain elements that occur in the predicate interpretations and are not constants are translated into variables. Then we can check that homomorphic interpretations are translated into homomorphic atomsets.

Definition I.8 (Variable renaming, safe renaming, safe extension)
A *variable renaming* is an injective substitution whose range is a set of variables. A *safe renaming* is a variable renaming to a set of "fresh" variables; a fresh variable x is an element of a totally ordered infinite set of variables $\mathcal{V}_f$, which is disjoint from the set of variables used in the input formulas, such that x is strictly greater than all elements of $\mathcal{V}_f$ already introduced in the considered context. A *safe extension* s^+ of a substitution s from X (to T) to a set of variables $Y \supseteq X$ is the substitution from Y (to $T \cup \mathcal{V}_f$) defined as follows: if $x \in X$, $s^+(x) = s(x)$ otherwise $s^+(x) = x'$ where x' is a fresh variable from V_f.

It is important to note that some atoms within a set of atoms may be redundant, in the sense that they do not convey additional information. To refer to a minimal subset equivalent to the whole set, we use the concept of *core*. Although this concept finds its roots in graph theory (see [Hell and Nesetril, 2004] for details), it is readily adaptable to sets of atoms. An atomset is a *core* if it is not homomorphically equivalent to one of its strict subsets (that is, it does not map to one of its strict subsets). Next, we define a core of an atomset.

Definition I.9 (Core)
A *core of a set of atoms* A is a subset of A that is homomorphically equivalent to A and inclusion-wise minimal with respect to this property.

In other words, a subset A' of A is a core of A if A maps to A' and A' is a core.

For a finite set of atoms A, there always exists a core of A and all the cores of A are isomorphic; that is why we can talk of "the" core of A, which we denote by core(A). However, this does not hold for A an infinite set of atoms: then A may have several non-isomorphic cores or even no core at all [Carral et al., 2018].

Example I.3: Core of a set of atoms

Consider A_1 and A_2 from Example I.2: A_2 is a core of A_1 because $A_2 \subseteq A_1$, A_1 maps to A_2 and A_2 is minimal. Here, the atoms $p(a,x)$, $p(x,b)$ and $p(x,y)$ are redundant in A_1.

I.3 Knowledge bases

In general, a knowledge base is composed of an ontology given as a set of axioms that represent some domain knowledge and a set of facts that represent specific situations. Two main families of ontology languages have been considered in the literature:

- *Description Logics* (DLs), which have been specially designed to represent ontologies and reason with them [Baader et al., 2007]; they notably underlie the Semantic Web ontological language OWL.
- *Existential rules*, a more recent family of languages, specifically oriented towards query answering [Baget et al., 2011, Calí et al., 2009].

I.3.1 Description Logics

A DL ontology essentially contains axioms that express inclusions between concepts and between binary relations (called roles). More specifically, a concept can be either a concept name or a complex concept built from concept names using a set of constructors. Similarly, a role can be either a role name or a complex role built from role names using a set of constructors. Then an axiom is generally a concept inclusion of the form $C_1 \sqsubseteq C_2$, where C_1 and C_2 are concepts, or a role inclusion of the form $r_1 \sqsubseteq r_2$, where r_1 and r_2 are roles. In some DLs, special axioms furthermore declare some properties of a role, like transitivity or functionality. The expressiveness of a specific DL depends on the allowed set of constructors and shape of the axioms.

In general, DLs can be translated into specific fragments of first-order logic for which classical reasoning tasks (like determining whether a knowledge base is satisfiable or whether a ground atom is entailed by the knowledge base) are decidable. The translation sketchily proceeds as follows: a concept name is seen as a unary predicate and a role name as a binary predicate. Then a concept C is translated into a formula $\phi_C(x)$ with free variable x; if C is a concept name, then $\phi_C(x) = C(x)$. Similarly, a role r is translated into a formula $\phi_r(x, y)$ with free variables x and y; if r is a role name, then $\phi_r(x, y) = r(x, y)$. Finally, a concept inclusion $C_1 \sqsubseteq C_2$ is translated into the formula $\forall x\, (\phi_{C_1}(x) \rightarrow \phi_{C_2}(x))$ and a role inclusion $r_1 \sqsubseteq r_2$ into $\forall x \forall y\, (\phi_{r_1}(x, y) \rightarrow \phi_{r_2}(x, y))$.

In this dissertation, we will refer to lightweight DLs that are used in the context of query answering, as they have good computational properties for this problem. The axioms in these DLs can be logically translated into existential rules, hence we will present them in Section I.3.3.

I.3.2 Existential rules

Existential rules can be seen as an extension of function-free Horn rules (aka Datalog) with existentially quantified variables in rule heads. Next, we call them *conjunctive* rules to distinguish them from their extension to *disjunctive* rules studied in Chapter III. Note that existential rules have the same logical form as very general integrity constraints that have long been studied in databases, called Tuple Generating Dependencies (TGDs) [Abiteboul et al., 1995].

Definition I.10 (Existential conjunctive rule)
An *existential conjunctive rule* (or simply a conjunctive rule) is a formula

$$R = \forall \mathbf{x} \forall \mathbf{y} (B[\mathbf{x}, \mathbf{y}] \rightarrow \exists \mathbf{z} H[\mathbf{y}, \mathbf{z}])$$

where $\mathbf{x}, \mathbf{y}$ and $\mathbf{z}$ are tuples of variables, $B = \mathsf{body}(R)$, and $H = \mathsf{head}(R)$ are conjunctions of atoms, respectively, called the body and head of R. The *frontier* of R, denoted by $\mathrm{fr}(R)$, is the set $\mathrm{vars}(B) \cap \mathrm{vars}(H) = \mathbf{y}$. The set of *existential variables* in R is the set $\mathrm{vars}(H) \setminus \mathrm{fr}(R) = \mathbf{z}$.

Example I.4: Conjunctive rule

Let us consider the conjunctive rule $R = \forall x \forall y (p(x,y) \rightarrow \exists z\ q(y,z))$. In this dissertation, we will omit universal quantifiers and note $R = p(x,y) \rightarrow \exists z\ q(y,z)$. Here, $p(x,y)$ is the body of R and $q(y,z)$ is its head; y is the only frontier variable and z the only existential variable.

Of particular interest is the subclass of rules with an empty set of existential variables, called *Datalog*.

We can now define a knowledge base in the existential rule framework. As already mentioned, a knowledge base is generally composed of a finite set of facts and a finite set of axioms. In the existential rule framework, the axioms are expressed by existential rules, and a finite set of facts is often called an instance. This instance may be ground, in which case we call it a database instance, or it may more generally have existentially quantified variables, i.e., correspond to an existentially closed conjunction of atoms.

Definition I.11 (Instance, Knowledge base)
An *instance* is an existential closed conjunction of atoms, which is also seen as a finite atomset. A *database instance* is ground. A *knowledge base* (KB) is a pair $\mathcal{K} = (I, \mathcal{R})$ where I is an instance and $\mathcal{R}$ is a set of existential rules.

Note: In the following, we will silently assume that an existential rule does not introduce a *new* constant, i.e., for any rule R, $\mathtt{consts}(\mathsf{head}(R)) \subseteq \mathtt{consts}(\mathsf{body}(R))$. This is to simplify technical developments in Chapters 6 and 7.

I.3.3 Lightweight DLs and existential rules

Lightweight description logics are generally used in the context of data querying, and this is even more true in the context of ontology-based data access, i.e. in the presence of mappings. We focus here on two families of lightweight DLs, namely $\mathcal{EL}$ [Baader et al., 2005, Lutz et al., 2009] and DL-Lite [Calvanese et al., 2007b, Artale et al., 2009] More precisely, for the first family we will define the specific DLs $\mathcal{EL}$, $\mathcal{ELH}$, $\mathcal{ELI}$ and $\mathcal{ELHI}$, and for the second family we will define the specific DLs DL-Lite$_{\text{core}}$, DL-Lite$_{\mathcal{R}}$ and DL-Lite$_{\text{RDFS}}$. Indeed, these DLs are relevant to describe related work.

Answering conjunctive queries with all these DLs can be done in polynomial time in terms of data complexity (see later). We will point out that these DLs can be translated into specific existential rule classes (which also feature polynomial time conjunctive query answering). The DL $\mathcal{EL}$ is the simplest member of the $\mathcal{EL}$ family. In this DL, the axioms are concept inclusions $C_1 \sqsubseteq C_2$, where the C_i have the following shape:

$$C_i := \top \mid A \mid C_1 \sqcap C_2 \mid \exists r.C_1$$

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

Table I.1: Translation of normal $\mathcal{EL}$-axioms (with implicit universal quantifiers)

with A and r concept and role names, respectively. The extensions to inverse roles and to role inclusions are indicated by the letters $\mathcal{I}$ and $\mathcal{H}$, respectively. This yields the DLs $\mathcal{ELI}$, $\mathcal{ELH}$ and $\mathcal{ELHI}$. Hence, axioms in $\mathcal{ELHI}$ are of the form $C_1 \sqsubseteq C_2$ and $s_1 \sqsubseteq s_2$, where:

$$C_i := \top \mid A \mid C_1 \sqcap C_2 \mid \exists s_1.C_1$$
$$s_i := r \mid r^-$$

where A and r are names, and r^- is the inverse of r.

DLs are usually provided with axioms in a normal form: then, all axioms that can be expressed can be rewritten as normal axioms, which may require to introduce new concept and role names, while preserving the semantics (more precisely, the new set of axioms is a conservative extension in the sense of [Baader et al., 2007]). For example, $\exists r.\exists s.A \sqsubseteq B_1 \sqcap B_2$ can be rewritten as three normalised axioms: $\exists s.A \sqsubseteq B_3$, $\exists r.B_3 \sqsubseteq B_1$ and $\exists r.B_3 \sqsubseteq B_2$, where B_3 is a fresh concept name. Table I.1 shows a set of normal axioms for $\mathcal{EL}$ and their logical translation (note that universal quantifiers are implicit). In $\mathcal{ELI}$, r can be a role or its inverse r^-, and we have $\phi_{r^-}(x,y) = r(y,x)$. In $\mathcal{ELH}$, role inclusions are of the form $r_1 \sqsubseteq r_2$, which is translated by the rule $r_1(x,y) \sqsubseteq r_2(x,y)$. Finally, $\mathcal{ELHI}$ combines both extensions.

The most well-known member of the DL-Lite family is DL-Lite$_R$, which underlines OWL2 QL (a tractable profile of OWL 2). In this DL, axioms have the following shape:

$$B_1 \sqsubseteq B_2 \quad B_1 \sqsubseteq \neg B_2 \quad s_1 \sqsubseteq s_2 \quad s_1 \sqsubseteq \neg s_2$$

where $B_i := A \mid \exists s$ and $s_i := r \mid r^-$, with A and r names. The inclusions without $\neg$ are said positive, the others are negative. Note that $B_1 \sqsubseteq \neg B_2$ could equivalently be written $B_1 \sqcap B_2 \sqsubseteq \bot$. And similarly for $s_1 \sqsubseteq \neg s_2$. In other words, negative axioms express concept and role disjointness.

DL-Lite$_{\text{core}}$ is the restriction of DL-Lite$_R$ to concept inclusions (i.e., the two first axiom shapes). DL-Lite$_{\text{RDFS}}$ is the restriction of DL-Lite$_R$ to positive inclusions without $\exists r$ in the right side, i.e., B_2 is restricted to a concept name in $B_1 \sqsubseteq B_2$.

Table I.2 shows the set of axioms for DL-Lite$_R$ and their logical translation.

Clearly, $\mathcal{ELHI}$ can be expressed in a fragment of existential rules. More specifically, the normal axioms given in Table I.1, as well as their extension with inverse roles and

DL-Axiom	Translated rule
$A_1 \sqsubseteq A_2$	$A_1(x) \rightarrow A_2(x)$
$A \sqsubseteq \exists r$	$A(x) \rightarrow \exists y\ r(x,y)$
$\exists r \sqsubseteq A$	$r(x,y) \rightarrow A(x)$
$r_1 \sqsubseteq r_2$	$r_1(x,y) \rightarrow r_2(x,y)$
$A_1 \sqsubseteq \neg A_2$	$A_1(x) \wedge A_2(x) \rightarrow \bot$
$r_1 \sqsubseteq \neg r_2$	$r_1(x,y) \wedge r_2(x,y) \rightarrow \bot$

In all the axioms, $r_{(i)}$ can be replaced by its inverse $r^-_{(i)}$ (then (x,y) is replaced by (y,x) in the translation)

Table I.2: Translation of normal DL-Lite$_R$ axioms (with implicit universal quantifiers)

role inclusions can be expressed as guarded existential rules, which are existential rules in which each rule body has an atom – called a guard – that contains all the body variables (see [Mugnier, 2020] for details).

The positive axioms of DL-Lite$_R$ can be translated into the class of linear existential rules, where the body and the head of rules are restricted to a single atom; the negative axioms can be translated into so-called negative constraints (see, e.g. [Calí et al., 2009]). Note that negative axioms play a role in the (un)satisfiability of a KB, but when the KB is satisfiable they can be ignored in query answering.

I.4 Query answering

Query answering is the fundamental task on databases. It has become one of the main tasks on knowledge bases with the development of applications making intensive use of data. However, an important distinction between databases and KBs is the assumption about the meaning of an instance: in databases, the instance is usually considered as a complete set of facts, i.e. missing facts are assumed to be false (this is the closed-world assumption, CWA) while in KBs the instance is only a set of known facts (this is the open-world assumption, OWA).

In line with this, a Boolean query Q has a positive answer on a database instance I if (the interpretation isomorphic to) I is a model of Q (i.e., $\mathcal{I}(I) \models Q$), while it has a positive answer on a KB's instance I if I entails Q (i.e., $I \models Q$).

To distinguish between both notions, we will talk about answer and query evaluation for a database instance and of certain answer and query answering for a KB. Note that in the sequel of this dissertation we will often simply say answer, when it is clear from the context.

Definition I.12 (FO-query)
A *first-order query* (in short FO-query) Q is an FO-formula. The free variables in Q are called *answer variables* and denoted by $\mathtt{ansVars}(Q)$. When convenient, we also denote a query by $Q[\mathbf{x}]$, where $\mathbf{x}$ is the set of answer variables. The *arity* of $Q[\mathbf{x}]$ is $|\mathbf{x}|$. A *Boolean*

query has no answer variables.

Definition I.13 (Answer)
Given an instance I and a query $Q[\mathbf{x}]$, a tuple of constants $\mathbf{c}$ with $|\mathbf{c}| = |\mathbf{x}|$ is an *answer* to Q on I if, given the substitution s from $\mathbf{x}$ to $\mathbf{c}$ that assigns to each variable from $\mathbf{x}$ the constant from $\mathbf{c}$ of the same rank, I is a model of $s(Q)$, i.e., $I \models_1 s(Q)$. The set of answers to Q on I is denoted by $Q(I)$.

We also say that the answer to a Boolean query $\mathcal{Q}$ on I is positive if $Q(I) = \{()\}$ (and it is negative if $Q(I) = \emptyset$).

Let us now consider knowledge bases and define query answering.

Definition I.14 (Certain answer)
Given a KB $\mathcal{K} = (I, \mathcal{R})$ and a query $Q[\mathbf{x}]$, a tuple of constants $\mathbf{c}$ with $|\mathbf{c}| = |\mathbf{x}|$ is a *certain answer* to Q if, given the substitution s from $\mathbf{x}$ to $\mathbf{c}$ that assigns to each variable from $\mathbf{x}$ the constant from $\mathbf{c}$ of the same rank, $\mathcal{K} \models s(Q)$. The set of certain answers to Q on $\mathcal{K}$ is denoted by $\mathtt{certain}(Q, \mathcal{K})$. When $\mathcal{K} = (I, \emptyset)$, we simply note $\mathtt{certain}(Q, I)$.

In other words, the set of certain answers to Q on $\mathcal{K}$ is the intersection of the sets of answers to Q on all the instances associated with the models of $\mathcal{K}$.

In the following, we will mainly consider the basic database queries, known as (unions of) conjunctive queries (UCQs). In chapter V, we deal with an extension of UCQs named $\text{UCQ}^{\mathbf{C},\neq}$.

Definition I.15 (Conjunctive Query, Union of Conjunctive Queries)
A *conjunctive query* (CQ) q is an FO-query of the form $\exists \mathbf{y}\ \phi[\mathbf{x}, \mathbf{y}]$, where $\mathbf{x}$ and $\mathbf{y}$ are disjoint tuples of variables, $\mathbf{x}$ are the answer variables, and ϕ is a finite conjunction of atoms with $\mathtt{vars}(\phi) = \mathbf{x} \cup \mathbf{y}$. In a *full* CQ, $\mathbf{y}$ is empty. An *atomic* CQ has a single atom. A *union of conjunctive queries* (UCQ) is a disjunction of CQs with the same tuple of answer variables $\mathbf{x}$.

For clarity, we will denote a UCQ by $\mathcal{Q}$ and a CQ by q. Note that we will sometimes consider UCQs with equality atoms whose role is only to ensure that all the CQs in the UCQ have the same tuple of answer variables. These equality atoms occur only between answer variables and/or constants of a CQ, in such a way that they can be removed by substituting variables and yield a CQ as defined above.

It is well known that for UCQs, and more generally all query languages closed by homomorphism (see Definition I.7), the notions of answers and certain answers coincide, in the sense that for any query Q in such language and for any instance I, $Q(I) = \mathtt{certain}(Q, I)$.

Finally, we formally define the fundamental decision problem of UCQ entailment.

Problem I.5: UCQ Entailment

The *UCQ entailment problem* takes as input a KB $\mathcal{K} = (I, \mathcal{R})$ and a Boolean UCQ $\mathcal{Q}$, and asks if $I, \mathcal{R} \models \mathcal{Q}$.

This problem has long been known to be undecidable with general existential rules [Beeri and Vardi, 1981]. However, many decidable subclasses are known, see, e.g., [Thomazo, 2013] for a synthesis.

In the following two sections, we will present the main approaches to answer queries on an existential rule KB. Very roughly, there are two ways of taking rules into account: either in the instance (this is forward chaining) or in the query (this is backward chaining). In both cases, this allows to "reduce" the query answering task to evaluating a UCQ on an instance, provided that the process terminates.

I.5 Query Answering based on Forward Chaining

We now define the fundamental notions related to forward chaining with (conjunctive) existential rules, also called *chase* in database theory.

Definition I.16 (Trigger)
Given an instance I and an existential conjunctive rule R, a *trigger* for R on I is a pair (R, h) where h is a homomorphism from $\mathsf{body}(R)$ to I.

When such a trigger exists, the rule R is said to be *applicable* on I.

Example I.6: Trigger

Consider the rule $R = p(x, y) \rightarrow \exists z\, q(y, z)$ and the instance $I = \{p(a, b)\}$. The only trigger for R on I is (R, h) with $h = \{x \mapsto a, y \mapsto b\}$.

Definition I.17 (Trigger application)
Given a trigger (R, h) for a rule R on an instance I, the *application of this trigger*, denoted by the operator α, is defined as

$$\alpha(I, R, h) = I \cup h^+(\mathsf{head}(R))$$

We recall that h^+ denotes a safe extension of the homomorphism h.

Example I.7: Trigger Application

Consider again the rule $R = p(x, y) \rightarrow \exists z\, q(y, z)$, the instance $I = \{p(a, b)\}$ and the trigger (R, h) with $h = \{x \mapsto a, y \mapsto b\}$. Then

$$\alpha(I, R, h) = I \cup h^+(q(y, z)) = \{p(a, b), q(b, z')\}$$

Here, z' is the fresh variable introduced by the safe renaming of z.

Definition I.18 (Derivation)
A *derivation* from an instance I and a set of rules $\mathcal{R}$ is a sequence of instances $\mathcal{D} = I_0, I_1, I_2, \ldots$ such that $I_0 = I$ and, for all $i \geq 1$, $I_i = \alpha(I_{i-1}, R, h_i)$ for some trigger (R, h_i) on I_{i-1} for $R \in \mathcal{R}$.

Example I.8: Derivation

Let us consider the instance $I = \{t(a)\}$ and the set of rules $\mathcal{R} = \{R = t(x) \rightarrow \exists y\, t(y) \wedge q(y,x)\}$. We can start a derivation with a trigger (R,h) where $h = \{x \mapsto a\}$. Applying the trigger gives $I_1 = \alpha(I,R,h) = I \cup h^+(\{t(y),q(y,x)\}) = \{t(a),t(y_1),q(y_1,a)\}$, where y_1 is a fresh variable introduced by the safe renaming.
We could continue the derivation with another application of R, this time with $h' = \{x \mapsto y_1\}$. This results in $I_2 = \alpha(I_1,R,h') = I_1 \cup h'^+(\{t(y),q(y,x)\}) = \{t(a),t(y_1),q(y_1,a),t(y_2),q(y_2,y_1)\}$, where y_2 is a fresh variable introduced by the safe renaming.
We can notice here that this derivation is not finite, since we can always apply a new trigger on what was produced in the previous step. In fact, we have $I_k = \alpha(I_{k-1},R,h_k = \{x \mapsto y_{k-1}\}) = I_{k-1} \cup h_k^+(\{t(y),q(y,x)\})$ for any $k > 1$. Thus, the derivation from I and $\mathcal{R}$ is the infinite sequence $\mathcal{D} = I_0, I_1, I_2, \ldots$.
Also note that, in this example, there is only one trigger to apply at each step and so there is a unique way to build the derivation. But in the general case, there can be several ways to build a derivation by choosing different orders of application of the triggers.

It is convenient to consider a derivation performed in a breadth-first manner, i.e., by computing all possible rule applications in parallel at each step. This leads to the following notion of k-saturation obtained after k parallel steps.

Definition I.19 (Direct saturation, k-Saturation)
Let I be an instance and $\mathcal{R}$ be a set of rules. We denote by $\Pi(\mathcal{R},I)$ the set of triggers for the rules in $\mathcal{R}$ on I: $\Pi(\mathcal{R},I) = \{(R,h) \mid R \in \mathcal{R}$ and h maps $\mathsf{body}(R)$ to $I\}$.

The *direct saturation* of I with $\mathcal{R}$ is defined as:

$$\alpha(I,\mathcal{R}) = I \cup \bigcup_{(R,h)\in\Pi(\mathcal{R},I)} h^+(\mathsf{head}(R))$$

The *k-saturation* of I with $\mathcal{R}$ is denoted by $\alpha_k(I,\mathcal{R})$ and is inductively defined as $\alpha_0(I,\mathcal{R}) = I$ and $\alpha_i(I,\mathcal{R}) = \alpha(\alpha_{i-1}(I,\mathcal{R}),\mathcal{R})$ for $i > 0$.

Example I.9: k-Saturation

Consider the set of rules $\mathcal{R} = \{R_1, R_2\}$ where:

- $R_1 : r(x_1,y_1) \wedge r(y_1,z_1) \rightarrow r(x_1,z_1)$;
- $R_2 : r(x_2,y_2) \wedge q(x_2) \wedge p(y_2) \rightarrow \exists z_2\, s(x_2,y_2,z_2)$.

Given the initial instance $I = \{q(a), r(a,b), r(b,c), r(c,t), p(t)\}$:

$$\alpha_0(I,\mathcal{R}) = I.$$

In the first saturation step, we identify the triggers:

$$(R_1, h_1) \text{ with } h_1 = \{x_1 \mapsto a, y_1 \mapsto b, z_1 \mapsto c\}$$

$$(R_1, h_2) \text{ with } h_2 = \{x_1 \mapsto b, y_1 \mapsto c, z_1 \mapsto t\}$$

Applying these triggers, we produce:

$$\alpha_1(I, \mathcal{R}) = \alpha_0(I, \mathcal{R}) \cup \{r(a, c), r(b, t)\}.$$

In the next saturation step, using the inferred facts from $\alpha_1(I, \mathcal{R})$, we can apply $(R_1, \{x_1 \mapsto a, y_1 \mapsto b, z_1 \mapsto t\})$. This gives:

$$\alpha_2(I, \mathcal{R}) = \alpha_1(I, \mathcal{R}) \cup \{r(a, t)\}.$$

Finally, in the last step, we can apply $(R_2, \{x_2 \mapsto a, y_2 \mapsto t\})$. This gives

$$\alpha_3(I, \mathcal{R}) = \alpha_2(I, \mathcal{R}) \cup \{s(a, t, z_2')\}.$$

After the third iteration, we have reached a point where no new facts can be derived.
Note that in the first step of the derivation, we applied two triggers at the same time instead of applying them one after the other like in a derivation (see Example I.8).

Definition I.20 (Saturation of a KB)
Let $\mathcal{K} = (I, \mathcal{R})$ be a KB. The *saturation of* $\mathcal{K}$, denoted by $\alpha_\infty(I, \mathcal{R})$ or $\alpha_\infty(\mathcal{K})$, is defined as:

$$\alpha_\infty(I, \mathcal{R}) = \bigcup_{k \in \mathbb{N}} \alpha_k(I, \mathcal{R})$$

In the database literature, the saturation process is called the chase. Therefore, instead of $\alpha_\infty(I, \mathcal{R})$, we simply note $\mathsf{chase}(I, \mathcal{R})$.

Example I.10: Saturation

Considering again Example I.8 with $I = \{t(a)\}$ and $\mathcal{R} = \{t(x) \to \exists y\ t(y) \wedge q(y, x)\}$, we have:

- $\alpha_0(I, \mathcal{R}) = I$,
- $\alpha_1(I, \mathcal{R}) = \{t(a), t(y_1), q(y_1, a)\}$,
- $\alpha_2(I, \mathcal{R}) = \{t(a), t(y_1), q(y_1, a), t(y_2), q(y_2, y_1)\}$
- ...

where each y_i is a new element introduced by the safe renaming. The saturation of I and $\mathcal{R}$ is the union of all these sets:

$$\alpha_\infty(I,\mathcal{R}) = \bigcup_{k\in\mathbb{N}} \alpha_k(I,\mathcal{R}) = \{t(a), t(y_1), q(y_1,a), t(y_2), q(y_2,y_1), t(y_3), q(y_3,y_2), \ldots\}$$

The saturation of a KB has a very nice property: not only its isomorphic interpretation is a model of the KB, but it furthermore maps by homomorphism to any model of the KB. It is called a *universal model* in the database literature.

Definition I.21 (Universal Model)
An interpretation $\mathcal{I}$ is a *universal model* of an instance I and a set of rules $\mathcal{R}$ if it satisfies the following conditions:

1. $\mathcal{I}$ is a model of I and $\mathcal{R}$, which means $\mathcal{I}$ satisfies I and all rules in $\mathcal{R}$.
2. For any other model $\mathcal{I}'$ of I and $\mathcal{R}$, there is a homomorphism from $\mathcal{I}$ to $\mathcal{I}'$.

In the following, we call *canonical model* of the KB $(I,\mathcal{R})$ the interpretation isomorphic to its saturation. To know if a Boolean UCQ Q is entailed by $\mathcal{K}$, it suffices to check whether the canonical model of $\mathcal{K}$ is a model of Q (note that this is not only true for the UCQ language, but also for any query language closed by homomorphism).

The next theorem follows form previous definitions.

Theorem I.11: Canonical model and entailment (e.g., [Baget et al., 2011])

Let I be an instance, $\mathcal{R}$ be a set of rules and q be a Boolean CQ. The following properties are equivalent:

1. $(I,\mathcal{R}) \models q$;
2. There exists a homomorphism from q to $\alpha_\infty(I,\mathcal{R})$;
3. There exists an integer k such that there is a homomorphism from q to $\alpha_k(I,\mathcal{R})$.

Query answering is not decidable with general existential rules, but, obviously, it becomes decidable when the chase is finite. This observation leads to the notion of "finite expansion set", which is a set of rules ensuring that, for any instance, there is a step k such that the possibly infinite saturation is homomorphically equivalent to the k-saturation.

Definition I.22 (Finite Expansion Set [Baget et al., 2011])
A set of rules $\mathcal{R}$ is said to be a *finite expansion set (FES)* if and only if, for every instance I, there exists an integer k such that $\alpha_k(I,\mathcal{R}) \equiv \alpha_\infty(I,\mathcal{R})$.

The problem of determining whether a set of rules is a FES is undecidable [Baget et al., 2011].

Example I.12: Finite Expansion Set (from [König, 2014])

Consider the instance $I = \{q(a)\}$ and the set of rules $\mathcal{R} = \{q(x) \rightarrow \exists y\ r(x,y) \wedge r(y,y) \wedge q(y)\}$. Applying the rules, we have:

$$\alpha_1(I,\mathcal{R}) = I \cup \{r(a,y_1), r(y_1,y_1), q(y_1)\}$$
$$\alpha_2(I,\mathcal{R}) = \alpha_1(I,\mathcal{R}) \cup \{r(y_1,y_2), r(y_2,y_2), q(y_2)\}$$
$$\alpha_3(I,\mathcal{R}) = \alpha_2(I,\mathcal{R}) \cup \{r(y_2,y_3), r(y_3,y_3), q(y_3)\}$$
$$\alpha_4(I,\mathcal{R}) = \ldots$$

We observe that $\alpha_\infty(I,\mathcal{R})$ is infinite, yet it is equivalent to $\alpha_1(I,\mathcal{R})$. This is because $\alpha_1(I,\mathcal{R}) \subseteq \alpha_\infty(I,\mathcal{R})$, and every set $\{r(y_i,y_{i+1}), r(y_{i+1},y_{i+1}), q(y_{i+1})\} \subseteq \alpha_\infty(I,\mathcal{R})$ can be mapped to $\{r(y_1,y_1), q(y_1)\}$ via the homomorphism $\{y_i \mapsto y_1, y_{i+1} \mapsto y_1\}$.

I.6 Query Answering based on Backward Chaining

Historically, backward chaining techniques were first used in logic programming, especially in Prolog. The aim was to prove that a CQ q is entailed by a logic program P by showing that $P \wedge \neg q$ cannot be satisfied using resolution. In the context of query answering with DL-Lite ontologies, this method was divided into two main steps. The first step, known as "query rewriting", involves turning the initial query into a set of rewritings, seen as a UCQ. The second step consists of evaluating this UCQ on the instance [Poggi et al., 2008].

The main reason for this split was to take advantage of the optimisations offered by relational database systems, assuming that the data are stored that way. This separation has several additional benefits. It works well when data is scattered across different databases or when there are limits on editing the facts. Also, backward chaining avoids problems related to expanding a fact base. The query rewriting process is not affected by changes in the instance; on the other hand, saturation must be recalculated when the instance is modified.

When it comes to existing rewriting techniques, they can be classified by the target query language. Initially, the aim was to rewrite a UCQ into another UCQ [Gottlob et al., 2011,Chortaras et al., 2011,Rodriguez-Muro et al., 2013,Baget et al., 2011,König, 2014, König et al., 2015]. This framework has been the focus of most studies. Since a UCQ cannot always be rewritten as a (finite) UCQ, two notions were introduced to define rule sets that ensure a finite rewriting exists, namely FO-rewritable and FUS (which we will also name UCQ-rewritable). Even though FO-rewritable seems broader than FUS, it has been pointed out that they cover the same rule classes. It remains that some forms of FO-queries are more compact than UCQs, even exponentially smaller in some cases, as highlighted in [Thomazo, 2013] about so-called unions of semi-conjunctive

queries. Note however that smaller queries are not always more efficiently evaluated, since it is important to take properties of the data into account as shown in [Bursztyn et al., 2015].

Later, other languages than FO-queries were considered for the target query. In particular, some techniques rewrite the query into a Datalog program, which provides more expressivity than first-order logic [Pérez-Urbina et al., 2010, Gottlob and Schwentick, 2011, Eiter et al., 2012, Trivela et al., 2015]. Hence, targeting Datalog instead UCQs increases the number of cases where a finite rewriting exists. In this dissertation, we focus on UCQ rewritings.

We first define sound and complete rewritings.

Definition I.23 (Sound and Complete Rewriting)

- **Sound Rewriting:** A query $\mathcal{Q}'$ is called a *sound rewriting* of a query $\mathcal{Q}$ with respect to a set of rules $\mathcal{R}$ if $\texttt{certain}(\mathcal{Q}', I) \subseteq \texttt{certain}(\mathcal{Q}, (I, \mathcal{R}))$ for any instance I.
- **Complete Rewriting:** Conversely, a query $\mathcal{Q}'$ is called a *complete rewriting* of a query $\mathcal{Q}$ with respect to a set of rules if $\mathcal{R}$ $\texttt{certain}(\mathcal{Q}, (I, \mathcal{R})) \subseteq \texttt{certain}(\mathcal{Q}', I)$ for any instance I.

Furthermore, when $\mathcal{Q}'$ is a UCQ and a sound and complete rewriting of $\mathcal{Q}$ (with respect to $\mathcal{R}$), we call it a *UCQ-rewriting*. We recall that when $\mathcal{Q}'$ is a UCQ, $\texttt{certain}(\mathcal{Q}', I) = \mathcal{Q}'(I)$.

The following example shows that no all UCQs admit a UCQ-rewriting, even with respect to very simple Datalog rule sets.

Example I.13: Transitivity

Let $R = p(x,y) \wedge p(y,z) \rightarrow p(x,z)$. The (Boolean) CQ $q_1 = p(a,b)$, where a and b are constants, has no finite rewriting with $\{R\}$, while the (Boolean) CQ $q_2 = p(u,v)$ has one, which is $\{q_2\}$. Indeed, any complete rewriting of q_1 is infinite as it contains all the "paths" of p-atoms from a to b, which are pairwise incomparable by homomorphism. In contrast, the atom $p(u,v)$ maps by homomorphism to any path of p-atoms.

Such observations have motivated the following definitions.

Definition I.24 (UCQ-rewritable)
The pair $(\mathcal{Q}, \mathcal{R})$ with $\mathcal{Q}$ a UCQ and $\mathcal{R}$ a set of rules, is said to be *UCQ-rewritable* if there exists a UCQ-rewriting of $\mathcal{Q}$ with respect to $\mathcal{R}$. Moreover, we say that $\mathcal{R}$ is *UCQ-rewritable* if every pair $(\mathcal{Q}, \mathcal{R})$ is UCQ-rewritable.

In the literature, a UCQ-rewritable rule set is also called a *finite unification set (FUS)* [Baget et al., 2011].

Definition I.25 (FO-Rewritable [Artale et al., 2007])
A set of rules $\mathcal{R}$ is *FO-rewritable* if for any UCQ $\mathcal{Q}$, there exists an FO-query Q' that is a sound and complete rewriting of $\mathcal{Q}$.

As pointed out in [Rudolph and Krötzsch, 2013], the concepts of FO-rewritable sets and FUS (ou UCQ-rewritable sets) actually coincide.

Theorem I.14: [Rudolph and Krötzsch, 2013]

Let $\mathcal{R}$ be a set of rules. The following two conditions are equivalent:

1. $\mathcal{R}$ is a FO-rewritable set,
2. $\mathcal{R}$ is a finite unification set.

It is worth noting that it is undecidable to determine whether a set of rules is a finite unification set as per [Baget et al., 2011].

We now turn our attention to the computation of sound and complete rewritings.

Using classical logic programming techniques does not work well with existential rules. Indeed, the usual logic unification on these rules can lead to unsound rewritings. It follows that existential variables in rule heads have to be dealt with in a special way. We first explain why the usual unification cannot be used for existential rules.

Definition I.26 (Datalog Unification)
Let q be a CQ and R be a Datalog rule. A *Datalog unifier* of q with R is a pair $\mu = (\alpha, u)$, where α is an atom of q and u is a substitution, which maps variables from $\texttt{vars}(\alpha) \cup \texttt{vars}(\texttt{head}(R))$ to $\texttt{terms}(\texttt{head}(R)) \cup \texttt{consts}(q)$, such that the application of u to α and $\texttt{head}(R)$ results in the same atom, i.e., $u(\alpha) = u(\texttt{head}(R))$.

Example I.15: Datalog Unification

Consider the following set of Datalog rules:

$$R_1 : \text{parent}(x_1, y_1) \rightarrow \text{ancestor}(x_1, y_1),$$
$$R_2 : \text{parent}(x_2, z_2) \wedge \text{ancestor}(z_2, y_2) \rightarrow \text{ancestor}(x_2, y_2),$$

and the Datalog query $q = \exists v\ \text{ancestor}(xerces, v)$.
The Datalog unifiers between q and the rules R_1 and R_2 are:

- with R_1: $\mu_1 = (\text{ancestor}(xerces, v), u_1)$ with $u_1 = \{x_1 \mapsto xerces, y_1 \mapsto v\}$.
- with R_2: $\mu_2 = (\text{ancestor}(xerces, v), u_2)$ with $u_2 = \{x_2 \mapsto xerces, y_2 \mapsto v\}$.

We have:

- $u_1(\text{ancestor}(xerces, v)) = u_1(\text{head}(R_1)) = \text{ancestor}(xerces, v)$
- $u_2(\text{ancestor}(xerces, v)) = u_2(\text{head}(R_2)) = \text{ancestor}(xerces, v)$.

Definition I.27 (Datalog Rewriting)
Given a CQ q and a Datalog rule R, if $\mu = (\alpha, u)$ is a unifier of q with R, then the *rewriting* of q according to μ is defined as: $q' = u(B) \cup u(q \setminus \{\alpha\})$.

Example I.16: Datalog Rewriting

Consider again the set of Datalog rules and the query from Example I.15:

$$R_1 : \text{parent}(x_1, y_1) \rightarrow \text{ancestor}(x_1, y_1),$$
$$R_2 : \text{parent}(x_2, z_2) \wedge \text{ancestor}(z_2, y_2) \rightarrow \text{ancestor}(x_2, y_2)$$

and $q = \exists v\ \text{ancestor}(\text{xerces}, v)$.
Also take the same unifiers, i.e., $\mu_1 = (\text{ancestor}(\text{xerces}, v), u_1 = \{x_1 \mapsto \text{xerces}, y_1 \mapsto v\})$ for R_1 and $\mu_2 = (\text{ancestor}(\text{xerces}, v), u_2 = \{x_2 \mapsto \text{xerces}, y_2 \mapsto v\})$ for R_2.
The following CQs are produced:

- With μ_1:

$$\begin{aligned} q' &= u_1(\{\text{parent}(x_1, y_1)\} \cup u_1(\{\text{ancestor}(\text{xerces}, v)\} \setminus \{\text{ancestor}(\text{xerces}, v)\})) \\ &= \exists v\ \text{parent}(\text{xerces}, v) \end{aligned}$$

- With μ_2:

$$\begin{aligned} q'' = u_2(&\{\text{parent}(x_2, z_2), \text{ancestor}(z_2, y_2)\} \\ &\cup u_2(\{\text{ancestor}(\text{xerces}, v)\} \setminus \{\text{ancestor}(\text{xerces}, v)\})) \\ = \exists v, z_2\ &\text{parent}(\text{xerces}, z_2) \wedge \text{ancestor}(z_2, v) \end{aligned}$$

The following example shows one of the issues that arise when we try to use classical unification with existential rules.

Example I.17: Unsound Datalog Rewriting

Consider the existential rule $R = \text{person}(x) \rightarrow \exists y\ \text{hasParent}(x, y)$, the conjunctive query $q = \exists v, w\ \text{hasParent}(v, w) \wedge \text{dentist}(w)$, and the instance $I = \{\text{person}(\text{Maria}), \text{dentist}(\text{Giorgos})\}$.
We find a Datalog unifier $\mu = (\text{hasParent}(v, w), u)$ for q and R, where u is a substitution mapping x to v and y to w.
Following the Datalog rewriting definition, we rewrite q according to μ as follows:

$$q' = u\,(\{person(x)\} \cup (\{\text{hasParent}(v, w), \text{dentist}(w)\} \setminus \{\text{hasParent}(v, w)\}))$$

$$= \{\text{person}(v), \text{dentist}(w)\}, \text{ i.e., } \exists v, w\ \text{person}(v) \wedge \text{dentist}(w)$$

However, while $I \models q'$ (as Maria is a person and Giorgos is a dentist), we do not have $I, R \models q$. This shows that q' is not a sound rewriting of q with respect to R.

In the setting of conjunctive existential rules, query rewriting can be performed using *piece-unifiers* [Baget et al., 2011]. These are a generalisation of classical unifiers that handle existential variables in rule heads by unifying sets of atoms instead of single atoms. To do that, we first need to define some notions about a partition of terms.

A partition P of a set of terms is said to be *admissible* if no class of P contains two constants. We *associate a substitution* u with an admissible partition P_u by selecting one term in each class with priority given to constants: for each class C in P_u, let t_i be the selected term, then for every $t_j \in C$, we set $u(t_j) = t_i$.

Definition I.28 (Separating variables)
Given a Boolean CQ q and a subset q' of q, a variable v in q' is a *separating variable* if it also appears in $q \setminus q'$.

Next, we define piece-unification for a Boolean CQ. We will present later two ways of extending this to a CQ (with a nonempty set of answer variables).

Definition I.29 (Piece-unifier)
Let q be a CQ and R be a conjunctive existential rule. A *piece-unifier* of q with R is a triple $\mu = (q', H', P_u)$ with $q' \neq \emptyset$, $q' \subseteq q$, $H' \subseteq \mathrm{head}(R)$, and P_u is an admissible partition on $\mathtt{terms}(q') \cup \mathtt{terms}(H')$ such that:

1. $u(q') = u(H')$, with u a substitution associated with P_u;
2. If a class $C \in P_u$ contains an existential variable (from H'), then the other terms in C are non-separating variables from q'.

Example I.18: Piece-unifier

Consider the set of rules $\mathcal{R}$ defined as:

$$R_1 : \mathrm{isManager}(x) \wedge \mathrm{isExperienced}(x) \rightarrow \exists y\ \mathrm{canSupervise}(x, y),$$
$$R_2 : \mathrm{isSeniorStaff}(x) \rightarrow \exists y\ \mathrm{canSupervise}(x, y),$$
$$R_3 : \mathrm{isProjectLead}(x) \rightarrow \exists y\ \mathrm{canDirectlySupervise}(x, y),$$
$$R_4 : \mathrm{worksWith}(x, y) \rightarrow \mathrm{worksWith}(y, x),$$
$$R_5 : \mathrm{canDirectlySupervise}(x, y) \rightarrow \mathrm{canSupervise}(x, y),$$

and let q be the following Boolean CQ:

$$q = \exists x_1, x_2, x_3 \mathrm{worksWith}(x_1, x_2) \wedge \mathrm{canSupervise}(x_1, x_3) \wedge \mathrm{canSupervise}(x_2, x_3),$$

which intuitively asks whether there are x_1 and x_2 such that x_1 works with x_2, and both x_1 and x_2 are capable of supervising x_3.
To illustrate the concept of a piece-unifier, we focus on $R_2 = \mathrm{isSeniorStaff}(x) \rightarrow \exists y\ \mathrm{canSupervise}(x, y)$.

Let $q' = \{\text{canSupervise}(x_1, x_3), \text{canSupervise}(x_2, x_3)\}$, $H' = \{\text{canSupervise}(x, y)\}$. Consider the partition P_u of $\texttt{terms}(q') \cup \texttt{terms}(H')$ to be $\{\{x_1, x_2, x\}, \{x_3, y\}\}$. Notice that P_u is admissible, as no class contains two constants. We associate with it the substitution u such that $u(x_1) = u(x_2) = u(x) = x_1$ and $u(x_3) = u(y) = x_3$. Then, we find:

$$\begin{aligned} u(q') &= u(\{\text{canSupervise}(x_1, x_3), \text{canSupervise}(x_2, x_3)\}) \\ &= \{\text{canSupervise}(x_1, x_3)\} \end{aligned}$$

$$\begin{aligned} u(H') &= u(\{\text{canSupervise}(x, y)\}) \\ &= \{\text{canSupervise}(x_1, x_3)\} \end{aligned}$$

Thus, $u(q') = u(H')$. Furthermore, the class in P_u that contains the existential variable y contains only the non-separating variable x_3 since all the atoms that contain x_3 are in q'.
This shows that $\mu = (q', H', P_u)$ is a piece unifier of q with R_2.

Definition I.30 (Application of a Piece-unifier)
Let $\mu = (q', H', P_u)$ be a piece-unifier of a CQ q with a rule $R : B \to H$ and u be a substitution associated with P_u. The *application of μ to q* produces a CQ defined as follows:

$$\beta(q, R, \mu) = u(B) \cup u(q \setminus q'),$$

The CQ $\beta(q, R, \mu)$ is called the *direct* query rewriting of q according to μ and R.

Example I.19: Application of a Piece-unifier

Consider the piece-unifier $\mu = (q', H', P_u)$ of q with $R_2 = \text{isSeniorStaff}(x) \to \exists y\, \text{canSupervise}(x, y)$ from Example I.18. Recall that:

- $q' = \{\text{canSupervise}(x_1, x_3), \text{canSupervise}(x_2, x_3)\}$,
- $H' = \{\text{canSupervise}(x, y)\}$,
- P_u is a partition of $\texttt{terms}(q') \cup \texttt{terms}(H')$ given by $\{\{x_1, x_2, x\}, \{x_3, y\}\}$.

We associated with P_u the substitution u such that $u(x_1) = u(x_2) = u(x) = x_1$ and $u(x_3) = u(y) = x_3$.
The application of the piece-unifier μ to q according to R_2 is given by:

$$\begin{aligned} \beta(q, R_2, \mu) &= u(\{\text{isSeniorStaff}(x)\}) \cup u(q \setminus q') \\ &= \{\text{isSeniorStaff}(x_1), \text{worksWith}(x_1, x_1)\} \end{aligned}$$

Definition I.31 ($\mathcal{R}$-Rewriting Sequence)
Given a UCQ $\mathcal{Q}$ and a set of rules $\mathcal{R}$, an *$\mathcal{R}$-rewriting sequence* from $\mathcal{Q}$ is defined as a finite sequence of UCQs $(\mathcal{Q}_0, \ldots, \mathcal{Q}_k)$ such that:

- $\mathcal{Q}_0 = \mathcal{Q}$,
- for each $1 \leq i \leq k$, there exists a piece-unifier μ_i of a CQ q_{i-1} in $\mathcal{Q}_{i-1}$ with $R_i \in \mathcal{R}$ such that $\mathcal{Q}_i = \mathcal{Q}_{i-1} \cup \{\beta(q_{i-1}, R_i, \mu_i)\}$.

Definition I.32 (Piece-rewriting)
A *piece-rewriting* of a UCQ $\mathcal{Q}$ with a (conjunctive) rule set $\mathcal{R}$ is a UCQ $\mathcal{Q}_k$ obtained from $\mathcal{Q}$ by an $\mathcal{R}$-rewriting sequence $(\mathcal{Q}_0 = \mathcal{Q}), \ldots, \mathcal{Q}_k$ $(k \geq 0)$.

Theorem I.20: Soundness and Completeness of $\mathcal{R}$-Rewriting

Let I be an instance, $\mathcal{R}$ be a set of existential rules, and $\mathcal{Q}$ be a Boolean UCQ. The following conditions are equivalent:

1. $I, \mathcal{R} \models \mathcal{Q}$
2. There exists a piece-rewriting with $\mathcal{R}$ from $\mathcal{Q}$ to $\mathcal{Q}_k$ such that $I \models \mathcal{Q}_k$.

In other words, a knowledge base $(I, \mathcal{R})$ entails a Boolean UCQ $\mathcal{Q}$ if and only if there exists an $\mathcal{R}$-rewriting sequence that transforms $\mathcal{Q}$ into a new query $\mathcal{Q}_k$ that is entailed by the instance I.

We now present two ways to handle answer variables in a piece-unifier: with a special predicate or with a slight modification of the notion of piece-unifier. We detail these two techniques:

- **Special Predicate for Answer Variables:** Let $q(\mathbf{x})$ be a CQ, we transform it into a boolean CQ by adding an atom $Ans(\mathbf{x})$, which contains all the answer variables. And then, we just have to rewrite this Boolean CQ: we have the guarantee that a variable in an Ans-atom will never be unified with an existential variable since it will always be a separating variable when we compute a piece-unification.
- **Modification of the Piece-Unifier:** Given a CQ $q(\mathbf{x})$, we can refine the notion of piece-unifier to ensure that no answer variable is in the same class as an existential variable. Furthermore, if an answer variable $x \in \mathbf{x}$ is unified with another answer variable $y \in \mathbf{x}$ or a constant c, we introduce an equality atom so that the answer variable still appears in the CQ, that is, we add $x = y$ in the first case or $x = c$ in the latter. With this approach, we obtain CQs that may contain equalities (but keep their answer variables, which is important when the set of CQs is seen as a UCQ).

We now focus on producing a set of CQs that is a sound and complete rewriting. Although a sound and complete rewriting may be an infinite set, one of its finite subsets

may still be complete. This is because some queries may be redundant (hence can be eliminated), a concept captured by the notion of query containment.

Definition I.33 (Query Containment)
Given two queries $\mathcal{Q}_1$ and $\mathcal{Q}_2$, we say that $\mathcal{Q}_1$ is *contained* in $\mathcal{Q}_2$, which is denoted by $\mathcal{Q}_1 \sqsubseteq \mathcal{Q}_2$, if $\texttt{certain}(\mathcal{Q}_1, I) \subseteq \texttt{certain}(\mathcal{Q}_2, I)$ for every instance I. We further more say that $\mathcal{Q}_1$ is *strictly contained* in $\mathcal{Q}_2$, which is denoted by $\mathcal{Q}_1 \sqsubset \mathcal{Q}_2$, if $\mathcal{Q}_1 \sqsubseteq \mathcal{Q}_2$ and $\mathcal{Q}_2 \not\sqsubseteq \mathcal{Q}_1$.

In the case of CQs, we can characterise query containment with the notion of query homomorphism.

Definition I.34 (Query Homomorphism)
Given two CQs $q_1(\mathbf{x_1})$ and $q_2(\mathbf{x_2})$, a *query homomorphism* from q_1 to q_2 is a homomorphism h from q_1 to q_2 such that $h(\mathbf{x_1}) = \mathbf{x_2}$.

It is well known that, for two CQs q_1 and q_2, we have $q_1 \sqsubseteq q_2$ if and only if q_2 maps to q_1 by a query homomorphism. Moreover, given two UCQs $\mathcal{Q}_1$ and $\mathcal{Q}_2$, we have $\mathcal{Q}_1 \sqsubseteq \mathcal{Q}_2$, if and only if for all $q_1 \in \mathcal{Q}_1$, there exists $q_2 \in \mathcal{Q}_2$ such that $q_1 \sqsubseteq q_2$; if $\mathcal{Q}_1$ and $\mathcal{Q}_2$ are Boolean UCQs, $\mathcal{Q}_1 \models \mathcal{Q}_2$ if and only if $\mathcal{Q}_1 \sqsubseteq \mathcal{Q}_2$.

We can now define the cover of a set of CQs:

Definition I.35 (Query Cover)
Let $\mathcal{Q}$ be a set of conjunctive queries. A *cover* of $\mathcal{Q}$ is a set $\mathcal{Q}_c \subseteq \mathcal{Q}$ such that:

1. For any q in $\mathcal{Q}$, there exists q' in $\mathcal{Q}_c$ such that $q' \sqsubseteq q$,
2. All elements of $\mathcal{Q}_c$ are pairwise incomparable with respect to $\sqsubseteq$.

Example I.21: Query Containment & Cover

Consider the UCQ

$$\mathcal{Q} = \begin{cases} q_1 = & \exists x_1, y_1, z_1 \; p(x_1, y_1) \wedge q(y_1, z_1), \\ q_2 = & \exists x_2, y_2 \; p(x_2, y_2) \wedge q(y_2, y_2), \\ q_3 = & \exists x_3, y_3, z_3 \; p(x_3, y_3) \wedge q(y_3, z_3) \wedge q(u_3, z_3) \end{cases}$$

.
Query Containment:
We have the following inclusions: $q_1 \sqsubseteq q_3$, $q_3 \sqsubseteq q_1$, $q_1 \sqsubseteq q_2$ and $q_3 \sqsubseteq q_2$.
Query Cover:
A cover of $\mathcal{Q}$ could be $\mathcal{Q}' = \{q_1\}$. This subset is minimal and $\mathcal{Q}' \equiv \mathcal{Q}$ as for each CQ in $\mathcal{Q}'$, there exists a homomorphic image in $\mathcal{Q}$ and vice versa. In particular, $\mathcal{Q}' = \{q_1\}$ seems to be a better choice than $\mathcal{Q}' = \{q_3\}$, as q_1 is simpler than q_3, but both are possible covers of $\mathcal{Q}$.

Using these notions, we can build a breadth-first rewriting algorithm (Algorithm 1), whose primary objective is to generate a sound and complete rewriting of any given UCQ $\mathcal{Q}$ with respect to any given set of rules $\mathcal{R}$.

The algorithm operates by maintaining two sets: the resulting set $\mathcal{Q}_F$, which comprises all generated CQs up to the current point, and the set of CQs $\mathcal{Q}_E$ to be explored.

While there are still CQs to explore (line 3), we compute all the possible piece-unifiers (line 7) with all the rules (line 6) and apply them to produce new CQs that are added to $\mathcal{Q}_t$ (line 8) that contains the CQs produced in a breadth-first step.

Then, a cover of all the CQs produced (comprising the ones produced in a previous step and the ones produced during the step) is computed (line 12). The cover is computed by following a priority scheme that favours queries that have already been explored. In essence, if two CQs are homomorphically equivalent and only one of them has already been explored, the function retains the explored CQ and discards the other. This feature ensures that two equivalent CQs are not explored redundantly, contributing to the termination guarantee of the algorithm.

Then, we select unexplored queries in the cover to explore them in the next step (line 13). The algorithm continues these iterations until $\mathcal{Q}_E$, is empty, at which point the resulting set $\mathcal{Q}_F$ provides a cover of the set of all the rewritings of the initial UCQ $\mathcal{Q}$.

Algorithm 1: Breadth-First Rewriting Algorithm

Input : A set of rules $\mathcal{R}$, a union of conjunctive queries $\mathcal{Q}$
Output: A cover of the set of $\mathcal{R}$-rewritings of $\mathcal{Q}$

```
1  Q_F ← Q ;                                                  // resulting set
2  Q_E ← Q ;                                          // queries to be explored
3  while Q_E ≠ ∅ do
4      Q_t ← ∅ ;              // queries generated at this rewriting step
5      for q_i ∈ Q_E do
6          for R ∈ R do
7              for μ piece-unifier of q_i with R do
8                  Q_t ← Q_t ∪ β(q_i, R, μ)
9              end
10         end
11     end
12     Q_c ← cover(Q_F ∪ Q_t) ;                                // update cover
13     Q_E ← Q_c \ Q_F ;          // select unexplored queries from the cover
14     Q_F ← Q_c
15 end
16 return QF
```

Theorem I.22 states that Algorithm 1 is sound and complete.

Theorem I.22: Soundness & completeness [König, 2014, Thomazo, 2013]

The output of Algorithm 1 is a sound and complete rewriting of $\mathcal{Q}$.

Note that we talk of a sound and complete rewriting of a UCQ, even when there is no such finite rewriting, i.e., no UCQ-rewriting. The following notions allow one to describe a possibly infinite set of CQs that is a sound and complete rewriting.

Definition I.36 (Breadth-first rewriting operator)
The *breadth-first rewriting operator* W_∞, takes as input a UCQ $\mathcal{Q}$ and a rule set $\mathcal{R}$, and returns a possibly infinite set of CQs inductively defined as follows:

- $W_0(\mathcal{Q},\mathcal{R}) = \mathcal{Q}$
- For $i > 0$:

$$W_i(\mathcal{Q},\mathcal{R}) = W_{i-1}(\mathcal{Q},\mathcal{R}) \cup \{\beta(W_{i-1}(\mathcal{Q},\mathcal{R}), R, \mu) \mid \mu \text{ piece-unifier with } R \in \mathcal{R}\}$$

- Finally, $W_\infty(\mathcal{Q},\mathcal{R}) = \bigcup_{i\in\mathbb{N}} W_i(\mathcal{Q},\mathcal{R})$.

Definition I.37 (Rewriting Function)
Let $\mathcal{Q}$ be a UCQ and $\mathcal{R}$ be a set of rules. The *rewriting function*, denoted by `rewriting`$(\mathcal{Q},\mathcal{R})$, is defined as a function that returns $W_k(\mathcal{Q},\mathcal{R})$ with k the smallest integer such that $W_k(\mathcal{Q},\mathcal{R}) \equiv W_{k+1}(\mathcal{Q},\mathcal{R})$, if such an integer exists, otherwise it returns $W_\infty(\mathcal{Q},\mathcal{R})$.

I.7 KBDM systems

One of the objectives of a KBDM system is to provide a high-level access to data from a variety of sources. Unlike in conventional knowledge-based systems, the knowledge base instance (or fact base) is not given, but *specified* by assertions that relate elements of the data sources with facts at the ontological level. Such set of assertions is called a *mapping*. Since mappings are classical objects in database theory (and specially used in data integration and data exchange), we will first define mappings independently from a KBDM system.

I.7.1 Mapping

Broadly speaking, a *mapping* is a (finite) set of *mapping assertions*, which allow one to (1) select elements using a source vocabulary and (2) translate them into elements in a target vocabulary. More specifically, a mapping assertion can be seen as a pair $(Q_S[\mathbf{x}], Q_T[\mathbf{x}])$, where $Q_S[\mathbf{x}]$ is a query over the source vocabulary and $Q_T[\mathbf{x}]$ is a query over the target vocabulary, both queries having the same tuple of answer variables $\mathbf{x}$. The intuitive meaning of such an assertion is that the answers to $Q_S[\mathbf{x}]$ on a source instance yield answers to $Q_T[\mathbf{x}]$ on the target instance.

Most work considers mappings in which assertions are pairs of conjunctive queries: we call them *conjunctive mappings*. A conjunctive mapping assertion can thus be seen as an existential rule such that the body an the head use disjoint sets of predicates. Such rule is also called a *source-to-target* (or simply s-to-t) rule.

Definition I.38 (Conjunctive Mapping)
Let V_S and V_T be two disjoint sets of predicates respectively called *source vocabulary* and *target vocabulary*. A *conjunctive mapping* (or simply *mapping*) is a set $\mathcal{M}$ of existential rules, such that each rule $R \in \mathcal{M}$ is source-to-target, that is, $\texttt{predicates}(\texttt{body}(R)) \subseteq V_S$ and $\texttt{predicates}(\texttt{head}(R)) \subseteq V_T$.

Different classes of conjunctive mappings have been considered in the literature, depending on whether the source or target vocabulary is considered as the reference vocabulary [Lenzerini, 2002]:

- *Global As View mappings (GAV)* consider the mapping assertions as views over the sources: each relevant element of the target vocabulary is associated with a query using the source vocabulary. Formally, a GAV mapping assertion is a Datalog rule, that is, of the form $B[\mathbf{x},\mathbf{y}] \rightarrow H[\mathbf{y}]$, where $H[\mathbf{y}]$ is often restricted to a single atom.
- *Local As View mappings (LAV)*, on the other hand, consider the mapping assertions as views over the target: each relevant element of the source vocabulary is associated with a query using the target vocabulary. A LAV mapping assertion is a linear existential rule, that is, of the form $p[\mathbf{x},\mathbf{y}] \rightarrow \exists z\, H[\mathbf{y},\mathbf{z}]$, where p is a predicate. Note that existential variables in rule heads allow for asserting the existence of target entities that may not be present in the source (aka value invention), which is not the case of GAV mappings.
- *Global-Local As View mappings (GLAV)* introduce more flexibility by generalising both GAV and LAV. Their mapping assertions are general existential rules.

Given a database D over V_S and a mapping $\mathcal{M}$, one difficulty is to define the resulting instance and in relation to this, the answers to a target query (i.e., a query expressed over V_T). Indeed, the target instance is not completely defined by D and $\mathcal{M}$. Actually, the target instance can be thought of as the set of all instances that "satisfy" $\mathcal{M}$ given D. This is formalised by the following notion of *solution*, originally coming from data exchange, where the aim is to actually build an instance produced from D and $\mathcal{M}$ (see, e.g., [Fagin et al., 2005]).

Let D be a (source) database, i.e., a ground instance on a vocabulary V_S, and let $\mathcal{M}$ be a mapping defined from V_S to a target vocabulary V_T. A *solution* for D w.r.t. $\mathcal{M}$ is an atomset I on V_T such that $\mathcal{I}(D \cup I)$ satisfies $\mathcal{M}$ (where $\mathcal{I}(D \cup I)$ is the interpretation isomorphic to $D \cup I$, considering that it is an interpretation of the logical language with set of predicates $V_S \cup V_T$ and set of constants including those occurring in D and $\mathcal{M}$). The *set of solutions* for D w.r.t. $\mathcal{M}$ is thus:

$$\texttt{sol}_{\mathcal{M}}(D) = \{I \mid I \text{ is an atomset on } V_T \text{ and } (D \cup I) \models_1 \mathcal{M}\}$$

Example I.23: Solutions

Let $D = \{q(a),\ r(a),\ s(b)\}$ be a database and $\mathcal{M}$ be a mapping from V_S to V_T such that:

$$\mathcal{M} = \begin{cases} q(x) & \rightarrow \exists z\ p(x,z), \\ r(x) & \rightarrow \exists z\ p(x,z) \wedge t(z) \end{cases}$$

Several instances I over $V_T = \{p(\cdot,\cdot), t(\cdot)\}$ are solutions for D with respect to $\mathcal{M}$, among them:

- $I_1 = \{p(a,z_0),\ t(z_0)\}$;
- $I_2 = \{p(a,a),\ t(a)\}$;
- $I_3 = \{p(a,a),\ t(a),\ p(b,b)\}$;
- $I_4 = \{p(a,z_0),\ p(a,z_1),\ t(z_1)\}$.

We can see that I_1 and $_4$ are homomorphically equivalent and can be mapped to I_2 and I_3.

The *certain answers* to a query $Q_T[\mathbf{x}]$ over a pair $(D,\mathcal{M})$ are then defined as the answers shared by all the solutions, i.e.,

$$\texttt{certain}_{\mathcal{M}}(Q_T, D) = \bigcap_{I \in \texttt{sol}_{\mathcal{M}}(D)} Q_T(I)$$

I.7.2 KBDM

As in the classical OBDA framework, we distinguish the specification of a KBDM system, which only defines the mapping and the ontology, from its instantiation on specific data sources.

Definition I.39 (KBDM specification)
A *KBDM specification* $\Sigma = (V_{\mathcal{O}}, \mathcal{R}_{\mathcal{O}}, V_S, \mathcal{M})$ is composed of:

- a finite set of predicates V_S;
- a finite set of predicates $V_{\mathcal{O}}$ (disjoint from V_S);
- an ontology $\mathcal{R}_{\mathcal{O}}$ composed of a set of existential rules defined over $V_{\mathcal{O}}$;
- a conjunctive mapping $\mathcal{M}$ from V_S to $V_{\mathcal{O}}$.

Note that $V_{\mathcal{O}}$ plays the role of the target vocabulary. By a slight abuse of notation, we will write that a component of Σ is an "element" of Σ, e.g. $V_S \in \Sigma$.

Definition I.40 (KDBM system)
A *KBDM system* $\mathcal{K} = (D, \Sigma)$ is composed of a database instance D on $V_S \in \Sigma$ and a KBDM specification $\Sigma = (V_{\mathcal{O}}, \mathcal{R}_{\mathcal{O}}, V_S, \mathcal{M})$.

The *certain answers* to a query $Q_\mathcal{O}$ (over $V_\mathcal{O}$) over $\mathcal{K}$ are defined as:

$$\texttt{certain}_\Sigma(Q_\mathcal{O}, D) = \bigcap_{I \in \texttt{sol}_\mathcal{M}(D)} \texttt{certain}(Q_\mathcal{O}, (I, \mathcal{R}_\mathcal{O}))$$

We can extend the KBDM framework to consider several data schemas and sources, which are integrated thanks to the ontological vocabulary. This leads to *Integrated KBDM specifications* (IKBDM) $\Sigma_I = (V_\mathcal{O}, \mathcal{R}_\mathcal{O}, \mathcal{S}_I, \mathcal{M}_I)$ where $\mathcal{S}_I = \{V_{\mathcal{S}_1}, \ldots, V_{\mathcal{S}_n}\}$ is a sequence of n data source vocabularies and $\mathcal{M}_I = \{\mathcal{M}_1, \ldots, \mathcal{M}_n\}$ is a sequence of n mappings, each of them defined from $V_{\mathcal{S}_i}$ to $\mathcal{O}$. Then an *IKBDM system* is a pair $\mathcal{K}_I = (D_I, \Sigma_I)$ where $D_I = \{D_1, \ldots, D_n\}$ where each D_i is a database on $V_{\mathcal{S}_i}$.

From an abstract viewpoint, an *IKBDM specification* $\Sigma_I = (V_\mathcal{O}, \mathcal{R}_\mathcal{O}, \{V_{\mathcal{S}_1}, \ldots, V_{\mathcal{S}_n}\}, \{\mathcal{M}_1, \ldots, \mathcal{M}_n\})$ and an *IKBDM system* $\mathcal{K}_I = (\{D_1, \ldots, D_n\}, \Sigma_I)$ can be seen as, respectively, a KBDM specification and a KBDM system. Indeed, we can define $\Sigma = (V_\mathcal{O}, \mathcal{R}_\mathcal{O}, V_{\mathcal{S}_1} \cup \ldots \cup V_{\mathcal{S}_n}, \mathcal{M}_1 \cup \ldots \cup \mathcal{M}_n)$ and $\mathcal{K} = (D_1 \cup \ldots \cup D_n, \Sigma)$ (where the union is defined in the obvious way) since we assume the sets of predicates are disjoint. So, all the properties we define on a KBDM specification / system hold for an IKBDM specification / system. Hence, in most of the dissertation, we will talk about KBDM specifications and systems and we will specifically mention IKBDM systems only when it is relevant to do so.

I.7.3 Querying a KBDM system

Since mappings are existential rules, querying a KBDM system with a UCQ can be performed using the forward or backward chaining techniques we have already introduced for knowledge bases. However, since a mapping describes a translation from a source to a target, we have to slightly adapt the chase and rewriting procedure by adding a post-processing that removes the atoms that are not on the expected vocabulary.

Definition I.41 (Mapping Chase)
Let $I_\mathcal{S}$ be an instance over $V_\mathcal{S}$ and $\mathcal{M}$ be a mapping from $V_\mathcal{S}$ to $V_\mathcal{O}$. The *mapping chase* of $I_\mathcal{S}$ by $\mathcal{M}$, denoted by $\mathcal{M}\text{–}\texttt{chase}(I_\mathcal{S})$, is defined as follows:

$$\mathcal{M}\text{–}\texttt{chase}(I_\mathcal{S}) = \texttt{chase}(I_\mathcal{S}, \mathcal{M}) \setminus I_\mathcal{S}$$

Definition I.42 (Mapping Rewriting Function)
Let $\mathcal{Q}_\mathcal{O}$ be a UCQ over $V_\mathcal{O}$ and $\mathcal{M}$ be a mapping from $V_\mathcal{S}$ to $V_\mathcal{O}$. The *mapping-rewriting function* of $\mathcal{Q}_\mathcal{O}$ by $\mathcal{M}$, denoted by $\mathcal{M}\text{–}\texttt{rewriting}(\mathcal{Q}_\mathcal{O})$, is defined as follows:

$$\mathcal{M}\text{–}\texttt{rewriting}(\mathcal{Q}_\mathcal{O}) = \{q_\mathcal{S} \mid q_\mathcal{S} \in \texttt{rewriting}(\mathcal{Q}_\mathcal{O}, \mathcal{M}), \text{ and } q_\mathcal{S} \text{ is over } V_\mathcal{S}\}$$

Both procedures always terminate since the rules in the mappings are not recursive. Moreover, since the rule bodies and heads are defined on disjoint vocabularies, a unique breath-first step is sufficient to compute the result. By applying the mapping chase to a database D, we obtain an instance on $V_\mathcal{O}$ that has a "universality" property, similar to that of a universal model. It is called a universal solution in database theory.

Definition I.43 (Universal Solution)
Let $\mathcal{K} = (D, \Sigma)$ be a KBDM system with an empty ontology, i.e., $\Sigma = (V_\mathcal{O}, \emptyset, V_\mathcal{S}, \mathcal{M})$. An instance I on $V_\mathcal{O}$ is a *universal solution* of $\mathcal{K}$ if $I \in \mathtt{sol}_\mathcal{M}(D)$ and I maps to each solution in $\mathtt{sol}_\mathcal{M}(D)$.

We can thus compute the certain answers to a UCQ $Q_\mathcal{O}$ over a KBDM system $\mathcal{K} = (D, \Sigma = (V_\mathcal{O}, \emptyset, V_\mathcal{S}, \mathcal{M}))$ with an empty ontology by evaluating $Q_\mathcal{O}$ on $\mathcal{M}\text{–}\mathtt{chase}(D)$:

Proposition I.24

$\mathtt{certain}_\mathcal{M}(Q_\mathcal{O}, D) = Q_\mathcal{O}(\mathcal{M}\text{–}\mathtt{chase}(D))$ for any UCQ $Q_\mathcal{O}$ on $V_\mathcal{O}$.

Similarly, with a query rewriting approach:

Proposition I.25

$\mathtt{certain}_\mathcal{M}(Q_\mathcal{O}, D) = \mathcal{M}\text{–}\mathtt{rewriting}(\mathcal{Q}_\mathcal{O})(D)$ for any UCQ $Q_\mathcal{O}$ on $V_\mathcal{O}$.

Example I.26: Query answering with a mapping

Take again Example I.23: let database $D = \{q(a),\ r(a),\ s(b)\}$ be a database and a $\mathcal{M}$ be a mapping from $V_\mathcal{S}$ to $V_\mathcal{T}$ such that:

$$\mathcal{M} = \begin{cases} q(x) & \rightarrow \exists z\ p(x,z), \\ r(x) & \rightarrow \exists z\ p(x,z) \wedge t(z) \end{cases}$$

Several instances I over $V_\mathcal{T} = \{p(\cdot,\cdot), t(\cdot)\}$ are solutions for D with respect to $\mathcal{M}$, among them:

- $I_1 = \{p(a, z_0),\ t(z_0)\}$;
- $I_2 = \{p(a,a),\ t(a)\}$;
- $I_3 = \{p(a,a),\ t(a),\ p(b,b)\}$;
- $I_4 = \{p(a,z_0),\ p(a,z_1),\ t(z_1)\}$.

Let $\mathcal{Q}_\mathcal{O}(u) = \exists v\ p(u,v)$ be a UCQ.
On I_1, I_2 and I_4, the only answer to $\mathcal{Q}_\mathcal{O}(u)$ is (a). On I_3, there is also (b). By definition, (b) cannot be a certain answer on D w.r.t. $\mathcal{M}$, since it is not an answer on all the solutions.
We have $\mathcal{M}\text{–}\mathtt{chase}(D) = I_4$, which implies that I_4 is a universal solution (note that I_1 is also a universal solution). Then, $\mathcal{Q}_\mathcal{O}(I_4) = \mathtt{certain}_\mathcal{M}(\mathcal{Q}_\mathcal{O}, D) = \{(a)\}$.
And we have $\mathcal{Q}_\mathcal{S}(u) = \mathcal{M}\text{–}\mathtt{rewriting}(\mathcal{Q}_\mathcal{O}) = q(u) \vee r(u)$. Then, $\mathcal{Q}_\mathcal{S}(D) = \mathtt{certain}_\mathcal{M}(\mathcal{Q}_\mathcal{O}, D) = \{(a)\}$.

The previous propositions hold for $Q_{\mathcal{O}}$ a UCQ, and more generally $Q_{\mathcal{O}}$ expressed in a query language closed by homomorphism. They can be extended to KBDM with a non-empty ontology, provided that the chase or query rewriting halts.

Proposition I.27

Let $\mathcal{K} = (D, \Sigma = (V_{\mathcal{O}}, \mathcal{R}_{\mathcal{O}}, V_{\mathcal{S}}, \mathcal{M}))$ be a KBDM system where $\mathcal{R}_{\mathcal{O}}$ is a finite expansion set, and $Q_{\mathcal{O}}$ be a UCQ on $V_{\mathcal{O}}$. Then:

$$\texttt{certain}_{\Sigma}(Q_{\mathcal{O}}, D) = Q_{\mathcal{O}}(\texttt{chase}(\mathcal{M}\texttt{–chase}(D), \mathcal{R}_{\mathcal{O}}))$$

Proposition I.28

Let $\mathcal{K} = (D, \Sigma = (V_{\mathcal{O}}, \mathcal{R}_{\mathcal{O}}, V_{\mathcal{S}}, \mathcal{M}))$ be a KBDM system where $\mathcal{R}_{\mathcal{O}}$ is a finite unification set, and $Q_{\mathcal{O}}$ be a UCQ on $V_{\mathcal{O}}$. Then:

$$\texttt{certain}_{\Sigma}(Q_{\mathcal{O}}, D) = \mathcal{M}\texttt{–rewriting}(\texttt{rewriting}(Q_{\mathcal{O}}, \mathcal{R}_{\mathcal{O}}))(D)$$

Note that, when $\mathcal{R}_{\mathcal{O}}$ is FUS, we can also rewrite $Q_{\mathcal{O}}$ with $\mathcal{R}_{\mathcal{O}}$ and evaluate the resulting query on $\mathcal{M}\texttt{–chase}(D)$.

Finally, we extend the query containment notation to ontological queries on a KBDM specification $\Sigma = (V_{\mathcal{O}}, \mathcal{R}_{\mathcal{O}}, V_{\mathcal{S}}, \mathcal{M})$. Let $\mathcal{Q}_1(\mathbf{x})$ and $\mathcal{Q}_2(\mathbf{x})$ be two queries over $V_{\mathcal{O}}$, we note:

- $\mathcal{Q}_1 \sqsubseteq_{\mathcal{M}} \mathcal{Q}_2$ when $\texttt{certain}_{\mathcal{M}}(Q_1, D) \subseteq \texttt{certain}_{\mathcal{M}}(Q_2, D)$ for each database D on $V_{\mathcal{S}}$;
- $\mathcal{Q}_1 \sqsubseteq_{\Sigma} \mathcal{Q}_2$ when $\texttt{certain}_{\Sigma}(Q_1, D) \subseteq \texttt{certain}_{\Sigma}(Q_2, D)$ for each database D on $V_{\mathcal{S}}$.

The following properties of $\sqsubseteq_{\mathcal{M}}$ and $\sqsubseteq_{\Sigma}$ follow from the definitions and from the soundness and completeness of query rewriting.

Proposition I.29

Let $\Sigma = (V_{\mathcal{O}}, \mathcal{R}_{\mathcal{O}}, V_{\mathcal{S}}, \mathcal{M})$ be a KBDM specification and $\mathcal{Q}_1(\mathbf{x})$ and $\mathcal{Q}_2(\mathbf{x})$ be two UCQs over $V_{\mathcal{O}}$:

- $\mathcal{Q}_1 \sqsubseteq_{\mathcal{M}} \mathcal{Q}_2$ iff $\mathcal{M}\texttt{–rewriting}(\mathcal{Q}_1) \sqsubseteq \mathcal{M}\texttt{–rewriting}(\mathcal{Q}_2)$;
- if $\mathcal{R}_{\mathcal{O}}$ is a finite unification set of rules, $\mathcal{Q}_1 \sqsubseteq_{\Sigma} \mathcal{Q}_2$ iff $\mathcal{M}\texttt{–rewriting}(\texttt{rewriting}(\mathcal{Q}_1, \mathcal{R}_{\mathcal{O}})) \sqsubseteq \mathcal{M}\texttt{–rewriting}(\texttt{rewriting}(\mathcal{Q}_2, \mathcal{R}_{\mathcal{O}}))$.

II - Translation framework

This chapter is devoted to the presentation of our translation framework. In Section II.1, we introduce the different kinds of translations in a KBDM system. Section II.2 defines fundamental notions for the translation of queries. Concerning the translation of constraints, we first discuss the semantics of constraints in a KBDM system in Section II.3, then we define fundamental notions in Section II.4. These notions mirror those defined for query translation up to differences between the semantics of query answering and constraint satisfaction. Then we formalize the notion of "selected data" through a mapping, which has an impact on the existence of a perfect translation (Section II.5). Finally, we illustrate potential uses of our framework on practical scenarios in Section II.6.

Preliminary note. In the literature pertaining to the analysis and transformation of queries, a multitude of terminologies have been used, such as “direct and reverse rewriting" [Lenzerini, 2019], “source and target rewriting" [Arenas et al., 2010, Pérez, 2011], as well as “abstraction" [Cima et al., 2020] and “realization” [Lutz et al., 2018] of a data query in the case of data-to-ontology translation. These terminologies, while descriptive in their contexts, may introduce ambiguities in the scope of this dissertation, particularly with respect to the term “rewriting” referring to backward chaining (e.g., Section I.6 and Chapter III). To delineate the specific operation of changing the vocabulary of a query, which is inherently bidirectional and distinct from the logical transformation involved in rewriting, we adopt the term "translation". The choice of this nomenclature is motivated by two main considerations. First, it avoids potential confusion by distinguishing the vocabulary transformation process from other forms of rewriting. Second, it aptly conveys the notion of changing vocabulary, drawing an appropriate metaphor from linguistic translation, where words are transposed from one language to another while preserving the underlying meaning. Thus, within the context of this work, "translation" will refer to the process of expressing queries or constraints in another vocabulary, while preserving their semantics as well as possible.

II.1 Different kinds of translations

Given a KBDM specification, a translation can be done in each of the two directions, from the source vocabulary to the ontological vocabulary and vice versa, and may consider two levels of the KBDM specification: either the level of the mappings only, or the whole KBDM specification, which includes the (existential) rules as well. For convenience, we also call translation the result of the translation. We consider here the translation of queries and integrity constraints.

Moreover, a translation through a KBDM specification goes from a constraint/query class $\mathbb{C}$ to a constraint/query class $\mathbb{C}'$, which we denote by a $\mathbb{C}$-to-$\mathbb{C}'$-translation. For the sake of clarity, constraint/query classes will be explicitly mentioned only when needed. When talking only of the target class $\mathbb{C}$, we say that it is a $\mathbb{C}$-translation.

Definition II.1 ($\mathcal{S}$-to-$\mathcal{O}$-translation)
An *$\mathcal{S}$-to-$\mathcal{O}$ translation* w.r.t. a KBDM specification $\Sigma = (V_\mathcal{O}, \mathcal{R}_\mathcal{O}, V_\mathcal{S}, \mathcal{M})$ translates constraints/queries from $V_\mathcal{S}$ to $V_\mathcal{O}$.

Definition II.2 ($\mathcal{O}$-to-$\mathcal{S}$-translation)
An *$\mathcal{O}$-to-$\mathcal{S}$ translation* w.r.t. a KBDM specification $\Sigma = (V_\mathcal{O}, \mathcal{R}_\mathcal{O}, V_\mathcal{S}, \mathcal{M})$ translates constraints/queries from $V_\mathcal{O}$ to $V_\mathcal{S}$.

In the following we define properties of a translation with respect to the mapping or to the entire specification. For $\mathcal{S}$-to-$\mathcal{O}$-translations, we talk of an *$\mathcal{M}$-translation* in the first case and a *Σ-translation* in the second case; for $\mathcal{O}$-to-$\mathcal{S}$-translations, we talk of *$\mathcal{M}^{-1}$-translation* and *Σ^{-1}-translation*. These notations are summarized in Table II.1.

Direction	with respect to $\mathcal{M}$	with respect to Σ
$\mathcal{S}$-to-$\mathcal{O}$	$\mathcal{M}$-translation	Σ-translation
$\mathcal{O}$-to-$\mathcal{S}$	$\mathcal{M}^{-1}$-translation	Σ^{-1}-translation

Table II.1: Kinds of translations

In the following, we formally define translations of queries (respectively, of constraints) that fulfil some semantic properties.

II.2 Query translation

In this section, we define a general framework for the translation of queries, which we will apply in particular on UCQs. This framework is largely based on notions introduced in [Cima et al., 2019] and [Arenas et al., 2010, Pérez, 2011]. Translations of UCQs will be building blocks for the translations of constraints.

We first introduce the concepts of *sound* and *complete* translations to characterise the quality of a translation. A *sound translation* is one that does not retrieve more answers than the translated query, while a *complete translation* is one that retrieves all the answers of the translated query. Finally, a *perfect translation* is both sound and complete.

Definition II.3 (Sound / complete translation)
Let $\mathcal{Q}_\mathcal{S}$ and $\mathcal{Q}_\mathcal{O}$ be two queries over vocabularies $V_\mathcal{S}$ and $V_\mathcal{O}$, respectively, $\Sigma = (V_\mathcal{O}, \mathcal{R}_\mathcal{O}, V_\mathcal{S}, \mathcal{M})$ be a KBDM specification, and $X \in \{\mathcal{M}, \Sigma\}$. We say that:

- $\mathcal{Q}_\mathcal{O}$ is a *sound X-translation* of $\mathcal{Q}_\mathcal{S}$ (and $\mathcal{Q}_\mathcal{S}$ *is a complete X^{-1}-translation* of $\mathcal{Q}_\mathcal{O}$) if for every database D, $\texttt{certain}_X(D, \mathcal{Q}_\mathcal{O}) \subseteq \mathcal{Q}_\mathcal{S}(D)$.
- $\mathcal{Q}_\mathcal{O}$ is a *complete X-translation* of $\mathcal{Q}_\mathcal{S}$ (and $\mathcal{Q}_\mathcal{S}$ *is a sound X^{-1}-translation* of $\mathcal{Q}_\mathcal{O}$) if for every database D, $\mathcal{Q}_\mathcal{S}(D) \subseteq \texttt{certain}_X(D, \mathcal{Q}_\mathcal{O})$.
- $\mathcal{Q}_\mathcal{O}$ is a *perfect X-translation* of $\mathcal{Q}_\mathcal{S}$ (and $\mathcal{Q}_\mathcal{S}$ is a *perfect X^{-1}-translation* of $\mathcal{Q}_\mathcal{O}$) if it is both sound and complete.

The next example illustrates this definition. In particular, it shows that a database query may have no perfect $\mathcal{S}$-to-$\mathcal{O}$-translation.

Example II.1: Sound, Complete, and Perfect Translations of Queries

Let $\Sigma = (V_\mathcal{O}, \mathcal{R}_\mathcal{O}, V_\mathcal{S}, \mathcal{M})$ be a KBDM specification where:

$$V_\mathcal{S} = \{s_1, s_2, s_3\}$$
$$V_\mathcal{O} = \{p, t\}$$
$$\mathcal{R}_\mathcal{O} = \{r(x,y) \to \exists z.p(y,z)\},$$
$$\mathcal{M} = \begin{cases} s_1(x,y) & \to p(x,y), \\ s_2(x,x) & \to \exists z.r(x,z), \\ s_3(x,y) & \to p(x,y) \end{cases}$$

$\mathcal{O}$-to-$\mathcal{S}$-translation:

- Query $\mathcal{Q}_\mathcal{O} = \{\exists u, v.p(u,v)\}$ admits:
  - a sound $\mathcal{M}^{-1}$-translation: $\mathcal{Q}^1_{\mathcal{O}\to\mathcal{S}} = \{\exists u, v.s_1(u,v)\}$
  - a perfect $\mathcal{M}^{-1}$-translation: $\mathcal{Q}^2_{\mathcal{O}\to\mathcal{S}} = \{\exists u, v.s_1(u,v), \exists u, v.s_3(u,v)\}$
  - a perfect Σ^{-1}-translation: $\mathcal{Q}^3_{\mathcal{O}\to\mathcal{S}} = \{\exists u, v.s_1(u,v), \exists u, v.s_3(u,v), \exists u.s_2(u,u)\}$

Note that $\mathcal{Q}^2_{\mathcal{O}\to\mathcal{S}}$ is a sound Σ^{-1}-translation but not a complete Σ^{-1}-translation. Indeed, let $D = \{s_2(a,a)\}$: then $I_{(D,\mathcal{M})} = \{\exists z.r(a,z)\}$ and $\text{chase}(I_{(D,\mathcal{M})}, \mathcal{O}) = I_{(D,\mathcal{M})} \cup \{p(a,z)\}$, hence $D, \mathcal{M}, \mathcal{R}_\mathcal{O} \models \mathcal{Q}_\mathcal{O}$ but $D \not\models \mathcal{Q}^2_{\mathcal{O}\to\mathcal{S}}$.

$\mathcal{S}$-to-$\mathcal{O}$-translation:

- Query $\mathcal{Q}^1_\mathcal{S} = \{\exists u, v.s_1(u,v)\}$ admits:
  - a complete $\mathcal{M}$-translation: $\mathcal{Q}^1_{\mathcal{S}\to\mathcal{O}} = \{\exists u, v.p(u,v)\}$ (hence, $\mathcal{Q}^1_{\mathcal{S}\to\mathcal{O}}$ is also a complete Σ-translation); however, $\mathcal{Q}^1_{\mathcal{S}\to\mathcal{O}}$ is not sound. Indeed, let $D = \{s_3(a,b)\}$, then $I_{(D,\mathcal{M})} = \{p(a,b)\}$, hence $I_{(D,\mathcal{M})} \models \mathcal{Q}^1_{\mathcal{S}\to\mathcal{O}}$, whereas $D \not\models \mathcal{Q}^1_\mathcal{S}$. Actually, the only sound $\mathcal{M}$-translation of $\mathcal{Q}^1_\mathcal{S}$ is the empty UCQ, equivalent to $\bot$. This shows that $\mathcal{Q}^1_\mathcal{S}$ has no perfect $\mathcal{M}$-translation.
- Query $\mathcal{Q}^2_\mathcal{S} = \{\exists u, v.s_2(u,v)\}$ admits:
  - a sound $\mathcal{M}$-translation: $\mathcal{Q}^2_{\mathcal{S}\to\mathcal{O}} = \{\exists u, w.r(u,w)\}$, which is however not complete, as can be checked by taking for instance $D = \{s_2(a,b)\}$.

Among sound or complete translations, some translations are more "faithful", or

closer to the meaning of the input query, than others. Intuitively, we prefer complete translations that lead to fewer additional answers and sound translations that preserve more answers. To compare different translations with this respect, we rely on the pre-order on queries defined by query containment. Note that we have to specify the set of instances to be considered for checking query containment: for queries on V_S, we consider all the databases on V_S; in contrast, for queries on $V_\mathcal{O}$ we do not consider all instances on $V_\mathcal{O}$, but only the instances on $V_\mathcal{O}$ that can be produced via $\mathcal{M}$ (or $\mathcal{M}$ and $\mathcal{R}_\mathcal{O}$), i.e, all $I_{(D,\mathcal{M})}$ (or $I_{(D,\Sigma)}$) for D on V_S. Hence, we consider classical query containment ($\sqsubseteq$, see Definition I.33) to compare queries over V_S, containment w.r.t. $\mathcal{M}$ ($\sqsubseteq_\mathcal{M}$) to compare $\mathcal{M}$-translations, and finally containment w.r.t. to Σ ($\sqsubseteq_\Sigma$) to compare Σ-translations (these two last notations are defined in Section I.7.3). In the following, we denote by $\sqsubseteq_X$ one of these three operators, according to the considered translations.

The following notions of minimally complete and maximally sound translations are adapted from [Cima et al., 2019]:

Definition II.4 (Minimally Complete Translation)
Given two queries $\mathcal{Q}_1$ and $\mathcal{Q}_2$, over vocabularies V_1 and V_2, respectively, we say that $\mathcal{Q}_2$ is a *minimally complete translation* of $\mathcal{Q}_1$ if (1) $\mathcal{Q}_2$ is a complete translation of $\mathcal{Q}_1$, and (2) there is no query $\mathcal{Q}$ over V_2 such that $\mathcal{Q} \sqsubset_X \mathcal{Q}_2$ and $\mathcal{Q}$ is a complete translation of $\mathcal{Q}_1$.

Definition II.5 (Maximally Sound Translation)
Given two queries $\mathcal{Q}_1$ and $\mathcal{Q}_2$ over vocabularies V_1 and V_2, respectively, we say that $\mathcal{Q}_2$ is a *maximally sound translation* of $\mathcal{Q}_1$ if (1) $\mathcal{Q}_2$ is a sound translation of $\mathcal{Q}_1$, and (2) there is no query $\mathcal{Q}$ over V_2 such that $\mathcal{Q}_2 \sqsubset_X \mathcal{Q}$ and $\mathcal{Q}$ is a sound translation of $\mathcal{Q}_1$.

The following proposition states relationships between the notions of (minimally) complete, (maximally) sound and perfect translations.

Proposition II.2

Let $\mathcal{Q}_1$ and $\mathcal{Q}_2$ be two queries over two disjoint vocabularies. The following assertions are equivalent:

1. $\mathcal{Q}_1$ is a perfect translation of $\mathcal{Q}_2$.
2. $\mathcal{Q}_2$ is a perfect translation of $\mathcal{Q}_1$.
3. $\mathcal{Q}_1$ is a sound and complete translation of $\mathcal{Q}_2$.
4. $\mathcal{Q}_1$ is a minimally complete and maximally sound translation of $\mathcal{Q}_2$.
5. $\mathcal{Q}_1$ is a minimally complete translation and a sound translation of $\mathcal{Q}_2$.
6. $\mathcal{Q}_1$ is a maximally sound translation and a complete translation of $\mathcal{Q}_2$.

Proof. By definition, if $\mathcal{Q}_1$ is a sound translation of $\mathcal{Q}_2$, $\mathcal{Q}_2$ is a complete translation of $\mathcal{Q}_1$, and conversely. This implies that $\mathcal{Q}_1$ is a perfect translation of $\mathcal{Q}_2$ if and only if $\mathcal{Q}_2$ is a perfect translation of $\mathcal{Q}_1$, so (1) $\Leftrightarrow$ (2).

By definition, a perfect translation is a sound and complete translation, thus (1 – 2) $\Leftrightarrow$ (3). The points (4–6) are all about sound and complete translations, and thus they imply (1 – 3). To show that (1) $\Rightarrow$ (3 – 6), we show that a perfect translation is always minimally complete and maximally sound. Assume $\mathcal{Q}_1$ is perfect but not minimally complete. Then, there exists another query $\mathcal{Q}_1'$ that is also complete such that there exists an instance I with $\mathcal{Q}_1'(I) \subsetneq \mathcal{Q}_1(I)$. This implies that there is an answer to $\mathcal{Q}_1$ that is not an answer to $\mathcal{Q}_2$ since $\mathcal{Q}_1'$ is also complete, thus contradicting the fact that $\mathcal{Q}_1$ is sound. The argument is similar to prove that $\mathcal{Q}_1$ is maximally sound. ■

The following example illustrates the previous notions on the class of UCQs.

Example II.3: Minimally Complete and Maximally Sound Translations

Consider again the KBDM specification $\Sigma = (V_\mathcal{O}, \mathcal{R}_\mathcal{O}, V_\mathcal{S}, \mathcal{M})$ from Example II.1, where $\mathcal{R}_\mathcal{O} = \{r(x,y) \rightarrow \exists z.p(y,z)\}$ and $\mathcal{M} = \{s_1(x,y) \rightarrow p(x,y), s_2(x,x) \rightarrow \exists z.r(x,z), s_3(x,y) \rightarrow p(x,y)\}$. Among the various queries we defined or obtained by translation, the following ones illustrate the notions of minimally complete, maximally sound, and perfect translations:

$\mathcal{O}$-to-$\mathcal{S}$-translation:

- For the query $\mathcal{Q}_\mathcal{O} = \{\exists u, v.p(u,v)\}$:
  - The query $\mathcal{Q}^1_{\mathcal{O}\rightarrow\mathcal{S}} = \{\exists u, v.s_1(u,v)\}$ is not maximally sound as an $\mathcal{M}^{-1}$-translation, as there exists the $\mathcal{M}^{-1}$-translation $\mathcal{Q}^2_{\mathcal{O}\rightarrow\mathcal{S}} = \{\exists u, v.s_1(u,v), \exists u, v.s_3(u,v)\}$, which is a perfect translation, therefore both minimally complete and maximally sound.
  - The query $\mathcal{Q}^3_{\mathcal{O}\rightarrow\mathcal{S}} = \{\exists u, v.s_1(u,v), \exists u, v.s_3(u,v), \exists u.s_2(u,u)\}$, being a perfect Σ^{-1}-translation, is both minimally complete and maximally sound.

$\mathcal{S}$-to-$\mathcal{O}$-translation:

- For the query $\mathcal{Q}^1_\mathcal{S} = \{\exists u, v.s_1(u,v)\}$:
  - The query $\mathcal{Q}^1_{\mathcal{S}\rightarrow\mathcal{O}} = \{\exists u, v.p(u,v)\}$ is a minimally complete (but not sound) Σ-translation.
- For the query $\mathcal{Q}^2_\mathcal{S} = \{\exists u, v.s_2(u,v)\}$:
  - The query $\mathcal{Q}^2_{\mathcal{S}\rightarrow\mathcal{O}} = \{\exists u, w.r(u,w)\}$ is a maximally sound (but not complete) $\mathcal{M}$-translation.

When a query $\mathcal{Q}$ does not have a perfect translation (in the target query class), it becomes interesting to identify a pair of queries $(\mathcal{Q}_{\text{sound}}, \mathcal{Q}_{\text{comp}})$, such that $\mathcal{Q}_{\text{sound}}$ is a maximally sound translation of $\mathcal{Q}$, while $\mathcal{Q}_{\text{comp}}$ is a minimally complete translation of $\mathcal{Q}$.

Finally, it is interesting to note that a sound Σ-translation is also a sound $\mathcal{M}$-translation and a complete $\mathcal{M}$-translation is also a complete Σ-translation, as shown by the next proposition.

Proposition II.4

Let $\Sigma = (V_{\mathcal{O}}, \mathcal{R}_{\mathcal{O}}, V_{\mathcal{S}}, \mathcal{M})$ be a KBDM specification, a query $\mathcal{Q}_{\mathcal{S}}$ over $V_{\mathcal{S}}$ and a query $\mathcal{Q}_{\mathcal{O}}$ over $V_{\mathcal{O}}$. We have the following properties:

1. If $\mathcal{Q}_{\mathcal{O}}$ is a sound Σ-translation of $\mathcal{Q}_{\mathcal{S}}$ then $\mathcal{Q}_{\mathcal{O}}$ is a sound $\mathcal{M}$-translation of $\mathcal{Q}_{\mathcal{S}}$ (or equivalently, if $\mathcal{Q}_{\mathcal{S}}$ is a complete Σ^{-1}-translation of $\mathcal{Q}_{\mathcal{O}}$ then $\mathcal{Q}_{\mathcal{S}}$ is a complete $\mathcal{M}^{-1}$-translation of $\mathcal{Q}_{\mathcal{O}}$);
2. If $\mathcal{Q}_{\mathcal{O}}$ is a complete $\mathcal{M}$-translation of $\mathcal{Q}_{\mathcal{S}}$ then $\mathcal{Q}_{\mathcal{O}}$ is a complete Σ-translation of $\mathcal{Q}_{\mathcal{S}}$ (or equivalently, if $\mathcal{Q}_{\mathcal{S}}$ is a sound $\mathcal{M}^{-1}$-translation of $\mathcal{Q}_{\mathcal{O}}$ then $\mathcal{Q}_{\mathcal{S}}$ is a sound Σ^{-1}-translation of $\mathcal{Q}_{\mathcal{O}}$).

Proof. Without loss of generality, we do the proof for Boolean queries.

1. Assume that for all databases D over $V_{\mathcal{S}}$, we have $D, \mathcal{M}, \mathcal{R}_{\mathcal{O}} \models \mathcal{Q}_{\mathcal{O}}$ implies $D \models \mathcal{Q}_{\mathcal{S}}$. By the monotonicity of first-order logic, we have $D, \mathcal{M} \models \mathcal{Q}_{\mathcal{O}}$ implies $D, \mathcal{M}, \mathcal{R}_{\mathcal{O}} \models \mathcal{Q}_{\mathcal{O}}$. And thus $D, \mathcal{M} \models \mathcal{Q}_{\mathcal{O}}$ implies $D \models \mathcal{Q}_{\mathcal{S}}$.
2. Assume that for all databases D over $V_{\mathcal{S}}$, we have that $D \models \mathcal{Q}_{\mathcal{S}}$ implies $D, \mathcal{M} \models \mathcal{Q}_{\mathcal{O}}$. By the monotonicity of first-order logic, we also have $D, \mathcal{M}, \mathcal{R}_{\mathcal{O}} \models \mathcal{Q}_{\mathcal{O}}$. ■

II.3 Semantics of constraints

In database systems, (integrity) constraints play a pivotal role in ensuring the consistency and reliability of the stored data. They delineate the set of permissible values that a database can take, thus serving to maintain the accuracy and quality of the data by preventing the introduction of incorrect or inconsistent data. Constraints are typically specified through formulas in first-order logic, acting as a formal tool to define the conditions that the data in a database must satisfy. However, as we shall see, determining the appropriate semantics for constraints in a KBDM system poses a significant challenge.

Traditionally in knowledge representation, the semantics adhere to an *open-world assumption* (OWA), a standard that presumes information to be potentially incomplete and allows for the assimilation of data from varied sources [Poggi et al., 2008,Calvanese et al., 2017,Kharlamov et al., 2017].

However, database theory usually applies a *closed-world assumption* (CWA) when it comes to interpreting integrity constraints (as well as queries) which states that any information not explicitly known to be true is false. In other words, if an assertion cannot be found in the database, then its negation is assumed to be true. This paradigm serves to establish whether the data set satisfies specific conditions. Representing these integrity constraints in an ontology language is not completely straightforward due to this divergence of semantics.

Next, we first define the satisfaction of a constraint on a specific instance, then we discuss how to take into account the mapping and the ontological rules to define the satisfaction of a constraint in a KBDM context. Importantly, a constraint in this chapter is simply any (closed) FO-formula. We study in detail in Chapter VI different syntactic forms of constraints.

Definition II.6 (Constraint satisfaction / violation)
A constraint C is said to be *satisfied* by an instance I if $I \models_1 C$ and *violated* otherwise.

One can see that this definition gives a closed-world semantic to the evaluation of constraints since they are checked on a specific model, which is the one isomorphic to the instance. When we consider an instance that is a database, the semantics is exactly the one usually considered in database theory.

Since our aim is to translate constraints through mappings we have to define which target instance must be considered to define the satisfaction/violation of a target constraint given a database D and a mapping $\mathcal{M}$. Indeed, all the solutions in $\mathtt{sol}_{\mathcal{M}}(D)$ do not have the same status with respect to the satisfaction of a constraint, as illustrated in the following example.

Example II.5: Solutions & constraint satisfaction

Take again Example I.23: let $D = \{q(a),\ r(a),\ s(b)\}$ be a database and

$$\mathcal{M} = \begin{cases} q(x) & \rightarrow \exists z\ p(x,z), \\ r(x) & \rightarrow \exists z\ p(x,z) \wedge t(z) \end{cases}$$

. Several instances I on $V_{\mathcal{O}} = \{p(\cdot,\cdot), t(\cdot)\}$ satisfy $\mathcal{M}$, among them:

- $I_1 = \{p(a,z_0),\ t(z_0)\}$;
- $I_2 = \{p(a,a),\ t(a)\}$;
- $I_3 = \{p(a,a),\ t(a),\ p(b,b)\}$;
- $I_4 = \{p(a,z_0),\ p(a,z_1),\ t(z_1)\}$.

Let $C = p(x,y) \rightarrow t(y)$ be a constraint, where universal quantifiers are implicit, like in a rule.
Concerning the satisfaction of C, we have I_1 and I_2 both satisfy C, but neither I_3 nor I_4. Note, moreover, that I_1 and I_4 are semantically equivalent.

In a CWA perspective, it seems relevant to select a solution that adds minimal knowledge, that is an instance $I \in \mathtt{sol}_{\mathcal{M}}(D)$ such as there is no $J \in \mathtt{sol}_{\mathcal{M}}(D)$ with $I \models_1 J$ and $I \not\models_1 J$. So, we have to choose I among the *universal solutions* of $(D, \mathcal{M})$. On the previous example, only I_1 and I_4 have this property.

Again, in a CWA perspective, since we consider the isomorphic interpretation of an instance to check the satisfaction of a constraint, we have to choose, among the universal solutions, the one that is minimal in terms of cardinality. Indeed such a universal solution contains a minimal number of variables, and given the UNA, it introduces a minimal number of individuals in the isomorphic interpretation. On the previous example, I_1 is the instance to select in order to check C; hence, we conclude that $(D, \mathcal{M})$ satisfies C.

In other words, the instance to select to check a constraint on the virtual instance defined from D by a mapping $\mathcal{M}$ is the core of any $\mathcal{M}$−$\mathtt{chase}(D)$: we call it the *target instance*.

Definition II.7 (Target instance $I_{(D,\mathcal{M})}$)
The instance associated with a database D on V_S and a mapping $\mathcal{M}$ from V_S to V_O is the minimal universal solution of $(D, \mathcal{M})$. It is called *target instance* and is denoted by $I_{(D,\mathcal{M})}$.

Definition II.8 (Constraint satisfaction / violation with respect to a mapping)
A constraint C_O defined on V_O is said to be *satisfied* by a KBDM system with empty ontology $\mathcal{K} = (D, \Sigma = (V_O, \emptyset, V_S, \mathcal{M}))$ if $I_{(D,\mathcal{M})} \models_1 C_O$; otherwise it is *violated*.

Now, we should define the satisfaction of a constraint in a KBDM system integrating an ontology $\mathcal{R}_O$. Unfortunately, the choice of the instance on which we should check the constraint is not as obvious as in the previous case.

An immediate option would be to make the same choice: take the minimal universal model of $I_{(D,\mathcal{M})}$ and $\mathcal{R}_O$, which would be $core(\mathtt{chase}(I_{(D,\mathcal{M})}, \mathcal{R}_O))$. But in some cases there is no finite universal model, then there may be several minimal universal models that are not isomorphic or even there may exist no minimal universal model [Carral et al., 2018] (see example below).

Example II.6: Multiple universal models & constraints

Let (D, Σ) be a KBDM system with $D = \{p(a)\}$ and $\Sigma = (V_O, \mathcal{R}_O, V_S, \mathcal{M})$ where:

$$\mathcal{M} = \begin{cases} p(u) & \rightarrow \exists x,\ y.s(x,y), \\ p(v) & \rightarrow \exists z, w.s(z,w) \end{cases}$$

$$\mathcal{R}_O = \begin{cases} r(x,y) & \rightarrow \exists z.s(y,z), \\ s(x,y) & \rightarrow \exists z.r(y,z) \end{cases}$$

Then, the following instances (which are infinite "paths" alternating r- and s-atoms) are associated with two distinct universal models:

- $J_1 = \bigwedge_{n\in\mathbb{N}} s(x_n, y_n) \wedge r(y_n, x_{n+1})$;
- $J_2 = \bigwedge_{n\in\mathbb{N}} r(z_n, w_n) \wedge s(w_n, z_{n+1})$.

Furthermore, J_1 and J_2 are homomorphically equivalent but not isomorphic (in particular, they do not begin with the same atom).
The constraint $C_1 = s(x, y) \rightarrow \exists z.r(z, x)$ checks if there is an r-atom before every s-atom. Thus, $J_1 \not\models_1 C_1$ (since we have $s(x_0, y_0)$ but there is no atom of the form $r(\_, y_0)$), while $J_2 \models_1 C_1$. On the other hand, the constraint $C_2 = r(x, y) \rightarrow \exists z.s(z, x)$ checks if there is a s-atom before every r-atom. Thus, $J_2 \not\models_1 C_2$ (since we have $r(x_0, y_0)$ but there is no atom of the form $s(\_, y_0)$), while $J_1 \models_1 C_2$.
This example shows that we cannot choose a specific model among the minimal ones with our criteria since there is no model that minimises the constraints that are violated.

This example is Example 12 in [Carral et al., 2018] adapted to the KBDM framework.

Note, however, that when $\mathcal{R}_\mathcal{O}$ is a finite expansion set, $core(\mathsf{chase}(I_{(D,\mathcal{M})}, \mathcal{O}))$ exists and is unique for any D, up to a bijective variable renaming.

Another way of getting around this difficulty is to focus only on constraints that behave in the same way on all homomorphically equivalent interpretations. Then, any universal model, minimal or not, is relevant to check the satisfaction of a constraint. This class of constraints, that we call *equivalence-stable constraints*, allows to give a definition of their satisfaction w.r.t. to a general KBDM system.

Definition II.9 (Equivalence-stable)
A constraint C is *equivalence-stable* if for any pair of homomorphically equivalent interpretations I_1 and I_2, $I_1 \models_1 C$ iff $I_2 \models_1 C$.

Any constraint whose satisfaction can be checked by a homomorphism on an interpretation belongs to this class. This is in particular the case of database denial (aka negative) constraints. We study in chapter VI other types of constraint that this class embraces.

We can now define the satisfaction of such a constraint w.r.t. a KBDM system.

Definition II.10 (Constraint satisfaction / violation with respect to a KBDM system)
An equivalence-stable constraint $C_\mathcal{O}$ defined on $V_\mathcal{O}$ is said to be *satisfied* by a KBDM system $\mathcal{K} = (D, \Sigma = (V_\mathcal{O}, \mathcal{R}_\mathcal{O}, V_\mathcal{S}, \mathcal{M}))$ if $\mathsf{chase}(I_{(D,\mathcal{M})}, \mathcal{R}_\mathcal{O}) \models_1 C_\mathcal{O}$ and *violated* otherwise.

Finally, we could also assume that a specific (and not necessarily minimal) universal model of $I_{(D,\mathcal{M})}$ and $\mathcal{R}_\mathcal{O}$ is chosen and that the satisfaction of an ontological constraint

is checked on this model. For instance, we could define satisfaction with respect to a specific deterministic chase (i.e., a chase that ensures that a unique result is obtained for each input, up to a bijective renaming of fresh variables) such that the oblivious, semi-oblivious or Skolem, breadth-first restricted (aka standard) chase, etc. See e.g. [Grahne and Onet, 2018] for a presentation of the main chase variants.

In the general framework presented in this chapter, we simply assume that a universal model of the KB $(I_{(D,\mathcal{M})}, \mathcal{R}_\mathcal{O})$ is chosen, without further specification. By a slight abuse of notation, we denote that atomset by $I_{(D,\Sigma)}$ (even if it is not necessarily an instance, i.e., a finite atomset, as may be suggested by the notation I). We note $I_{(D,\Sigma)} = \mathtt{chase}(I_{(D,\mathcal{M})}, \mathcal{R}_\mathcal{O})$. In Chapter VI, we will consider constraints that are equivalence-stable: then, any universal model can be chosen.

II.4 Constraint translation

As discussed in the previous section, we consider the satisfaction of an ontological constraint against a KBDM system $\mathcal{K} = (D, \Sigma = (V_\mathcal{O}, \mathcal{R}_\mathcal{O}, V_\mathcal{S}, \mathcal{M}))$ either with respect to the mapping, thus by checking the constraint on $I_{(D,\mathcal{M})}$, or with respect to Σ, by checking the constraint on $I_{(D,\Sigma)}$. Next, a *set of constraints* is logically seen as the conjunction of its elements, like it is the case for a set of rules.

Our objective is to provide translations that map a set of constraints $\mathcal{C}_1$ to a set of constraints $\mathcal{C}_2$ that behave similarly on "corresponding" instances. We first define the desired properties of constraint translations; these properties mirror those defined for query translation, modulo the semantic differences between query answering and constraint satisfaction.

Definition II.11 (Preservation of the satisfaction / violation)
Let $\Sigma = (V_\mathcal{O}, \mathcal{R}_\mathcal{O}, V_\mathcal{S}, \mathcal{M})$ be a KBDM specification and let $\mathcal{C}_\mathcal{S}$ and $\mathcal{C}_\mathcal{O}$ be two constraint sets respectively defined on $V_\mathcal{S}$ and $V_\mathcal{O}$. We say that:

- $\mathcal{C}_\mathcal{O}$ *preserves the satisfaction* of $\mathcal{C}_\mathcal{S}$ (or $\mathcal{C}_\mathcal{S}$ *preserves the violation* of $\mathcal{C}_\mathcal{O}$) with respect to Σ if for every database D, when $D \models_1 \mathcal{C}_\mathcal{S}$ then $I_{(D,\Sigma)} \models_1 \mathcal{C}_\mathcal{O}$;
- $\mathcal{C}_\mathcal{O}$ *preserves the violation* of $\mathcal{C}_\mathcal{S}$ (or $\mathcal{C}_\mathcal{S}$ *preserves the satisfaction* of $\mathcal{C}_\mathcal{O}$) with respect to Σ if for every database D, when $I_{(D,\Sigma)} \models_1 \mathcal{C}_\mathcal{O}$ then $D \models_1 \mathcal{C}_\mathcal{S}$;
- $\mathcal{C}_\mathcal{O}$ *perfectly preserves* $\mathcal{C}_\mathcal{S}$ (or $\mathcal{C}_\mathcal{S}$ *perfectly preserves* $\mathcal{C}_\mathcal{O}$) with respect to Σ if $\mathcal{C}_\mathcal{O}$ preserves both the satisfaction and the violation of $\mathcal{C}_\mathcal{S}$, i.e., if for every database D, $D \models_1 \mathcal{C}_\mathcal{S}$ iff $I_{(D,\Sigma)} \models_1 \mathcal{C}_\mathcal{O}$. We also say that $\mathcal{C}_\mathcal{O}$ is a *perfect translation* of $\mathcal{C}_\mathcal{S}$.

In the following, when we write "with respect to $\mathcal{M}$" instead of "with respect to Σ", we mean that we do not consider the set of rules; in other words, we refer to the restriction of $\Sigma = (V_\mathcal{O}, \mathcal{R}_\mathcal{O}, V_\mathcal{S}, \mathcal{M})$ to the KBDM specification $(V_\mathcal{O}, \emptyset, V_\mathcal{S}, \mathcal{M})$.

These notions are illustrated by the next example. This example also shows that, as in the case of queries, perfect translations do not always exist.

Example II.7: Translations of constraints

Let $\Sigma = (V_{\mathcal{O}}, \mathcal{R}_{\mathcal{O}}, V_{\mathcal{S}}, \mathcal{M})$ be a KBDM specification, where:

$$\mathcal{M} = \begin{cases} r(x,y) \rightarrow & p(x,y), \\ q(x) \rightarrow & \exists z.p(x,z), \\ s(x,y) \rightarrow & t(x,y) \end{cases}$$

and $\mathcal{R}_{\mathcal{O}} = \{p(x,y) \rightarrow t(x,y)\}$.
Consider the following (negative) constraints over $V_{\mathcal{S}}$:

- $C^1_{\mathcal{S}} = q(x) \rightarrow \bot$
- $C^2_{\mathcal{S}} = r(x,y) \rightarrow \bot$
- $C^3_{\mathcal{S}} = s(x,y) \rightarrow \bot$

Let $C^1_{\mathcal{O}} = p(x,y) \rightarrow \bot$. This constraint preserves the violation of both $C^1_{\mathcal{S}}$ and $C^2_{\mathcal{S}}$ w.r.t. $\mathcal{M}$. However, it does not preserve their satisfaction. For example, with $D_1 = \{r(a,b)\}$ leading to $I_{(D_1,\mathcal{M})} = \{p(a,b)\}$, we have $D_1 \models_1 C^1_{\mathcal{S}}$ but $I_{(D_1,\mathcal{M})} \not\models_1 C^1_{\mathcal{O}}$. Similarly, for $D_2 = \{q(a)\}$, which yields $I_{(D_2,\mathcal{M})} = \{p(x,z_0)\}$, we have $D_2 \models C^2_{\mathcal{S}}$ but $I_{(D_2,\mathcal{M})} \not\models C^1_{\mathcal{O}}$.
In contrast, $C^1_{\mathcal{O}}$ perfectly preserves $\mathcal{C}_{\mathcal{S}} = \{C^1_{\mathcal{S}}, C^2_{\mathcal{S}}\}$, i.e., $\mathcal{C}_{\mathcal{S}} = C^1_{\mathcal{S}} \wedge C^2_{\mathcal{S}}$, w.r.t. both $\mathcal{M}$ and Σ.
Now, let $C^2_{\mathcal{O}} = t(x,y) \rightarrow \bot$: it perfectly preserves $C^3_{\mathcal{S}}$ w.r.t. to $\mathcal{M}$, but does not perfectly preserve $C^3_{\mathcal{S}}$ w.r.t. Σ. Indeed, consider D_1: then $I_{(D_1,\Sigma)} = \{p(a,b), t(a,b)\}$, and, while $D_1 \models_1 C^3_{\mathcal{S}}$, $I_{(D_1,\Sigma)} \not\models_1 C^2_{\mathcal{O}}$.

We now introduce a preorder on sets of constraints (defined on the same vocabulary) in order to compare them. Just as for query containment, this preorder has to consider different sets of instances depending on the context. For constraints on V_S, these instances are all the databases on V_S, while for constraints on $V_{\mathcal{O}}$, these are all the $I_{(D,\Sigma)}$ for all D on V_S. Next, the set of instances to be considered is called *relevant set of instances*.

Definition II.12 (Stronger/weaker constraints)
Let $\mathcal{I}$ denote the relevant set of instances. Given two constraints sets $\mathcal{C}_1$ and $\mathcal{C}_2$ over the same vocabulary, $\mathcal{C}_1$ is said to be *stronger than* $\mathcal{C}_2$ with respect to $\mathcal{I}$ (or: $\mathcal{C}_2$ is said to be *weaker than* $\mathcal{C}_1$ with respect to $\mathcal{I}$) if every instance $I \in \mathcal{I}$ that satisfies $\mathcal{C}_1$ also satisfies $\mathcal{C}_2$, i.e: for all $I \in \mathcal{I}$, if $I \models_1 \mathcal{C}_1$ then $I \models_1 \mathcal{C}_2$.

Moreover, $\mathcal{C}_1$ is *strictly* stronger than $\mathcal{C}_2$ with respect to $\mathcal{I}$ (or: $\mathcal{C}_2$ is *strictly* weaker than $\mathcal{C}_1$ with respect to $\mathcal{I}$) if $\mathcal{C}_1$ is stronger than $\mathcal{C}_2$ but $\mathcal{C}_2$ is not stronger than $\mathcal{C}_1$.

Example II.8 illustrates this preorder on constraint sets. As extreme cases, an always-satisfied constraint set (i.e., semantically equivalent to $\top$) is weaker than any

other set, and an always-violated constraint set (i.e., semantically equivalent to $\bot$) is stronger than any other set.

Example II.8

Take again the KBDM specification and the constraints over V_S of Example II.7:

- $C_S^1 = q(x) \rightarrow \bot$
- $C_S^2 = r(x,y) \rightarrow \bot$
- $C_S^3 = s(x,y) \rightarrow \bot$

We saw that $C_O^1 = p(x,y) \rightarrow \bot$ preserves the violation of C_S^1 and C_S^2 w.r.t. $\mathcal{M}$. There are other constraints that preserve the violation of C w.r.t. $\mathcal{M}$ like $C_O^\bot = \bot$ or $C_O^3 = (t(x,y) \rightarrow \bot) \wedge (p(x,y) \rightarrow \bot)$. But one can intuitively see that C_O^1 is the best one, as it is the weakest constraint that preserves the violation. $C_O^\bot$ is useless as it does not give any information about the translated constraint: it is a translation that preserves the violation of any constraint. C_O^3 is less "accurate" than C_O^1 as it is more frequently violated.
We also saw that $C_O^2 = t(x,y) \rightarrow \bot$ perfectly preserves C_S^3 w.r.t. $\mathcal{M}$ and so it preserves satisfaction. There are other constraints that preserve the satisfaction of C_S^3 w.r.t. $\mathcal{M}$, such as $C_O^\top = \top$ or $C_O^4 = p(x,y) \wedge t(x,y) \rightarrow \bot$. C_O^2 is intuitively the best one preserving satisfaction as it is the strongest that preserves satisfaction. $C_O^\top$ is useless as it preserves the satisfaction of any translated constraint, and C_O^4 is less "accurate" than C_O^2 as it is more often satisfied.

Similarly to the notions of maximally sound / minimally complete translations for queries, we now define sets of constraints that maximally preserve satisfaction, which are the strongest ones that preserve satisfaction, and sets of constraints that minimally preserve violation, which are the weakest ones that preserve violation.

Definition II.13 (Maximally preserve satisfaction)
Given two constraint sets $\mathcal{C}_1$ and $\mathcal{C}_2$, over vocabularies V_1 and V_2, respectively, we say that $\mathcal{C}_2$ *maximally preserves satisfaction* of $\mathcal{C}_1$ if (1) $\mathcal{C}_2$ preserves the satisfaction of $\mathcal{C}_1$, and (2) there is no constraint set $\mathcal{C}$ over V_2 such that $\mathcal{C}$ is strictly stronger than $\mathcal{C}_2$ and $\mathcal{C}$ preserves the satisfaction of $\mathcal{C}_1$.

Definition II.14 (Minimally preserve violation)
Given two constraint sets $\mathcal{C}_1$ and $\mathcal{C}_2$, over vocabularies V_1 and V_2, respectively, we say that $\mathcal{C}_2$ *minimally preserves violation* of $\mathcal{C}_1$ if (1) $\mathcal{C}_2$ preserves the violation of $\mathcal{C}_1$, and (2) there is no constraint set $\mathcal{C}$ over V_2 such that $\mathcal{C}$ is strictly weaker than $\mathcal{C}_2$ and $\mathcal{C}$ preserves the violation of $\mathcal{C}_1$.

The relationships between the previous preservation notions can be stated in the same way as for queries (Proposition II.2):

Proposition II.9

Let us consider two sets of constraints $\mathcal{C}_1$ and $\mathcal{C}_2$ over disjoint vocabularies. The following assertions are equivalent:

1. $\mathcal{C}_1$ perfectly preserves $\mathcal{C}_2$.
2. $\mathcal{C}_2$ perfectly preserves $\mathcal{C}_1$.
3. $\mathcal{C}_1$ preserves the satisfaction and the violation of $\mathcal{C}_2$.
4. $\mathcal{C}_1$ minimally preserves the violation and maximally preserves the satisfaction of $\mathcal{C}_2$.
5. $\mathcal{C}_1$ minimally preserves the violation and preserves the satisfaction of $\mathcal{C}_2$.
6. $\mathcal{C}_1$ preserves the violation and maximally preserves the satisfaction of $\mathcal{C}_2$.

Proof. The proof is similar to that of Proposition II.2 for queries: a (maximally) sound or (minimally) complete query is replaced by a set of constraints that (maximally) preserves satisfaction or (minimally) preserves violation, respectively. ■

II.5 Impact of selected information

An interesting question is the following: why does a perfect translation of a given query or constraint not always exist? Obviously, it may already come from the fact that the mapping selects only part of the data, since a query or constraint may behave differently on a restricted part of the data and on the whole data.

To describe that, we first define the information from a database D that is selected by a mapping $\mathcal{M}$, which we call the "selected part" of D. Note that when the body of a mapping rule is mapped to D, only the database values that are images of frontier variables are transferred by the mapping. As we will see, the selected part of D is an instance that yields the same target instance as D.

Definition II.15 (Selected part of the database)
Given a KBDM system $(D, \Sigma = (V_\mathcal{O}, \mathcal{R}_\mathcal{O}, V_\mathcal{S}, \mathcal{M}))$, the *selected part of the database* D, denoted by $\texttt{selected}(D, \mathcal{M})$ is defined as follows:

$$\texttt{selected}(D, \mathcal{M}) = \{h'(\text{body}(m)) \mid \forall m \in \mathcal{M}, \forall h \text{ homomorphism from } \text{body}(m) \text{ to } D\}$$

where h' is defined as follows: if $x \in \texttt{fr}(m)$ then $h'(x) = h(x)$ else $h'(x)$ is a fresh variable.

The following example illustrates the consequences of selecting data on the existence of a perfect translation of a constraint.

Example II.10

Let $\mathcal{K} = (D = \{p(a,b),\ r(b,c),\ s(a,b)\},\ \Sigma = (V_\mathcal{O}, \mathcal{R}_\mathcal{O}, V_\mathcal{S}, \mathcal{M}))$ where be a KBDM system:

$$\mathcal{M} = \begin{cases} p(x,y) & \to q(x), \\ r(x,y) & \to t(y,x) \end{cases}$$

Then $\texttt{selected}(D, \mathcal{M}) = \{p(a, y_0),\ r(b,c)\}$.
Let us define the constraints over $V_\mathcal{S}$ as:

- $C^1_\mathcal{S} = p(x,y) \wedge r(y,z) \to \bot$;
- $C^2_\mathcal{S} = p(x,y) \wedge s(x,y) \to \bot$.

These two constraints do not have a perfect $\mathcal{M}$-translation. In both cases, it is due to some information that is not translated into the ontological level:

- We have $D \not\models_1 C^1_\mathcal{S}$ but $\texttt{selected}(D, \mathcal{M}) \models_1 C^1_\mathcal{S}$ - we can see with the selected part of the database that the information that $p(a,b)$ has in its second position the same constant as in first position of $r(b,c)$ is not transferred by the mapping;
- Similarly, for $C^2_\mathcal{S}$, we have $D \not\models_1 C^2_\mathcal{S}$ and $\texttt{selected}(D, \mathcal{M}) \models_1 C^2_\mathcal{S}$, but the situation is even worse: none of the content in s-atoms is transferred to the ontological level, thus $s(-,-)$ can even not be translated.

Propositions II.11 and II.12 help us to better understand the properties of the selected part of the database.

Proposition II.11

Given a KBDM specification $\Sigma = (V_\mathcal{O}, \mathcal{R}_\mathcal{O}, V_\mathcal{S}, \mathcal{M})$, for every database D, $\texttt{selected}(D, \mathcal{M})$ maps to D.

Proof. It is easy to see that for each $h'(\texttt{body}(m)) \subseteq \texttt{selected}(D, \mathcal{M})$ (defined in the definition of $\texttt{selected}(D, \mathcal{M})$), we can create a substitution h'' such that $h''(\texttt{body}(m)) \subseteq D$:

- $h''(x) = h'(x) = h(x)$ for each $x \in \texttt{fr}(m)$;
- $h''(h'(y)) = h(y)$ for each $y \in \texttt{vars}(\texttt{body}(m)) \setminus \texttt{fr}(m)$.

By aggregating all these h'', we obtain a homomorphism from $\texttt{selected}(D, \mathcal{M})$ to D. ■

Proposition II.12

Given a KBDM specification $\Sigma = (V_{\mathcal{O}}, \mathcal{R}_{\mathcal{O}}, V_{\mathcal{S}}, \mathcal{M})$, $I_{(D,\mathcal{M})}$ and $I_{(\mathtt{selected}(D,\mathcal{M}),\mathcal{M})}$[1] are isomorphic for every database D.

Proof. It is easy to see that for each $h'(\mathtt{body}(m)) \subseteq \mathtt{selected}(D,\mathcal{M})$ (defined in the definition of $\mathtt{selected}(D,\mathcal{M})$), the application of the triggers[2] (m,h) and (m,h') produce equivalent outputs: $h^+(\mathtt{head}(m))$ and $h'^+(\mathtt{head}(m))$ are isomorphic since existential variables are bijectively renamed. Since it is true for all triggers on D and $\mathtt{selected}(D,\mathcal{M})$, we have $I_{(D,\mathcal{M})}$ and $I_{(\mathtt{selected}(D,\mathcal{M}),\mathcal{M})}$ are isomorphic. ■

The following example illustrates Proposition II.12.

Example II.13

Let $\mathcal{K}_1 = (D_1, \Sigma)$ and $\mathcal{K}_2 = (D_2, \Sigma)$ be two KBDM systems where $\Sigma = (V_{\mathcal{O}}, \mathcal{R}_{\mathcal{O}}, V_{\mathcal{S}}, \mathcal{M})$, $D_1 = \{p(a,b)\}$, $D_2 = \{r(a,b)\}$ and

$$\mathcal{M} = \begin{cases} p(x,y) & \rightarrow t(x,y), \\ r(x,y) & \rightarrow t(x,y) \end{cases}$$

We have:

- $\mathtt{selected}(D_1,\mathcal{M}) = \{p(a,b)\}$;
- $\mathtt{selected}(D_2,\mathcal{M}) = \{r(a,b)\}$;
- $I_{(D_1,\mathcal{M})} = I_{(\mathtt{selected}(D_1,\mathcal{M}),\mathcal{M})} = I_{(\mathtt{selected}(D_2,\mathcal{M}),\mathcal{M})} = I_{(D_2,\mathcal{M})} = \{t(a,b)\}$.

From the notion of selected part of a database, we can define the notion of a query / constraint "stable by selection" (Definition II.16), which, obviously, is only relevant for $\mathcal{S}$-to-$\mathcal{O}$-translation. The main idea behind this notion is that the queries or constraints should be over the part of the database that the user selects. In other words, the selected part of a database has to behave like the whole database with respect to these objects. As we shall see in Proposition II.14, the stability by selection is a necessary condition to the existence of a perfect $\mathcal{S}$-to-$\mathcal{O}$-translation. Note that this is independent from any specific input or target query/constraint class.

Definition II.16 (Stability by selection)

1. A query $\mathcal{Q}_{\mathcal{S}}$ over $V_{\mathcal{S}}$ is *stable by selection* (w.r.t. $\mathcal{M}$) if for all databases D over $V_{\mathcal{S}}$, $\mathcal{Q}_{\mathcal{S}}(D) = \mathcal{Q}_{\mathcal{S}}(\mathtt{selected}(D,\mathcal{M}))$.

[1] Note that $\mathtt{selected}(D,\mathcal{M})$ is not a ground instance. However, we can still define what a universal model is, so we make a slight abuse of notation.

[2] See Definition I.17

2. A constraint or set of constraints $\mathcal{C}_\mathcal{S}$ over $V_\mathcal{S}$ is *stable by selection* (w.r.t. $\mathcal{M}$) if for all databases D over $V_\mathcal{S}$, $D \models_1 \mathcal{C}_\mathcal{S}$ if and only if $\texttt{selected}(D, \mathcal{M}) \models_1 C_\mathcal{S}$.

Proposition II.14

Let $\mathcal{M}$ be a mapping from $V_\mathcal{S}$ to $V_\mathcal{O}$ and $\mathcal{Q}_\mathcal{S}$ be a query (respectively $\mathcal{C}_\mathcal{S}$ be a constraint set) over $V_\mathcal{S}$. If $\mathcal{Q}_\mathcal{S}$ (resp $\mathcal{C}_\mathcal{S}$) is not stable by selection w.r.t. $\mathcal{M}$ then it does not admit a perfect $\mathcal{S}$-to-$\mathcal{O}$-translation.

Proof. If $\mathcal{Q}_\mathcal{S}$ is not stable by selection, then, there exists a database D such that $\mathcal{Q}_\mathcal{S}(D) \neq \mathcal{Q}_\mathcal{S}(\texttt{selected}(D, \mathcal{M}))$. By definition of a perfect $\mathcal{M}$-translation $\mathcal{Q}_\mathcal{O}$ of $\mathcal{Q}_\mathcal{S}$, we would have $\mathcal{Q}_\mathcal{S}(D) = \mathcal{Q}_\mathcal{O}(I_{(D,\mathcal{M})})$ and $\mathcal{Q}_\mathcal{S}(\texttt{selected}(D, \mathcal{M})) = \mathcal{Q}_\mathcal{O}(I_{(\texttt{selected}(D,\mathcal{M}),\mathcal{M})})$. But we proved in Proposition II.12 that $I_{(\texttt{selected}(D,\mathcal{M}),\mathcal{M})}$ and $I_{(D,\mathcal{M})}$ are isomorphic. So we cannot have a perfect $\mathcal{M}$-translation, this would lead to a contradiction. The proof is similar for constraints. ■

Next, we give an operational characterisation of a Boolean UCQ stable by selection.[3] This can easily be extended to non-Boolean UCQs.

In the next proposition, we abuse notation and unify a query with a conjunction of rule bodies. Indeed, this conjunction of rule bodies can be seen as a rule head, in the sense that we distinguish between frontier variables and the other variables, considered as existential variables, which allows us to use the notion of piece-unifier.

Proposition II.15: Stability by selection of CQs

A *Boolean* CQ q is stable by selection if and only if there exists a piece-unifier μ such that μ unifies the whole q with some conjunction of bodies of rules in $\mathcal{M}$.

Proof. Note that for all databases D, if $\texttt{selected}(D, \mathcal{M}) \models q$ then $D \models q$ (because there is a homomorphism from $\texttt{selected}(D, \mathcal{M})$ to D). For the other direction of the proof, is implied by Lemma II.16. ■

Lemma II.16

A *Boolean* CQ q is such that, for all databases D, $D \models q$ implies $\texttt{selected}(D, \mathcal{M}) \models q$ if and only if there exists a piece unifier μ such that μ unifies the whole of q with a conjunction of bodies of rules in $\mathcal{M}$.

Proof. For both directions of the proof, we introduce a special mapping that allows another way to define the selected part of a database. Let us define a mapping built with bodies of $\mathcal{M}$: $\mathcal{M}_S = \{B[\mathbf{x}, \mathbf{y}] \rightarrow \exists z.\hat{B}[\mathbf{x}, \mathbf{z}] \mid B[\mathbf{x}, \mathbf{y}] \rightarrow H \in \mathcal{M}\}$. The hat means that

[3] As this is a side result, we prefer to include it here rather than in the chapter devoted to UCQ translation.

we rename the predicates in B with a hat on top of their name (that is, a predicate p is renamed into a predicate $\hat{p}$). The idea is that $I_{(D,\mathcal{M}_S)}$ is just $\texttt{selected}(D,\mathcal{M})$ whose predicates were renamed: $I_{(D,\mathcal{M}_S)} \equiv \widehat{\texttt{selected}(D,\mathcal{M})}$. Then, we introduce a renaming of the CQ with hats on top of the predicates, which we denote $\hat{q}$. We have $I_{(D,\mathcal{M}_S)} \models \hat{q}$ if and only if $\texttt{selected}(D,\mathcal{M}) \models q$ if and only if $D,\mathcal{M}_S \models \hat{q}$. Moreover, since the heads of $\mathcal{M}_S$ are just bodies of $\mathcal{M}$ with renamed predicates, we see that there exists a unifier μ between bodies of $\mathcal{M}$ and q that covers q if and only if we have a unifier $\hat{\mu}$ (that is, μ but with renamed predicates) between heads of $\mathcal{M}_S$ and $\hat{q}$ that covers $\hat{q}$, which is equivalent to $\mathcal{M}_S\texttt{−rewriting}(\hat{q}) \neq \emptyset$.

($\Rightarrow$) "if $D \models q$ then $\texttt{selected}(D,\mathcal{M}) \models q$" implies "if $D \models q$ then $I_{(D,\mathcal{M}_S)} \models \hat{q}$" which implies "if $D \models q$ then $D,\mathcal{M}_S \models \hat{q}$" that implies "if $D \models q$ then $D \models \mathcal{M}_S\texttt{−rewriting}(\hat{q})$" which implies "if $D \models q$ then $\mathcal{M}_S\texttt{−rewriting}(\hat{q}) \neq \emptyset$" and thus "if $D \models q$ then there exists a unifier μ between bodies of $\mathcal{M}$ and q that covers q".

($\Leftarrow$) We have that "there exists a piece unifier μ such that μ unifies the whole of q with a conjunction of bodies of rules in $\mathcal{M}$" implies "$\mathcal{M}_S\texttt{−rewriting}(\hat{q}) \neq \emptyset$". Note that $D \models q$ implies $D \models \mathcal{M}_S\texttt{−rewriting}(\hat{q})$ because there is $q' \in \mathcal{M}_S\texttt{−rewriting}(\hat{q})$ that maps to q (rewriting with $\mathcal{M}_S$ consists of removing the hats and replacing some variables by fresh variables). This is equivalent to "$D \models q$ implies $D,\mathcal{M}_S \models \hat{q}$" which is finally equivalent to "$D \models q$ implies $\texttt{selected}(D,\mathcal{M}) \models q$". ■

Proposition II.15 can be generalised to Boolean UCQs.

Proposition II.17: Stability by selection of UCQs

A Boolean UCQ $\mathcal{Q}$ is stable by selection if and only if each CQ $q \in \mathcal{Q}$ is stable by selection.

Proof. ($\Rightarrow$) We take the contrapositive: we assume that a CQ $q \in \mathcal{Q}$ is not stable by selection and show that it implies that $\mathcal{Q}$ is not stable by selection. Without loss of generality, we assume that there is no CQ in $\mathcal{Q}$ that is more general than q. Let D_q be the database that is q where the variables are replaced by fresh constants. Then, $D_q \models q$ and thus $D_q \models \mathcal{Q}$. Since q is not stable by selection, $\texttt{selected}(D_q,\mathcal{M}) \not\models q$. And since there are no more general CQs than q, we have $D_q \not\models q'$ and thus $\texttt{selected}(D_q,\mathcal{M}) \not\models q'$ (since $\texttt{selected}(D_q,\mathcal{M})$ maps to D_q) for all $q' \neq q$ in $\mathcal{Q}$. Therefore $\texttt{selected}(D_q,\mathcal{M}) \not\models \mathcal{Q}$: $\mathcal{Q}$ is not stable by selection.

($\Leftarrow$) For each database D such that $D \models \mathcal{Q}$, we have $q \in \mathcal{Q}$ such that $D \models q$. Since q is stable by selection, $\texttt{selected}(D,\mathcal{M}) \models q$ and thus $\texttt{selected}(D,\mathcal{M}) \models \mathcal{Q}$. On the contrary, for each database D such that $D \not\models \mathcal{Q}$, we do not have $q \in \mathcal{Q}$ such that $D \models q$. Therefore, since each of them is stable by selection, there are no CQs $q \in \mathcal{Q}$ such that $\texttt{selected}(D,\mathcal{M}) \models q$. This implies $\texttt{selected}(D,\mathcal{M}) \not\models \mathcal{Q}$. ■

II.6 Practical uses of constraint translation

The practical interest of query translation has already been well covered in the literature, being from the ontology level to the data level (which is the priviledged way of answering ontological queries in OBDA), or vice-versa (see in particular [Cima et al., 2023]). In contrast, constraint translation has been much less considered. That is why in this section we provide some practical scenarios that illustrate the interest of translating constraints in a KBDM context. These scenarios also illustrate how our translation framework can be applied.

One could imagine many motivations for translating constraints, whatever the direction. Among them:

1. This can facilitate the user's understanding of data.
2. This can help improving data quality.
3. This can allow one to detect mismatches between different data sources.
4. This can be used to optimize ontological query rewriting.

Next, we illustrate these different cases.

II.6.1 Facilitate user's understanding of data

It is often the case that data are provided with integrity constraints. Then, different kinds of translation may help the user to get a better understanding of data.

A translation of the source constraints that minimally preserves violation helps the user to understand when the source constraints are satisfied. Since this is equivalent to saying that the source constraints maximally preserve satisfaction of this translation, when the translation is satisfied, we are sure that the source constraints are also satisfied.

Example II.18: Scenario 1

In the scenario, a user has an ontology about plants and a data source and wants to build a mapping. Given a mapping under construction, translating the constraints from the source allows to check if these constraints are consistent with the semantics of its ontology.
Let $\Sigma = (V_{\mathcal{O}}, \mathcal{R}_{\mathcal{O}}, V_{\mathcal{S}}, \mathcal{M})$ be a KBDM specification about agronomy:

- $V_{\mathcal{O}} = \{Solanaceae, Cucurbitaceae\}$ (all predicates are of arity 1)
- $V_{\mathcal{S}}$ contains the following predicates of arity 1:
  - $S{:}Solanum\_lycopersicum$[a]
  - $S{:}Cucurbita\_pepo$[b]
  - $S{:}Cucurbita\_maxima$[c]

- The user wants to test a mapping $\mathcal{M}$ that contains:
  - $S{:}Solanum\_lycopersicum(x) \rightarrow Solanaceae(x)$
  - $S{:}Cucurbita\_pepo(x) \rightarrow Cucurbitaceae(x)$
  - $S{:}Cucurbita\_maxima(x) \rightarrow Cucurbitaceae(x)$

There is a set of constraints $\mathcal{C}_S$ over V_S that contains two denial constraints:

- $S{:}Solanum\_lycopersicum(x) \wedge S{:}Cucurbita\_pepo(x) \rightarrow \bot$
- $S{:}Solanum\_lycopersicum(x) \wedge S{:}Cucurbita\_maxima(x) \rightarrow \bot$

An $\mathcal{M}$-translation of $\mathcal{C}_S$ that minimally preserves the violation is $C_\mathcal{O} = Solanaceae(x) \wedge Cucurbitaceae(x) \rightarrow \bot$. It means that it follows from the data constraints and the mapping that plants cannot be in both families. This is consistent with scientific knowledge, hence should agree with the ontology. The translated constraint might be added to the ontology.

[a] *Solanum lycopersicum* is the scientific name of the plant that produces tomatoes.
[b] *Cucurbita pepo* includes a variety of squashes and pumpkins. Zucchini, acorn squash, and spaghetti squash are some of the plants under this species.
[c] *Cucurbita maxima* is another species of squash, which includes types like the Hubbard and buttercup squashes.

If we have a translation of source constraints that preserves satisfaction and consider only legal databases w.r.t. the source constraints, then we have the guarantee that the translation will always be satisfied at the ontological level. This gives us information about consistent databases.

Example II.19: Scenario 2

Take again the KBDM specification from Example II.18. We add the following to the specification:

- the unary predicate $S{:}heavy\_feeder$
- the unary predicate $heavy\_feeder$ to $V_\mathcal{O}$;
- the rule $S{:}heavy\_feeder(x) \rightarrow heavy\_feeder(x)$ to $\mathcal{M}$;

We consider the following set of constraints $\mathcal{C}_S^+$:

- $S{:}Solanum\_lycopersicum(x) \rightarrow S{:}heavy\_feeder(x)$;
- $S{:}Cucurbita\_pepo(x) \rightarrow S{:}heavy\_feeder(x)$;
- $S{:}Cucurbita\_maxima(x) \rightarrow S{:}heavy\_feeder(x)$.

An $\mathcal{M}$-translation that maximally preserves the satisfaction of $\mathcal{C}_S^+$ is $\mathcal{C}_O^+ = \{Solanaceae(x) \rightarrow heavy\_feeder(x), Cucurbitaceae(x) \rightarrow heavy\_feeder(x)\}$. Then we know that the properties described by $\mathcal{C}_O^+$ are always true at the ontological level. We could use them as rules in the ontology without modifying the answers to the queries.

A user of the KBDM system may also want to check hypotheses about data. A way to do that is to allow the user to write constraints using the ontological vocabulary and then translate them at the data level.

Example II.20: Scenario 3

Take again again the KBDM specification of the example II.18. We add the following:

- the unary predicate $S{:}Edible\_fruit$ to V_S;
- the constraint $C_S^+ = S{:}Solanum\_lycopersicum(x) \rightarrow S{:}Edible\_fruit(x)$;
- the unary predicate $Edible\_fruit$ to V_O;
- the rule $S{:}Edible\_fruit(x) \rightarrow Edible\_fruit(x)$ to $\mathcal{M}$;

The user may think that all plants in the Cucurbitaceae family produce edible fruits. To test this hypothesis, he can write the constraint $C_O^{H1} = Cucurbitaceae(x) \rightarrow Edible\_fruit(x)$. A perfect $\mathcal{M}^{-1}$-translation of C_O^{H1} is the set of constraints $\mathcal{C}_S^{H1} = \{S{:}Cucurbita\_maxima(x) \rightarrow S{:}Edible\_fruit(x), S{:}Cucurbita\_maxima(x) \rightarrow S{:}Edible\_fruit(x)\}$. Since we cannot prove $\mathcal{C}_S \models \mathcal{C}_S^{H1}$, we cannot conclude that the hypothesis is always correct. We can check if the current database satisfies $\mathcal{C}_S^{H1}$, but, if it does, there is no guarantee that the hypothesis will remain true if the data changes. Assume now that the user hypothesises that the plants in the Solanacee family represented in the system all produce edible fruits, writing the constraint $C_O^{H2} = Solanaceae(x) \rightarrow Edible\_fruit(x))$. Here, we can conclude that the hypothesis is always correct, because a perfect $\mathcal{M}^{-1}$-translation of C_O^{H2} is C_S^+.

II.6.2 Improve data quality / data cleaning assistance

Users of a KBDM system may want the data to respect specific properties. So, a user may write a constraint using the ontological vocabulary to assert such property: indeed, it is much easier for a user to write constraints using his own vocabulary. However, if the ontological constraint is found to be violated, the user wants to understand what the cause of the problem is. One can then translate the constraint to the source level: we obtain a set of constraints and each constraint can be cheked, which allows to identify

the trouble in the data.

Example II.21: Scenario 4

We take the example of an Integrated KBDM system containing two databases that use distinct schemas: the first one on music titles in compact discs, the second one is about music on an online platform. Let $\Sigma_I = (\mathcal{O}, \{V_{\mathcal{S}_1}, V_{\mathcal{S}_2}\}, \{\mathcal{M}_1, \mathcal{M}_2\})$ be an IKBDM specification with:

- $\mathcal{O} = (V_\mathcal{O}, \mathcal{C}_\mathcal{O})$ with $V_\mathcal{O} = \{Title, Author, Has\_author\}$ and $\mathcal{C}_\mathcal{O} = \{Title(x) \rightarrow \exists z\, Author(z) \wedge Has\_author(x,z)\}$;
- $\mathcal{S}_1 = (V_{\mathcal{S}_1})$ with $V_{\mathcal{S}_1} = \{S_1 : Compact\_Disc, S_1 : Title, S_1 : Author, S_1 : Compact\_Disc\_Title, S_1 : Title\_Author\}$ with the first three predicates of arity 1 and the two others of arity 2;
- $\mathcal{S}_2 = (V_{\mathcal{S}_2})$ with $V_{\mathcal{S}_2} = \{S_2 : Title\_name, S_2 : Author\_name, S_2 : Title\_Author\}$ with the first two predicates of arity 1 and the last one of arity 2;
- $\mathcal{M}_1$ contains:
  - $S_1 : Title(x) \rightarrow Title(x)$
  - $S_1 : Author(x) \rightarrow Author(x)$
  - $S_1 : Title\_Author(x,y) \rightarrow Has\_author(x,y)$
- $\mathcal{M}_2$ contains:
  - $S_2 : Title\_name(x) \rightarrow Title(x)$
  - $S_2 : Author\_name(x) \rightarrow Author(x)$
  - $S_2 : Title\_Author(x,y) \rightarrow Has\_author(x,y)$

Let $\mathcal{K}_I = (\{D_1, D_2\}, \Sigma_I)$ be an IKBDM System with:

- D_1 contains:
  - $S_1 : Compact\_Disc = \{Single\ A\}$
  - $S_1 : Title = \{Song\ 1\}$
  - $S_1 : Author = \{John\ Smith\}$
  - $S_1 : Compact\_Disc\_Title = \{(Album\ A, Song\ 1)\}$
  - $S_1 : Title\_Author = \{\}$
- D_2 contains:
  - $S_2 : Title\_name = \{Melody\ C\}$

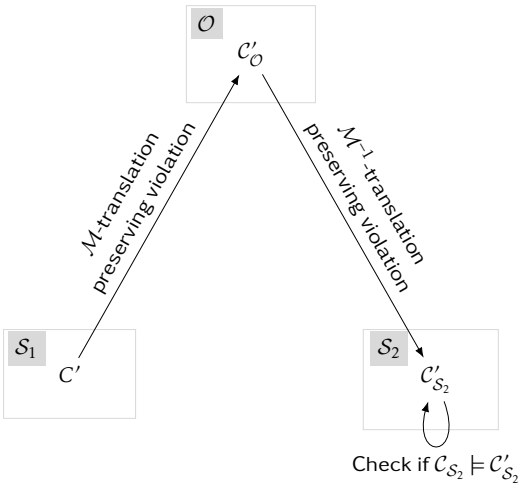


Figure II.1: Detect potential mismatches between data sources

- $S_2 : Author\_name = \{Emily\ Johnson\}$
- $S_2 : Title\_Author = \{(Melody\ C, Emily\ Johnson)\}$

If we evaluate $\mathcal{C}_{\mathcal{O}}$ on $\mathcal{K}_I$, we notice that the constraint is violated, but without knowing why. To know why, we can compute the perfect $\mathcal{M}_1^{-1}$-translation and perfect $\mathcal{M}_2^{-1}$-translation of this constraint (which exist). One of the constraints in the perfect $\mathcal{M}_1^{-1}$-translation is $C^+_{\mathcal{S}_1} = S_1 : Title(x) \rightarrow \exists z\ S_1 : Author(z) \wedge S_1 : Title\_Author(x,z)$, which is violated when evaluated on D_1 (Table $S_1 : Title\_Author$ does not give the author of the title "Song 1") and so we know that D_1 leads to the violation of $\mathcal{C}_{\mathcal{O}}$. This information can be given to the user to help him understand the issue and correct it.

II.6.3 Detect mismatches between data sources

In an IKBDM system, we integrate several sources of data. This can lead to some conflicting properties between the sources, which is either inherent to their conception or caused by the mapping or the ontology.

To check if there is such an issue, we can take a constraint C' from a source schema $\mathcal{S}_1$, translate it at the ontological level, and then translate it back at the schema level to a source schema $\mathcal{S}_2$ to check the compatibility of these schemas. The Figure II.1 summarizes this process.

Example II.22: Scenario 5

Take again the example II.21. We add binary predicates $S_1 : Genre$, $S_2 : Genre$, and $Genre$, respectively, to $V_{\mathcal{S}_1}$, $V_{\mathcal{S}_2}$, and $V_{\mathcal{O}}$ to link the title of a song to its genre. We add the corresponding rules $S_1 : Genre(x,y) \rightarrow Genre(x,y)$ and $S_2 : Genre(x,y) \rightarrow Genre(x,y)$ to respectively $\mathcal{M}_1$ and $\mathcal{M}_2$.
We add the constraint $C^+_{\mathcal{S}_1} = S_1 : Genre(x,y) \wedge S_1 : Genre(x,z) \rightarrow y = z$ to $\mathcal{C}_{\mathcal{S}_1}$, which asserts that a title has a single genre. This is not what is assumed in $\mathcal{S}_2$. We can do an $\mathcal{M}_1^{-1}$-translation, then an $\mathcal{M}_2$-translation that both preserve violation to obtain the constraint $C^+_{\mathcal{S}_2} = S_2 : Genre(x,y) \wedge S_2 : Genre(x,z) \rightarrow y = z$. $C^+_{\mathcal{S}_2}$ cannot be proved from the set of constraints $\mathcal{C}_{\mathcal{S}_2}$. This allows one to detect that the two schemas do not make the same hypotheses about the world.

Identifying mismatches between different sources leads to different possibilities of correction. The first one is to modify the mapping and/or the ontology to take these into account. The second one is, as seen previously, to use the translated constraints to help the user to correct the sources, if feasible (in particular the user needs to have the right to update the data).

II.6.4 Optimize query rewriting

Let us now consider the optimisation of query rewriting with an ontology in the presence of ontological constraints. There is some work on this subject for DL-Lite ontologies [Rosati, 2012, Console et al., 2013]. The main idea is that if some axioms in the ontology (the TBox) are always satisfied by an instance (ABox) that satisfies the ontological constraints (called here extensional constraints, EBox), these axioms can be ignored when rewriting queries.

Example II.23: Query Rewriting with Extensional Constraints

This example comes from [Rosati, 2012].
Let the rule set $\mathcal{R} = \{R_1 = \text{Student}(x) \rightarrow \text{Person}(x)\}$ and the constraint $C_1 = \text{Student}(x) \rightarrow \text{Person}(x)$.
The CQ $q(x) = Person(x)$ has a rewriting with respect to $\mathcal{R}$ which is a disjunction of two CQs:

$$q_1(x) = \text{Person}(x)$$
$$q_2(x) = \text{Student}(x)$$

If C_1 is taken into account, the rewriting can be restricted to $q(x)$ itself (i.e., q_1), since all the individuals of class $Student$ are also of class $Person$ for all the instances that satisfy C_1. Therefore, one can ignore the rule R_1 when rewriting queries.

In [Console et al., 2013], this technique is extended to OBDA by considering also constraints in data sources. They translate data constraints into the vocabulary of the ontology to use them in order to optimise query rewriting of ontological queries.

II.7 Summary

In this chapter, we presented a general framework for translating queries and constraints in a KBDM system (D, Σ) with $\Sigma = (V_{\mathcal{O}}, \mathcal{R}_{\mathcal{O}}, V_{\mathcal{S}}, \mathcal{M})$ (Section II.1). More precisely, we presented:

- **A framework for query translation:** We first define a framework to describe query translation and its desired properties (Section II.2). This framework is mainly based on the notions introduced by Cima et al. extended to our setting. In particular, we take from that work the fundamental notions of perfect (Definition II.3), minimally complete (Definition II.4) and maximally sound (Definition II.5) translations, while distinguishing between translations with respect to the mapping only or to the whole specification.
- **A framework for constraint translation:** The main elements of this framework are the following:
  - **Constraint satisfaction at the ontological level:** Constraint satisfaction w.r.t. a mapping $\mathcal{M}$ is checked against the minimal universal solution of $(D, \mathcal{M})$, called the target instance and denoted by $I_{(D,\mathcal{M})}$. Constraint satisfaction w.r.t. a whole specification Σ is checked against a chosen universal model of $(I_{(D,\mathcal{M})}, \mathcal{R}_{\mathcal{O}})$ (Section II.3).
  - **Equivalence-Stable Constraints:** The property of equivalence-stability ensures that a constraint behaves similarly on all the universal models of $(I_{(D,\mathcal{M})}, \mathcal{R}_{\mathcal{O}})$ (Definition II.9). For instance, negative constraints have this property.
  - **Desired properties of translations:** We consider translations of constraints from the data source to the ontology, and reciprocally. Since a *perfect* translation may not exist, we define the properties of *(maximally) preserving satisfaction* and *(minimally) preserving violation* (Definitions II.11, II.13 and II.14).

This chapter also contains:

- A discussion on the impact of the selection performed by the mapping: We formalised the notion of selected part of a database (Definition II.15) and defined the notion of a query / constraint stable by selection (Definition II.16). We also gave an operational characterization of this notion in the specific case of UCQs.
- An illustration of the interest of constraint translation on concrete use-cases (Section II.6).

Looking Ahead: Our next objective is to apply this general framework to the translation of specific classes of queries and constraints. We will build on some existing tools in this endeavour, however the introduction of novel techniques appeared to be necessary. That is why we will first introduce a technique to rewrite (U)CQs with disjunctive existential rules in Chapter III.

III - Disjunctive rewriting

Looking for techniques to translate constraints from source to ontology, we encountered the need to rewrite a UCQ into another UCQ through disjunctive *rules, and more precisely, through a disjunctive mapping (see Chapter V, and more precisely Section V.2.3 on the maximally sound $\mathcal{M}$-translation of a UCQ). It appeared that there was little work on this subject (Section III.1). This motivated the development of a novel technique to perform disjunctive rewriting (Section III.3). Before presenting this technique, we need to introduce the disjunctive chase (Section III.2), since in our technique a rewriting step is closely related to a disjunctive chase step. Then, we explore the possibility of defining classes of disjunctive rules that are FUS (Section III.4). Finally, we study the specific case of disjunctive mappings (Section III.5).*

Most results presented in this chapter have been published in [Leclère et al., 2023]. A prototype implementing our disjunctive rewriting technique is available as the library Pie[1].

III.1 Introduction

In a nutshell, disjunctive existential rules are an extension of conjunctive existential rules in which the head of a rule is a disjunction of conjunctive heads. Each conjunctive head has its own existential variables and its frontier variables form a subset of the rule's frontier variables.

Definition III.1 (Disjunctive existential rule, Disjunctive mapping)
A *disjunctive existential rule* (or simply disjunctive rule) is a rule of the form:

$$\forall \mathbf{x} \forall \mathbf{y} (B[\mathbf{x}, \mathbf{y}] \rightarrow \bigvee_{i=1}^{n} \exists \mathbf{z_i} \, H_i[\mathbf{x_i}, \mathbf{z_i}]) \;\; \texttt{with} \;\; \mathbf{x_i} \subseteq \mathbf{x}$$

where $n \geq 1$, $B = \text{body}(R)$ (the body) and $H_i = \text{head}_i(R)$ (the conjunctive heads) are non-empty conjunctions of atoms with $\texttt{vars}(B) = \mathbf{x} \cup \mathbf{y}$ and $\texttt{vars}(H_i) = \mathbf{x_i} \cup \mathbf{z_i}$; furthermore $\mathbf{x} = \bigcup_{i=1}^{n} \mathbf{x_i}$.

A disjunctive existential rule is a *disjunctive s-to-o rule* if the vocabulary of the body is disjoint from the vocabulary of the heads. A *disjunctive mapping* is a set of a disjunctive s-to-o rules (considered as a mapping).

Example III.1

The following rule says that any entity x who is grandparent of an entity y is

[1] https://bitbucket.org/guillaume-perution-kihli/pie

parent of a mother of y or parent of a father of y:

$$\forall x \forall y (\text{isGrandParent}(x,y) \rightarrow \exists z_1\ (\text{isParent}(x,z_1) \wedge \text{isMother}(z_1,y)) \\ \vee \exists z_2\ (\text{isParent}(x,z_2) \wedge \text{isFather}(z_2,y)))$$

So far, reasoning with disjunctive existential rules has been mainly studied through the *chase*. It was shown that decidable classes of (conjunctive) existential rules, based on the behaviour of the chase, can be generalised to disjunctive rules in a quite natural way, whether in relation to acyclicity notions [Carral et al., 2017] or based on a property called guardedness [Alviano et al., 2012, Gottlob et al., 2012, Bourhis et al., 2016], although these generalisations come with a huge increase in the complexity of query answering.

In contrast, *query rewriting* within UCQs has barely been addressed yet. A notable exception is the work in [Alfonso et al., 2021], which provides a rewriting technique based on first-order resolution. We will give details on this technique later in this section. Furthermore, a large body of work has studied the rewritability of ontology-mediated queries, i.e., pairs of the form $(Q, \mathcal{O})$ with Q a (U)CQ and $\mathcal{O}$ an ontology, into query languages of various expressivity. However, for ontologies expressed in fragments of disjunctive existential rules, most studies target expressive rewriting languages, such as disjunctive Datalog [Bienvenu et al., 2014, Ahmetaj et al., 2018]. As far as we are aware, the only result directly relevant to our purpose comes from the fine-grained complexity study in [Gerasimova et al., 2020], which provides syntactic rewritability conditions for ontology-mediated queries where the ontology is composed of a single specific disjunctive rule, called a covering axiom (see Section III.4 for more details).

As we have just mentioned, [Alfonso et al., 2021] is, to the best of our knowledge, the only previous work that proposes a UCQ rewriting technique for general disjunctive existential rules. This technique is based on a restricted form of first-order resolution, where at each step a CQ is unified with a disjunct of a rule head (using a conjunctive piece-unifier, see Definition I.29), which produces a new disjunctive rule with fewer disjunctions; when the unified rule is conjunctive (the negation of) a CQ is produced.

Example III.2

Let $R = p(x,y) \rightarrow \exists z_1\ r(x,z_1) \vee \exists z_2\ r(y,z_2)$ be a disjunctive rule and $\mathcal{Q} = \{q\}$ be a UCQ with $q = \exists u, v\ s(u) \wedge r(u,v)$.
We have a piece-unifier $\mu_1 = (r(u,v), r(x,z_1), \{\{u,x\},\{v,z_1\}\})$ between the first disjunct of $\text{head}(R)$ and q.
A step of rewriting with the disjunctive rule R consists in building a new disjunctive rule by removing the unified disjunct from R and adding to its body the

remaining atoms of the CQ (those that were not unified) and finally applying the unifier. Here, we obtain the new rule $R' = p(x,y) \wedge s(x) \rightarrow \exists z_2\ r(y,z_2)$.
Then, we can use R' to rewrite again q. Since R' is a conjunctive rule, we obtain (the negation of) a CQ. We have the piece-unifier $\mu_1 = (r(u,v), r(y,z_2), \{\{u,y\},\{v,z_2\}\})$, which leads to the CQ $q' = \exists x,y\ p(x,y) \wedge s(x) \wedge s(y)$.

One of the disadvantages of this technique of rewriting is that it produces intermediate rules, whereas we are only interested in producing CQs. Clearly, some intermediate rules may be useless to produce new CQs; furthermore, it is more difficult to study the properties of a changing set of rules.

Example III.3

Take a variant of Example III.2: let $R = p(x,y) \rightarrow \exists z_1\ r(x,z_1) \vee \exists z_2\ t(y,z_2)$ be a disjunctive rule and $\mathcal{Q} = \{q\}$ be a UCQ with $q = \exists u,v\ s(u) \wedge r(u,v)$.
We have the same piece-unifier with the first disjunct as in Example III.2. We produce the new rule $R' = p(x,y) \wedge s(x) \rightarrow \exists z_2\ t(y,z_2)$. But there is no unification of q with the head of R', so this rule is useless.

In comparison, the main advantages of our proposal, presented in Section III.3, are the following:

1. A rewriting step directly produces a CQ and not a rule.
2. Intermediate rules, which may not lead to a CQ, are avoided.
3. There is a direct correspondence between a chase step and a rewriting step, which makes it easier to study the properties of query rewriting, especially as the rule set is not updated.

III.2 Disjunctive chase

We now introduce the disjunctive chase, which is a generalisation of the chase for conjunctive rules (Section I.5). The following presentation of this technique is inspired by the work of [Bourhis et al., 2016, Carral et al., 2017].

A trigger for disjunctive rules is defined similarly to conjunctive rules (Definition I.16).

Definition III.2 (Disjunctive Trigger)
Let I be an instance and R be a disjunctive rule. A *disjunctive trigger* on I is a pair (R,h), where h is a homomorphism from body(R) that maps R to I. When such a trigger exists, the rule R is said to be *applicable* on I.

Example III.4: Disjunctive Trigger

Consider the disjunctive rule $R = vertex(x) \rightarrow green(x) \vee red(x)$ (which says that each vertex can be green or red) and the instance $I = \{edge(a,b), vertex(a), vertex(b)\}$ that represents a graph with two vertices a and b connected by and edge. There are two disjunctive triggers for R on I:

- (R, h_a), where $h_a = \{x \mapsto a\}$,
- (R, h_b), where $h_b = \{x \mapsto b\}$.

This indicates that the disjunctive rule R can be applied to each of the vertices a, and b separately.

The result of an application of a Disjunctive Trigger (Definition III.4) is not a single instance, but a set of instances, seen as a disjunction of instances.

Definition III.3 (Set of Instances)
A *set of instances*, denoted $\mathcal{I}$, is viewed as a disjunction of instances $\mathcal{I} = \bigvee_{i=1}^{n} I_i$, where each I_i is a conjunction of atoms.

Definition III.4 (Application of a Disjunctive Trigger)
Given a disjunctive rule R and a disjunctive trigger $t = (R, h)$ on an instance I, the *application* of t (or: of R according to h) to I produces a set of instances

$$\alpha_\vee(I, R, h) = \{I \cup h^{+_i}(\mathsf{head}_i(R)) \mid 1 \leq i \leq n\}$$

$\alpha_\vee(I, R, h)$ is called an *immediate derivation* from I. The trigger t is said *satisfied* by I if there exists an extension h' of h with $h'(\mathsf{head}_i(R)) \subseteq I$ for some i.

Example III.5: Application of a Disjunctive Trigger

Continuing with the same rule $R = vertex(x) \rightarrow green(x) \vee red(x)$ and triggers from example III.4, let us consider the application of $(R, h_a = \{x \mapsto a\})$ on the instance $I = \{edge(a,b), vertex(a), vertex(b)\}$. We obtain a set of two instances defined as follows:

$$\alpha_\vee(I, R, h_a) = \{\{edge(a,b), vertex(a), vertex(b), green(a)\}, \{edge(a,b), vertex(a), vertex(b), red(a)\}\}$$

The trigger (R, h_a) is satisfied by every instance $I' \in \alpha_\vee(I, R, h_a)$, because there exists an extension h' of h_a with $h'(\mathsf{head}_i(R)) \subseteq I'$ for some i.

As for the chase on conjunctive rules, the disjunctive chase procedure iteratively applies triggers towards a fixpoint. This procedure is often seen as the construction of

a tree, see, in particular, [Bourhis et al., 2016, Carral et al., 2017]. Next, we formally define the derivation tree built by a disjunctive chase (Definition III.6), where each node represents an instance, the root represents the initial instance, and the children of a nodes are generated by application of a disjunctive trigger. To define it, we need first to introduce the notion of branch of a tree.

Definition III.5 (Branch of a Tree)
Let $\mathcal{T} = (V, E)$ be a tree where V is the set of vertices and E the set of edges. Let r_T be the root of $\mathcal{T}$. A *branch* of $\mathcal{T}$ is a subset of nodes $\gamma \subseteq V$ such that:

1. The root node r_T is in γ.
2. For any node $v \in \gamma$, if v is not the root node, then the parent of v is also in γ.
3. Each node in γ either:
   - is a leaf node of $\mathcal{T}$ (that is, it has no child nodes in V), or
   - has exactly one child node in γ.

The set of all branches of a tree $\mathcal{T}$ is denoted by $\Gamma(\mathcal{T})$.

This definition captures the intuitive idea of a branch as a maximal path through the tree: a path that cannot be extended by including more nodes. In particular, each node on the path has a unique successor, except for the final node, which is a leaf and has no successors. Note that a branch can also be infinite (in this case, there is no leaf node in the branch).

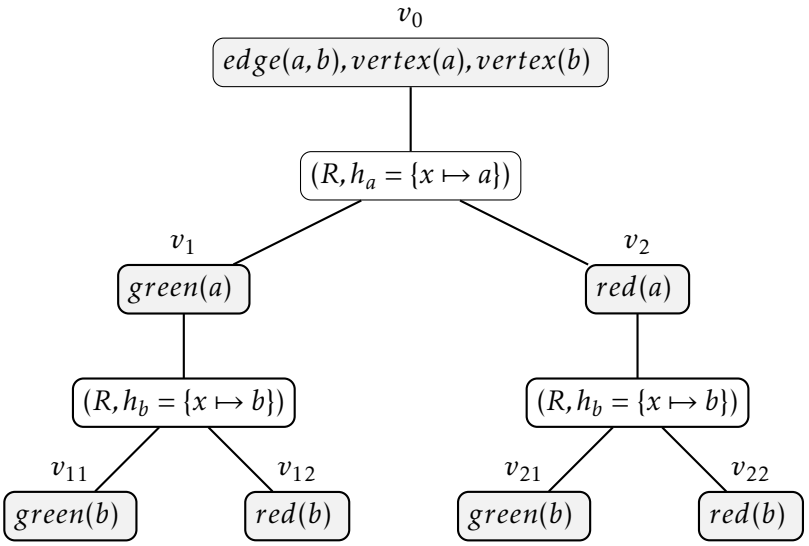
Definition III.6 (Derivation Tree)
A *derivation tree* $\mathcal{T}$ for a KB $(I, \mathcal{R})$ is a labelled rooted tree (V, E, λ), where V is the set of vertices, E is the set of edges, and λ is a function that labels each vertex $v \in V$ by an instance. The tree is defined inductively as follows:

- For the root r of $\mathcal{T}$, $\lambda(r) = I$.
- For each vertex v with children $v_1, ..., v_n$, there is a disjunctive trigger (R, h) on $\lambda(v)$ with $R = B[\mathbf{x}, \mathbf{y}] \rightarrow H_1[\mathbf{x}, \mathbf{z_1}] \vee \ldots \vee H_n[\mathbf{x}, \mathbf{z_n}] \in \mathcal{R}$. The restriction of λ to the domain $\{v_1, ..., v_n\}$ forms a bijection with $\alpha_\vee(\lambda(v), R, h)$.

A derivation tree (V, E, λ) is *fair* if, for each branch γ and each vertex $v \in nodes(\gamma)$, any disjunctive trigger on $\lambda(v)$ is satisfied in $\lambda(v')$ with $v' \in nodes(\gamma)$. Finally, a *chase tree* is a fair derivation tree.

Example III.6: Derivation Tree

Given a set of rules $\mathcal{R}$ composed only of $R = vertex(x) \rightarrow green(x) \vee red(x)$, and a singleton set of instances $\mathcal{I} = \{\{edge(a, b), vertex(a), vertex(b)\}\}$, let us build a



Note: to save space, we only put the new atoms in the nodes and not the whole instances.

Figure III.1: Graphical view of the derivation tree of the example III.6

derivation tree.
At the root of the tree, we have a node v_0 labelled with the initial instance $\{edge(a,b), vertex(a), vertex(b)\}$.
We saw in Example III.4 that we have a disjunctive trigger $(R, h_a = \{x \mapsto a\})$ on the instance $\lambda(v_0)$. Applying the rule through this trigger, we obtain two instances that form the labels for the children nodes of the root: v_1 is labelled with $\{edge(a,b), vertex(a), vertex(b), green(a)\}$ and v_2 is labelled with $\{edge(a,b), vertex(a), vertex(b), red(a)\}$.
Now, we identify another disjunctive trigger using R and a homomorphism h_b that assigns x to b. Applying the rule to the instance at v_1, we get two child nodes of v_1: v_{11} labelled with $\{edge(a,b), vertex(a), vertex(b), green(a), green(b)\}$ and v_{12} labelled with $\{edge(a,b), vertex(a), vertex(b), green(a), red(b)\}$. Similarly, applying the rule to the instance at v_2 yields two more child nodes: v_{21} labelled with $\{edge(a,b), vertex(a), vertex(b), red(a), green(b)\}$ and v_{22} labelled with $\{edge(a,b), vertex(a), vertex(b), red(a), red(b)\}$.
The obtained derivation tree is shown in Figure III.1. The tree itself has the grey nodes: we added the triggers for clarity.
In this example, intuitively, we generated all the possible colorations of the graph represented in the initial instance.

Definition III.7 (Disjunctive chase result)
The *result of a disjunctive chase* of an instance I by a set of rules $\mathcal{R}$, denoted $\mathsf{chase}_\vee(I,\mathcal{R})$, is defined as the set of all atomsets that can be obtained by the union of the labels of the nodes in every branch of a chase tree $\mathcal{T}$. Formally,

$$\mathsf{chase}_\vee(I,\mathcal{R}) = \{\bigcup_{v\in\gamma} \lambda(v) \mid \gamma \in \Gamma(\mathcal{T})\}$$

where $\mathcal{T}$ is a fair chase tree and λ is its labelling function.

Although each node in the tree has a finite degree (bounded by the maximal number of disjuncts in a rule head), the tree may contain an infinite number of branches, and a branch can also be infinite. Consequently, the result of the chase may contain an infinite number of atomsets, each potentially comprising an infinite number of atoms. From a logical perspective, the result of a (finite) disjunctive chase is a disjunction of existentially closed conjunctions of atoms.

Although the chase tree and the chase result might not be unique, they all entail the same queries.

Example III.7: Disjunctive chase result

Consider a set of rules $\mathcal{R} = \{R\}$ with $R = vertex(x) \rightarrow green(x) \vee red(x)$ and an initial instance $I = \{edge(a,b), vertex(a), vertex(b)\}$ from Example III.6.
The disjunctive chase result, $\mathsf{chase}_\vee(I,\mathcal{R})$, for this particular set of rules and initial instance is as follows:

$$\begin{aligned}\mathsf{chase}_\vee(I,\mathcal{R}) = \{&\{edge(a,b), vertex(a), vertex(b), green(a), green(b)\},\\ &\{edge(a,b), vertex(a), vertex(b), green(a), red(b)\},\\ &\{edge(a,b), vertex(a), vertex(b), red(a), green(b)\},\\ &\{edge(a,b), vertex(a), vertex(b), red(a), red(b)\}\}\end{aligned}$$

In this example, the chase is finite as the rule cannot be infinitely applied (it is disjunctive Datalog). Once all vertices are colored (either green or red), no further applications of the rule are possible.

It is sometimes convenient to consider a linearization of a finite derivation tree, which we call a disjunctive derivation.

Definition III.8 (Disjunctive derivation)
A *disjunctive derivation* of an instance I and a set of rules $\mathcal{R}$ is a finite sequence of sets of instances and triggers $\mathcal{D} = (\mathcal{I}_0 = I) \xrightarrow{t_1} \mathcal{I}_1 \xrightarrow{t_2} \ldots \xrightarrow{t_k} \mathcal{I}_k$ where $t_i = (R,h)$ is a trigger of $R \in \mathcal{R}$ on an instance $I_j \in \mathcal{I}_{i-1}$ and $\mathcal{I}_i = (\mathcal{I}_{i-1} \setminus I_j) \cup \alpha_\vee(I_j,R,h)$, for all $1 \leq i \leq k$.

A disjunctive derivation is a linearisation of a finite derivation tree obtained by a total ordering of the trigger applications associated with the inner vertices in the tree, which is compatible with the parent-child partial order in the tree.

When $\mathcal{R}$ is a set of conjunctive rules, a derivation tree is a path and the instances I_i in a derivation are singletons, then, a disjunctive derivation becomes a regular derivation (as previously defined for the chase with conjunctive rules, see definition I.18), which can be viewed as a sequence of instances.

The following theorem provides the formal guarantee that the disjunctive chase is a sound and complete procedure to determine whether a disjunctive KB entails a given (Boolean) UCQ. Indeed, a UCQ is entailed by a KB if and only if it is also entailed by all instances in the result of the disjunctive chase on the KB.

Theorem III.8: from [Bourhis et al., 2016]

Let $\mathcal{Q}$ be a (Boolean) UCQ and $(I, \mathcal{R})$ be a disjunctive KB. Then $I, \mathcal{R} \models \mathcal{Q}$ if and only if $\mathsf{chase}_\vee(I, \mathcal{R}) \models \mathcal{Q}$ (that is, $I_i \models \mathcal{Q}$ for all $I_i \in \mathsf{chase}_\vee(I, \mathcal{R})$).

Example III.9: Link between models and disjunctive chase

Consider the set of rules $\mathcal{R} = \{R\}$ with $R = vertex(x) \rightarrow green(x) \vee red(x)$ and the initial instance $I = \{edge(a,b), vertex(a), vertex(b)\}$ from Example III.6, and a UCQ $\mathcal{Q} = (\exists x_1, y_1\ green(x_1) \wedge edge(x_1, y_1) \wedge green(y_1)) \vee (\exists x_2, y_2\ red(x_2) \wedge edge(x_2, y_2) \wedge red(y_2))$. This query $\mathcal{Q}$ checks if there exist two adjacent nodes that have the same colour in the graph.
According to Theorem III.8, $I, \mathcal{R} \models \mathcal{Q}$ if and only if $\mathsf{chase}_\vee(I, \mathcal{R}) \models \mathcal{Q}$, that is, $I_i \models \mathcal{Q}$ for all $I_i \in \mathsf{chase}_\vee(I, \mathcal{R})$.
We can examine the instances in $\mathsf{chase}_\vee(I, \mathcal{R})$ from Example III.7:

- $\mathcal{Q}$ has an answer in $\{edge(a,b), vertex(a), vertex(b), green(a), green(b)\}$ and $\{edge(a,b), vertex(a), vertex(b), red(a), red(b)\}$,
- but $\mathcal{Q}$ has no answer in $\{edge(a,b), vertex(a), vertex(b), green(a), red(b)\}$ and $\{edge(a,b), vertex(a), vertex(b), red(a), green(b)\}$.

Therefore, the instance I and the set of rules $\mathcal{R}$ do not entail the UCQ $\mathcal{Q}$, since there exist instances in the result of the chase where $\mathcal{Q}$ does not have an answer. This means that the initial graph represented by the instance I is 2-colorable, as there exists some 2-colorations described by the instances $\{edge(a,b), vertex(a), vertex(b), green(a), red(b)\}$ and $\{edge(a,b), vertex(a), vertex(b), red(a), green(b)\}$.

III.3 A novel technique of disjunctive rewriting

Our generalisation of query rewriting to disjunctive rules is based on a simple idea: a query $\mathcal{Q}$ can be rewritten with a rule $R = B \to H_1 \vee \cdots \vee H_n$ if *each* H_i contributes to partially answer $\mathcal{Q}$. Therefore, a unification step consists of unifying *each* H_i (using a piece-unifier) with a safe copy q_i of a CQ from $\mathcal{Q}$; safe copies ensure that the CQs involved in the unification have pairwise disjoint sets of variables. Note that several safe copies of the same CQ from $\mathcal{Q}$ can be involved. This yields a new CQ made of body(R) and the remaining parts of the unified CQs, according to some aggregation of the piece-unifiers. We need a few auxiliary notions to specify this aggregation.

Definition III.9 (Join of partitions)
Let $\mathcal{P}$ be a set of partitions (that are not necessarily partitions of the same set). The *join* of $\mathcal{P}$, denoted by join($\mathcal{P}$), is the partition obtained from $\mathcal{P}$ by making the union of the partitions in $\mathcal{P}$, then merging all non-disjoint classes until a fixed point.

Example III.10: Join of partitions

Given $\mathcal{P}$ composed of partitions $\{\{x,u\},\{y,v\},\{z,w\}\}$ and $\{\{x,y,a\},\{z',t\}\}$, we obtain $join(\mathcal{P}) = \{\{x,u,y,v,a\},\{z,w\},\{z',t\}\}$.

We say that a set of partitions associated with piece-unifiers is *admissible* if its join is an admissible partition (that is, it does not contain a class with two constants).

The following definition generalises the notion of piece-unifiers with a conjunctive rule to a disjunctive rule.

Definition III.10 (Disjunctive Piece-Unifier and One-step Piece-Rewriting)
Let a rule $R = B \to H_1 \vee \cdots \vee H_n$ and a UCQ $\mathcal{Q}$. A *disjunctive piece-unifier* $\mu_\vee$ of $\mathcal{Q}$ with R is a set $\{\mu_1,\ldots,\mu_n\}$ such that:

- for $1 \leq i \leq n$, $\mu_i = (q_i', H_i', P_{u_i})$ is a (conjunctive) piece-unifier of q_i, a safe copy of a CQ from $\mathcal{Q}$, with the (conjunctive) rule $B \to H_i$;
- and $\mathcal{P}_{u_\vee} = \{P_{u_1},\ldots,P_{u_n}\}$ is admissible.

Given a substitution $u_\vee$ associated with join($\mathcal{P}_{u_\vee}$), the application of $\mu_\vee$ produces the CQ

$$\beta_\vee(\mathcal{Q},R,\mu_\vee) = u_\vee(B) \cup \bigcup_{1\leq i\leq n} u_\vee(q_i \setminus q_i')$$

The *one-step piece-rewriting* of $\mathcal{Q}$ with respect to $\mu_\vee$ is

$$\mathcal{Q} \cup \{\beta_\vee(\mathcal{Q},R,\mu_\vee)\}$$

Example III.11

Let the disjunctive rule $R = p(x,y) \rightarrow \exists z_1\ r(x,z_1) \ \vee\ \exists z_2\ r(y,z_2)$ and the UCQ $\mathcal{Q} = \{q\}$ with $q = \exists u,v\ s(u) \wedge r(u,v)$ from Example III.2.
First, we make safe copies $q_1 = \exists u_1,v_1\ s(u_1) \wedge r(u_1,v_1)$ and $q_2 = \exists u_2,v_2\ s(u_2) \wedge r(u_2,v_2)$ of q.
Then, let the disjunctive piece-unifier $\mu_\vee = \{\mu_1,\mu_2\}$ with $\mu_1 = (\{r(u_1,v_1)\}, \{r(x,z_1)\}, \{\{u_1,x\}, \{v_1,z_1\}\})$ and $\mu_2 = (\{r(u_2,v_2)\}, \{r(y,z_2)\}, \{\{u_2,y\}, \{v_2,z_2\}\})$.
Finally, assume that we give priority to the variables from R, that is, we take the substitution $u_\vee = \{u_1 \mapsto x,\ v_1 \mapsto z_1,\ u_2 \mapsto y,\ v_2 \mapsto z_2\}$. Then:

$$\beta_\vee(\mathcal{Q}, R, \mu_\vee) = \exists x,y\ p(x,y) \wedge s(x) \wedge s(y)$$

Definition III.11 (Disjunctive Piece-Rewriting)
Given a disjunctive rule set $\mathcal{R}$, a UCQ $\mathcal{Q}'$ is a *disjunctive piece-rewriting* (or simply *rewriting* when clear from the context) of a UCQ $\mathcal{Q}$ with $\mathcal{R}$ if there is a finite sequence (called *disjunctive rewriting sequence*) $\mathcal{Q} = \mathcal{Q}_0, \mathcal{Q}_1, \ldots, \mathcal{Q}_k = \mathcal{Q}'$ $(k \geq 0)$, such that for all $0 < i \leq k$, there is a disjunctive piece-unifier $\mu_\vee$ of $\mathcal{Q}_{i-1}$ with $R \in \mathcal{R}$ such that $\mathcal{Q}_i$ is the one-step rewriting of $\mathcal{Q}_{i-1}$ with respect to $\mu_\vee$.

The following lemmas highlight the fundamental properties of $\alpha_\vee$ and $\beta_\vee$.

Lemma III.12: Preservation of entailment by $\alpha_\vee$ and $\beta_\vee$

Let R be a disjunctive rule.

1. For any instances I_1 and I_2 such that $I_2 \models I_1$: if there is a trigger (R,h_1) on I_1 then there is a trigger (R,h_2) on I_2 such that $\alpha_\vee(I_2,R,h_2) \models \alpha_\vee(I_1,R,h_1)$.
2. For any UCQs $\mathcal{Q}_1$ and $\mathcal{Q}_2$ such that $\mathcal{Q}_2 \models \mathcal{Q}_1$: if there is a (disjunctive) piece-unifier μ_2 of $\mathcal{Q}_2$ with R then either $\beta_\vee(\mathcal{Q}_2,R,\mu_2) \models \mathcal{Q}_1$, or there is a (disjunctive) piece-unifier μ_1 of $\mathcal{Q}_1$ with R such that $\beta_\vee(\mathcal{Q}_2,R,\mu_2) \models \beta_\vee(\mathcal{Q}_1,R,\mu_1)$.

The second lemma clarifies the tight relationship between $\alpha_\vee$ and $\beta_\vee$ (we recall that instances and CQs have the same logical form; this is also true of finite sets of instances and UCQs).

Lemma III.13: Composition of $\alpha_\vee$ and $\beta_\vee$

Let R be a disjunctive rule.

1. For any instance I: if there is a trigger (R,h) on I then there is a (disjunctive) piece-unifier μ of $\alpha_\vee(I,R,h)$ with R such that $I \models \beta_\vee(\alpha_\vee(I,R,h),R,\mu)$.
2. For any UCQ $\mathcal{Q}$: if there is a piece-unifier μ of $\mathcal{Q}$ with R then there is a trigger (R,h) on $\beta_\vee(\mathcal{Q},R,\mu)$ such that $\alpha_\vee(\beta_\vee(\mathcal{Q},R,\mu),R,h) \models \mathcal{Q}$.

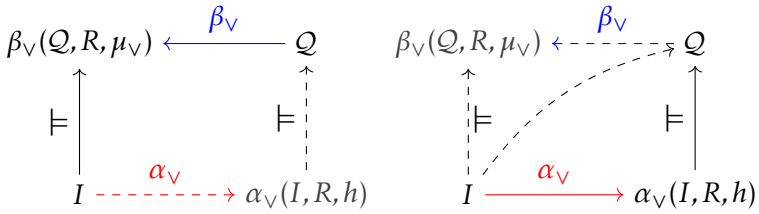


Figure III.2: Correspondences between $\beta_\vee$ (in blue) and $\alpha_\vee$ (in red)

The proofs of Lemma III.12 and Lemma III.13 can be found in Appendix A (see lemmas A.5, A.6, A.7 and A.8). These two lemmas are keys to establish the soundness and completeness of piece-rewriting, as stated in Theorem III.14.

Theorem III.14: Soundness and completeness of disjunctive piece-rewriting

Let $\mathcal{R}$ be a set of disjunctive rules and $\mathcal{Q}$ be a UCQ. Then, for any instance I, holds $I, \mathcal{R} \models \mathcal{Q}$ if and only if there is a disjunctive piece-rewriting $\mathcal{Q}'$ of $\mathcal{Q}$ such that $I \models \mathcal{Q}'$.

In the following, we give the main ideas to prove Theorem III.14. The full proof of the Theorem can be found in Appendix A.

Proof. (Sketch) We show that there is a derivation of $(\{I\}, \mathcal{R})$ leading to an $\mathcal{I}_i$ such that $\mathcal{I}_i \models \mathcal{Q}$ if and only if there is a rewriting $\mathcal{Q}_j$ of $\mathcal{Q}$ with $\mathcal{R}$ such that $I \models \mathcal{Q}_j$ (with, moreover, $j \leq i$). This equivalence is based on the following two lemmas, which are corollaries of previous Lemmas III.12 and III.13. Given any Boolean UCQ $\mathcal{Q}$, disjunctive rule R, and instance I, the following holds (see Figure III.2):

- (Backward-forward Lemma) For any disjunctive piece-unifier $\mu_\vee$ of $\mathcal{Q}$ with R, if $I \models \beta_\vee(\mathcal{Q}, R, \mu_\vee)$ then there is a trigger (R, h) on I such that $\alpha_\vee(I, R, h) \models \mathcal{Q}$;
- (Forward-backward Lemma) For any trigger (R, h) on I, if $\alpha_\vee(I, R, h) \models \mathcal{Q}$ then either $I \models \mathcal{Q}$ or there is a disjunctive piece-unifier $\mu_\vee$ of $\mathcal{Q}$ with R, such that $I \models \beta_\vee(\mathcal{Q}, R, \mu_\vee)$.

The ($\Rightarrow$) direction of the theorem is proved by induction on the length k of a derivation from $\{I\}$ to $\mathcal{I}_k$ such that $\mathcal{I}_k \models \mathcal{Q}$, using forward-backward Lemma (which itself follows from Lemma III.13 (Point 1) and Lemma III.12 (Point 2)). The ($\Leftarrow$) direction is proved by induction on the length k of a rewriting sequence from $\mathcal{Q}$ to $\mathcal{Q}_k$ such that $I \models \mathcal{Q}_k$, using backward-forward Lemma (which itself follows from Lemma III.13 (Point 2) and Lemma III.12 (Point 1)). ■

To actually compute a UCQ-rewriting of $\mathcal{Q}$ when one exists, it is convenient to proceed in a breadth-first manner, that is, extend $\mathcal{Q}$ at each step with all the CQs that can

be generated with (new) disjunctive piece-unifiers. More specifically, we inductively define the following operator $W_\infty^\vee$.

Definition III.12 (Breadth-first disjunctive rewriting operator)
The *breadth-first disjunctive rewriting operator* $W_\infty^\vee$ takes as input a UCQ $\mathcal{Q}$ and a disjunctive rule set $\mathcal{R}$, and returns a possibly infinite set of CQs inductively defined as follows:

- $W_0^\vee(\mathcal{Q},\mathcal{R}) = \mathcal{Q}$
- For $i > 0$:

$$W_i^\vee(\mathcal{Q},\mathcal{R}) = W_{i-1}^\vee(\mathcal{Q},\mathcal{R}) \cup \{\beta_\vee(W_{i-1}^\vee(\mathcal{Q},\mathcal{R}), R, \mu_\vee) \mid \mu_\vee \text{ piece-unifier with } R \in \mathcal{R}\}$$

- Finally, $W_\infty^\vee(\mathcal{Q},\mathcal{R}) = \bigcup_{i\in\mathbb{N}} W_i^\vee(\mathcal{Q},\mathcal{R})$.

Proposition III.15: Properties of $W_\infty^\vee$

For any UCQ $\mathcal{Q}$ and disjunctive rule set $\mathcal{R}$, the following holds:

1. $W_\infty^\vee(\mathcal{Q},\mathcal{R})$ is a complete rewriting of $(\mathcal{Q},\mathcal{R})$.
2. If $(\mathcal{Q},\mathcal{R})$ admits a UCQ-rewriting $\mathcal{Q}'$, then there is $i \geq 0$ such that $\mathcal{Q}' \equiv W_i^\vee(\mathcal{Q},\mathcal{R})$.

Proof. (1) Each $W_i^\vee(\mathcal{Q},\mathcal{R})$ is a piece-rewriting of $\mathcal{Q}$ with $\mathcal{R}$ and, for any piece-rewriting $\mathcal{Q}'$ of $\mathcal{Q}$ with $\mathcal{R}$, there is i such that $\mathcal{Q}' \subseteq W_i^\vee(\mathcal{Q},\mathcal{R})$. Therefore, the union of all the $W_i^\vee(\mathcal{Q},\mathcal{R})$ is a complete rewriting of $\mathcal{Q}$. (2) If $(\mathcal{Q},\mathcal{R})$ admits a UCQ-rewriting $\mathcal{Q}'$, then by Theorem III.14 it admits a complete piece-rewriting $\mathcal{Q}''$, and both are necessarily equivalent. Then, $\mathcal{Q}'' \subseteq W_i^\vee(\mathcal{Q},\mathcal{R})$ for some i and since $\mathcal{Q}''$ is complete, $\mathcal{Q}'' \equiv W_i^\vee(\mathcal{Q},\mathcal{R})$.■

We propose a query rewriting algorithm (see Algorithm 2) that mimics the computation of $W_\infty^\vee(\mathcal{Q},\mathcal{R})$, while including two optimisations at each step $i > 0$.

First, it only considers *new* disjunctive piece-unifiers, that is, those that involve at least one CQ generated at step $i-1$. Second, it removes redundant CQs in the rewriting under construction by the computation of a cover.

More specifically, $\mathcal{Q}^\star$ denotes the rewriting under construction and $\mathcal{Q}_{new}$ the set of CQs generated at a given step. The function `cover` (Lines 1 and 6) returns a cover of the given set. The function `generate` (Line 5) takes as input the current rewriting $\mathcal{Q}^\star$, its subset $\mathcal{Q}_{prev}$ of CQs generated at the previous step, as well as $\mathcal{R}$, and returns the set of generated CQs, that is, all the $\beta_\vee(\mathcal{Q}^\star, R, \mu_\vee)$ where $\mu_\vee$ is a new disjunctive piece-unifier. This yields the set $\mathcal{Q}_{new}$.

To compute a cover of $\mathcal{Q}^\star \cup \mathcal{Q}_{new}$, priority is given to $\mathcal{Q}^\star$ in case of query equivalence, for termination reasons. The function `removeMoreSpecific` takes as input two sets of CQs and returns the first set minus its queries more specific than a query of the second set.

The computation of a cover of $\mathcal{Q}^\star \cup \mathcal{Q}_{new}$ is decomposed into three steps (Lines 6-8): compute a cover of $\mathcal{Q}_{new}$; remove from $\mathcal{Q}_{new}$ the queries more specific than a query from $\mathcal{Q}^\star$; and remove from $\mathcal{Q}^\star$ the queries more specific than a query from $\mathcal{Q}_{new}$. Then, $\mathcal{Q}_{new}$ is added to $\mathcal{Q}^\star$ (Line 9).

We remind that a query may have rewritings of unbounded size but still a UCQ-rewriting (see Example I.13), hence the role of the cover computation is not only to remove redundancies but also to ensure that the algorithm halts when a UCQ-rewriting has been found.

Algorithm 2: Breadth-First Disjunctive Rewriting

Data: Boolean UCQ $\mathcal{Q}$ and set of disjunctive rules $\mathcal{R}$
Result: A sound and complete rewriting of $\mathcal{Q}$

1 $\mathcal{Q}_{new} \leftarrow \texttt{cover}(\mathcal{Q})$; *// new CQs*
2 $\mathcal{Q}^\star \leftarrow \mathcal{Q}_{new}$; *// result*
3 **while** $\mathcal{Q}_{new} \neq \emptyset$ **do**
4 $\quad \mathcal{Q}_{prev} \leftarrow \mathcal{Q}_{new}$ *// CQs from the preceding step*
5 $\quad \mathcal{Q}_{new} \leftarrow \texttt{generate}(\mathcal{Q}^\star, \mathcal{Q}_{prev}, \mathcal{R})$; *// new CQs*
6 $\quad \mathcal{Q}_{new} \leftarrow \texttt{cover}(\mathcal{Q}_{new})$
7 $\quad \mathcal{Q}_{new} \leftarrow \texttt{removeMoreSpecific}(\mathcal{Q}_{new}, \mathcal{Q}^\star)$
8 $\quad \mathcal{Q}^\star \leftarrow \texttt{removeMoreSpecific}(\mathcal{Q}^\star, \mathcal{Q}_{new})$
9 $\quad \mathcal{Q}^\star \leftarrow \mathcal{Q}^\star \cup \mathcal{Q}_{new}$
10 **end**
11 **return** $\mathcal{Q}^\star$

The correctness of the algorithm is based on the soundness and completeness of the $W_\infty^\vee$ operator, however, attention should be paid to the potential impact of query removal on completeness (Lines 6 to 8). Indeed, when a CQ q_2 is removed because it is more specific than another CQ q_1, we have to ensure that any CQ that could be generated using q_2 is more specific than another CQ already present in the current rewriting, or than a CQ that can be generated using q_1. Fortunately, this property is ensured by Lemma A.10, considering $\mathcal{Q}^\star$ and $\mathcal{Q}_{new}$ at the end of Line 5, then taking $\mathcal{Q}_2 = \mathcal{Q}^\star \cup \mathcal{Q}_{new}$ and $\mathcal{Q}_1 = \mathcal{Q}_2 \setminus \{q_2\}$.

Theorem III.16

Algorithm 2 computes a sound and complete rewriting. Moreover, it halts and outputs a minimal rewriting when $(\mathcal{Q}, \mathcal{R})$ is UCQ-rewritable.

Proof. By induction on the number of iterations of the while loop, we prove the following invariant of the algorithm, using Lemma A.10: after step i, $\mathcal{Q}^\star$ is equivalent to $W_i^\vee(\mathcal{Q}, \mathcal{R})$. Then, soundness and completeness follow from Proposition III.15. Line 7 ensures that $\mathcal{Q}_{new}$ becomes empty when $\mathcal{Q}^\star$ is a complete rewriting. Since a cover of $\mathcal{Q}^\star$ is computed at each step, the output set is of minimal size. ■

Further remarks on completeness. When it comes to practical implementations, one may find simpler to rely on (conjunctive) piece-unifiers that unify the smallest possible subsets of a CQ. Such piece-unifiers are called *single-piece* [König et al., 2015]. In the specific case of Datalog, a single-piece-unifier unifies a single atom of a CQ with a rule head. Piece-rewriting restricted to single-piece-unifiers is complete for conjunctive rules [König et al., 2015], but it is no longer so with disjunctive rules. This occurs already in the case of disjunctive Datalog, as illustrated next.

Example III.17

Consider again the colorability example (Example III.4) with $R = vertex(x) \rightarrow green(x) \vee red(x)$ and $\mathcal{Q} = \{q_1, q_2\}$ with $q_1 = green(u) \wedge edge(u,w) \wedge green(w)\}$ and $q_2 = red(u) \wedge edge(u,w) \wedge red(w)$.
With single-piece-unifiers we obtain CQs that have the shape of "chains" with a *green*-atom or an *red*-atom at each extremity. However, there are also rewritings without any occurrence of *green* nor *red*, and the only way of obtaining them is to unify two query atoms together.
For instance, the CQ $\{vertex(u), edge(u,u)\}$ is obtained by unifying, on the one hand both *green*-atoms of a safe copy of q_1 with $green(x)$, and on the other hand both *red*-atoms of a safe copy of q_2 with $red(x)$.
More generally, using such piece-unifiers, one can produce all the CQs that describe the odd-length cycles in the graph. Note that these CQs are incomparable with the CQs generated with single-piece-unifiers. This example also shows that a UCQ may have no UCQ-rewriting although each of its CQs has one (which is here the CQ itself).

Similarly as the rewriting function for conjunctive rules (Definition I.37), we define below a disjunctive rewriting function.

Definition III.13 (Disjunctive rewriting Function)
Let $\mathcal{Q}$ be a UCQ and $\mathcal{R}$ be a set of disjunctive rules. The *disjunctive rewriting function*, denoted by $\texttt{rewriting}_\vee(\mathcal{Q}, \mathcal{R})$, is defined as a function that returns $W_k^\vee(\mathcal{Q}, \mathcal{R})$ with k the smallest integer such that $W_k^\vee(\mathcal{Q}, \mathcal{R}) \equiv W_{k+1}^\vee(\mathcal{Q}, \mathcal{R})$, if such an integer exists, otherwise it returns $W_\infty^\vee(\mathcal{Q}, \mathcal{R})$.

III.4 Is FUS relevant for disjunctive rules?

We now address the question of identifying classes of disjunctive rules that are UCQ-rewritable.

By extension of the term coined for conjunctive existential rules (Definition I.24), we also call them *FUS*. To the best of our knowledge, the only *FUS* class of disjunctive rules mentioned in the literature [Alfonso et al., 2021] is actually a slight extension of *FUS* conjunctive rules: this class consists of disjunctive rules with an empty frontier and it is shown that such rules can be safely added to a set of *FUS* conjunctive rules. As

a matter of fact, known *FUS* classes of conjunctive rules do not seem to be extensible to the disjunctive case. And worse, the straightforward extension of syntactic criteria that underlie *FUS* in the conjunctive case seems to easily lead to undecidability of query answering, as shown for example in [Morak, 2021] for the syntactic restriction called stickiness [Calì et al., 2010]. At first glance, one may expect *nonrecursive* (i.e., s-to-o) disjunctive rule sets to be *FUS*, as it happens for conjunctive rules. However, it is not the case, as shown by the next example: a CQ (on unary predicates) may have no UCQ-rewriting even with a *single non-recursive body-atomic* (disjunctive) *Datalog* rule.

Example III.18

Let the rule $R = p(x,y) \rightarrow t_1(x) \vee t_2(y)$ and the BCQ $q = t_1(u) \wedge t_2(u)$. Then the pair $(\{q\},\{R\})$ has no UCQ-rewriting. Indeed, a complete rewriting contains all the CQs of the following shape for any $n \in \mathbb{N}$:

$$t_2(u_0) \wedge \left(\bigwedge_{i=1}^{n} p(u_{i-1}, u_i) \right) \wedge t_1(u_n)$$

All these queries are pairwise incomparable with respect to homomorphism. Let us detail the first rewriting step. To unify $\{q\}$ with R, we have to make two safe copies of q, let q_1 and q_2, which are respectively unified with $t_1(x)$ and $t_2(y)$. This produces the CQ $\{t_2(x), p(x,y), t_1(y)\}$, isomorphic to $\{t_2(u_0), p(u_0,u_1), t_1(u_1)\}$. If we switch the unified atoms of $\mathsf{head}(R)$, we obtain an isomorphic CQ. All subsequent rewriting steps lead to longer paths of p-atoms.

A similar observation follows from [Gerasimova et al., 2020], which focuses on a specific disjunctive rule of the form $A(x) \rightarrow T(x) \vee F(x)$, called a covering axiom and denoted by cov_A; their complexity results imply that the singleton set $\{cov_A\}$ is not *FUS*[2], which can be checked, for instance, by considering the query $Q = \{T(u), p(u,v), F(v)\}$.

Next, we show that such observations can be generalized to almost *any* source-to-target disjunctive rule. Evidently, we have to exclude disjunctive rules that are equivalent to a conjunctive rule, as classes of *FUS* conjunctive rules are known. We also exclude *disconnected* rules, that is, rules R such that $\mathsf{body}(R) \cup \mathsf{head}(R)$ is not a connected set of atoms (where connectivity is defined in the obvious way based on shared variables). Note that a rule with a head H_i that has an empty frontier is disconnected, as well as a rule whose body has a connected component with an empty frontier. However, a rule with a disconnected body may not be disconnected, since head atoms may connect several connected components of the body (e.g., a "product"

[2] That paper studies syntactic conditions on ontology-mediated CQs of the form (Q, cov_A) that determine the data complexity of query answering and the rewritability in some target query language. In particular, it is shown that if a (connected) CQ Q has no term x with both atoms $T(x)$ and $F(x)$ and contains at least one F-atom and one T-atom then answering (Q, cov_A) is L-hard for data complexity. Since answering a UCQ-rewritable ontology-mediated query is in AC^0 for data complexity, and $AC^0 \subset L$, it follows that no cov_A is FUS.

rule like $b_1(x) \wedge b_2(y) \rightarrow t_1(x) \vee t_2(y) \vee p(x,y)$ is not disconnected).

Example III.19: *FUS* disconnected rule

Let the disconnected rule $R = b(x) \rightarrow t_1(x) \vee \exists z\ t_2(z)$. R is not equivalent to a conjunctive rule. Let us check that it is *FUS*. Given any UCQ $\mathcal{Q}$, let $\mathcal{Q}_2$ be the subset of $\mathcal{Q}$ that contains all the CQs that can be unified with $\exists z\ t_2(z)$. Any $Q \in \mathcal{Q}_2$ necessarily contains a disconnected component of the form $\exists u\ t_2(u)$. Moreover, it is useless to unify Q with $t_1(x)$: in such case, let Q_2 be the CQ unified with $\exists z\ t_2(z)$, then the obtained rewriting is more specific than Q_2. Hence, we can ignore all the produced CQs that contain a connected component of the form $\exists u\ t_2(u)$. Rewriting $\mathcal{Q}$ with $\{R\}$ amounts to rewriting $\mathcal{Q} \setminus \mathcal{Q}_2$ with the conjunctive rule set $\mathcal{R} = \{b(x) \wedge (q_2 \setminus \{\exists u\ t_2(u)\}) \rightarrow t_1(x) \mid q_2 \in \mathcal{Q}_2\}$, which belongs to the *FUS* class called *domain restricted* [Baget et al., 2011].

In the next theorem, we restrict the head of the rule to a disjunction of two atomsets, to keep the proof simple.

Theorem III.20

Let $R = B \rightarrow H_1 \vee H_2$ be a source-to-target rule that is not disconnected nor equivalent to a conjunctive rule. Then, there is a CQ q such that $(\{q\}, \{R\})$ is not UCQ-rewritable.

Proof. Let $R = B[\mathbf{x_1}, \mathbf{x_2}, \mathbf{y}] \rightarrow \exists \mathbf{z_1}\ H_1[\mathbf{x_1}, \mathbf{z_1}] \vee \exists \mathbf{z_2}\ H_2[\mathbf{x_2}, \mathbf{z_2}]$, where:

- $\mathrm{fr}(R) = \mathbf{x_1} \cup \mathbf{x_2}$; $\mathbf{x_1}$ and $\mathbf{x_2}$ may share variables;
- $\mathbf{x_i} \neq \emptyset$ $(i = 1, 2)$ since R is not disconnected.

We build the following Boolean CQ:

$$q = H_1^s[\mathbf{v_1}, \mathbf{w_1}] \wedge p(\mathbf{v_1}, \mathbf{v_2}) \wedge H_2^s[\mathbf{v_2}, \mathbf{w_2}]$$

where each $H_i^s[\mathbf{v_i}, \mathbf{w_i}]$ is a safe copy of $H_i[\mathbf{x_i}, \mathbf{z_i}]$ and p is a fresh predicate. Note that, since R is connected, both H_1 and H_2 have a frontier variable, and frontier variables being safely renamed in each H_i^s, we have $\mathbf{v_1} \cap \mathbf{v_2} = \emptyset$, hence the arity of p is at least 2. In $p(\mathbf{v_1}, \mathbf{v_2})$ the order of the variables is important: a fixed order is chosen in $\mathbf{x_i}$ (hence, $\mathbf{v_i}$) and the tuple $\mathbf{v_1}$ comes before the tuple $\mathbf{v_2}$. Hence, $p(\mathbf{v_1}, \mathbf{v_2})$ can be seen as "directed" from $\mathbf{v_1}$ to $\mathbf{v_2}$. We then proceed in two steps.

1. We show that we can produce an infinite set $\mathcal{Q}$ whose element CQs are pairwise incomparable by homomorphism. Let $q_0 = q$. At each step $i \geq 1$, q_i is produced from a safe copy of q unified with H_1 and a safe copy of q_{i-1} unified with H_2. The piece-unifiers unify H_1^s (resp. H_2^s) in q (resp. q_{i-1}) according to the isomorphism from H_1^s (resp. H_2^s) to H_1 (resp. H_2). Any CQ q_k in $\mathcal{Q}$ is connected and follows

the "pattern" $H_1^s.p.(B.p)^k.H_2^s$, where the occurrences of p atoms all have the same direction; hence, two "adjacent" p-atoms, i.e., that share variables with the same copy B_i of a B, cannot be mapped one onto the other (by a homomorphism that maps B_i to itself).

2. We show that no CQ q' that can be produced by piece-rewriting maps by homomorphism to a CQ from $\mathcal{Q}$, except by isomorphism. When there is no (conjunctive) piece-unifier that unifies $H_1[\mathbf{v_1}, \mathbf{w_1}]$ in q with $H_2[\mathbf{x_2}, \mathbf{z_2}]$ (then, the same holds if we exchange H_1 and H_2), all the produced q' are more specific than (including isomorphic to) CQs from $\mathcal{Q}$. Otherwise, assume that a CQ Q' is produced by unifying $H_1[\mathbf{v_1}, \mathbf{w_1}]$ with $H_2[\mathbf{x_2}, \mathbf{z_2}]$. If Q' can be mapped by homomorphism to a $Q_n \in \mathcal{Q}$, the arguments of any p-atom in Q' must be pairwise distinct variables. We show that it leads to have R equivalent to the conjunctive rule $B \to H_i$ (with $i = 1$ or $i = 2$), which contradicts the hypothesis on R.

It follows that $\mathcal{Q}$ is a subset of any sound and complete rewriting of $\{q\}$ with $\{R\}$, therefore the pair $(\{q\}, \{R\})$ does not admit a UCQ-rewriting.

Details on step 1. We consider the infinite sequence $\mathcal{Q}_0, \ldots, \mathcal{Q}_i, \ldots$, where $\mathcal{Q}_0 = \{q_0 = q\}$ and for all $i > 0$, $\mathcal{Q}_i = \mathcal{Q}_{i-1} \cup \{q_i\}$, where q_i is obtained by a (disjunctive) piece-unifier that unifies safe copies of q_0 and q_{i-1}, with H_1 and H_2 respectively, according to the isomorphism from H_1^s (resp. H_2^s) to H_1 (resp. H_2). By an easy induction on the length k of the rewriting sequence leading to $\mathcal{Q}_k$ ($k \geq 0$), we check that all the CQs q_k are of the following form:

$$H_1^s[\mathbf{v_1^0}, \mathbf{w_1}] \wedge p(\mathbf{v_1^0}, \mathbf{v_2^0}) \wedge \left(\bigwedge_{i=1}^{k} B[\mathbf{v_2^{i-1}}, \mathbf{v_1^i}, \mathbf{y_i}] \wedge p(\mathbf{v_1^i}, \mathbf{v_2^i}) \right) \wedge H_2^s[\mathbf{v_2^k}, \mathbf{w_2}]$$

Moreover, q_k is connected. Indeed, by hypothesis, R is connected, hence B is connected, or we have $\mathbf{v_1^{i-1}} \cap \mathbf{v_2^i} \neq \emptyset$, for all $i > 0$, i.e., two p-atoms adjacent to a B share a variable.

Since the two p-atoms connected to an occurrence of B are "in the same direction", they do not fold one onto the other. Hence, if a CQ q_i maps to a CQ q_j ($i \neq j$), it is necessarily by an injective homomorphism. However, this is impossible because the "chains" that underlie these CQs are of different length, while the copies of H_1 and H_2 at their extremities should be mapped one onto the other. Hence, the set $\mathcal{Q}$ defined as the union of all the $\mathcal{Q}_i$ for $i \in \mathbb{N}$, is composed of pairwise incomparable CQs.

Details on step 2. **(1)** We first consider the case where there is no (conjunctive) piece-unifier that unifies $H_1[\mathbf{v_1}, \mathbf{w_1}]$ in q with $H_2[\mathbf{x_2}, \mathbf{z_2}]$ (then, the same holds if we exchange H_1 and H_2) and show that the produced CQs are more specific than (including isomorphic to) CQs from $\mathcal{Q}$. Indeed, in this case, all the CQs produced are of the above general form, except that the p-atoms may be specialized, as well as the B's on their frontier (it is the case if we consider more specific unifiers than the ones used to build

$\mathcal{Q}$). Let us prove it by induction on the length l of a rewriting sequence. This is true for $l = 0$. Assume this is true until $l = n$. For $l = n + 1$, let $\mathcal{Q}_j$ and $\mathcal{Q}_k$, with (in simplified form) $\mathcal{Q}_j = H_1^s.p.(B.p)^j.H_2^s$ unified with H_1 and $\mathcal{Q}_k = H_1^s.p.(B.p)^k.H_2^s$ unified with H_2. The produced CQ has the form $H_1^s.p.(B.p)^{j+k+1}.H_2^s$, hence it is more specific than q_{j+k+1}, as defined in the step 1 of the proof. **(2)** Otherwise, let Q_k be a CQ produced by unifying $H_1^s[\mathbf{v_1}, \mathbf{w_1}]$ with $H_2[\mathbf{x_2}, \mathbf{z_2}]$ (if we exchange H_1 and H_2, the case is similar). If Q_k can be mapped by homomorphism to a $Q_n \in \mathcal{Q}$, any p-atom in Q_k must have pairwise distinct variables. Hence, when an atom set of the form H_1 is unified with an atom set of the form H_2, the (copies of the) frontier variables in each set have to remain distinct (i.e., no frontier variable can be unified with another frontier variable in the same set). From this observation and the fact that two existential variables of H_2 cannot be unified together, there is a homomorphism from $H_1^s[\mathbf{v_1}, \mathbf{w_1}]$ to $H_2[\mathbf{x_2}, \mathbf{z_2}]$, with $\mathbf{v_1}$ mapped to $\mathbf{x_2}$. Since by construction of the rewriting, an H_1^s is never specialized (by merging two variables or replacing a variable by a constant), H_1^s is isomorphic to H_1 (with frontier variables mapped to frontier variables). Hence, there is a homomorphism h from H_1 to H_2, with frontier variables mapped to frontier variables. Now, two cases: either $h(B)$ maps to B by a homomorphism invariant on the frontier variables of $h(B)$, and $B \to H_2 \models B \to H_1$, hence R is equivalent to the conjunctive rule $B \to H_2$, which is excluded by hypothesis; or $h(B)$ does not map to B by a homomorphism invariant on the frontier variables of $h(B)$, and it does not map to a B by a homomorphism from Q_k to Q_n, hence there is no homomorphism from Q_k to Q_n. ■

One interest of the above proof is to provide a general construction that applies to any rule (fulfilling the conditions of the theorem). Also, the proof can be generalized to a rule head with k disjuncts, taking q containing a safe copy of each H_i plus a p-atom that connects these copies through their frontier variables. Given this result, the notion of *FUS* disjunctive rules does not seem to be particularly relevant. Studying the problem of deciding whether a pair $(\mathcal{Q}, \mathcal{R})$ is UCQ-rewritable seems more interesting, although it is known to be undecidable already for (conjunctive) Datalog rules.[3] Again, little is known about classes of disjunctive rules and UCQs for which this problem would be decidable. Let us point out a few immediate cases of UCQ-rewritable pairs $(\mathcal{Q}, \mathcal{R})$:

- $\mathcal{Q}$ is composed of atomic CQs and $\mathcal{R}$ is a set of disjunctive linear existential rules (that is, rules with an atomic body). Indeed, only atomic CQs can be produced, and there is a finite number of them on a given set of predicates. This case was already noticed in [Bourhis et al., 2016].

- $\mathcal{Q}$ is composed of atomic queries and $\mathcal{R}$ is a set of $\mathcal{S}$-to-$\mathcal{T}$ rules. The produced CQs are obtained from the rule bodies by specializing their frontier (that is, merging

[3] This follows from the undecidability of determining whether a Datalog program is uniformly bounded [Gaifman et al., 1993]. Indeed, a Datalog program $\mathcal{R}$ is uniformly bounded if and only if the pair $(q, \mathcal{R})$ is UCQ-rewritable for any *full* atomic query q. In turn, UCQ-rewritability of $(q, \mathcal{R})$ can be reduced to UCQ-rewritability of $(q', \mathcal{R})$ with q' a Boolean CQ.

variables and replacing them by constants occurring in $\mathcal{Q}$ and rule heads). Hence, there is a finite number of them.

- $\mathcal{Q}$ is composed of variable-free CQs[4] and $\mathcal{R}$ is a set of lossless existential rules (that is, such that all the variables in a rule body are frontier). Then, no variable is introduced by rewriting, hence the number of terms in a CQ is bounded by $|\mathtt{consts}(\mathcal{Q}) \cup \mathtt{consts}(\mathcal{R})|$.

III.5 Disjunctive Mappings

We now consider UCQ-rewritability with (disjunctive) mappings. Let $\mathcal{S}$ and $\mathcal{T}$ be the sets of source and target predicates, respectively, and let $\mathcal{M}$ be a mapping on $(\mathcal{S},\mathcal{T})$. Given a query on $\mathcal{T}$, the aim is to obtain a complete rewriting with respect to instances on $\mathcal{S}$. Because $\mathcal{S}$ and $\mathcal{T}$ are disjoint, CQs that contain atoms on $\mathcal{T}$ are useless in a rewriting. Hence, we define a *mapping rewriting* as a rewriting on $\mathcal{S}$ and use the notation $\mathcal{S}$*-rewriting* to distinguish it from a rewriting on $\mathcal{S} \cup \mathcal{T}$. An $\mathcal{S}$-rewriting $\mathcal{Q}'$ of a UCQ $\mathcal{Q}$ with $\mathcal{M}$ is *complete* if, for all instance I on $\mathcal{S}$, if $I, \mathcal{M} \models \mathcal{Q}$ then $I \models \mathcal{Q}'$. A finite complete $\mathcal{S}$-rewriting is called a *UCQ-$\mathcal{S}$-rewriting*.

Example III.21: Colorability

We adapt Example III.17 to transform the rule into a mapping. Let $\mathcal{S} = \{vertex, edge\}$, $\mathcal{T} = \{t\_edge, green, red\}$ and $\mathcal{M} = \{m_1, m_2\}$, with:
$m_1 = edge(x,y) \rightarrow t\_edge(x,y)$
$m_2 = vertex(x) \rightarrow green(x) \vee red(x)$.
Let $\mathcal{Q} = \{q_1, q_2\}$ with $q_1 = green(u) \wedge t\_edge(u,w) \wedge green(w)$ and $q_2 = red(u) \wedge t\_edge(u,w) \wedge red(w)$. Any complete $\mathcal{S}$-rewriting of $\mathcal{Q}$ contains CQs that describe all the cycles of odd length (in other words, it defines non-2-colorability). All the other CQs that can be produced by piece-rewriting contain predicates *green* and *red*, hence are discarded.

Note that a query may have a UCQ-$\mathcal{S}$-rewriting, while it does not have any UCQ-rewriting (on $\mathcal{S} \cup \mathcal{T}$), as illustrated by the next example.

Example III.22

Let $\mathcal{S} = \{p\}$ and $\mathcal{T} = \{t_1, t_2\}$. Consider the (Boolean) CQ $q = t_1(u) \wedge t_2(u)$ and the rule $R = p(x,y) \rightarrow t_1(x) \vee t_2(y)$ from Example III.18. Although the pair $(\{q\}, \{R\})$ has no UCQ-rewriting, it has a UCQ-$\mathcal{S}$-rewriting, which is empty. In fact, all CQs that can be obtained by piece-rewriting contain an atom on $\mathcal{T}$.

[4] If non-Boolean CQs are considered, $\mathcal{Q}$ can be extended to a set of full CQs.

III.5.1 Undecidability of disjunctive mapping rewritability

Let *disjunctive mapping rewritability* be the following problem: Given a disjunctive mapping $\mathcal{M}$ on $(\mathcal{S},\mathcal{T})$ and a UCQ $\mathcal{Q}$ on $\mathcal{T}$, does $(\mathcal{Q},\mathcal{M})$ have a UCQ-$\mathcal{S}$-rewriting ?

Theorem III.23

Disjunctive mapping rewritability is undecidable.

In the following, we give the main ideas to prove Theorem III.23. The full proof of the Theorem can be found in Appendix B.

Proof. (Sketch) We build a reduction from the following undecidable problem: Given a (Boolean) CQ q and a set of (conjunctive) Datalog rules $\mathcal{R}$, is the pair $(\{q\},\mathcal{R})$ UCQ-rewritable? W.l.o.g. we assume that rules in $\mathcal{R}$ have no constants (and an atomic head). The reduction translates each instance $(q,\mathcal{R})$ defined on a set of predicates $\mathcal{P}$, into an instance $(\mathcal{Q}^{Q,\mathcal{R}},\mathcal{M}^{Q,\mathcal{R}})$ of the disjunctive mapping rewritability problem, defined on a pair of predicats sets $(\mathcal{S},\mathcal{T})$ such that:

- $\mathcal{S} = \mathcal{P} \cup \{T\}$, where T is a fresh unary predicate,
- $\mathcal{T}$ is the union of: (1) a set of predicates in bijection with $\mathcal{S}$, where $\hat{p}$ denotes the predicate obtained from $p \in \mathcal{S}$, and (2) a set of fresh predicates in bijection with $\mathcal{R}$, where p_{R_i} denotes the predicate associated with the rule R_i; the arity of each p_{R_i} is $|\mathrm{fr}(R_i)|$.

Given a conjunction Q (on $\mathcal{P}$), we denote by Q^T the conjunction (on $\mathcal{S}$) obtained from Q by adding a T-atom on each term; given a conjunction Q (on $\mathcal{S}$), we denote by $\hat{Q}$ the conjunction (on $\mathcal{T}$) obtained from Q by renaming all the predicates p into $\hat{p}$. Hence, $\widehat{Q^T}$ is obtained by performing the first operation, then the second. Given $\mathbf{x} = x_1,\dots,x_n$, $T[\mathbf{x}]$ denotes the conjunction $T(x_1) \wedge \dots \wedge T(x_n)$. Similarly, $\hat{T}[\mathbf{x}] = \hat{T}(x_1) \wedge \dots \wedge \hat{T}(x_n)$.

Let q and $\mathcal{R} = \{R_1,\dots,R_n\}$, where $R_i = B_i[\mathbf{x_i},\mathbf{y_i}] \to H_i[\mathbf{x_i}]$. The instance $(\mathcal{Q}^{q,\mathcal{R}},\mathcal{M}^{q,\mathcal{R}})$ is defined as follows:

- $\mathcal{Q}^{q,\mathcal{R}} = \{q_q\} \cup \mathcal{Q}_\mathcal{R}$ with:
  $q_Q = \widehat{q^T}$,
  $\mathcal{Q}_\mathcal{R} = \{q_{R_i} = \exists \mathbf{x_i},\mathbf{y_i}\ \widehat{(B_i)^T}[\mathbf{x_i},\mathbf{y_i}] \wedge p_{R_i}(\mathbf{x_i}) | R_i \in \mathcal{R}\}$
- $\mathcal{M}^{q,\mathcal{R}} = \mathcal{M}_\mathcal{R} \cup \mathcal{M}_{trans}$ with:
  $\mathcal{M}_\mathcal{R} = \{m_{R_i} = T[\mathbf{x_i}] \to p_{R_i}(\mathbf{x_i}) \vee \hat{H}_i(\mathbf{x_i}) \mid R_i \in \mathcal{R}\}$
  $\mathcal{M}_{trans} = \{p(\mathbf{x}) \to \hat{p}(\mathbf{x}) \mid p \in \mathcal{S}\}$

Based on the natural bijection between the CQs $Q_\mathcal{P}$ defined on $\mathcal{P}$ and the CQs $(Q_\mathcal{P})^T$ defined on $\mathcal{S}$, we prove that $Q_\mathcal{P}$ belongs to a rewriting of $\{q\}$ with $\mathcal{R}$ if and only if $(Q_\mathcal{P})^T$ belongs to a rewriting of $\mathcal{Q}^{q,\mathcal{R}}$ with $\mathcal{M}^{q,\mathcal{R}}$. Note that set membership is up to isomorphism throughout the proof. More specifically, we first prove the following lemmas:

1. For any CQ Q_w in a piece-rewriting of $\{q\}$ with $\mathcal{R}$, $(Q_w)^T$ belongs to a piece-rewriting of $\mathcal{Q}^{q,\mathcal{R}}$ with $\mathcal{M}^{q,\mathcal{R}}$. Indeed, to each R_i are associated a CQ q_{R_i} and a rule m_{R_i} that allow to simulate any rewriting step performed with R_i, using fresh predicate p_{R_i}.

2. Any CQ Q_S in an $\mathcal{S}$-rewriting of $\mathcal{Q}^{q,\mathcal{R}}$ with $\mathcal{M}^{q,\mathcal{R}}$ is of the form $Q_S = (Q_\mathcal{P})^T$, with $Q_\mathcal{P}$ the subset of Q_S on $\mathcal{P}$.

3. For any CQ of the form $(Q_\mathcal{P})^T$, with $Q_\mathcal{P}$ on $\mathcal{P}$, that belongs a piece-rewriting of $\mathcal{Q}^{q,\mathcal{R}}$ with $\mathcal{M}^{q,\mathcal{R}}$, $Q_\mathcal{P}$ belongs to a piece-rewriting of $\{q\}$ with $\mathcal{R}^\star$, where $\mathcal{R}^\star$ is the reflexive and transitive closure of $\mathcal{R}$ by unfolding (that is, rule composition). Note that $\mathcal{R}^\star$ is logically equivalent to $\mathcal{R}$.

We rely on these lemmas to prove the following: if there is a UCQ-rewriting of $(\{Q\},\mathcal{R})$ then there is a UCQ-$\mathcal{S}$-rewriting of $(\mathcal{Q}^{q,\mathcal{R}},\mathcal{M}^{q,\mathcal{R}})$. The proof of the opposite direction is similar. Let $\mathcal{Q}$ be a UCQ-rewriting of $(\{Q\},\mathcal{R})$. Then there is a piece-rewriting $\mathcal{Q}_i$ of $\{Q\}$ with $\mathcal{R}$ such that $\mathcal{Q}_i \equiv \mathcal{Q}$. By Lemma 1, there is a piece-rewriting $\mathcal{Q}_j$ of $\mathcal{Q}^{q,\mathcal{R}}$ with $\mathcal{M}^{q,\mathcal{R}}$ that contains all the CQs of the form $(Q_w)^T$ in bijection with the Q_w in $\mathcal{Q}_i$. By definition, $\mathcal{Q}_j$ is a finite rewriting of $(\mathcal{Q}^{q,\mathcal{R}},\mathcal{M}^{q,\mathcal{R}})$ and the subset $\mathcal{Q}_j^\mathcal{S}$ of $\mathcal{Q}_j$ that contains only the CQs on $\mathcal{S}$ is a finite $\mathcal{S}$-rewriting of $(\mathcal{Q}^{q,\mathcal{R}},\mathcal{M}^{q,\mathcal{R}})$. Now, assume $\mathcal{Q}_j^\mathcal{S}$ is not complete, that is, there is a CQ that belongs to an $\mathcal{S}$-rewriting of $(\mathcal{Q}^{q,\mathcal{R}},\mathcal{M}^{q,\mathcal{R}})$ but that is not more specific than a CQ in $\mathcal{Q}_j^\mathcal{S}$; by Lemma 2, such CQ is of the form $(Q_\mathcal{P})^T$. Then there is a piece-rewriting $\mathcal{Q}'_j$ of $\mathcal{Q}^{q,\mathcal{R}}$ with $\mathcal{M}^{q,\mathcal{R}}$ that contains a CQ entailed by $(Q_\mathcal{P})^T$; hence such CQ is also on $\mathcal{S}$, and by Lemma 2 it is of the form $(Q'_\mathcal{P})^T$. By Lemma 3, $Q'_\mathcal{P}$ belongs to a piece-rewriting of $\{q\}$ with $\mathcal{R}^\star$. Since $\mathcal{R}^\star \equiv \mathcal{R}$, there is a CQ equivalent to $Q'_\mathcal{P}$ in some rewriting of $(\{Q\},\mathcal{R})$. Since $\mathcal{Q}_i$ is complete, there is $Q_c \in \mathcal{Q}_i$ such that $Q'_\mathcal{P} \models Q_c$. Hence, $(Q'_\mathcal{P})^T \models (Q_c)^T$, so $(Q_\mathcal{P})^T \models (Q_c)^T$; by Lemma 1, $(Q_c)^T \in \mathcal{Q}_j$, hence $(Q_c)^T \in \mathcal{Q}_j^\mathcal{S}$, which contradicts the fact that $(Q_\mathcal{P})^T$ is not more specific than a CQ in $\mathcal{Q}_j^\mathcal{S}$. ■

III.5.2 Disjunctive mapping rewriting / chase operators

In Chapter V, we will need to rewrite (U)CQs through a disjunctive mapping and do the chase with such mappings. Therefore, we define the following operators.

Chasing through a disjunctive mapping. The disjunctive mapping chase is a natural extension of the (conjunctive) mapping chase (Definition I.41).

Definition III.14 (Disjunctive Mapping Chase)
Let I be an instance over $V_\mathcal{S}$ and $\mathcal{M}$ be a disjunctive mapping from $V_\mathcal{S}$ to $V_\mathcal{O}$. The *disjunctive mapping chase* of I by $\mathcal{M}$, denoted as $\mathcal{M}\text{–}\mathsf{chase}_\vee(I)$, is defined as follows: for each instance J in $\mathsf{chase}_\vee(I,\mathcal{M})$, we remove all atoms of the initial instance I from J. Formally,

$$\mathcal{M}\text{–}\mathsf{chase}_\vee(I) = \{J \setminus I \mid J \in \mathsf{chase}_\vee(I,\mathcal{M})\}$$

As for the mapping chase, the atoms of the initial instance I are removed from every instance produced by the chase. This removal does not alter the answers to queries defined over $V_{\mathcal{O}}$.

The result of the disjunctive mapping chase is always finite. This holds because the disjunctive mapping consists of nonrecursive rules.

Rewriting through a disjunctive mapping. The disjunctive mapping rewriting operator $\mathcal{M}$-rewriting$_\vee$ is also a direct extension of the standard mapping rewriting as in Definition I.42.

Definition III.15 (Disjunctive mapping rewriting)
Let $\mathcal{Q}$ be a union of conjunctive queries over $V_{\mathcal{S}}$ and $\mathcal{M}$ be a disjunctive mapping from $V_{\mathcal{S}}$ to $V_{\mathcal{O}}$. The *disjunctive mapping rewriting* of $\mathcal{Q}$ by $\mathcal{M}$, denoted as $\mathcal{M}$-rewriting$_\vee(\mathcal{Q})$, is defined by applying the disjunctive rewriting function (Definition III.13) and then removing all conjunctive queries that contain atoms with predicates over $V_{\mathcal{O}}$. Formally,

$$\mathcal{M}\text{-rewriting}_\vee(\mathcal{Q}) = \{q \mid q \in \texttt{rewriting}_\vee(\mathcal{Q}, \mathcal{M}), \text{ and } q \text{ is over } V_{\mathcal{S}}\}$$

This process produces a set of queries that only include the new conjunctive queries that were generated over $V_{\mathcal{S}}$ during the rewriting.

Unlike the standard mapping rewriting, this operator is not guaranteed to produce a finite result, and it remains an open problem to find an algorithm that terminates when there exists a (finite) UCQ that is a $\mathcal{M}$-rewriting$_\vee$.

In addition to the basic form of the operator, there are two optional parameters that can be used to control the rewriting process. The first optional parameter is the query inclusion operator, allowing for the computation of a cover for the result that takes into account either a specific mapping $\mathcal{M}$ with $\sqsubseteq_{\mathcal{M}}$ or a KBDM specification Σ with $\sqsubseteq_\Sigma$.

Definition III.16 (Disjunctive mapping rewriting with special cover)
Let $\mathcal{Q}$ be a union of conjunctive queries over $V_{\mathcal{S}}$ and $\mathcal{M}$ be a disjunctive mapping from $V_{\mathcal{S}}$ to $V_{\mathcal{O}}$. The *disjunctive mapping rewriting of $\mathcal{Q}$ by $\mathcal{M}$ provided with inclusion operator* $\sqsubseteq_X$, where $X \in \{\mathcal{M}, \Sigma\}$, denoted $\mathcal{M}_\vee$-rewriting$_\vee^{\sqsubseteq_X}(\mathcal{Q})$, is an inclusion-minimal subset of $\mathcal{M}_\vee$-rewriting$_\vee(\mathcal{Q})$ such that:

1. For any q in $\mathcal{M}_\vee$-rewriting$_\vee(\mathcal{Q})$, there exists q' in $\mathcal{M}_\vee$-rewriting$_\vee^{\sqsubseteq_X}(\mathcal{Q})$ such that $q' \sqsubseteq_X q$,
2. All elements of $\mathcal{M}_\vee$-rewriting$_\vee^{\sqsubseteq_X}(\mathcal{Q})$ are pairwise incomparable with respect to $\sqsubseteq_X$.

This operator will be useful in Chapter V to compute the maximally sound $\mathcal{M}$-translation of a UCQ (see Theorem V.25).

The second optional parameter limits the number of steps of rewriting steps, providing an approximation of the rewriting. Thus, the result is guaranteed to be a UCQ, although it might not be a complete rewriting. This is denoted as $\mathcal{M}_\vee$-rewriting$_\vee(\mathcal{Q}, k)$, where k is the limit on the number of steps.

Finally, we can combine a special cover with a finite number of steps, which we denote $\mathcal{M}_\vee$-rewriting$_\vee^{\sqsubseteq_X}(\mathcal{Q}, k)$.

III.5.3 Existence of a non-empty rewriting

We have shown that one cannot decide if a UCQ has a *finite* sound and complete rewriting through a mapping. We now show that we can however decide if a UCQ has a *non-empty* sound and complete rewriting. This is a corollary of Theorem III.25.

We recall that an empty UCQ is trivially considered a sound rewriting for any UCQ. There exist certain UCQs for which the only sound rewriting is the empty UCQ, as illustrated in Example III.24.

Example III.24

Let $\mathcal{M} = \{m = p(x) \rightarrow r(x) \vee s(x)\}$ be a disjunctive mapping and $\mathcal{Q}_\mathcal{O} = \exists u\ s(u)$ be a UCQ. There is no disjunctive piece-unifier between m and $\mathcal{Q}_\mathcal{O}$. Which implies that $\mathcal{M}$-rewriting$_\vee(\mathcal{Q}_\mathcal{O})$ is empty, and thus, $\mathcal{Q}_\mathcal{O}$ has no answer for every instance over $\mathcal{S}$.

As we will later in Theorem V.22 in Chapter V, Theorem III.25 will also be useful to know whether there exists a nontrivial maximally sound $\mathcal{M}$-translation.

Before stating this theorem, we need to define the notion of the critical instance on a vocabulary.

Definition III.17 (Critical instance of $\mathcal{S}$)
Let $\mathcal{S}$ be a database schema. The *critical instance* of $\mathcal{S}$ over a nonempty set of constants $\mathcal{C}$ is defined as $I_\mathcal{C} = \{p(\mathbf{c}) \mid p \in \text{predicates}(\mathcal{S})$ of arity k and $\mathbf{c} \in \mathcal{C}^k\}$.

Theorem III.25

Let $\mathcal{M}$ be a disjunctive mapping, $\mathcal{Q}_\mathcal{T}$ be a Boolean UCQ and $\mathcal{C}$ be the set of all constants that appear in $\mathcal{M}$ and $\mathcal{Q}_\mathcal{T}$ or $\mathcal{C} = \{c\}$, with c that is a constant, if they contain no constant.
Then, there is a non-empty $\mathcal{S}$-rewriting of $\mathcal{Q}_\mathcal{T}$ through $\mathcal{M}$ if and only if $\mathcal{M}$–chase$_\vee(I_\mathcal{C}) \models \mathcal{Q}_\mathcal{T}$ where $I_\mathcal{C}$ is the critical instance of $\mathcal{S}$ over $\mathcal{C}$.

To prove the theorem, we use the following lemma.

Lemma III.26

Let $q_\mathcal{S}$ be a CQ over $\mathcal{S}$, and a nonempty set of constants $\mathcal{C}$. Then, $q_\mathcal{S}$ can be sent by homomorphism into the critical instance $I_\mathcal{C}$ of $\mathcal{S}$.

Proof. We just have to build a homomorphism h from $q_\mathcal{S}$ to $I_\mathcal{C}$ that maps the variables of $q_\mathcal{S}$ on any constant of $\mathcal{C}$. ■

We recall that the following assertions are equivalent for any Boolean UCQ q:

- $I, \mathcal{M} \models \mathcal{Q}$;

- $\mathcal{M}\text{–chase}_\vee(I) \models \mathcal{Q}$;
- $I \models \mathcal{M}\text{-rewriting}_\vee(\mathcal{Q})$.

Proof of theorem III.25. **(⇒)** Let $q_\mathcal{S} \in \mathcal{M}\text{-rewriting}_\vee(\mathcal{Q}_\mathcal{T})$ be a BCQ. By the Lemma III.26, $I_\mathcal{C} \models q_\mathcal{S}$. Since $I_\mathcal{C} \models q_\mathcal{S}$, we have $I_\mathcal{C} \models \mathcal{M}\text{-rewriting}_\vee(\mathcal{Q}_\mathcal{T})$ and so, we also have $\mathcal{M}\text{–chase}_\vee(I_\mathcal{C}) \models \mathcal{Q}_\mathcal{T}$.

(⇐) Since $\mathcal{M}\text{–chase}_\vee(I_\mathcal{C}) \models \mathcal{Q}_\mathcal{T}$, we have $I_\mathcal{C} \models \mathcal{M}\text{-rewriting}_\vee(\mathcal{Q}_\mathcal{T})$, which implies that $\mathcal{M}\text{-rewriting}_\vee(\mathcal{Q}_\mathcal{T})$ cannot be empty. Indeed, $I_\mathcal{C} \models \mathcal{M}\text{-rewriting}_\vee(\mathcal{Q}_\mathcal{T})$ implies that there exists a CQ $q \in \mathcal{M}\text{-rewriting}_\vee(\mathcal{Q}_\mathcal{T})$ such that $I_\mathcal{C} \models q$ which cannot be the case if it is empty. ■

III.6 Summary

In this chapter, we focused on the issue of rewriting a UCQ into a UCQ with respect to disjunctive existential rules and disjunctive mappings. Our main contributions are the following:

- We first define a sound and complete query rewriting operator which has the advantage of establishing a tight relationship between a chase step and a rewriting step (Theorem III.14). The associated breadth-first query rewriting algorithm outputs a minimal UCQ-rewriting when one exists (Theorem III.16).
- We then show that the notion of UCQ-rewritable (or: FUS) ruleset seems to have little relevance for disjunctive rules. Indeed, we show that, for *any* "truly disjunctive" s-to-o rule, there is a CQ that is not UCQ-rewritable (Theorem III.20). Hence, studying the problem of whether a pair $(\mathcal{Q},\mathcal{R})$ is UCQ-rewritable seems more promising. We point out some cases of UCQ-rewritable pairs.
- Finally, considering (disjunctive) mappings, we show that the problem of determining whether a given UCQ on the target vocabulary admits a UCQ-rewriting on the source vocabulary is undecidable (Theorem III.23). However, determining whether a UCQ on the target vocabulary admits a non-empty sound rewriting on the source vocabulary is decidable (Theorem III.25).

This work leaves open a number of problems that we will list in the concluding chapter.

IV - Query translation: related background

In this chapter, we present notions about and related to the translation of queries. In Section IV.1 we present a slight extension of UCQs (in particular, studied by [Arenas et al., 2010, Pérez, 2011]), which is the query language (for both source and target queries) we use in this chapter and the next one (Chapter V). Section IV.2 reviews related work on $\mathcal{S}$-to-$\mathcal{O}$-translation. Finally, in Section IV.3, we introduce the fundamental notion of maximum recovery, imported from data exchange.

IV.1 Slight extensions of UCQs

In the following, we will consider extensions of UCQs / mappings obtained by adding some special predicates, as in [Arenas et al., 2010, Pérez, 2011]. This will allow us to obtain more powerful translations in the context of $\mathcal{S}$-to-$\mathcal{O}$-translation, in the sense that we can achieve more faithful translations than by just considering nonextended UCQs.

These special predicates are the following: [1]

- $\mathbf{C}(\cdot)$: to express that the argument is a constant.
- $\cdot \neq \cdot$ (inequality predicate): to express that the two arguments are distinct entities (i.e., $\neq$ is the negation of equality).

We furthermore denote by $\mathbf{C}[\mathbf{x}]$ the conjunction $\bigwedge_{x_i \in \mathbf{x}} \mathbf{C}(x_i)$ and by $\top[\mathbf{x}]$ the conjunction $\bigwedge_{x_i \in \mathbf{x}} \top(x_i)$.

These special predicates are more precisely interpreted as follows. Let $\mathcal{I} = (\Delta, .^{\mathcal{I}})$ be any interpretation of a logical language with set of constants $\mathcal{C}$. Then:

- $\mathbf{C}^{\mathcal{I}} = \mathcal{C}$.
- $\neq^{\mathcal{I}} = \{(d_1, d_2) \in \Delta^2 \mid d_1 \neq d_2\}$.

We will consider the two following extensions of the UCQ class:

- (U)CQ$^{\mathbf{C}}$: the class (U)CQ provided with the $\mathbf{C}$ predicate, which may occur in any term that also occurs in a standard atom. Note that for any constant a, $\mathbf{C}(a)$ is true in any interpretation. Also, adding to a query the atom $\mathbf{C}(x)$ for an answer variable x does not change its semantics since an answer variable has to be mapped to a constant.
- (U)CQ$^{\neq,\mathbf{C}}$ (resp. (U)CQ$^{\neq}$): the class (U)CQ$^{\mathbf{C}}$ (resp. (U)CQ) provided with the $\neq$ predicate, such that the arguments of $\neq$ are either constants, answer variables or variables that occur in a $\mathbf{C}$-atom. In other words, the arguments of $\neq$ are necessarily mapped to constants in a homomorphism from a CQ to a database instance

[1] Besides, we will refer to the special unary predicate $\top(\cdot)$ when reviewing related work in description logics. In any interpretation $\mathcal{I} = (\Delta, .^{\mathcal{I}})$, $\top^{\mathcal{I}} = \Delta$.

(and, more generally, considering a non-ground instance, mapped to terms that are known to be constants, which also includes variables occurring in a **C**-atom).

Note that $\neq$ may make a query inconsistent if the query contains an atom of the form $t \neq t$ or if it contains equalities that conflict with the inequalities. More precisely, we say that a $\text{UCQ}^{\neq,\mathbf{C}}$ is consistent if there is an instantiation of its answer variables by constants that yields a satisfiable Boolean query, otherwise we say that it is inconsistent. Next, we implicitely assume that queries are consistent.

When needed, to distinguish between the different kinds of atoms in a conjunction (or set) of atoms, we write it as $\phi \wedge \mathbf{C} \wedge \delta \wedge \eta$, where:

- $\mathbf{C}$ denotes the subset of **C**-atoms,
- δ denotes the subset of inequalities, and
- η denotes for the subset of equalities,
- ϕ denotes the subset of other atoms (i.e., standard atoms).

Given a set of atoms or $\text{CQ}^{\neq,\mathbf{C}}$ q, we denote by $\mathbf{C}(q)$ its **C**-atoms, $\delta(q)$ its inequalities, $\eta(q)$ its equalities, and $\phi(q)$ its other atoms.

The notion of *homomorphism* is naturally extended to take the predicates $\mathbf{C}$ and $\neq$ into account. We recall we assume that, when dealing with atomsets that may contain equalities, we implicitly perform the substitutions associated with equalities before seeking a homomorphism. Given a $\text{CQ}^{\neq,\mathbf{C}}$ q_1 and an instance I, a homomorphism from q_1 to I is a homomorphism from $\phi(q_1)$ to I such that:

- for all atom $\mathbf{C}(t) \in q_1$, $h(t)$ is a constant;
- for all atom $t_1 \neq t_2 \in q_1$, $h(t_1)$ and $h(t_2)$ are distinct constants.

In other words, in a homomorphism, special predicates $\mathbf{C}$ and $\neq$ are handled like standard predicates if we assume that implicit atoms in I are made explicit, that is, for any constant a, there is an atom $\mathbf{C}(a)$ and for any pair of distinct constants (a_1, a_2), there is an atom $a_1 \neq a_2$. Note however that, in the following, special predicates never occur explicitely in an instance.

The notion of query homomorphism is extended similarly, taking into account that $\mathbf{C}$ and $\neq$ may also occur in the target query: given $\text{CQ}^{\neq,\mathbf{C}}$ q_1 and q_2, a *query homomorphism* from q_1 to q_2 is a a query homomorphism from $\phi(q_1)$ to $\phi(q_2)$ such that:

- for all atom $\mathbf{C}(t) \in q_1$, $h(\mathbf{C}(t)) \in q_2$ or $h(t)$ is a constant;
- for all atom $t_1 \neq t_2 \in q_1$, $h(t_1 \neq t_2) \in q_2$, or $h(t_1)$ and $h(t_2)$ are distinct constants.

Now, the question is whether logical entailment can still be based on homomorphism. It is not difficult to check that the addition of the **C**-predicate does not make any difference: for any Boolean $CQ^{\mathbf{C}}$ q and instance (or Boolean $CQ^{\mathbf{C}}$) I, there is a homomorphism from q to I if and only if $I \models q$. In constrast, it is well-known that inequalities generally make reasoning more complex. However, we consider here a constrained form of inequalities. Actually, we have to distinguish two cases:

- **Homomorphism from a (Boolean) CQ$^{\neq,\mathbf{C}}$ q to an instance I.** If I is a ground set of atoms (e.g., I is a database instance) and I is a model of q, for any atom $(t_1 \neq t_2)$, t_1 and t_2 are assigned to distinct domain elements, which are necessarily constants, hence there is a homomorphism from q to I. If I may contain variables (e.g., I is an ontological instance obtained by a GLAV mapping), let us consider its isomorphic model $M = (\Delta, .^M)$. If $M \models q$, there is an assignment of the variables in q to Δ that satisfies the atoms in q. In particular, for an inequality atom $(t_1 \neq t_2)$, t_1 and t_2 are assigned to distinct elements of Δ, and each t_i $(i = 1, 2)$ is either a constant or a variable necessarily assigned to a constant from Δ because of the atom $\mathbf{C}(t_i)$. Hence, the assignment from q to Δ defines a homomorphism from q to I. To sum up, for any instance I, if $I \models q$ then there is a homomorphism from q to I (and reciprocally).
- **Homomorphism within (Boolean) CQ$^{\neq,\mathbf{C}}$**: then, entailment does not ensure the existence of a homomorphism as witnessed by the following example.

Example IV.1: Homomorphism is not complete for CQ$^{\neq,\mathbf{C}}$

Consider the following CQ$^{\neq,\mathbf{C}}$ queries:
$q_1 = \exists u, v.\ p(u, v) \wedge \mathbf{C}[u, v] \wedge u \neq v$
$q_2 = \exists x, y, z.\ p(x, y) \wedge p(x, z) \wedge \mathbf{C}[x, y, z] \wedge y \neq z.$
Note that all the variables occur in a $\mathbf{C}$-atom, which amounts to allow for inequalities between any (distinct) terms. There is no homomorphism from q_1 to q_2, however $q_2 \sqsubseteq q_1$. Indeed, for any database instance I that answers q_2, either x and y are mapped to distinct constants, and I answers q_1, or x and y are mapped to the same constant, in which case x and z are necessarily mapped to distinct constants (because of the atom $y \neq z$), and I again answers q_1.

In fact, it is known that query containment of (Boolean) CQs provided with inequalities (that may occur anywhere) is at the second level of the polynomial hierarchy (precisely, Π_2^p-complete) [van der Meyden, 1997]. The constraints we enforce in the class CQ$^{\neq,\mathbf{C}}$ do not simplify the problem because one can add a $\mathbf{C}$-atom on any term in the queries without changing query inclusion (we remind that database instance are assumed to be ground), and then the constraints on $\neq$ are satisfied, as illustrated by Example IV.1.

IV.2 $\mathcal{S}$-to-$\mathcal{O}$-translation of queries: State of the art

In the landscape of knowledge-based data management, the issue of target (query) translation has long been overshadowed by the more explored terrain of $\mathcal{O}$-to-$\mathcal{S}$-translation. While substantial literature has been dedicated to unraveling the complexities of $\mathcal{O}$-to-$\mathcal{S}$-translation, $\mathcal{S}$-to-$\mathcal{O}$-translation has remained a relatively peripheral area of study. The first glimpses into this subject can be traced back to the works of Arenas et al. [Arenas et al., 2010] and Perez [Pérez, 2011]. However, these pioneering

studies were not primarily focused on $\mathcal{S}$-to-$\mathcal{O}$-translation and did not provide formal algorithms for this purpose. The domain remained largely uncharted until the arrival of more concentrated efforts, such as those found in the works of Lutz et al. [Lutz et al., 2018], Lenzerini et al. [Lenzerini, 2019] and Cima [Cima, 2020]. These contributions mark the beginning of a focused study on $\mathcal{S}$-to-$\mathcal{O}$-translation.

IV.2.1 $\mathcal{M}$-translation

In [Arenas et al., 2010], an algorithm named *TargetRewriting* computes the perfect UCQ$^{\neq,\mathbf{C}}$-to-UCQ$^{\neq,\mathbf{C}}$ $\mathcal{M}$-translation when there exists one. However, this algorithm was not precisely defined. Indeed, the previous work only mentioned that following a proof of one of its theorems, there exists an algorithm that is then used has a black box in the paper. Since the proof is not provided in this work, we have to rely on Pérez' Phd thesis dissertation [Pérez, 2011] that contains a description of the algorithm in several parts mixed in the proofs of lemmas 6.2.3, 6.2.6 and A.2.3. We do not give details about the algorithm here: the main steps of the algorithm are given in the appendix (section D.1) and for more details, the reader can read the lemmas from [Pérez, 2011] .

[Arenas et al., 2010, Pérez, 2011] also give us an important property: the class of queries UCQ$^{\neq,\mathbf{C}}$ is expressive enough to express any perfect $\mathcal{M}$-translation of a query in UCQ$^{\neq,\mathbf{C}}$.

Proposition IV.2: [Arenas et al., 2010, Pérez, 2011]

Let $\mathcal{M}$ be a mapping specified by a set of rules where inequalities are allowed in the body, and $\mathcal{Q}$ be a UCQ$^{\neq,\mathbf{C}}$ that has a perfect $\mathcal{M}$-translation in some query language. Then, there exists a query in UCQ$^{\neq,\mathbf{C}}$ that is a perfect $\mathcal{M}$-translation of $\mathcal{Q}$.

IV.2.2 Σ-translation

In the existing literature on $\mathcal{S}$-to-$\mathcal{O}$-translation, the primary emphasis has been placed on the context of Ontology-Based Data Access (OBDA). As for $\mathcal{M}$-translation, a perfect Σ-translation may not always be attainable, since a Σ-translation with an empty ontology is the same problem as the $\mathcal{M}$-translation. Thus, [Lutz et al., 2018], one of the first works on Σ-translation, studies two reasoning problems: checking if a query is a perfect Σ-translation (Problem IV.3) and the problem of expressibility, that is, checking if there exists a perfect Σ-translation (Problem IV.4).

Problem IV.3: Verification problem from [Lutz et al., 2018]

Input: OBDA specification $\Sigma = (V_\mathcal{O}, \mathcal{R}_\mathcal{O}, V_\mathcal{S}, \mathcal{M})$, a query $\mathcal{Q}_\mathcal{S}$ over $V_\mathcal{S}$, a query $\mathcal{Q}_\mathcal{O}$ over $V_\mathcal{O}$ of the same arity as $\mathcal{Q}_\mathcal{S}$.
Output: Is $\mathcal{Q}_\mathcal{O}$ a perfect UCQ-Σ-translation of $\mathcal{Q}_\mathcal{S}$?

Problem IV.4: Expressibility problem from [Lutz et al., 2018]

Input: OBDA specification $\Sigma = (V_\mathcal{O}, \mathcal{R}_\mathcal{O}, V_\mathcal{S}, \mathcal{M})$, a query $\mathcal{Q}_\mathcal{S}$ over $V_\mathcal{S}$.
Output: Is there a perfect UCQ-Σ-translation of $\mathcal{Q}_\mathcal{S}$?

The study in [Lutz et al., 2018] considers cases where the query class is UCQ and the mapping is GAV, both restricted to unary and binary predicates. It is shown that the complexity of both problems is Π_2^p-complete for the main dialects of the DL-LITE family, CoNExpTime-complete between $\mathcal{EL}$ and $\mathcal{ELHI}$ when source queries are rooted (every variable is reachable from an answer variable in the query graph of every CQ), and 2ExpTime-complete for unrestricted source queries.

[Cima et al., 2019, Cima et al., 2020, Cima et al., 2022] study how to practically compute Σ-translations.

Minimally complete Σ-translation. The first algorithm proposed only performs a $\mathcal{M}$-chase of each CQ to obtain a new query that is a minimally complete UCQ: see Algorithm 3 from [Cima et al., 2019]. The algorithm computes a UCQ-to-UCQ minimally complete Σ-translation considering an OBDA framework in which the ontology is in DL-LITE$_{RDFS}$ and it is assumed that we add rules of the form $p(x_1,\ldots,x_n) \rightarrow \top(x_1) \wedge \ldots \wedge \top(x_n)$ for each predicate $p \in V_\mathcal{S}$ to the mapping. DL-LITE$_{RDFS}$ can be seen as a restriction of DL-LITE$_\mathcal{R}$ which amounts to forbid existential rules in the rule heads - more specifically, the ontology contains only rules of the form $a(x) \rightarrow b(x)$, $p(x,y) \rightarrow a(x)$, $p(x,y) \rightarrow b(x)$, $p(x,y) \rightarrow q(x,y)$ and $p(x,y) \rightarrow q(y,x)$.

Algorithm 3: minimally complete UCQ-Σ-translation [Cima et al., 2019]

Input: A KBDM specification $\Sigma = (V_\mathcal{O}, \mathcal{R}_\mathcal{O}, V_\mathcal{S}, \mathcal{M})$ where $\mathcal{R}_\mathcal{O}$ is in DL-LITE$_{RDFS}$ and $\mathcal{M}$ also contains rules of the form $p(\mathbf{x}) \rightarrow \top[\mathbf{x}]$ for each predicate $p \in V_\mathcal{S}$, a UCQ $\mathcal{Q}_\mathcal{S}(\mathbf{x}) = q^1_\mathcal{S}(\mathbf{x}) \vee \ldots \vee q^n_\mathcal{S}(\mathbf{x})$ over $V_\mathcal{S}$
Output: A UCQ $\mathcal{Q}_\mathcal{O}(\mathbf{x})$ over $V_\mathcal{O}$
$\mathcal{Q}_\mathcal{O}(\mathbf{x}) \leftarrow \bigvee_{i=1}^{n} \{\mathcal{M}\text{-chase}(q^i_\mathcal{S}) \mid 1 \leq i \leq n\}$;
return $\mathcal{Q}_\mathcal{O}(\mathbf{x})$;

We can notice in Algorithm 3 that only the mapping is used, the ontology does not play any role.

Proposition IV.5: from [Cima et al., 2019]

Given Algorithm 3, the following properties hold:

1. The algorithm computes a minimally complete UCQ-Σ-translation of $\mathcal{Q}_\mathcal{S}$.
2. A minimally complete UCQ-Σ-translation of $\mathcal{Q}_\mathcal{S}$ always exists if we add to $\mathcal{M}$ the special rules $p(x_1,\ldots,x_n) \rightarrow \top(x_1) \wedge \ldots \wedge \top(x_n)$ for each predicate $p \in V_\mathcal{S}$.
3. If $\mathcal{Q}_\mathcal{S}$ is a CQ, then it remains a CQ.

4. The complexity of Algorithm 3 does not depend on $\mathcal{R}_\mathcal{O}$ and is in PTIME in the size of $\mathcal{Q}_\mathcal{S}$, and in EXPTIME in the size of $\mathcal{M}$, as it essentially applies the chase.

5. An algorithm for computing a minimally complete UCQ-Σ-translations that is in PTIME in the size of all inputs would imply a PTIME algorithm for CQ containment. Thus, assuming PTIME $\neq$ NP, the computation problem cannot be solved in PTIME.

Let us make comments about these results.

Remark IV.1

1. The point 1 in Proposition IV.5 does not imply that we compute the minimally complete Σ-translation of $\mathcal{Q}_\mathcal{S}$. It is minimally complete when the target class is UCQ, otherwise we can have a translation in $\text{UCQ}^{\neq,\mathbf{C}}$ that is more minimal (see Example IV.6).

2. The point 2 in Proposition IV.5 assumes that some special rules are added to $\mathcal{M}$. In this dissertation, we do not make this assumption because such rules amount to transfer all the data values at the ontological level, which does not seem in line with the idea of selecting only relevant data. Hence, we will see later in Section V.2.1 that a complete translation does not always exist when we consider any mapping, and we will characterize when it exists.

Example IV.6: Result of Algorithm 3 is not minimally complete

Let $V_\mathcal{S} = \{A(\cdot,\cdot), B(\cdot)\}$ and $V_\mathcal{O} = \{S(\cdot,\cdot), U(\cdot)\}$, and consider the mapping $\mathcal{M}$ from $V_\mathcal{S}$ to $V_\mathcal{O}$ such that:

$$\mathcal{M} = \begin{cases} A(x,y) & \rightarrow S(x,y), \\ B(x) & \rightarrow S(x,x), \\ A(x,x) & \rightarrow U(x). \end{cases}$$

Let $\mathcal{Q}_\mathcal{S} = \exists x,y.A(x,y)$. Algorithm 3 returns $\mathcal{Q}_\mathcal{O} = \exists x,y.S(x,y)$. $\mathcal{Q}_\mathcal{O}$ is a UCQ-minimally complete Σ-translation of $\mathcal{Q}_\mathcal{S}$. It is not perfect because it is not sound (taking the database $D = \{B(a)\}$, we have $I_{(D,\mathcal{M})} = \{S(a,a)\}$, and thus $D \not\models \mathcal{Q}_\mathcal{S}$ but $I_{(D,\mathcal{M})} \models \mathcal{Q}_\mathcal{O}$).
However, it can be shown that the $\text{UCQ}^{\neq,\mathbf{C}}$ $\mathcal{Q} = (\exists x,y.S(x,y) \wedge x \neq y \wedge \mathbf{C}(x) \wedge \mathbf{C}(y)) \vee (\exists x.U(x))$ is a perfect Σ-translation of $\mathcal{Q}_\mathcal{S}$. Thus, since $\mathcal{Q}$ is perfect, it is more minimal than $\mathcal{Q}_\mathcal{O}$ with respect to completeness.

This example comes from example 32 in [Pérez, 2013].

A variant of Algorithm 3 is proposed in [Cima et al., 2020], with the aim of computing a complete Σ-translation that is more minimal than with UCQs. It uses a more

expressive target query class, namely EQL-LITE(UCQ) (see [Calvanese et al., 2007a] for more information on this query language). The principle of the algorithm is similar, as it is also based on the chase. But in addition, the epistemic modal operator **K** surrounds the UCQ that would be produced in Algorithm 3. Intuitively, **K** checks if "a fact is known to be true". Its main role, in this context, is to ensure that the variables can only be mapped to constants. This can also be done with the special predicate **C**, so with much less expressive language. In the context of their work, this allows one to filter out some answers that we couldn't have on a database which contains only constants. Example IV.7 illustrates this algorithm.

Example IV.7

Take again the mapping and the CQs from Example IV.6:
Let $V_{\mathcal{S}} = \{A(\cdot,\cdot), B(\cdot)\}$ and $V_{\mathcal{O}} = \{S(\cdot,\cdot), U(\cdot)\}$, and consider the mapping $\mathcal{M}$ from $V_{\mathcal{S}}$ to $V_{\mathcal{O}}$ such that:

$$\mathcal{M} = \begin{cases} A(x,y) & \rightarrow S(x,y), \\ B(x) & \rightarrow S(x,x), \\ A(x,x) & \rightarrow U(x). \end{cases}$$

Let $\mathcal{Q}_{\mathcal{S}} = \exists x, y.A(x,y)$. We have a specification $\Sigma = (V_{\mathcal{O}}, \emptyset, V_{\mathcal{S}}, \mathcal{M})$.
Then, a minimally complete EQL-LITE-Σ-translation, as described in [Cima et al., 2020], is $\mathcal{Q}_{\mathcal{O}} = \exists x, y.\mathbf{K}(S(x,y))$.

Another work ([Cima et al., 2022]) proposes another variant of the two previous algorithms, whose target class is non-recursive Datalog rules with the epistemic operator **K** and $\neq$-atoms in the body (we denote that by *non-recursive Datalog*$_{\mathbf{K},\neq}$). Algorithm 4 takes all the "specializations" of the CQs in the input (that is, we do all the possible partitioning of the variables of the CQs and create new CQs for each partition, in which we add inequalities atoms between each pair of variables that are not in the same class of the partition) and then do a $\mathcal{M}$-chase on the resulting CQs. Finally, the operator **K** is used to surround the result that is then transformed into a Datalog program (the CQs become the body of the Datalog rules, and a target predicate "Ans" is put on the head). The result of this algorithm is proved to be a minimally complete monotone-Σ-translation (that is, there exists no other monotone query that is strictly more minimal with respect to completeness). Example IV.8 illustrates this algorithm.

Example IV.8

Take again the mapping and the CQs from Example IV.6:
Let $V_{\mathcal{S}} = \{A(\cdot,\cdot), B(\cdot)\}$ and $V_{\mathcal{O}} = \{S(\cdot,\cdot), U(\cdot)\}$, and consider the mapping $\mathcal{M}$ from $V_{\mathcal{S}}$

to $V_{\mathcal{O}}$ such that:

$$\mathcal{M} = \begin{cases} A(x,y) & \rightarrow S(x,y), \\ B(x) & \rightarrow S(x,x), \\ A(x,x) & \rightarrow U(x). \end{cases}$$

Let $\mathcal{Q}_{\mathcal{S}} = \exists x, y.A(x,y)$. We have a specification $\Sigma = (V_{\mathcal{O}}, \emptyset, V_{\mathcal{S}}, \mathcal{M})$.
Then, a minimally complete monotone-Σ-translation, as described in [Cima et al., 2022], is:

$$\mathcal{Q}_{\mathcal{O}} = \begin{cases} \mathbf{K}(S(x,y) \wedge x \neq y) & \rightarrow Ans(), \\ \mathbf{K}(U(x)) & \rightarrow Ans() \end{cases}$$

Algorithm 4: Non-recursive Datalog$_{\mathbf{K},\neq}$-Minimally Complete Σ-translation [Cima et al., 2022]

Input: OBDM specification $\Sigma = (V_{\mathcal{O}}, \mathcal{R}_{\mathcal{O}}, V_{\mathcal{S}}, \mathcal{M})$ with $\mathcal{R}_{\mathcal{O}}$ a DL-LITE$_{RDFS}$ ontology and $\mathcal{M}$ also contains rules of the form $p(\mathbf{x}) \rightarrow \top[\mathbf{x}]$ for each predicate $p \in V_{\mathcal{S}}$, UCQ $\mathcal{Q}_{\mathcal{S}}$ over $V_{\mathcal{S}}$
Output: Non-recursive Datalog$_{\mathbf{K},\neq}$-Minimally Complete Σ-translation of $\mathcal{Q}_{\mathcal{S}}$
foreach $q_i = \{\mathbf{x} \mid \exists \mathbf{y}.\phi[\mathbf{x},\mathbf{y}] \wedge \delta[\mathbf{x},\mathbf{y}]\}$ *in* SaturateQ($\mathcal{Q}_{\mathcal{S}}$) **do**
  $r_{q_i} \leftarrow \mathbf{K}(\exists \mathbf{z}.\top[\mathbf{x}] \wedge \mathcal{M}\text{−chase}(\phi[\mathbf{x},\mathbf{y}]) \wedge \delta[\mathbf{x},\mathbf{y}']) \rightarrow \text{Ans}(\mathbf{x})$;
end
return $\{r_{q_1}, \ldots, r_{q_n}\}$;

function SaturateQ($\mathcal{Q}_{\mathcal{S}}$);
$\mathcal{Q}'_{\mathcal{S}} \leftarrow \emptyset$;
foreach *disjunct* $q_{\mathcal{S}} = \exists \mathbf{y}.\phi[\mathbf{x},\mathbf{y}]$ *of* $\mathcal{Q}_{\mathcal{S}}$ **do**
  foreach *unifier* μ *on* $\mathbf{x} \cup \mathbf{y}$ *such that* $\mu(x) \in \mathbf{x}$ *for each* $x \in \mathbf{x}$ **do**
    $q'_{\mathcal{S}} \leftarrow \mu(q_{\mathcal{S}})$;
    foreach *pair of distinct variables* t_1, t_2 *in* $q'_{\mathcal{S}}$ **do**
      $q'_{\mathcal{S}} \leftarrow q'_{\mathcal{S}} \cup \{t_1 \neq t_2\}$;
    end
    $\mathcal{Q}'_{\mathcal{S}} \leftarrow \mathcal{Q}'_{\mathcal{S}} \cup \{q'_{\mathcal{S}}\}$;
  end
end
return $\mathcal{Q}'_{\mathcal{S}}$;

Maximally sound Σ-translation. In the study of Σ-translations of UCQs, a surprising finding was obtained: a maximally sound UCQ-Σ-translation does not always exist, even when considering an empty ontology. [Cima et al., 2019] gives a case where there is always a maximally sound UCQ-Σ-translation as described in Proposition IV.9. Proposition IV.10 gives cases where there is no maximally sound UCQ-Σ-translation.

To see an illustration, read Example V.27 in Section V.2.3.

Before introducing the two next propositions, we need to define the notions of pure GAV mapping and UCQJFE. A GAV mapping is a pure GAV if the head of all its rules is composed of a (binary) atom without constant nor repeated variables (that is, an atom of the form $p(x)$ or $p(x,y)$). A UCQJFE is a union of CQJFEs (Conjunctive Queries with Join-Free Existential variables). A CQJFE is a CQ whose existential variables are not repeated more than once.

Proposition IV.9: from [Cima et al., 2019]

Let $\Sigma = (V_\mathcal{O}, \mathcal{R}_\mathcal{O}, V_\mathcal{S}, \mathcal{M})$ be a KBDM specification where the ontology is DL-LITE$_{RDFS}$, $\mathcal{M}$ is pure GAV and $\mathcal{Q}_\mathcal{S}$ is a UCQJFE over $V_\mathcal{S}$.
There is always a UCQ-maximally sound Σ-translation of $\mathcal{Q}_\mathcal{S}$, in which CQs have fewer atoms than the following bound:

$$\text{bound}(\mathcal{M}, \mathcal{Q}_\mathcal{S}) = \sum_{i=0}^{\text{size}(\mathcal{Q}_\mathcal{S})} |\mathcal{M}|^i,$$

where size($\mathcal{Q}_\mathcal{S}$) is the total number of atoms in $\mathcal{Q}_\mathcal{S}$, and $|\mathcal{M}|$ the total number of rules in $\mathcal{M}$.

In [Cima et al., 2019], an algorithm is proposed in the restricted setting introduced in Proposition IV.9. Algorithm 5 finds a translation using a generate-and-test approach: it generates all the possible CQs of a size less than bound($\mathcal{M}, \mathcal{Q}_\mathcal{S}$) and then checks if it is a sound Σ-translation of the UCQJFE $\mathcal{Q}_\mathcal{S}$ in the input. To check that a CQ q is a sound Σ-translation of $\mathcal{Q}_\mathcal{S}$, we just need to test if $q \sqsubseteq_\Sigma \mathcal{Q}_\mathcal{S}$ (which is decidable in this case since the ontology is in DL-LITE$_{RDFS}$, so it is a FUS).

Algorithm 5: Compute maximally sound UCQ-Σ-translation of a UCQJFE [Cima et al., 2019]

Input: A KBDM specification $\Sigma = (V_\mathcal{O}, \mathcal{R}_\mathcal{O}, V_\mathcal{S}, \mathcal{M})$ where $\mathcal{R}_\mathcal{O}$ is DL-LITE$_{RDFS}$ and $\mathcal{M}$ is pure GAV, a (U)CQJFE $\mathcal{Q}_\mathcal{S}$ over $V_\mathcal{S}$
Output: $\mathcal{Q}_\mathcal{O}$ over $V_\mathcal{O}$
$\mathcal{Q}_\mathcal{O} \leftarrow \bot$;
for *each CQ q over $V_\mathcal{O}$ with at most bound($\mathcal{M}, \mathcal{Q}_\mathcal{S}$) atoms, including constants from $\mathcal{Q}_\mathcal{S}$ and $\mathcal{M}$ only* **do**
  if *q is a sound Σ-translation of $\mathcal{Q}_\mathcal{S}$* **then**
    $\mathcal{Q}_\mathcal{O} \leftarrow \mathcal{Q}_\mathcal{O} \vee q$;
  end
end
return $\mathcal{Q}_\mathcal{O}$;

Slight modifications of the restricted setting introduced in Proposition IV.9 can lead

to the nonexistence of a maximally sound UCQ-Σ-translation, as presented in Proposition IV.10.

Proposition IV.10: from [Cima et al., 2019]

Let $\Sigma = (V_{\mathcal{O}}, \mathcal{R}_{\mathcal{O}}, \mathcal{S}, \mathcal{M})$ be a KBDM specification where the ontology is DL-LITE$_{RDFS}$, $\mathcal{M}$ is pure GAV and $\mathcal{Q}_{\mathcal{S}}$ be a UCQJFE over $V_{\mathcal{S}}$. The UCQ-maximally sound Σ-translations of $\mathcal{Q}_{\mathcal{S}}$ may not exist if we extend this restricted setting with any of the following features:

1. disjointness axioms in the ontology $\mathcal{R}_{\mathcal{O}}$;
2. inclusion axioms with $\exists R$ as right-hand side in the ontology $\mathcal{R}_{\mathcal{O}}$;
3. LAV mapping assertions, even without joins involving existential variables in the right-hand side;
4. non-pure GAV mapping assertions;
5. $\mathcal{Q}_{\mathcal{S}}$ in a fragment of CQs going beyond CQJFEs.

Another algorithm is proposed in [Cima et al., 2020] which computes an EQL-LITE(UCQ)-Σ-translation of a UCQ: the class EQL-LITE(UCQ) is powerful enough to represent a maximally sound Σ-translation for every UCQ (that is, there always exists one in this class). But it comes with the price of a non-monotone class of queries. The main idea of this algorithm is to compute a minimally complete EQL-LITE(UCQ)-Σ-translation $\mathcal{Q}_{\mathcal{O}}$ of a UCQ $\mathcal{Q}_{\mathcal{S}}$ and then add relevant negations to remove unwanted answers that are not sound. To do that, for every CQ $q_{\mathcal{O}} \in \mathcal{Q}_{\mathcal{O}}$ that is not sound, $q_{\mathcal{O}}$ is rewritten to obtain a UCQ $\mathcal{Q}_{\mathcal{O}\to\mathcal{S}}$ over $V_{\mathcal{S}}$. Then, for each $q_i \in \mathcal{Q}_{\mathcal{O}\to\mathcal{S}}$ that is such that $q_i \not\sqsubseteq \mathcal{Q}_{\mathcal{S}}$, a minimally complete Σ-translation q'_i is computed. Finally, $q_{\mathcal{O}}$ is replaced by a query of the form $q_{\mathcal{O}} \wedge \neg\mathbf{K}(q'_i)$ which allows one to remove unwanted answers. For more details on this process, see [Cima et al., 2020].

The last known work on this subject is about computing monotone-maximally sound Σ-translation [Cima et al., 2022], again in the context of a DL-LITE$_{RDFS}$ ontology. The target language is a set of disjunctive rules using the epistemic operator **K** and ≠-atoms in the body, and can contain existential variables in the head (we denote it $R^{\vee}_{\mathbf{K},\neq}$). The idea is to compose the mapping and the ontology to obtain a new mapping, then invert it, and put it in a rule set.

This result extends considerably the preceding ones. Indeed, all maximally sound Σ-translation are captured. However, their inversion of the mapping is rather intricate and seems very close to the already known notion of maximum recovery (introduced in [Arenas et al., 2009a] and detailed next). So, here, we do not give more detail about their computation of an inverse mapping and refer the reader to the appendix where the main algorithms are given (Appendix D.2). For more details, see [Cima et al., 2022].

Proposition IV.11: from [Cima et al., 2022]

Given a KBDM specification $\Sigma = (V_{\mathcal{O}}, \mathcal{R}_{\mathcal{O}}, V_{\mathcal{S}}, \mathcal{M})$ where $\mathcal{R}_{\mathcal{O}}$ is a DL-LITE$_{RDFS}$ ontology and a UCQ $\mathcal{Q}_{\mathcal{S}}$ over $V_{\mathcal{S}}$, the algorithm *M-MaxSound* in [Cima et al., 2022][2] terminates and returns the unique (up to Σ-equivalence) maximally sound $R^{\vee}_{\mathbf{K},\neq}$-$\Sigma$-translation of $\mathcal{Q}_{\mathcal{S}}$.

Perfect Σ-translation. Two notable works in the literature have considered this problem, each computing a perfect translation with different target query languages.

The first technique, presented by [Cima et al., 2019], uses a two-step process to compute a UCQ as the target query. The algorithm first computes a UCQ-minimally complete Σ-translation (Algorithm 3) of a UCQ $\mathcal{Q}_{\mathcal{S}}$, denoted as $\mathcal{Q}_{\mathcal{O}}$. Subsequently, it verifies whether $\mathcal{Q}_{\mathcal{O}}$ is also a sound Σ-translation of $\mathcal{Q}_{\mathcal{S}}$. If the answer is affirmative, $\mathcal{Q}_{\mathcal{O}}$ is returned as the perfect translation; otherwise, it reports that there is no perfect translation.

The second technique, as described in [Cima et al., 2022], works with $R^{\vee}_{\mathbf{K},\neq}$ rules. But it uses the same ideas: it computes the $R^{\vee}_{\mathbf{K},\neq}$-minimally complete query and then check whether it is sound.

Algorithm 6 summarises the main ideas of these two techniques to compute a perfect $\mathcal{K}^{-1}$-translation.

Algorithm 6: C-Perfect Sound Σ-translation

Input: KBDM specification Σ, Source query $\mathcal{Q}_{\mathcal{S}}$
Output: C-Perfect translation $\mathcal{Q}_{\mathcal{O}}$, that is a query of class C, or a failure report
$\mathcal{Q}_{\mathcal{O}} \leftarrow$ `C-Compute-Minimally-Complete`$(\mathcal{Q}_{\mathcal{S}})$; `// Algorithm 3 or 4`
if `Check-Soundness`$(\mathcal{Q}_{\mathcal{O}}, \mathcal{Q}_{\mathcal{S}}, \Sigma)$ **then**
  return $\mathcal{Q}_{\mathcal{O}}$;
else
  return *"No perfect translation exists for the target class C"*;
end

IV.3 A key notion: maximum recovery

The concept of maximum recovery has not originally been presented in the context of query translation but for data exchange [Arenas et al., 2008]. However, we will see in Section V.2.3.1 its relevance and utility: notably, we will use the notion of maximum recovery to compute maximally sound $\mathcal{S}$-to-$\mathcal{O}$ translations of UCQs.

Before defining a maximum recovery, we need to introduce the notion of abstract mapping. Usually, we define mappings as a set of nonrecursive rules from one vocabulary to another. But we can have a more abstract view of mappings. We can describe a mapping $\mathcal{M}$ as a binary relation from a set of instances over the source to a set of

[2] Algorithm 20 in the apprendix.

instances over the target, such that for each pair of instances (I, J) in the relation, we have $I \cup J \models_1 \mathcal{M}$. It is with this notion of mapping that a maximum recovery is defined.

Definition IV.1 (Abstract mapping [Pérez, 2011])
An *abstract mapping* $\mathcal{M}_\mathcal{A}$ from a set of predicates V_1 to a set of predicates V_2 (which we will simply call an abstract mapping from V_1 to V_2) is a non-empty subset of $inst(V_1) \times inst(V_2)$ (where $inst(V_\mathcal{P})$ is the set of all instances that we can build on a set of predicates $V_\mathcal{P}$).

We say that a mapping $\mathcal{M}$ composed of rules is a *specification* of an abstract mapping $\mathcal{M}_\mathcal{A}$ from V_1 to V_2 if for each pair (I, J) where I is an instance over V_1 and J is an instance over V_2, we have $(I, J) \in \mathcal{M}_\mathcal{A}$ if and only if $I \cup J \models_1 \mathcal{M}$. Note that a specification in the form of conjunctive or disjunctive rules does not always exist.

Since abstract mappings are simply binary relations on sets of instances, the *composition of abstract mappings* can be defined by considering the classical definition of composition of binary relations. Given abstract mappings $\mathcal{M}_\mathcal{A}^{12}$ from V_1 to V_2 and $\mathcal{M}_\mathcal{A}^{23}$ from V_2 to V_3, the composition of $\mathcal{M}_\mathcal{A}^{12}$ and $\mathcal{M}_\mathcal{A}^{23}$, denoted by $\mathcal{M}_\mathcal{A}^{12} \bullet \mathcal{M}_\mathcal{A}^{23}$, is defined as $\mathcal{M}_\mathcal{A}^{12} \bullet \mathcal{M}_\mathcal{A}^{23} = \{(I_1, I_3) \mid \exists I_2 : (I_1, I_2) \in \mathcal{M}_\mathcal{A}^{12} \text{ and } (I_2, I_3) \in \mathcal{M}_\mathcal{A}^{23}\}$ [Melnik, 2004, Fagin et al., 2004].

The *domain of an abstract mapping* $\mathcal{M}_\mathcal{A}$, denoted $\mathrm{dom}(\mathcal{M}_\mathcal{A})$, is the set of instances I over its source such that there exists $(I, J) \in \mathcal{M}_\mathcal{A}$, that is, $\mathrm{dom}(\mathcal{M}_\mathcal{A}) = \{I \mid (I, J) \in \mathcal{M}_\mathcal{A}\}$.

We now extend the notions of *solutions* and *certain answers* to abstract mappings as in [Arenas et al., 2009a]. Given an abstract mapping $\mathcal{M}_\mathcal{A}$ from V_1 to V_2 and I an instance over V_1, we say that an instance J over V_2 is a *solution* for I under $\mathcal{M}_\mathcal{A}$, if $(I, J) \in \mathcal{M}_\mathcal{A}$. The set of solutions for I under $\mathcal{M}_\mathcal{A}$ is denoted by $\mathtt{sol}_{\mathcal{M}_\mathcal{A}}(I)$. Given a source instance I, the set of *certain answers* of $\mathcal{Q}$ over I under $\mathcal{M}_\mathcal{A}$, is the set of tuples that belong to the evaluation of $\mathcal{Q}$ over every solution in $\mathtt{sol}_{\mathcal{M}_\mathcal{A}}(I)$. We denote this set by $\mathrm{certain}_{\mathcal{M}_\mathcal{A}}(\mathcal{Q}, I)$. Thus, $\mathrm{certain}_{\mathcal{M}_\mathcal{A}}(\mathcal{Q}, I) = \bigcap_{J \in \mathtt{sol}_{\mathcal{M}_\mathcal{A}}(I)} \mathcal{Q}(J)$. Note that these notions coincide with those on the specification of a mapping: $\mathrm{certain}_{\mathcal{M}_\mathcal{A}}(\mathcal{Q}, I) = \mathrm{certain}_{\mathcal{M}}(\mathcal{Q}, I)$ and $\mathtt{sol}_{\mathcal{M}}(I) = \mathtt{sol}_{\mathcal{M}_\mathcal{A}}(I)$ with $\mathcal{M}$ a specification of $\mathcal{M}_\mathcal{A}$.

A recovery is a kind of inverse mapping. Intuitively, an inverse of a mapping $\mathcal{M}$ serves as a reverse mapping that undoes the application of $\mathcal{M}$. The essential requirement for a recovery mapping is soundness, which ensures that only information that was originally present before the exchange is restored.

Definition IV.2 (Recovery Mapping [Arenas et al., 2009a])
An abstract mapping $\mathcal{M}'_\mathcal{A}$ is a *recovery mapping* of $\mathcal{M}_\mathcal{A}$ if and only if for all instances I over $V_\mathcal{S}$ and all queries $\mathcal{Q}$, it holds that:

$$\mathtt{certain}_{\mathcal{M}_\mathcal{A} \bullet \mathcal{M}'_\mathcal{A}}(\mathcal{Q}, I) \subseteq \mathcal{Q}(I).$$

We do not only want to recover sound information from the source, but ideally, we would like to recover all the sound information that it is possible to recover: this is performed by a *maximum recovery*.

Definition IV.3 (Maximum Recovery [Arenas et al., 2009a])
A *recovery* $\mathcal{M}'_{\mathcal{A}}$ is considered *maximum* if, for every other recovery $\mathcal{M}''_{\mathcal{A}}$ of $\mathcal{M}_{\mathcal{A}}$, we have for every instance I over $V_{\mathcal{S}}$ and every query $\mathcal{Q}$,

$$\mathtt{certain}_{\mathcal{M}_{\mathcal{A}} \bullet \mathcal{M}''_{\mathcal{A}}}(\mathcal{Q}, I) \subseteq \mathtt{certain}_{\mathcal{M}_{\mathcal{A}} \bullet \mathcal{M}'_{\mathcal{A}}}(\mathcal{Q}, I).$$

In other words, a mapping $\mathcal{M}'_{\mathcal{A}}$ is maximum if no other recovery allows one to recover more sound information from the original data. It guarantees the most extensive recovery possible while preserving soundness.

Next Proposition IV.12 gives properties that will be useful in some proofs that involve a maximum recovery. This proposition uses the notion of *reduced recovery*: an abstract mapping $\mathcal{M}'_{\mathcal{A}}$ is a *reduced recovery* of $\mathcal{M}_{\mathcal{A}}$ if $\mathcal{M}'_{\mathcal{A}}$ is a recovery of $\mathcal{M}_{\mathcal{A}}$ and for every $(I_1, I_2) \in \mathcal{M}_{\mathcal{A}} \bullet \mathcal{M}'_{\mathcal{A}}$, we have $I_2 \in \mathrm{dom}(\mathcal{M}_{\mathcal{A}})$.

Proposition IV.12: From [Arenas et al., 2009b]

Let $\mathcal{M}_{\mathcal{A}}$ and $\mathcal{M}'_{\mathcal{A}}$ be abstract mappings. Then the following conditions are equivalent:

1. $\mathcal{M}'_{\mathcal{A}}$ is a maximum recovery of $\mathcal{M}_{\mathcal{A}}$.
2. $\mathcal{M}'_{\mathcal{A}}$ is a reduced recovery of $\mathcal{M}_{\mathcal{A}}$ and $\mathcal{M}_{\mathcal{A}} = \mathcal{M}_{\mathcal{A}} \bullet \mathcal{M}'_{\mathcal{A}} \bullet \mathcal{M}_{\mathcal{A}}$.
3. $\mathcal{M}'_{\mathcal{A}}$ is a recovery of $\mathcal{M}_{\mathcal{A}}$ and for every $(I_1, I_2) \in \mathcal{M}_{\mathcal{A}} \bullet \mathcal{M}'_{\mathcal{A}}$, it is the case that $\emptyset \subsetneq \mathtt{sol}_{\mathcal{M}_{\mathcal{A}}}(I_2) \subseteq \mathtt{sol}_{\mathcal{M}_{\mathcal{A}}}(I_1)$.

IV.3.1 Maximum recoveries in practice

We defined the notion of maximum recovery on abstract mappings, but in practice we do not use abstract mappings. One can ask the question: when we have a mapping composed of conjunctive rules, can we specify a maximum recovery with a set of rules, and if so, what kind of rules? Proposition IV.13 answers this question.

Proposition IV.13: From [Arenas et al., 2009b]

Let $\mathcal{M}_{\mathcal{A}}$ be an abstract mapping and $\mathcal{M}'_{\mathcal{A}}$ be a maximum recovery of $\mathcal{M}_{\mathcal{A}}$. If the domain of $\mathcal{M}_{\mathcal{A}}$ contains only databases (that is, ground instances) and $\mathcal{M}_{\mathcal{A}}$ is specified by conjunctive rules, then $\mathcal{M}'_{\mathcal{A}}$ can be specified with disjunctive rules with special predicate **C** occurring in rule bodies and equalities occurring in rule heads.

Remark IV.2

1. Proposition IV.13 does not hold if we assume that the source instance is not ground: to manage such a case, the notion of maximum extended recovery was introduced in [Fagin et al., 2011], but, as far as we know, the question of whether

a maximum extended recovery can be specified by a set of rules remains an open problem.

2. Since we consider only databases in the domain of $\mathcal{M}_\mathcal{A}$ in Proposition IV.13, and every solution to a maximum recovery of $\mathcal{M}_\mathcal{A}$ is in the domain of $\mathcal{M}_\mathcal{A}$ by Proposition IV.12, we have that each solution in a maximum recovery of $\mathcal{M}_\mathcal{A}$ is a database.

3. In the general case, not only can we specify a maximum recovery with the kind of rules described in Proposition IV.13, but we cannot rely on a less expressive class. Indeed, we need disjunctive rules and the special predicate **C** [Arenas et al., 2009b]. Intuitively, we need disjunction because the content transferred into a predicate at the ontological level can be from different set of atoms from the database; and the predicate **C** is used to ensure that a term comes from the database, i.e., was not created from an existential variable of a rule. As we shall see, the equalities in the head can be replaced with inequalities in the body, but one of the two is necessary to specify the maximum recovery of a conjunctive mapping [Arenas et al., 2009a].

To compute a maximum recovery of a mapping $\mathcal{M}$, the main idea is to rewrite through $\mathcal{M}$ all the heads H of the rules in $\mathcal{M}$, to create, for each head H, a rule of the form $H \rightarrow \mathcal{M}\text{–}\texttt{rewriting}(H)$; note that H is seen as a UCQ whose answer variables are the rule frontier. Intuitively, since the rewriting is sound and complete, this will capture back all the information transferred via the heads of the rules. Algorithm 7 from [Arenas et al., 2009b] uses this principle.

Algorithm 7: MaximumRecovery($\mathcal{M}$) [Arenas et al., 2009b]

Input: A conjunctive mapping $\mathcal{M}$
Output: A disjunctive mapping $\mathcal{M}'$ that is a maximum recovery of $\mathcal{M}$; Rules in $\mathcal{M}'$ may contain head equalities

$\mathcal{M}' \leftarrow \emptyset$;
for $B[\mathbf{x}] \rightarrow H[\mathbf{x},\mathbf{y}]$ *in* $\mathcal{M}$ **do**
  $\alpha(\mathbf{x}) \leftarrow \mathcal{M}\text{–}\texttt{rewriting}(H[\mathbf{x},\mathbf{y}](\mathbf{x}))$; `// α(x) is a UCQ with equality`
  $\mathcal{M}' \leftarrow \mathcal{M}' \cup \{H[\mathbf{x},\mathbf{y}] \wedge \mathbf{C}[\mathbf{x}] \rightarrow \alpha(\mathbf{x})\}$;
end
return $\mathcal{M}'$;

Example IV.14 illustrates Algorithm 7.

Example IV.14: Maximum Recovery

Let us consider the mapping:

$$\mathcal{M} = \begin{cases} m_1 = s_1(x) & \rightarrow \exists y\ p(x,y) \\ m_2 = s_2(x,y) & \rightarrow p(x,y) \\ m_3 = s_3(x) & \rightarrow r(x,x) \\ m_4 = s_4(x,y) & \rightarrow r(x,y) \end{cases}$$

Algorithm 7 outputs the following maximum recovery:

$$\mathcal{M}' = \begin{cases} m'_1 = p(x,y) \wedge \mathbf{C}(x) & \rightarrow s_1(x) \vee \exists z.s_2(x,z) \\ m'_2 = p(x,y) \wedge \mathbf{C}(x) \wedge \mathbf{C}(y) & \rightarrow s_2(x,y) \\ m'_3 = r(x,y) \wedge \mathbf{C}(x) \wedge \mathbf{C}(y) & \rightarrow (s_3(x) \wedge x = y) \vee s_4(x,y) \\ m'_4 = r(x,x) \wedge \mathbf{C}(x) & \rightarrow s_3(x) \end{cases}$$

Indeed, to obtain the rules in $\mathcal{M}'$, we have performed the following:

- m'_1: let $q_1(x) = \exists y\ p(x,y)$, we have $\alpha_1(x) = \mathcal{M}\text{–}\texttt{rewriting}(q_1(x)) = s_1(x) \vee \exists z.s_2(x,z)$, we set $q_1(x) \wedge \mathbf{C}(x)$ as the body and $\alpha_1(x)$ as the head;
- m'_2: let $q_2(x,y) = p(x,y)$, we have $\alpha_2(x,y) = \mathcal{M}\text{–}\texttt{rewriting}(q_2(x,y)) = s_2(x,y)$, we set $q_2(x,y) \wedge \mathbf{C}(x) \wedge \mathbf{C}(y)$ as the body and $\alpha_2(x,y)$ as the head;
- m'_3: let $q_3(x,y) = r(x,y)$, we have $\alpha_3(x,y) = \mathcal{M}\text{–}\texttt{rewriting}(q_3(x,y)) = (s_3(x) \wedge x = y) \vee s_4(x,y)$, we set $q_3(x,y) \wedge \mathbf{C}(x) \wedge \mathbf{C}(y)$ as the body and $\alpha_3(x,y)$ as the head;
- m'_4: let $q_4(x) = r(x,x)$, we have $\alpha_4(x) = \mathcal{M}\text{–}\texttt{rewriting}(q_4(x)) = s_3(x)$, we set $q_4(x) \wedge \mathbf{C}(x)$ as the body and $\alpha_4(x)$ as the head.

IV.3.2 Maximum recovery without equalities

In this dissertation, we will do rewritings through a maximum recovery, but we have not defined how to manage rules with equality in a disjunctive rewriting. Fortunately, we can find algorithms in the literature to remove equalities in a maximum recovery. Here, we use the algorithm `EliminateEqualities` from [Arenas et al., 2009a], provided in Algorithm 8. By combining Algorithm 8 and Algorithm 7, we obtain a third one that computes a Maximum Recovery without equalities (Algorithm 9). In a nutshell, to remove head equalities, we add inequalities in the body of rules and create a maximum recovery that may have an exponential size compared to the maximum recovery with equalities.

Algorithm 8: EliminateEqualities($\mathcal{M}'$)

Input: A disjunctive mapping $\mathcal{M}'$ where a rule head may contain equalities
Output: A disjunctive mapping $\mathcal{M}''$ where a rule body may contain **C**-atoms and inequalities, and equality does not occur in rules.

$\mathcal{M}'' \leftarrow \emptyset$;
for $m = B[\mathbf{x}] \wedge \mathbf{C}[\mathbf{x}] \rightarrow H_1[\mathbf{x}] \vee \ldots \vee H_k[\mathbf{x}] \in \mathcal{M}'$ **do**
  if *m does not contain equalities in the head* **then**
    $\mathcal{M}'' \leftarrow \mathcal{M}'' \cup \{m\}$;
  else
    for *every substitution s from* **x** *to* **x** **do**
      $H_\vee \leftarrow \bot$;
      Let δ_s be the inequalities induced by s;
      for *i from* 1 *to k* **do**
        if *the equalities in* $H_i[s(\mathbf{x})]$ *are consistent with* δ_s **then**
          drop the equalities in $H_i[s(\mathbf{x})]$ and add the resulting formula as a disjunct in $H_\vee$;
        end
      end
      if $H_\vee \neq \bot$ **then**
        $\mathcal{M}'' \leftarrow \mathcal{M}'' \cup \{B[s(\mathbf{x})] \wedge \mathbf{C}[s(\mathbf{x})] \wedge \delta_s \rightarrow H_\vee\}$;
      end
    end
  end
end
return $\mathcal{M}''$;

Algorithm 9: MaximumRecoveryWithoutEqualities($\mathcal{M}$)

Input: A conjunctive mapping $\mathcal{M}$
Output: A disjunctive mapping $\mathcal{M}''$ where a rule body may contain **C**-atoms and inequalities, and equality does not occur in rules.

$\mathcal{M}' \leftarrow$ MaximumRecovery($\mathcal{M}$);
$\mathcal{M}'' \leftarrow$ EliminateEqualities($\mathcal{M}'$);
return $\mathcal{M}''$;

Example IV.15 illustrates Algorithm 9.

Example IV.15: Maximum Recovery without equalities

Take again the mapping from Example IV.14:

$$\mathcal{M} = \begin{cases} m_1 = s_1(x) & \rightarrow \exists y.p(x,y) \\ m_2 = s_2(x,y) & \rightarrow p(x,y) \\ m_3 = s_3(x) & \rightarrow r(x,x) \\ m_4 = s_4(x,y) & \rightarrow r(x,y) \end{cases}$$

Algorithm 7 outputs the following maximum recovery:

$$\mathcal{M}' = \begin{cases} m'_1 = p(x,y) \wedge \mathbf{C}(x) & \rightarrow s_1(x) \vee \exists z.s_2(x,z) \\ m'_2 = p(x,y) \wedge \mathbf{C}(x) \wedge \mathbf{C}(y) & \rightarrow s_2(x,y) \\ m'_3 = r(x,y) \wedge \mathbf{C}(x) \wedge \mathbf{C}(y) & \rightarrow (s_3(x) \wedge x = y) \vee s_4(x,y) \\ m'_4 = r(x,x) \wedge \mathbf{C}(x) & \rightarrow s_3(x) \vee s_4(x,x) \end{cases}$$

Algorithm 9 will remove the equality from m'_3 by considering that either $x = y$ or $x \neq y$. A new rule is created for each case. For the first case, we obtain exactly the rule m'_4, so we do not add this new rule. We just create a new rule for the case $x \neq y$ and we obtain the following maximum recovery:

$$\mathcal{M}'' = \begin{cases} m'_1 = p(x,y) \wedge \mathbf{C}(x) & \rightarrow s_1(x) \vee \exists z.s_2(x,z) \\ m'_2 = p(x,y) \wedge \mathbf{C}(x) \wedge \mathbf{C}(y) & \rightarrow s_2(x,y) \\ m''_3 = r(x,y) \wedge x \neq y \wedge \mathbf{C}(x) \wedge \mathbf{C}(y) & \rightarrow s_4(x,y) \\ m'_4 = r(x,x) \wedge \mathbf{C}(x) & \rightarrow s_3(x) \vee s_4(x,x) \end{cases}$$

IV.3.3 CQ-maximum recovery

One could ask the question: if we restrict the class of queries considered over the target, can we have a specification of a maximum recovery that does not use disjunctive rules ? There is some work on this topic in [Arenas et al., 2009a] about a notion of maximum recovery called CQ-maximum recovery, that is specific to conjunctive queries and that can be specified with conjunctive rules. It would have given very nice properties to the $\mathcal{M}$-translation of CQs but, unfortunately, the results of [Arenas et al., 2009a] are not correct, as shown in Example V.27.

We first explain what a CQ-Maximum Recovery is and then show with a counterexample why it is a recovery that is not maximum. This requires to define the Cartesian product of queries.

Definition IV.4 (Cartesian Product of Queries [Arenas et al., 2009a])
Let q_1 and q_2 be two n-ary conjunctive queries with tuple of answer variables $\mathbf{x}$. The *Cartesian product* of q_1 and q_2, denoted by $q_1 \times q_2$, is a k-ary conjunctive query with

$k \leq n$, constructed by defining a one-to-one function $f(\cdot,\cdot)$ from pairs of variables to variables, satisfying:

1. $f(x,x) = x$ for every variable x in $\mathbf{x}$,
2. $f(y,z)$ is a fresh variable (i.e., not occurring in q_1 nor q_2) in any other case.

Then, for every pair of atoms $R(y_1,\ldots,y_m)$ in q_1 and $R(z_1,\ldots,z_m)$ in q_2, the atom $R(f(y_1,z_1),\ldots,f(y_m,z_m))$ belongs to $q_1 \times q_2$. The free variables of $q_1 \times q_2$ are those of $\mathbf{x}$ that still occur in $q_1 \times q_2$.

Example IV.16: Cartesian Product of Queries [Arenas et al., 2009a]

Consider the following CQs:

$$q_1(x_1,x_2) = P(x_1,x_2) \wedge R(x_1,x_1),$$
$$q_2(x_1,x_2) = \exists y(P(x_1,y) \wedge R(x_2,x_2)),$$

Their product $q_1 \times q_2$ is the following CQ:

$$(q_1 \times q_2)(x_1) = \exists z_1 \exists z_2 (P(x_1,z_1) \wedge R(z_2,z_2)).$$

The above definition, inspired by the standard notion of Cartesian product of graphs [Hell and Nesetril, 2004], plays a key role in the following algorithm that eliminates disjunctions in a disjunctive mapping (Algorithm 10), which is then used in the algorithm that computes a CQ-Maximum Recovery (Algorithm 11).

Algorithm 10: Eliminate Disjunctions

Input: A disjunctive mapping as output by Algorithm 8.
Output: A conjunctive mapping $\mathcal{M}^\star$ (with $\mathbf{C}$-atoms and inequalities in rule bodies and no occurrence of equality)

$\mathcal{M}^\star \leftarrow \emptyset$;
for $\psi[\mathbf{x}] \wedge \mathbf{C}[\mathbf{x}] \wedge \delta[\mathbf{x}] \rightarrow H_1[\mathbf{x}] \vee \ldots \vee H_k[\mathbf{x}]$ *in* $\mathcal{M}''$ **do**
  if $H_1[\mathbf{x}] \times \ldots \times H_k[\mathbf{x}] \neq \emptyset$ **then**
    $\mathcal{M}^\star \leftarrow \mathcal{M}^\star \cup \{\psi[\mathbf{x}] \wedge \mathbf{C}[\mathbf{x}] \wedge \delta[\mathbf{x}] \rightarrow H_1[\mathbf{x}] \times \ldots \times H_k[\mathbf{x}]\}$;
  end
end
return $\mathcal{M}^\star$;

We have first to point out that the use of the Cartesian product of queries in Algorithm 10 is not correct. Indeed, according to Definition IV.4, the resulting arity of the Cartesian product can be strictly smaller than the arity of the involved original queries. This becomes problematic when chaining Cartesian products of queries, which are all supposed to be of the same arity. Since an intermediate result may have a strictly lower

Algorithm 11: Compute CQ-Maximum Recovery

Input: A conjunctive mapping $\mathcal{M}$
Output: A conjunctive mapping $\mathcal{M}^\star$ (with **C**-atoms and inequalities in rule bodies and no occurrence of equality)
$\mathcal{M}' \leftarrow$ MaximumRecovery($\mathcal{M}$) ; // Algorithm 7
$\mathcal{M}'' \leftarrow$ EliminateEqualities($\mathcal{M}'$) ; // Algorithm 8
$\mathcal{M}^\star \leftarrow$ EliminateDisjunctions($\mathcal{M}''$) ; // Algorithm 10
return $\mathcal{M}^\star$;

arity, the subsequent behaviour of the Cartesian product of queries becomes undefined. Fortunately, in Algorithm 10, we can consider that the disjuncts within the head of a disjunctive rule are Boolean queries, while preserving the desired properties of the cartesian product.

More fundamentally, contrarily to its claim, Algorithm 11 does not compute a CQ-Maximum Recovery. And worse, there are conjunctive mappings for which no CQ-maximum recovery is conjunctive. This is stated in the next proposition.

Proposition IV.17

Algorithm 11 does not compute a CQ-Maximum Recovery. Furthermore, there is a conjunctive mapping $\mathcal{M}$ that does not admit any CQ-maximum recovery that is conjunctive.

Proof of Proposition IV.17. Let us consider the following mapping:

$$\mathcal{M} = \begin{cases} s_1(x) & \rightarrow q(x), \\ s_2(x) \wedge s_5(x) & \rightarrow t(x), \\ s_1(y) \wedge s_3(x,y) & \rightarrow p(x,y), \\ s_2(x) \wedge s_4(x,y) & \rightarrow p(x,y). \end{cases}$$

Its maximum recovery $\mathcal{M}'$ is as follows :

$$\mathcal{M}' = \begin{cases} m'_1 : q(x) \wedge \mathbf{C}(x) & \rightarrow s_1(x), \\ m'_2 : t(x) \wedge \mathbf{C}(x) & \rightarrow s_2(x) \wedge s_5(x), \\ m'_3 : p(x,y) \wedge \mathbf{C}(x) \wedge \mathbf{C}(y) & \rightarrow (s_1(y) \wedge s_3(x,y)) \vee (s_2(x) \wedge s_4(x,y)). \end{cases}$$

Let $q_{\mathcal{S}} = s_1(u) \wedge s_2(u)$ be the CQ from the same example.

Algorithm 11 computes a disjunctive mapping $\mathcal{M}^\star$ that is the same as $\mathcal{M}'$ but without the last rule. Indeed, as part of its computation, the algorithm first derives $\mathcal{M}'$ based on the given input. As there are no equalities present, we directly get $\mathcal{M}'' = \mathcal{M}'$. Subsequently, Algorithm 10 (*EleminateDisjunctions*) retains the rules m'_1 and m'_2, while

discarding the rule m_3'. This is due to the fact that the Cartesian product of the disjuncts in the head of m_3' is empty. Thus, the final computed disjunctive mapping $\mathcal{M}^\star$ consists of $\{m_1', m_2'\}$.

Consider a database $D = \{s_1(a), s_2(a), s_4(a,b), s_2(b), s_5(b)\}$. The mapping chase on D yields: $\mathcal{M}\text{–chase}(D) = \{q(a), t(b), p(a,b)\}$. We then obtain:

$$\mathcal{M}'\text{–chase}_\vee(\mathcal{M}\text{–chase}(D)) = \{ \quad I_1 = \{s_1(a), s_2(b), s_5(b), s_1(b), s_3(a,b)\},$$
$$I_2 = \{s_1(a), s_2(b), s_5(b), s_2(a), s_4(a,b)\}\},$$
$$\mathcal{M}^\star\text{–chase}_\vee(\mathcal{M}\text{–chase}(D)) = \{ \quad I_3 = \{s_1(a), s_2(b), s_5(b)\}\}.$$

Now, let us analyze the behavior of these objects w.r.t. the entailment of $q_\mathcal{S}$:

- $D \models q_\mathcal{S}$, as we can map $q_\mathcal{S}$ to D with the homomorphism $h = \{u \mapsto a\}$.
- $\mathcal{M}'\text{–chase}_\vee(\mathcal{M}\text{–chase}(D)) \models q_\mathcal{S}$, as we can map $q_\mathcal{S}$ to I_1 with the homomorphism $h_1 = \{u \mapsto b\}$ and to I_2 with the homomorphism $h_2 = \{u \mapsto a\}$.
- $\mathcal{M}^\star\text{–chase}_\vee(\mathcal{M}\text{–chase}(D)) \not\models q_\mathcal{S}$ as we cannot find any homomorphism that maps $q_\mathcal{S}$ to I_3.

This proves that $\mathcal{M}^\star$ is not maximum since $\mathcal{M}'$ recovers strictly more answers than $\mathcal{M}^\star$, so $\mathcal{M}^\star$ is not a maximum recovery.

Now, we claim that there is no conjunctive mapping that can be a CQ-Maximum Recovery of $\mathcal{M}$. This is a consequence of Theorem V.25, stated later in this chapter. Theorem V.25 shows that we can obtain a maximally sound $\mathcal{M}$-translation of a CQ by rewriting it through a maximum recovery. However, we previously saw that a CQ may not have a finite maximally sound $\mathcal{M}$-translation. If we could compute a maximum recovery that is a conjunctive mapping for such CQ, we would have a contradiction: indeed, the rewriting through a conjunctive mapping is always finite. ■

IV.4 Summary

Let us briefly review the content of this chapter.

- **Extension of UCQs:** We have chosen to study translations within a slight extension of UCQs, as considered in the work of [Arenas et al., 2008, Pérez, 2011], yielding (U)CQ$^{\mathbf{C},\neq}$ (Section IV.1).
- **State of the art of $\mathcal{S}$-to-$\mathcal{O}$-translation:** We reviewed existing techniques related to $\mathcal{S}$-to-$\mathcal{O}$ translation (Section IV.2).
- **Maximum recovery:** We then bring into the framework the notion of maximum recovery, which was introduced for other purposes in the literature on data exchange. In passing, we exhibit a wrong claim in the literature IV.3.3, namely that all CQs admit a maximum recovery that is a conjunctive mapping.

V - Query translation: new results

In this chapter, we apply the query translation framework defined in Chapter II.2 to UCQs and UCQ$^{\mathbf{C},\neq}$. Section V.1 is devoted to $\mathcal{O}$-to-$\mathcal{S}$ translations within UCQs. Then, in Section V.2, we present new techniques for $\mathcal{O}$-to-$\mathcal{S}$ translations of UCQ$^{\mathbf{C},\neq}$.

V.1 $\mathcal{O}$-to-$\mathcal{S}$-translation of UCQs

We first consider the $\mathcal{M}^{-1}$-translation of a UCQ, then its Σ^{-1}-translation.

One of the most direct approaches to $\mathcal{M}^{-1}$-translate a UCQ into another UCQ through a conjunctive mapping $\mathcal{M}$, is by using well-known query rewriting techniques. These techniques are introduced in Section I.6 and can be easily applied for this purpose. In particular, the $\mathcal{M}$-rewriting operator (Definition I.42) always produces a perfect $\mathcal{M}^{-1}$-translation.

Theorem V.1: Soundness and Completeness of $\mathcal{M}$-Rewriting

Let $\mathcal{Q}_\mathcal{O}$ be a UCQ over $V_\mathcal{O}$ and $\mathcal{M}$ be a mapping from $V_\mathcal{S}$ to $V_\mathcal{O}$. Then, $\mathcal{M}$–`rewriting`$(\mathcal{Q}_\mathcal{O})$ is a perfect $\mathcal{M}^{-1}$-translation of $\mathcal{Q}_\mathcal{O}$ that is also a UCQ.

Proof. It is a direct consequence of the fact that the $\mathcal{M}$-rewriting operator is sound and complete, and that it always produces a finite set of CQs. ■

Unlike $\mathcal{M}^{-1}$-translation, where existing techniques can be directly used, the computation of a Σ^{-1}-translation brings about substantial complexity and novel challenges.

We remind the reader that the question of whether a given UCQ admits a UCQ-rewriting with a given set of existential rules is not decidable, but it is semi-decidable, in the sense that an algorithm exists that, given a pair $(\mathcal{Q},\mathcal{R})$ produces a UCQ-rewriting of $(\mathcal{Q},\mathcal{R})$ when one exists. See, for instance, Algorithm 1. Note that this algorithm always halts if the set of rules $\mathcal{R}$ is FUS.

In contrast, one cannot find any generic algorithm in the literature to tackle the problem of computing a perfect Σ^{-1}-translation of a UCQ into another UCQ for a specification Σ, that is, when taking into account existential rules. In fact, there is a good reason for that: we show below that there does not exist any algorithm that computes a perfect Σ^{-1}-translation and terminates when such finite translation exists. More precisely, even when we know that a finite perfect Σ^{-1}-translation exists for a pair $(\mathcal{Q}_\mathcal{O},\Sigma)$, we are not able to compute it.

We first define the computation problem and then demonstrate that the associated function is not computable.

Problem V.2: Computing a UCQ-Σ^{-1}-translation when we know its existence

Input: A KBDM specification Σ^{-1} and a (U)CQ $\mathcal{Q}_\mathcal{O}$ over $V_\mathcal{O}$, such that $\mathcal{Q}_\mathcal{O}$ has a perfect Σ^{-1}-translation as a UCQ
Output: A UCQ that is a perfect Σ^{-1}-translation of $\mathcal{Q}_\mathcal{O}$

Note that, in Problem V.2, we know that the input has a perfect Σ^{-1}-translation which is a UCQ. However, as stated in Theorem V.3, we cannot compute this UCQ.

Theorem V.3: Impossibility of generic UCQ-Σ^{-1}-translation

There exists no algorithm that can compute the output of Problem V.2.

To prove Theorem V.3, we introduce two decision problems; the first one is known to be undecidable, and we show that it can be reduced to the second one, which is thus undecidable as well; then we show that the second problem is decidable if there exists an algorithm that can compute the output of Problem V.2.

Problem V.4: Entailment 1

Input: A set of conjunctive rules $\mathcal{R}$, a CQ q, a set of predicates V, such that $(\mathcal{R}, q, V)$ is UCQ-rewritable, that is, there exists a finite, sound and complete rewriting of q with $\mathcal{R}$ over V, and a database D on V.
Question: Does $D, \mathcal{R} \models q$ hold?

It is known that there is no algorithm that can solves the Problem V.4, it is not even semi-decidable[1].

Problem V.5: Entailment 2

Input: A KBDM system $\mathcal{K} = (D, \Sigma)$ with $\Sigma = (V_\mathcal{O}, \mathcal{R}_\mathcal{O}, V_\mathcal{S}, \mathcal{M})$, a (U)CQ $\mathcal{Q}_\mathcal{O}$ over $V_\mathcal{O}$ such that $\mathcal{Q}_\mathcal{O}$ has a finite perfect Σ^{-1}-translation
Question: Does $I_{(D,\mathcal{M})}, \mathcal{R}_\mathcal{O} \models \mathcal{Q}$ hold?

Lemma V.6

Problem V.5 is undecidable.

Proof of lemma V.6. The proof is done by reducing Problem V.4 to Problem V.5. To do that, we take $V_\mathcal{O}$ in bijection with the set of predicates occurring in D, $\mathcal{R}$ and q: we note $\widehat{p}$ the predicate assigned to p. Then, $\mathcal{R}_\mathcal{O} = \widehat{\mathcal{R}}$, that is, we rename all predicates in the rules with a hat, and similarly $\mathcal{Q} = \{\widehat{q}\}$. Then, we build a mapping that simulates the restriction of the vocabulary: $\mathcal{M} = \{p(\mathbf{x}) \to \widehat{p}(\mathbf{x}) \mid p \in V\}$. We then have $I_{(D,\mathcal{M})}, \mathcal{R}_\mathcal{O} \models \mathcal{Q}$

[1] Personal communication from David Carral, June 2023

if and only if $D, \mathcal{R} \models q$. Since we know that there exists a UCQ $\mathcal{Q}'$ on V such that for every database D, $D \models \mathcal{Q}'$ if and only if $D, \mathcal{R} \models q$, we also have $D \models \mathcal{Q}'$ if and only if $I_{(D,\mathcal{M})}, \mathcal{R}_\mathcal{O} \models \mathcal{Q}$. Therefore, $\mathcal{Q}'$ is a finite perfect Σ^{-1}-translation of Σ if and only if it is a rewriting of q through $\mathcal{R}$. Thus, since there is no algorithm for Problem V.4, then there is no algorithm for Problem V.5 ■

Proof of Theorem V.3. If we can compute a perfect Σ^{-1}-translation $\mathcal{Q}_\mathcal{S}$ of $\mathcal{Q}_\mathcal{O}$, then, we are able to solve Problem V.5 since $D \models \mathcal{Q}_\mathcal{S}$ if and only if $I_{(D,\mathcal{M})}, \mathcal{R} \models \mathcal{Q}_\mathcal{O}$. But we proved in Lemma V.6 that this problem is undecidable, so there is no algorithm to compute a Σ^{-1}-translation. ■

This result may seem strange, as, in the idea, we could just use a rewriting algorithm through $\mathcal{R}_\mathcal{O}$ and then through $\mathcal{M}$, then the rewriting would be finite if it is finite through $\mathcal{R}_\mathcal{O}$. However, the point is that there are cases where a query has no finite rewriting with $\mathcal{R}_\mathcal{O}$ but a UCQ-perfect Σ^{-1}-translation. The next example illustrates that.

Example V.7: Finite Σ^{-1}-translation with Non-FUS Rules

Consider the KBDM specification $\Sigma = (V_\mathcal{O}, \mathcal{R}_\mathcal{O}, V_\mathcal{S}, \mathcal{M})$ where we have:

- $\mathcal{R}_\mathcal{O} = \{p(x,y) \wedge p(y,z) \rightarrow p(x,z)\}$ and $V_\mathcal{O} = \{p\}$.
- $\mathcal{M} = \{s_1(x) \wedge s_2(y) \rightarrow p(x,y)\}$.

The rule in $\mathcal{R}_\mathcal{O}$ is a classical example of a non-FUS rule, as it defines the transitive closure of a predicate.
Let $q(u,v) = p(u,v)$ be a CQ and let us check that it has a finite perfect Σ^{-1}-translation. More generally, any CQ has a finite Σ^{-1}-translation.

1. By rewriting $q(u,v)$ with the only rule in $\mathcal{R}_\mathcal{O}$, we obtain a set of pairwise incomparable CQs, composed of q and CQs of the following form:

$$q_n(u,v) = p(u,y_0) \cup \left(\bigcup_{i \in [1,n]} p(y_{i-1}, y_i) \right) \cup p(y_n, v), \text{for } n \in \mathbb{N}.$$

2. Rewriting any CQ $q_n(u,v)$ or the initial CQ $q(u,v)$ with $\mathcal{M}$ replaces each occurrence $p(x,y)$ by $s_1(x) \wedge s_2(y)$. So, if we remove the redundancies, we obtain the following result:

$$\mathcal{M}\text{–}\texttt{rewriting}(q_n(u,v)) = \{s_1(u) \wedge s_2(v)\}, \text{for } n \in \mathbb{N}.$$

$$\mathcal{M}\text{–}\texttt{rewriting}(q(u,v)) = \{s_1(u) \wedge s_2(v)\}.$$

So, the Σ^{-1}-translation contains only one CQ that is $q_\mathcal{S}(u,v) = s_1(u) \wedge s_2(v)$.

In this example, although $\mathcal{R}_\mathcal{O}$ is not FUS, the Σ^{-1}-translation is finite due to the structure of the mapping $\mathcal{M}$. Rewriting through $\mathcal{R}_\mathcal{O}$ creates CQs that form chains, but subsequent rewriting through $\mathcal{M}$ destroys these chains, resulting in a finite set of CQs. This is not only true for the CQ in our example but for any CQ over $V_\mathcal{O}$, since the rewriting through $\mathcal{M}$ can only produce connected components of the shape $s_1(x) \wedge s_2(x)$, $s_1(x)$ or $s_2(x)$.

Although Theorem V.3 is a strong negative result, this does not mean that we can never compute Σ^{-1}-translations of UCQs. If we consider a KBDM specification where $\mathcal{R}_\mathcal{O}$ is a finite unification set, then we can rely on the usual rewriting techniques. In fact, it is the classical setting of an OBDA framework, where most of the time a light Description Logic ontology is used to ensure termination of CQ rewriting . The Σ^{-1}-translation of a UCQ in this context can be computed with Algorithm 12.

Algorithm 12: Algorithm for Σ^{-1}-translation with Finite Unification Set

Input: KBDM specification $\Sigma = (V_\mathcal{O}, \mathcal{R}_\mathcal{O}, V_\mathcal{S}, \mathcal{M})$ with $\mathcal{R}_\mathcal{O}$ a finite unification set, UCQ $\mathcal{Q}_\mathcal{O}$

Output: Perfect Σ^{-1}-translation of $\mathcal{Q}_\mathcal{O}$

$\mathcal{Q}'_\mathcal{O} \leftarrow$ `rewriting`$(\mathcal{Q}_\mathcal{O}, \mathcal{R}_\mathcal{O})$; `// Rewrite` $\mathcal{Q}_\mathcal{O}$ `through` $\mathcal{R}_\mathcal{O}$

$\mathcal{Q}_\mathcal{K} \leftarrow \mathcal{M}$`-rewriting`$(\mathcal{Q}'_\mathcal{O})$; `// Rewrite result through mapping` $\mathcal{M}$

return $\mathcal{Q}_\mathcal{K}$

Proposition V.8: Termination of Algorithm 12 when the ontoogy is FUS

Let $\Sigma = (V_\mathcal{O}, \mathcal{R}_\mathcal{O}, V_\mathcal{S}, \mathcal{M})$ be a KBDM specification where $\mathcal{R}_\mathcal{O}$ is a finite unification set, and $\mathcal{Q}_\mathcal{O}$ be a union of conjunctive queries. Then Algorithm 12 always terminates on such an input and produces the Σ^{-1}-translation of $\mathcal{Q}_\mathcal{O}$.

Proof. It is a direct consequence that the rewriting with $\mathcal{R}_\mathcal{O}$ terminates because it is a finite unification set and the rewriting through a conjunctive mapping also always terminates. The result is a perfect Σ^{-1}-translation since the rewriting algorithms used are sound and complete. ■

Note that Algorithm 12 is not optimal in the sense that rewriting through $\mathcal{R}_\mathcal{O}$ can produce some CQs that cannot be rewritten through $\mathcal{M}$ (that is, when rewriting such a CQ through $\mathcal{M}$, we obtain an empty UCQ).

V.2 Novel techniques for $\mathcal{S}$-to-$\mathcal{O}$-translation of UCQs

In this section, we present new methods developed to tackle the problem of $\mathcal{S}$-to-$\mathcal{O}$-translation. Our contributions are mainly as follows:

- We extend the results of Cima and al. with existential existential rules instead of DL-LITE.
- We identify the conditions under which a (minimally) complete rewriting exists in this extended setting (Theorem V.11, Section V.2.1).
- We introduce an algorithm to compute a minimally complete $\mathcal{M}/\Sigma$-translation within the target class $\text{UCQ}^{\neq,\mathbf{C}}$. Although this class is less expressive than some of the query languages seen in Section IV.2, it is still able to express any minimally complete Σ-translation (Algorithm 14, Section V.2.1).
- We propose an algorithm to compute a set of $\text{CQ}^{\neq,\mathbf{C}}$ that constitutes a maximally sound $\mathcal{M}$-translation. However, this set may be infinite, hence it is not necessarily a $\text{UCQ}^{\neq,\mathbf{C}}$ (Theorem V.21, Section V.2.3.1).
- We characterise the existence of a non-trivial sound $\mathcal{M}$-translation (Theorem V.22, Section V.2.3.1).
- We generalise the algorithm from [Cima et al., 2022] that computes a maximally sound monotone-Σ-translation[2] (see Proposition IV.11 as well as Appendix D.2). We propose an algorithm that outputs a set of disjunctive rules that constitutes a maximally sound Σ-translation when the set of rules is parallelisable. These rules are more precisely disjunctive rules with special predicates $\mathbf{C}$ and $\neq$ in their body. Instead of using an "ad hoc" notion of inverse mapping, we use the already-known notion of maximum recovery (Section V.2.3.2).

V.2.1 Minimally complete $\mathcal{S}$-to-$\mathcal{O}$-translation

In this section, we discuss the results about minimally complete translation. Here we treat both $\mathcal{M}$ and Σ-translation since we will see that we can use exactly the same algorithm.

On the Absence of Minimally Complete $\mathcal{S}$-to-$\mathcal{O}$-translations We saw in Section IV.2.2 that in [Cima et al., 2019, Cima et al., 2020, Cima et al., 2022], to ensure the existence of complete $\mathcal{S}$-to-$\mathcal{O}$-translations, special rules of the form $p(x_1,\ldots,x_n) \rightarrow \top(x_1) \wedge \ldots \wedge \top(x_n)$ are added to the mapping for each predicate $p \in V_{\mathcal{S}}$. In this work, we do not allow ourselves to alter the user's mapping. Indeed, the aim of the mapping is precisely to select relevant data: so, we do not import all data values. Therefore, in our framework, there is not always a complete translation: see Example V.9. Moreover, these special rules can lead to outcomes that are not informative and disconnected from the original query: see Example V.10.

[2] A maximally sound monotone-Σ-translation is a query which is a Σ-translation maximally sound in the class of monotone queries.

Example V.9: Why do complete translations not always exist?

Let $V_{\mathcal{S}} = \{s\text{-}cat(\cdot), s\text{-}owner(\cdot,\cdot)\}$ be a database schema about cats and their owners, and consider a KDBM specification $\Sigma = (V_{\mathcal{O}}, \emptyset, V_{\mathcal{S}}, \mathcal{M})$ with an empty ontology and a mapping $\mathcal{M} = \{s\text{-}cat(x) \rightarrow cat(x)\}$.
Now, let $q_{\mathcal{S}}(x) = s\text{-}cat(Felix) \wedge s\text{-}owner(Felix, x)$ be a CQ, which retrieves the owners of a cat named Felix.
In this example, since the data in the second column of the relation $owner(\cdot,\cdot)$ are not transferred by mapping to the ontology, it is impossible to retrieve the owners of $Felix$ since we know none of the owners at the ontological level. Therefore, there is no a complete translation.

Example V.10: Result of adding special rules in mapping

Take the same KBDM specification and CQ as in Example V.9. If we add to the mapping the special rule $s\text{-}owner(x,y) \rightarrow \top(x) \wedge \top(y)$ to $\mathcal{M}$, we would have the minimally complete $\mathcal{S}$-to-$\mathcal{O}$-translation $q_{\mathcal{O}}(x) = cat(Felix) \wedge \top(x)$, which is very odd. In fact, as soon as a cat named Felix would exist in the database, all constants in the database would be answers to $q_{\mathcal{O}}$, hence this would not give any relevant information about the original query and would have no utility in practice.

Our approach avoids producing translations that lack utility or a coherent connection with the original query. Users still have the flexibility to include these special rules if they prefer to ensure the existence of complete translations.

We characterise the conditions under which a complete translation exists. Briefly, each answer variable in a CQ must be the target of a frontier variable from a rule of the mapping by a homomorphim from the body of this rule to the CQ.

Theorem V.11: Existence of a complete $\mathcal{M}$-translation of a UCQ$^{\neq,\mathbf{C}}$

Let $\Sigma = (V_{\mathcal{O}}, \mathcal{R}_{\mathcal{O}}, V_{\mathcal{S}}, \mathcal{M})$ be a KBDM specification and $\mathcal{Q}_{\mathcal{S}}(\mathbf{x})$ be a UCQ$^{\neq,\mathbf{C}}$ over $V_{\mathcal{S}}$. There exists a complete $\mathcal{M}$-translation of $\mathcal{Q}_{\mathcal{S}}(\mathbf{x})$, which is a UCQ$^{\neq,\mathbf{C}}$, if and only if for all $q_i \in \mathcal{Q}_{\mathcal{S}}$ and for all $x \in \mathbf{x}$, there exist $m \in \mathcal{M}$ and a homomorphism h from $\mathsf{body}(m)$ to q_i such that $x \in h(\mathtt{fr}(m))$.

We first do the proof for a CQ$^{\neq,\mathbf{C}}$, then we extend the proof to UCQ$^{\neq,\mathbf{C}}$.

Lemma V.12: Existence of a complete $\mathcal{M}$-translation of a CQ$^{\neq,\mathbf{C}}$

Let $\Sigma = (V_\mathcal{O}, \mathcal{R}_\mathcal{O}, V_\mathcal{S}, \mathcal{M})$ be a KBDM specification and $q_\mathcal{S}(\mathbf{x})$ be a CQ$^{\neq,\mathbf{C}}$ over $V_\mathcal{S}$. There exists a complete $\mathcal{M}$-translation of $q_\mathcal{S}(\mathbf{x})$, which is a CQ$^{\neq,\mathbf{C}}$, if and only if for all $x \in \mathbf{x}$ there are $m \in \mathcal{M}$ and a homomorphism h from $\mathtt{body}(m)$ to $q_\mathcal{S}(\mathbf{x})$ such that $x \in h(\mathtt{fr}(m))$.

Proof. ($\Rightarrow$) We have a query $q_\mathcal{S}(\mathbf{x})$ and we know that there exists a complete $\mathcal{M}$-translation $q_\mathcal{O}(\mathbf{x})$. Since $q_\mathcal{O}(\mathbf{x})$ is complete, we have that, for all databases D, $q_\mathcal{S}(D) \subseteq q_\mathcal{O}(I_{(D,\mathcal{M})})$. We denote by $freeze$ a substitution that replaces each variable x by a fresh constant c_x. Let $\mathbf{t} = freeze(\mathbf{x})$ and $D_{q_\mathcal{S}} = freeze(\phi(q_\mathcal{S}))$[3]. $\mathbf{t}$ is an answer tuple in $q_\mathcal{S}(D_{q_\mathcal{S}})$ and so it is also in $q_\mathcal{O}(I_{(D_{q_\mathcal{S}},\mathcal{M})})$. Let $q'_\mathcal{S}(\mathbf{x}) = \mathcal{M}\text{–}\mathtt{rewriting}(q_\mathcal{O})$. Since $q'_\mathcal{S}(\mathbf{x})$ is a sound and complete rewriting of $q_\mathcal{O}$ (Theorem I.20), we know that $\mathbf{t} \in q'_\mathcal{S}(D_{q_\mathcal{S}})$. Thus there exists a homomorphism from $q'_\mathcal{S}(\mathbf{x})$ to $D_{q_\mathcal{S}}$ that sends $\mathbf{x}$ to $\mathbf{t}$. Since $D_{q_\mathcal{S}}$ is the query $q_\mathcal{S}(\mathbf{x})$ that was frozen, we have a query homomorphism h from $q'_\mathcal{S}(\mathbf{x})$ to $q_\mathcal{S}(\mathbf{x})$ that is the identity on $\mathbf{x}$. By construction, $q'_\mathcal{S}(\mathbf{x})$ is a conjunction of the specialisations of the bodies of rules of $\mathcal{M}$ and, in addition, all answer variables in $q'_\mathcal{S}(\mathbf{x})$ can only be from frontier variables of these rules since a variable of a body that is not in the frontier of a rule can only become an existential variable in a rewriting. Thus, we have a homomorphism h' from these rules' bodies to $q'_\mathcal{S}(\mathbf{x})$ that covers all the answer variables with frontier variables. By the transitivity of the homomorphism, we have a homomorphism $h'' = h' \circ h$ that sends these bodies to $q_\mathcal{S}(\mathbf{x})$, and for all $x \in \mathbf{x}$, there exists a frontier variable y in these bodies such that $h(y) = x$.

($\Leftarrow$) Let $q'(\mathbf{x}) = \phi(q_\mathcal{S})$ be a CQ. We have $q_\mathcal{S} \sqsubseteq q'$, so a complete $\mathcal{M}$-translation of q' will also be a complete $\mathcal{M}$-translation of $q_\mathcal{S}$. Let $q_\mathcal{O}(\mathbf{x}) = \mathcal{M}\text{–}\mathtt{chase}(q'(\mathbf{x}))$. It follows from the second part of the theorem that all the answer variables of q' are transferred to $q_\mathcal{O}(\mathbf{x})$. If $q_\mathcal{O}(\mathbf{x})$ is empty (which can happen if it is Boolean), it is equivalent to $\top$ (and so is complete). Otherwise, there exists a conjunctive query $q_{\mathcal{O}\to\mathcal{S}}(\mathbf{x})$ obtained by a finite sequence of rewriting steps from $q_\mathcal{O}$, such that there exists a query homomorphism h from $q_{\mathcal{O}\to\mathcal{S}}(\mathbf{x})$ to q' (Theorem III.14). Therefore, we have $q_\mathcal{S} \sqsubseteq q' \sqsubseteq q_{\mathcal{O}\to\mathcal{S}} \sqsubseteq q_\mathcal{O}$. Hence, $q_\mathcal{O}(\mathbf{x})$ is a complete $\mathcal{M}$-translation of $q_\mathcal{S}(\mathbf{x})$. ■

Proof of theorem V.11. We just have to apply the lemma V.12 on each CQ$^{\neq,\mathbf{C}}$ in $\mathcal{Q}_\mathcal{S}$. ■

The syntactic impossibility of constructing an empty CQ of arity different from 0 makes it impossible to define a non-Boolean UCQ whose answer set includes all tuples of constants from the considered instance. Consequently, any complete translation of a non-Boolean UCQ is informative in the sense that if it has no answer, the original query does not either. However, this is no longer true for Boolean UCQs. Moreover, a Boolean UCQ always has a complete translation, since it always satisfies the conditions of Theorem V.11. In the following theorem, we characterise Boolean UCQs that only

[3]We recall that $\phi(q_\mathcal{S})$ is the set of atoms in $q_\mathcal{S}$ which do not use special predicates (Section IV.1))

have complete translation containing an empty CQ. In this case, it is unnecessary to attempt to compute a complete translation.

Theorem V.13

Let $\Sigma = (V_\mathcal{O}, \mathcal{R}_\mathcal{O}, V_\mathcal{S}, \mathcal{M})$ be a KBDM specification and $\mathcal{Q}_\mathcal{S}(\mathbf{x})$ be a boolean UCQ$^{\neq,\mathbf{C}}$ over $V_\mathcal{S}$. Any UCQ$^{\neq,\mathbf{C}}$-complete translation of $\mathcal{Q}_\mathcal{S}$ is a UCQ$^{\neq,\mathbf{C}}$ containing the empty CQ if and only if there is a CQ $q_i \in \mathcal{Q}_\mathcal{S}$ such that there is no trigger from a rule in $\mathcal{M}$ to q_i.

Proof. We prove the contrapositive: For all CQ $q_i \in \mathcal{Q}_\mathcal{S}$, there exists a trigger from a rule in $\mathcal{M}$ to q_i if and only if there exists a UCQ$^{\neq,\mathbf{C}}$-complete translation that does not contain the empty CQ.

($\Rightarrow$) As a consequence of Theorem V.11, we will have no empty CQ in the complete $\mathcal{M}$-translation, since we have at least one trigger from each CQ.

($\Leftarrow$) As we shall see in Theorem V.14, there exist triggers from each CQ in $\mathcal{Q}_\mathcal{S}$ that allow one to compute this translation.

Now, we turn our attention to the computation of a minimally complete translation. Using Theorem V.11, the algorithm *TargetRewriting* from [Pérez, 2011] (Section IV.2.1) and Algorithm 4, we can build a new algorithm that computes a minimally complete $\mathcal{M}$-translation of a UCQ$^{\neq,\mathbf{C}}$.

Algorithm 14 systematically explores every specialisation of a given conjunctive query (CQ) within a UCQ. To do that, it uses Algorithm 13 that computes specialisations of the CQs in input by applying all possible substitutions from the variables to the terms of the CQs. It preserves the answer variables by introducing equalities. It adds inequalities between all distinct terms and special predicates $\mathbf{C}$ on all variables. This process transforms each CQ into a series of specialisations such that the union of all these specialisations is equivalent to the original CQ when considering only ground instances. Then, when chasing each CQ, Algorithm 14 achieves a minimally complete translation.

Theorem V.14: Minimally Complete $\mathcal{M}$-translation of a UCQ$^{\neq,\mathbf{C}}$

Let $\Sigma = (V_\mathcal{O}, \mathcal{R}_\mathcal{O}, V_\mathcal{S}, \mathcal{M})$ be a KBDM specification and $\mathcal{Q}_\mathcal{S}(\mathbf{x})$ be a UCQ$^{\neq,\mathbf{C}}$ over $V_\mathcal{S}$. Algorithm 14 computes a Minimally Complete $\mathcal{M}$-translation of $\mathcal{Q}_\mathcal{S}(\mathbf{x})$ into a UCQ$^{\neq,\mathbf{C}}$ query $\mathcal{Q}_\mathcal{O}(\mathbf{x})$ over $V_\mathcal{O}$, if a complete $\mathcal{M}$-translation exists.

Before doing the proof of Theorem V.14, we need to prove that Algorithm 13 returns a UCQ$^{\neq,\mathbf{C}}$ that has the same answers as the one in input when considering ground databases.

Algorithm 13: Specialise UCQ$^{\neq,\mathbf{C}}$

Data: A UCQ$^{\neq,\mathbf{C}}$ $\mathcal{Q}_\mathcal{S}$
Result: $UCQ^{\neq,\mathbf{C}}$ specialised versions of the CQs within $\mathcal{Q}_\mathcal{S}$ that are consistent
$\mathcal{Q}'_\mathcal{S} \leftarrow \emptyset$;
foreach $q_i[\mathbf{x}]$ *in* $\mathcal{Q}_\mathcal{S}$ **do**
  foreach *substitution* s *from* $\mathtt{vars}(q_i)$ *to* $\mathtt{terms}(q_i)$ *such that* $s(\mathtt{ansVars}(q_i)) \subseteq \mathtt{ansVars}(q_i)$ **do**
    $q'_i[\mathbf{x}] \leftarrow s(q_i)$;
    if q'_i *is consistent* **then**
      foreach *distinct pair of terms* t_1, t_2 *in* $\mathtt{terms}(s(q_i))$ **do**
        $q'_i \leftarrow q'_i \wedge t_1 \neq t_2$;
      end
      foreach $x \in \mathtt{ansVars}(q_i)$ *such that* $x \notin \mathtt{vars}(s(q_i))$ **do**
        $q'_i \leftarrow q'_i \wedge x = s(x)$;
      end
      $q'_i \leftarrow q'_i \wedge \mathbf{C}[\mathtt{vars}(q'_i)]$;
      $\mathcal{Q}'_\mathcal{S} \leftarrow \mathcal{Q}'_\mathcal{S} \cup q'_i$;
    end
  end
end
return $\mathcal{Q}'_\mathcal{S}$

Algorithm 14: Minimally Complete $\mathcal{M}$-translation of UCQ$^{\neq,\mathbf{C}}$

Input: KBDM specification $\Sigma = (V_\mathcal{O}, \mathcal{R}_\mathcal{O}, V_\mathcal{S}, \mathcal{M})$, UCQ $\mathcal{Q}_\mathcal{S}$ over $V_\mathcal{S}$
Output: Consistent UCQ$^{\neq,\mathbf{C}}$ query $\mathcal{Q}_\mathcal{O}$ over Σ if a complete translation exists
$\mathcal{Q}_\mathcal{O} \leftarrow \emptyset$;
// specialise is Algorithm 13
foreach q_i *in* *specialise*$(\mathcal{Q}_\mathcal{S})$ **do**
  // The following if test the condition of Theorem V.11
  if q_i *has a complete* $\mathcal{M}$*-translation* **then**
    $q_\mathcal{O} \leftarrow \mathcal{M}$−chase$(q_i)$;
    Remove in $q_\mathcal{O}$ every $\neq$-atom and $\mathbf{C}$-atom that contains a variable that does not appear in $\phi(q_\mathcal{O})$;
    $\mathcal{Q}_\mathcal{O} \leftarrow \mathcal{Q}_\mathcal{O} \cup \{q_\mathcal{O}\}$;
  else
    return $\mathcal{Q}_\mathcal{S}$ has no complete translation
  end
end
return $\mathcal{Q}_\mathcal{O}$;

Lemma V.15

Let $\mathcal{Q}_S$ be a UCQ$^{\neq,\mathbf{C}}$ over V_S and let $\mathcal{Q}'_S = \texttt{specialise}(\mathcal{Q}_S)$ be the result of Algorithm 13. For any database D over V_S, $\mathcal{Q}_S(D) = \mathcal{Q}'_S(D)$.

To prove Lemma V.15, we use another Lemma from [Lembo et al., 2015].

Lemma V.16: from [Lembo et al., 2015]

Let D be a database, and let $\mathcal{Q}$ be a UCQ$^{\neq}$. Then, $\mathcal{Q}(D) = \text{Saturate}(\mathcal{Q})(D)$ where Saturate outputs the same result as Algorithm 13 but without the special predicate $\mathbf{C}$.

Proof of Lemma V.15. Note that adding the predicate $\mathbf{C}$ on variables of a CQ that is over V_S is without loss of generality since we consider only databases over V_S, which are ground instances. Since it is the only difference with the algorithm Saturate in [Lembo et al., 2015], we can conclude from Lemma V.16 that $\mathcal{Q}_S(D) = \mathcal{Q}'_S(D)$. ■

Now, we can prove Theorem V.14.

Proof of Theorem V.14. *(Termination)* Trivial.

(Completeness) We want to prove that, for every database D over V_S, $\mathcal{Q}_S(D) \subseteq \mathcal{Q}_\mathcal{O}(I_{(D,\mathcal{M})})$, that is, for every tuple of constants $\mathbf{c} \in \mathcal{Q}_S(D)$, we also have $\mathbf{c} \in \mathcal{Q}_\mathcal{O}(I_{(D,\mathcal{M})})$. By Lemma V.15, we have $\mathcal{Q}'_S = \text{specialise}(\mathcal{Q}_S)$ such that for all databases D over V_S, $\mathcal{Q}'_S(D) = \mathcal{Q}_S(D)$. This implies that, for each tuple of constants $\mathbf{c} \in \mathcal{Q}_S(D)$, there exists a CQ$^{\neq,\mathbf{C}}$ $q^i \in \mathcal{Q}'_S$ such that $\mathbf{c} \in q^i(D)$. So, we have a homomorphism h from q_i to D. As seen in the proof of Lemma V.12, we have that $q^i_\mathcal{O} = \mathcal{M}\text{–chase}(\phi(q^i))$ is a complete $\mathcal{M}$-translation of q^i. We should now prove that by adding inequalities and $\mathbf{C}$-atoms to $q^i_\mathcal{O}$, it remains a complete $\mathcal{M}$-translation of q^i. First, note that $q^i_\mathcal{O} = \{h^+_j(H) \mid (h_j, B \to H) \in \Pi(\mathcal{M}, q^i)\}$ (where Π is the set of all triggers (of the rules in $\mathcal{M}$ on q_i) as defined in Definition I.19) by the definition of a mapping rewriting. Let $q^{i,\mathbf{c}}_\mathcal{O} = \{(h \circ h_j)^+ \mid (h \circ h_j, B \to H) \in \Pi(\mathcal{M}, D)\}$. By construction, there exists a homomorphism h' from $q^i_\mathcal{O}$ to $q^{i,\mathbf{c}}_\mathcal{O}$ and $\mathbf{c} \in q^{i,\mathbf{c}}_\mathcal{O}(I_{(D,\mathcal{M})})$ due to a homomorphism h''. Let $h^\mathbf{c} = h'' \circ h'$. Clearly, it is a homomorphism from $q^i_\mathcal{O}$ to $I_{(D,\mathcal{M})}$ such that $h^\mathbf{c}(\mathbf{x}) = \mathbf{c}$. Moreover, for each $x \neq y \in q^i$ such that $x, y \in \texttt{vars}(q^i_\mathcal{O})$, we have $h^\mathbf{c}(x)$ and $h^\mathbf{c}(y)$ are two distinct constants in $I_{(D,\mathcal{M})}$ since these two variables were already sent on two distinct constants by h and that applying a trigger does not merge two terms together. And for each $\mathbf{C}(x) \in q^i$ such that $x \in \texttt{vars}(q^i_\mathcal{O})$ we have that $h^\mathbf{c}(x)$ is a constant since h maps x to a constant of D. This means we can define the CQ$^{\neq,\mathbf{C}}$ $q_\mathcal{O}$ with $\phi(q_\mathcal{O}) = q^i_\mathcal{O}$, to which we add all the inequalities and $\mathbf{C}$-atoms to $q^i_\mathcal{O}$ having the property of having their variables in $\texttt{vars}(q^i_\mathcal{O})$ without losing the answer $\mathbf{c}$. And since $q_\mathcal{O} \in \mathcal{Q}_\mathcal{O}$ by construction, $\mathcal{Q}_\mathcal{O}$ is a complete $\mathcal{M}$-translation.

(Minimality) We now prove that $\mathcal{Q}_\mathcal{O}$ is a minimally complete $\mathcal{M}$-translation of $\mathcal{Q}_S$, that is, for each query (irrespective of its class) $\mathcal{Q}'_\mathcal{O}$ that is a complete $\mathcal{M}$-translation,

we have $\mathcal{Q}_\mathcal{O} \sqsubseteq_\mathcal{M} \mathcal{Q}'_\mathcal{O}$. We prove this by contradiction. Let $\mathcal{Q}$ be a complete $\mathcal{M}$-translation such that $\mathcal{Q} \sqsubset_\mathcal{M} \mathcal{Q}_\mathcal{O}$. Then, there exists a database D over $V_\mathcal{S}$ such that $\mathbf{c} \in \mathtt{certain}_\mathcal{M}(\mathcal{Q}_\mathcal{O}, D) = \mathcal{Q}_\mathcal{O}(I_{(D,\mathcal{M})})$ but $\mathbf{c} \notin \mathtt{certain}_\mathcal{M}(\mathcal{Q}, D)$.

We prove that $\mathcal{Q}$ is, in fact, not complete. Since $\mathbf{c} \in \mathcal{Q}_\mathcal{O}(I_{(D,\mathcal{M})})$, there is $q^i_\mathcal{O} \in \mathcal{Q}_\mathcal{O}$ such that $\mathbf{c} \in q^i_\mathcal{O}(I_{(D,\mathcal{M})})$ - and so we have a homomorphism h^i from $q^i_\mathcal{O}$ to $I_{(D,\mathcal{M})}$ such that $h^i(\mathbf{x}) = \mathbf{c}$. $q^i_\mathcal{O}$ was obtained by chasing a $\mathrm{CQ}^{\neq,\mathbf{C}}$ $q^i_\mathcal{S}$ with $\mathcal{M}$. Let $D^i = freeze(h^i(\phi(q^i_\mathcal{S}(\mathbf{c}))))$. Then $\mathbf{c} \in q^i_\mathcal{S}(D^i)$ and considering that $q^i_\mathcal{O}$ is a complete $\mathcal{M}$-translation of $q^i_\mathcal{S}$, we also have $\mathbf{c} \in q^i_\mathcal{O}(I_{(D^i,\mathcal{M})})$.

Moreover, there exists a homomorphism h from $I_{(D^i,\mathcal{M})}$ to $I_{(D,\mathcal{M})}$. Indeed, $I_{(D^i,\mathcal{M})}$ is, by construction, isomorphic to $h^i(q^i_\mathcal{O})$ and we have $h^i(q^i_\mathcal{O}) \subseteq I_{(D,\mathcal{M})}$.

This implies $\mathtt{sol}_\mathcal{M}(D) \subseteq \mathtt{sol}_\mathcal{M}(D^i)$. Indeed, for each J such that $D \cup J \models_1 \mathcal{M}$, we have $I_{(D,\mathcal{M})}$ that can be mapped into J (because its isormophic interpretation is a universal model). Thus, $I_{(D^i,\mathcal{M})}$ can be mapped to J by the transitivity of the homomorphism. Therefore, $D^i \cup J \models_1 D^i \cup I_{(D^i,\mathcal{M})} \models_1 \mathcal{M}$.

Since $\mathbf{c} \notin \mathtt{certain}_\mathcal{M}(\mathcal{Q}, D)$, we have an instance in $\mathtt{sol}_\mathcal{M}(D)$ such that $\mathbf{c}$ is not an answer to $\mathcal{Q}$, which is also an instance in $\mathtt{sol}_\mathcal{M}(D^i)$. Therefore, $\mathbf{c} \notin \mathtt{certain}_\mathcal{M}(\mathcal{Q}, D_i)$ but $\mathbf{c} \in \mathcal{Q}_\mathcal{S}(D^i)$: $\mathcal{Q}$ is not a complete $\mathcal{M}$-translation of $\mathcal{Q}_\mathcal{S}$. ■

The following proposition extends the first point of Proposition II.4 to the case where the ontological query is monotone.

Proposition V.17: Link between complete $\mathcal{M}$ and Σ-translation

Let $\Sigma = (V_\mathcal{O}, \mathcal{R}_\mathcal{O}, V_\mathcal{S}, \mathcal{M})$ be a KBDM specification. If $\mathcal{Q}_\mathcal{O}(\mathbf{x})$ is a monotone query and a complete $\mathcal{M}$-translation of a query $\mathcal{Q}_\mathcal{S}(\mathbf{x})$ over $V_\mathcal{S}$ with respect to Σ, then $\mathcal{Q}_\mathcal{O}(\mathbf{x})$ is also a complete Σ-translation of $\mathcal{Q}_\mathcal{S}(\mathbf{x})$.

In other words, the complete translation with respect to a set of mappings $\mathcal{M}$ remains complete when taking into account the entire KBDM specification.

Proof. Let $\mathcal{Q}_\mathcal{O}(\mathbf{x})$ be a complete $\mathcal{M}$-translation of a query $\mathcal{Q}_\mathcal{S}(\mathbf{x})$ with respect to $\Sigma = (V_\mathcal{O}, \mathcal{R}_\mathcal{O}, V_\mathcal{S}, \mathcal{M})$. We want to show that $\mathcal{Q}_\mathcal{O}(\mathbf{x})$ is also a complete Σ-translation of $\mathcal{Q}_\mathcal{S}(\mathbf{x})$.

Since $\mathcal{Q}_\mathcal{O}(\mathbf{x})$ is a complete $\mathcal{M}$-translation, it captures all the answers that can be obtained by applying the rules in $\mathcal{M}$.

Now, consider the entire KBDM specification, including $\mathcal{R}_\mathcal{O}$. Taking into account these rules can only add answers to the query. It cannot remove answers, since the rules in $\mathcal{R}_\mathcal{O}$ only allow one to deduce new facts and $\mathcal{Q}_\mathcal{O}(\mathbf{x})$ is monotone, the set of answers can only grow when new facts are added to an instance. Thus, for every database D, we have $\mathcal{Q}_\mathcal{S}(D) \subseteq \mathtt{certain}_\mathcal{M}(\mathcal{Q}_\mathcal{O}, D) \subseteq \mathtt{certain}_\Sigma(\mathcal{Q}_\mathcal{O}, D)$.

Hence, $\mathcal{Q}_\mathcal{O}(\mathbf{x})$ remains complete when taking into account the entire KBDM specification, and so it is a complete Σ-translation of $\mathcal{Q}_\mathcal{S}(\mathbf{x})$. ■

Moreover, the result of Algorithm 14 is not only a minimally complete $\mathcal{M}$-translation, it is also a minimally complete Σ-translation.

Theorem V.18: Minimally Complete Σ-translation of a UCQ$^{\neq,\mathbf{C}}$

Let $\Sigma = (V_\mathcal{O}, \mathcal{R}_\mathcal{O}, V_\mathcal{S}, \mathcal{M})$ be a KBDM specification and $\mathcal{Q}_\mathcal{S}(\mathbf{x})$ be a UCQ$^{\neq,\mathbf{C}}$ over $V_\mathcal{S}$. Algorithm 14 computes a Minimally Complete Σ-translation of $\mathcal{Q}_\mathcal{S}(\mathbf{x})$ (for any class of query), that is a UCQ$^{\neq,\mathbf{C}}$ query $\mathcal{Q}_\mathcal{O}(\mathbf{x})$ over $V_\mathcal{O}$, if a complete Σ-translation exists.

To do the proof, we use an extension of the notion of solutions to a mapping to an entire KBDM specification: $\mathtt{sol}_\Sigma(D) = \{I \mid I \text{ is an atomset on } V_\mathcal{T} \text{ and } (D \cup I) \models_1 \mathcal{M} \cup \mathcal{R}_\mathcal{O}\}$.

Proof. *(Completeness)* It is just a consequence of Proposition V.17.

(Minimality) We do a proof similar to the proof of Theorem V.14: we prove that $\mathcal{Q}_\mathcal{O}$ is a minimally complete Σ-translation of $\mathcal{Q}_\mathcal{S}$, that is, for each query (irrespective of its class) $\mathcal{Q}'_\mathcal{O}$ that is a complete Σ-translation, we have $\mathcal{Q}_\mathcal{O} \sqsubseteq_\Sigma \mathcal{Q}'_\mathcal{O}$. We prove this by contradiction. Let $\mathcal{Q}$ be a complete Σ-translation such that $\mathcal{Q} \sqsubset_\Sigma \mathcal{Q}_\mathcal{O}$. Then, there exists a database D over $V_\mathcal{S}$ such that $\mathbf{c} \in \mathtt{certain}_\Sigma(\mathcal{Q}_\mathcal{O}, D) = \mathcal{Q}_\mathcal{O}(\alpha_\infty(I_{(D,\mathcal{M})}, \mathcal{R}_\mathcal{O}))$ but $\mathbf{c} \notin \mathtt{certain}_\Sigma(\mathcal{Q}, D)$.

We prove that $\mathcal{Q}$ is, in fact, not complete. Since $\mathbf{c} \in \mathcal{Q}_\mathcal{O}(\alpha_\infty(I_{(D,\mathcal{M})}, \mathcal{R}_\mathcal{O}))$, there is $q^i_\mathcal{O} \in \mathcal{Q}_\mathcal{O}$ such that $\mathbf{c} \in q^i_\mathcal{O}(\alpha_\infty(I_{(D,\mathcal{M})}, \mathcal{R}_\mathcal{O}))$ - and so we have a homomorphism h^i from $q^i_\mathcal{O}$ to $\alpha_\infty(I_{(D,\mathcal{M})}, \mathcal{R}_\mathcal{O})$ such that $h^i(\mathbf{x}) = \mathbf{c}$. $q^i_\mathcal{O}$ was obtained by chasing a CQ$^{\neq,\mathbf{C}}$ $q^i_\mathcal{S}$ with $\mathcal{M}$. Let $D^i = freeze(h^i(\phi(q^i_\mathcal{S}(\mathbf{c}))))$. Then $\mathbf{c} \in q^i_\mathcal{S}(D^i)$ and considering that $q^i_\mathcal{O}$ is a complete $\mathcal{M}$-translation of $q^i_\mathcal{S}$, we also have $\mathbf{c} \in q^i_\mathcal{O}(I_{(D^i,\mathcal{M})})$.

Moreover, there exists a homomorphism h from $I_{(D^i,\mathcal{M})}$ to $\alpha_\infty(I_{(D,\mathcal{M})}, \mathcal{R}_\mathcal{O})$. Indeed, $I_{(D^i,\mathcal{M})}$ is, by construction, isomorphic to $h^i(q^i_\mathcal{O})$ and we have $h^i(q^i_\mathcal{O}) \subseteq \alpha_\infty(I_{(D,\mathcal{M})}, \mathcal{R}_\mathcal{O})$.

This implies $\mathtt{sol}_\Sigma(D) \subseteq \mathtt{sol}_\Sigma(D^i)$. First, note that for every $J \in \mathtt{sol}_\Sigma(D)$, we have $J \models \mathcal{R}_\mathcal{O}$ (since $\mathcal{R}_\mathcal{O}$ uses only the same vocabulary as J). Moreover, for each J such that $D \cup J \models_1 \mathcal{M} \cup \mathcal{R}_\mathcal{O}$, we have $\alpha_\infty(I_{(D,\mathcal{M})}, \mathcal{R}_\mathcal{O})$ that can be mapped to J (because its isormophic interpretation is a universal model). Thus, $I_{(D^i,\mathcal{M})}$ can be mapped to J by the transitivity of the homomorphism. Therefore, $D^i \cup J \models_1 D^i \cup I_{(D^i,\mathcal{M})} \models_1 \mathcal{M}$. And since we know that $J \models_1 \mathcal{R}_\mathcal{O}$, we have $D^i \cup J \models_1 \mathcal{M} \cup \mathcal{R}_\mathcal{O}$.

Since $\mathbf{c} \notin \mathtt{certain}_\Sigma(\mathcal{Q}, D)$, we have an instance in $\mathtt{sol}_\Sigma(D)$ such that $\mathbf{c}$ is not an answer to $\mathcal{Q}$, which is also an instance in $\mathtt{sol}_\Sigma(D^i)$. Therefore, $\mathbf{c} \notin \mathtt{certain}_\Sigma(\mathcal{Q}, D^i)$ but $\mathbf{c} \in \mathcal{Q}_\mathcal{S}(D^i)$: $\mathcal{Q}$ is not a complete Σ-translation of $\mathcal{Q}_\mathcal{S}$. ■

V.2.2 Perfect $\mathcal{S}$-to-$\mathcal{O}$-translation

We already introduced how to compute a perfect $\mathcal{S}$-to-$\mathcal{O}$-translation with Algorithm 6. In this algorithm, we first compute a minimally complete $\mathcal{S}$-to-$\mathcal{O}$-translation and then we check its soundness. Algorithm 15 and Algorithm 16 can respectively be used to check the soundness of a $\mathcal{M}$-translation and Σ-translation in which the set of rules is FUS.

Algorithm 15: Checking the Soundness of a $\mathcal{M}$-translation

Input: UCQ$^{\neq,\mathbf{C}}$ $\mathcal{Q}_\mathcal{S}$, $\mathcal{M}$-translation $\mathcal{Q}_\mathcal{O}$, Mapping $\mathcal{M}$
Output: `true` if $\mathcal{Q}_\mathcal{O}$ is a sound $\mathcal{M}$-translation of $\mathcal{Q}_\mathcal{S}$, `false` otherwise
return $\mathcal{M}$−`rewriting`$(\mathcal{Q}_\mathcal{O}) \sqsubseteq \mathcal{Q}_\mathcal{S}$;

Algorithm 16: Checking the Soundness of a Σ-translation

Input: KBDM specification $\Sigma = (V_\mathcal{O}, \mathcal{R}_\mathcal{O}, V_\mathcal{S}, \mathcal{M})$ with $\mathcal{R}_\mathcal{O}$ being FUS, UCQ$^{\neq,\mathbf{C}}$ $\mathcal{Q}_\mathcal{S}$, Σ-translation $\mathcal{Q}_\mathcal{O}$
Output: `true` if $\mathcal{Q}_\mathcal{O}$ is a sound Σ-translation of $\mathcal{Q}_\mathcal{S}$, `false` otherwise
return $\mathcal{M}$−`rewriting`(`rewriting`$(\mathcal{Q}_\mathcal{O}, \mathcal{R}_\mathcal{O})) \sqsubseteq \mathcal{Q}_\mathcal{S}$;

Corollary V.19

If there exists a perfect $\mathcal{M}$-translation of a UCQ$^{\neq,\mathbf{C}}$ query, then Algorithm 6 computes it, taking Algorithm 14 to compute a minimally complete $\mathcal{M}$-translation.

Proof. Assume that there exists a perfect target $\mathcal{M}$ translation for a given UCQ$^{\neq,\mathbf{C}}$ query.

First, from Algorithm 14, we know that it computes the minimally complete $\mathcal{M}$-translation.

Now, suppose for contradiction that Algorithm 14 does not compute the perfect $\mathcal{S}$-to-$\mathcal{O}$-translation when it exists. Since a perfect translation is, by Proposition II.2, minimally complete, it would mean that it is more minimal than the result of Algorithm 14. This leads to a contradiction.

Thus, the existence of a perfect target $\mathcal{M}$ translation ensures that Algorithm 6, used with Algorithm 15 and Algorithm 14, indeed computes it. ■

The previous corollary can be easily extended to the case where the set of rules in a KBDM specification is FUS.

Corollary V.20

If there exists a perfect Σ-translation of a UCQ$^{\neq,\mathbf{C}}$ query, then Algorithm 6 computes it, taking Algorithm 14 to compute a minimally complete Σ-translation.

Proof. The proof is the same as V.19, but with Algorithm 15 instead of Algorithm 16.■

V.2.3 Maximally sound $\mathcal{S}$-to-$\mathcal{O}$-translation

In this section, we will exploit the connection between a maximally sound $\mathcal{S}$-to-$\mathcal{O}$-translation and disjunctive mapping rewriting through a maximum recovery. We will see how this helps to understand some odd properties of this kind of translation and what are the remaining challenges to obtain an algorithm that computes a UCQ that is a maximally sound $\mathcal{S}$-to-$\mathcal{O}$-translation when there exists one.

V.2.3.1 $\mathcal{M}$-translation

By a slight abuse of notation, we use "maximally sound $\mathcal{M}$-translation" to talk about a query that is not always a UCQ but can be an infinite set of CQs (seen as an infinite union, that is, like with UCQs, the answers to the set of CQs is the union of all the answers to the CQs in the set).

Theorem V.21

Let $\mathcal{Q}_\mathcal{S}$ be a (U)CQ over $V_\mathcal{S}$, $\mathcal{M}$ be a conjunctive mapping from $V_\mathcal{S}$ to $V_\mathcal{O}$, and $\mathcal{M}'$ be a specification of a maximum recovery of $\mathcal{M}$. Then, $\mathcal{Q}_\mathcal{O} = \mathcal{M}'\text{-rewriting}_\vee(\mathcal{Q}_\mathcal{S})$ is a maximally sound $\mathcal{M}$-translation of $\mathcal{Q}_\mathcal{S}$ through $\mathcal{M}$ (for any query class).

Proof. Let $\mathcal{M}_\mathcal{A}$ and $\mathcal{M}'_\mathcal{A}$ be the abstract mappings specified, respectively, by $\mathcal{M}$ and $\mathcal{M}'$.

(Soundness) By definition of a maximum recovery, we have, for all databases D, $\texttt{certain}_{\mathcal{M}_\mathcal{A}\bullet\mathcal{M}'_\mathcal{A}}(\mathcal{Q}_\mathcal{S}, D) \subseteq \mathcal{Q}_\mathcal{S}(D)$. So, to prove that we have $\texttt{certain}_{\mathcal{M}_\mathcal{A}}(\mathcal{Q}_\mathcal{O}, D) \subseteq \mathcal{Q}_\mathcal{S}(D)$ for all databases D (that is, $\mathcal{Q}_\mathcal{O}$ is a sound $\mathcal{M}$-translation of $\mathcal{Q}_\mathcal{S}$), we just have to prove $\texttt{certain}_{\mathcal{M}_\mathcal{A}\bullet\mathcal{M}'_\mathcal{A}}(\mathcal{Q}_\mathcal{S}, D) = \texttt{certain}_{\mathcal{M}_\mathcal{A}}(\mathcal{Q}_\mathcal{O}, D)$.

For all databases $D \in dom(\mathcal{M}_\mathcal{A})$:

$$
\begin{aligned}
\texttt{certain}_{\mathcal{M}_\mathcal{A}\bullet\mathcal{M}'_\mathcal{A}}(\mathcal{Q}_\mathcal{S}, D) &= \bigcap_{(D,D')\in\mathcal{M}_\mathcal{A}\bullet\mathcal{M}'_\mathcal{A}} \mathcal{Q}_\mathcal{S}(D') \text{ (by definition of certain answers)} \\
&= \bigcap_{(D,J)\in\mathcal{M}_\mathcal{A} \text{ and } (J,D')\in\mathcal{M}'_\mathcal{A}} \mathcal{Q}_\mathcal{S}(D') \text{ (by definition of composition)} \\
&= \bigcap_{(D,J)\in\mathcal{M}_\mathcal{A}} \left(\bigcap_{(J,D')\in\mathcal{M}'_\mathcal{A}} \mathcal{Q}_\mathcal{S}(D') \right) \text{ (by definition of intersection)} \\
&= \bigcap_{(D,J)\in\mathcal{M}_\mathcal{A}} \texttt{certain}_{\mathcal{M}'_\mathcal{A}}(\mathcal{Q}_\mathcal{S}, J) \text{ (by definition of certain answers)} \\
&= \bigcap_{(D,J)\in\mathcal{M}_\mathcal{A}} \mathcal{Q}_\mathcal{O}(J) \text{ (since it is a sound and complete rewriting)} \\
&= \texttt{certain}_{\mathcal{M}_\mathcal{A}}(\mathcal{Q}_\mathcal{O}, D)
\end{aligned}
$$

(Maximality) We do the proof by contradiction. We assume that there exists a query $\mathcal{Q}'_\mathcal{O}$, which is a sound $\mathcal{M}$-translation of $\mathcal{Q}_\mathcal{S}$ and there exists a database D such that $\texttt{certain}_{\mathcal{M}_\mathcal{A}}(\mathcal{Q}_\mathcal{O}, D) \subsetneq \texttt{certain}_{\mathcal{M}_\mathcal{A}}(\mathcal{Q}'_\mathcal{O}, D)$. This implies there is $\mathbf{c} \in \texttt{certain}_{\mathcal{M}_\mathcal{A}}(\mathcal{Q}'_\mathcal{O}, D)$ such that $\mathbf{c} \notin \texttt{certain}_{\mathcal{M}_\mathcal{A}}(\mathcal{Q}_\mathcal{O}, D)$. And since we previously proved that $\texttt{certain}_{\mathcal{M}_\mathcal{A}\bullet\mathcal{M}'_\mathcal{A}}(\mathcal{Q}_\mathcal{S}, D) = \texttt{certain}_{\mathcal{M}_\mathcal{A}}(\mathcal{Q}_\mathcal{O}, D)$, we have $\mathbf{c} \notin \texttt{certain}_{\mathcal{M}_\mathcal{A}\bullet\mathcal{M}'_\mathcal{A}}(\mathcal{Q}_\mathcal{S}, D)$. This implies there exists $(D, D') \in \mathcal{M}_\mathcal{A} \bullet \mathcal{M}'_\mathcal{A}$ such that $\mathbf{c} \notin \mathcal{Q}_\mathcal{S}(D')$. By Proposition IV.12, we have $\texttt{sol}_{\mathcal{M}_\mathcal{A}}(D') \subseteq \texttt{sol}_{\mathcal{M}_\mathcal{A}}(D)$. Thus, $\texttt{certain}_{\mathcal{M}_\mathcal{A}}(\mathcal{Q}'_\mathcal{O}, D) \subseteq \texttt{certain}_{\mathcal{M}_\mathcal{A}}(\mathcal{Q}'_\mathcal{O}, D')$. But since $\mathcal{Q}'_\mathcal{O}$ is a sound $\mathcal{M}$-translation of $\mathcal{Q}_\mathcal{S}$, we also have $\texttt{certain}_{\mathcal{M}_\mathcal{A}}(\mathcal{Q}'_\mathcal{O}, D') \subseteq \mathcal{Q}_\mathcal{S}(D')$.

Therefore, $\texttt{certain}_{\mathcal{M}_\mathcal{A}}(\mathcal{Q}'_\mathcal{O}, D) \subseteq \mathcal{Q}_\mathcal{S}(D')$, which contradicts $\mathbf{c} \in \texttt{certain}_{\mathcal{M}_\mathcal{A}}(\mathcal{Q}'_\mathcal{O}, D)$ and $\mathbf{c} \notin \mathcal{Q}_\mathcal{S}(D')$. ∎

It can happen that the only sound translation of a UCQ is the empty UCQ. Although this translation is sound, it provides no information about the translated query, since it never admits any answer (in the case of a Boolean query, it will always be false). The following theorem characterises the queries that have a non-empty sound translation. When this is not the case, it is unnecessary to try to compute a (non-empty) maximally sound translation.

Theorem V.22

Let $\mathcal{Q}_\mathcal{S}$ be a (U)CQ over $V_\mathcal{S}$ and $\mathcal{M}$ be a conjunctive mapping from $V_\mathcal{S}$ to $V_\mathcal{O}$ and $\mathcal{M}'$ be a maximum recovery of $\mathcal{M}$. Let $\mathcal{C}$ be the set of all constants that occur in $\mathcal{M}'$ and $\mathcal{Q}_\mathcal{S}$, or $\mathcal{C} = \{c\}$, with c a constant, if no constant occur in $\mathcal{M}'$ and $\mathcal{Q}_\mathcal{S}$.
Then, there exists a sound $\mathcal{M}$-translation of $\mathcal{Q}_\mathcal{S}$ that is not the empty UCQ if and only if $\mathcal{Q}_\mathcal{S}$ can be mapped to $\mathcal{M}'\text{–chase}_\vee(I_\mathcal{C})$ where $I_\mathcal{C}$ is the critical instance of $V_\mathcal{O}$ over $\mathcal{C}$.

Proof. It is a consequence of Theorem V.21 and Theorem III.25. ■

If this link between maximum recovery and maximally sound $\mathcal{M}$-translation is interesting, it does not allow to compute a finite maximally sound $\mathcal{M}$-translation (when it exists) by naively computing a disjunctive mapping rewriting through the maximum recovery.

Theorem V.23

There exist a mapping $\mathcal{M}$ from $V_\mathcal{S}$ to $V_\mathcal{O}$ and a CQ q on $V_\mathcal{S}$ such that there exists a finite maximally sound $\mathcal{M}$-translation of q but there exists no UCQ-rewriting of q through $\mathcal{M}'$ where $\mathcal{M}'$ is a maximum recovery of $\mathcal{M}$.

Proof. Let a mapping be

$$\mathcal{M} = \begin{cases} s_1(x) & \to q(x), \\ s_2(x) & \to t(x), \\ s_1(y) \wedge s_3(x,y) & \to p(x,y), \\ s_2(x) \wedge s_4(x,y) & \to p(x,y) \end{cases}$$

A maximum recovery of $\mathcal{M}$ is the mapping

$$\mathcal{M}' = \begin{cases} q(x) \wedge \mathbf{C}(x) & \to s_1(x), \\ t(x) \wedge \mathbf{C}(x) & \to s_2(x), \\ p(x,y) \wedge \mathbf{C}(x) \wedge \mathbf{C}(y) & \to (s_1(y) \wedge s_3(x,y)) \vee (s_2(x) \wedge s_4(x,y)) \end{cases}$$

■

Let $q() = s_1(u) \wedge s_2(u)$ be a CQ. Then, a maximally sound $\mathcal{M}$-translation of q is $q_\mathcal{O}() = q(u) \wedge t(u) \wedge \mathbf{C}(u)$ but we have:

$$\mathcal{Q}_\mathcal{O} = \mathcal{M}'\text{-rewriting}_\vee(q) = \bigvee_{i=0}^{n} q(u_0) \wedge \left(\bigwedge_{j=1}^{i} p(u_{j-1}, u_j) \right) \wedge t(u_n) \wedge \mathbf{C}[u_0, \ldots, u_n].$$

Indeed, we can map the only CQ in $\mathcal{M}\text{–}\texttt{rewriting}(q)$ to each CQ in $\mathcal{M}\text{–}\texttt{rewriting}(q_i)$, for each $q_i \in \mathcal{Q}_\mathcal{O}$. For example, we can see for $i = 1$:

$$\begin{aligned}\mathcal{Q}_1 &= \mathcal{M}\text{–}\texttt{rewriting}(q)\\ &= s_1(u_0) \wedge s_2(u_0) \wedge \mathbf{C}(u_0)\end{aligned}$$

$$\begin{aligned}\mathcal{Q}_2 &= \mathcal{M}\text{–}\texttt{rewriting}(q(u_0) \wedge p(u_0, u_1) \wedge t(u_1) \wedge \mathbf{C}(u_0) \wedge \mathbf{C}(u_1))\\ &= \begin{cases} s_1(u_0) \wedge s_2(u_1) \wedge s_1(u_1) \wedge s_3(u_0, u_1) \wedge \mathbf{C}(u_0) \wedge \mathbf{C}(u_1)\\ s_1(u_0) \wedge s_2(u_1) \wedge s_2(u_0) \wedge s_4(u_0, u_1) \wedge \mathbf{C}(u_0) \wedge \mathbf{C}(u_1)\end{cases}\end{aligned}$$

Thus, $\mathcal{Q}_\mathcal{O} \sqsubseteq_\mathcal{M} q_\mathcal{O}$ (in fact, they are even $\mathcal{M}$-equivalent).

Note that this observation does not contradict Theorem V.21 since the disjunctive rewriting through $\mathcal{M}'$ is effectively a maximally sound $\mathcal{M}$-translation through $\mathcal{M}$ but it can be an infinite set of CQs, even when there exists a UCQ that is a maximally sound $\mathcal{M}$-translation. This is due to an improper way of computing the cover of the result. We need not compute the cover with the classical query containment, but with the $\mathcal{M}$-query containment.

Proposition V.24

Let $\mathcal{M}$ be a conjunctive mapping. If a set of queries $\mathcal{Q}_1$ is $\mathcal{M}$-equivalent to a UCQ $\mathcal{Q}_2$, then $\mathcal{Q}_1$ admits a finite $\mathcal{M}$-cover (i.e., an $\mathcal{M}$-equivalent finite subset).

Proof. The UCQ $\mathcal{Q}_2$ has a finite $\mathcal{M}$-rewriting since $\mathcal{M}$ is conjunctive and non-recursive. If $\mathcal{Q}_1$ and $\mathcal{Q}_2$ are $\mathcal{M}$-equivalent, then $\mathcal{M}\text{–}\texttt{rewriting}(\mathcal{Q}_1)$ and $\mathcal{M}\text{–}\texttt{rewriting}(\mathcal{Q}_2)$ are equivalent (by Proposition I.29).

Since $\mathcal{M}\text{-rewriting}_\vee(\mathcal{Q}_2)$ is finite, it has a finite cover, and $\mathcal{M}\text{-rewriting}_\vee(\mathcal{Q}_1)$ has a finite cover of the same size (Theorem 1 from [König et al., 2015]).

Let C_1 be a finite cover of $\mathcal{M}\text{-rewriting}_\vee(\mathcal{Q}_1)$. Each CQ in C_1 is (equivalent to a CQ) obtained by rewriting a CQ of $\mathcal{Q}_1$ (e.g., by piece-rewriting). Let $\mathcal{Q}'_1$ be the finite subset of $\mathcal{Q}_1$ defined in this way, that is, such that $\mathcal{M}\text{-rewriting}_\vee(\mathcal{Q}'_1)$ is equivalent to C_1.

Since $\mathcal{M}\text{-rewriting}_\vee(\mathcal{Q}'_1)$ is equivalent to $\mathcal{M}\text{-rewriting}_\vee(\mathcal{Q}_1)$, $\mathcal{Q}'_1$ is $\mathcal{M}$-equivalent to $\mathcal{Q}_1$. Therefore, $\mathcal{Q}_1$ admits a finite $\mathcal{M}$-cover. ■

Next Theorem V.25 uses the notion of disjunctive mapping rewriting with a special cover, denoted by $\mathcal{M}_\vee\text{-rewriting}_\vee^{\sqsubseteq_X}(\mathcal{Q})$ in Definition III.16: this is an inclusion-minimal subset of $\mathcal{M}_\vee\text{-rewriting}_\vee(\mathcal{Q})$ w.r.t. $\sqsubseteq_X$. Here, we specifically use the containment relation $\sqsubseteq_\mathcal{M}$. The theorem states two key results: (1) one can still obtain a maximally sound $\mathcal{M}$-translation through the maximum recovery of $\mathcal{M}$ when considering the $\mathcal{M}$-cover of the rewriting, (2) this $\mathcal{M}$-cover ensures that we get a finite maximally sound $\mathcal{M}$-translation if and only if there exists one in $\text{UCQ}^{\neq,\mathbf{C}}$.

Theorem V.25

Let $\mathcal{Q}_\mathcal{S}$ be a (U)CQ over $V_\mathcal{S}$, $\mathcal{M}$ be a conjunctive mapping from $V_\mathcal{S}$ to $V_\mathcal{O}$ and $\mathcal{M}'$ be a maximum recovery without equalities of $\mathcal{M}$. Then:

1. $\mathcal{M}'$-rewriting$_\vee^{\sqsubseteq,\mathcal{M}}(\mathcal{Q}_\mathcal{S})$ is a maximally sound $\mathcal{M}$-translation of $\mathcal{Q}_\mathcal{S}$.
2. If there exists a maximally sound $\mathcal{M}$-translation of $\mathcal{Q}_\mathcal{S}$ in UCQ$^{\neq,\mathbf{C}}$, then $\mathcal{M}'$-rewriting$_\vee^{\sqsubseteq,\mathcal{M}}(\mathcal{Q}_\mathcal{S})$ is a UCQ$^{\neq,\mathbf{C}}$.

Proof. This theorem is a direct consequence of the previous ones. By Theorem V.21, we know that $\mathcal{Q}_\mathcal{O} = \mathcal{M}'$-rewriting$_\vee(\mathcal{Q}_\mathcal{S})$ is a maximally sound $\mathcal{M}$-translation of $\mathcal{Q}_\mathcal{S}$ through $\mathcal{M}$. Additionally, Proposition V.24 shows that the $\mathcal{M}$-cover computation is properly handled. ■

The following examples illustrate the previous theorem. We saw, in the example used in the proof of Theorem V.23, a case where we were not able to compute the finite maximally sound translation: Example V.26 shows that with the $\mathcal{M}$-cover, we can compute this finite maximally sound translation. Example V.27 illustrates that, as we can expect, there are still cases where we cannot have a finite maximally sound translation.

Example V.26

We take the example of the proof of Theorem V.23:

$$\mathcal{M} = \begin{cases} s_1(x) & \rightarrow q(x), \\ s_2(x) & \rightarrow t(x), \\ s_1(y) \wedge s_3(x,y) & \rightarrow p(x,y), \\ s_2(x) \wedge s_4(x,y) & \rightarrow p(x,y). \end{cases} \tag{V.1}$$

A maximum recovery of $\mathcal{M}$ is the following mapping:

$$\mathcal{M}' = \begin{cases} q(x) \wedge \mathbf{C}(x) & \rightarrow s_1(x), \\ t(x) \wedge \mathbf{C}(x) & \rightarrow s_2(x), \\ p(x,y) \wedge \mathbf{C}(x) \wedge \mathbf{C}(y) & \rightarrow (s_1(y) \wedge s_3(x,y)) \vee (s_2(x) \wedge s_4(x,y)). \end{cases} \tag{V.2}$$

Let $q_\mathcal{S} = s_1(u) \wedge s_2(u)$ be a CQ. Then, a maximally sound $\mathcal{M}$-translation of $q_\mathcal{S}$ is $q_\mathcal{O}() = \mathcal{M}'$-rewriting$_\vee^{\sqsubseteq,\mathcal{M}}(q_\mathcal{S}) = q(u) \wedge t(u) \wedge \mathbf{C}(u)$.
In this example, there is a finite maximally sound $\mathcal{M}$-translation.

Example V.27

We take again the mapping of the example of the proof of Theorem V.23 but with a slight modification: we add $s_5(x)$ to the body of $s_2(x) \rightarrow t(x)$ - we this modification, we will have an infinite number of incomparable CQs with respect to $\mathcal{M}$ in the rewriting.

$$\mathcal{M} = \begin{cases} s_1(x) & \rightarrow q(x), \\ s_2(x) \wedge s_5(x) & \rightarrow t(x), \\ s_1(y) \wedge s_3(x,y) & \rightarrow p(x,y), \\ s_2(x) \wedge s_4(x,y) & \rightarrow p(x,y). \end{cases}$$

A maximum recovery of $\mathcal{M}$ is the mapping

$$\mathcal{M}' = \begin{cases} q(x) \wedge \mathbf{C}(x) & \rightarrow s_1(x), \\ t(x) \wedge \mathbf{C}(x) & \rightarrow s_2(x) \wedge s_5(x), \\ p(x,y) \wedge \mathbf{C}(x) \wedge \mathbf{C}(y) & \rightarrow (s_1(y) \wedge s_3(x,y)) \vee (s_2(x) \wedge s_4(x,y)). \end{cases}$$

Let $q_{\mathcal{S}} = s_1(u) \wedge s_2(u)$ be a CQ. Then, there exists no UCQ that is a maximally sound $\mathcal{M}$-translation of $q_{\mathcal{S}}$. We can only describe it with the infinite set of CQs:

$$\mathcal{Q}_{\mathcal{O}} = \mathcal{M}'\text{-rewriting}_{\vee}^{\sqsubseteq_{\mathcal{M}}}(q) = \bigvee_{i=0}^{\infty} q(u_0) \wedge \left(\bigwedge_{j=1}^{i} p(u_{j-1}, u_j) \right) \wedge t(u_i) \wedge \mathbf{C}[u_0, \ldots, u_i].$$

If we take the two first CQs, we can see why they are incomparable with respect to $\mathcal{M}$:

$$\begin{aligned} \mathcal{Q}_1 &= \mathcal{M}\text{–}\texttt{rewriting}(q(u_0) \wedge t(u_0) \wedge \mathbf{C}(u_0)) \\ &= s_1(u_0) \wedge s_2(u_0) \wedge s_5(u_0) \wedge \mathbf{C}(u_0) \end{aligned}$$

$$\begin{aligned} \mathcal{Q}_2 &= \mathcal{M}\text{–}\texttt{rewriting}(q(u_0) \wedge p(u_0, u_1) \wedge t(u_1) \wedge \mathbf{C}(u_0) \wedge \mathbf{C}(u_1)) \\ &= \begin{cases} s_1(u_0) \wedge s_2(u_1) \wedge s_5(u_1) \wedge s_1(u_1) \wedge s_3(u_0, u_1) \wedge \mathbf{C}(u_0) \wedge \mathbf{C}(u_1) \\ s_1(u_0) \wedge s_2(u_1) \wedge s_5(u_1) \wedge s_2(u_0) \wedge s_4(u_0, u_1) \wedge \mathbf{C}(u_0) \wedge \mathbf{C}(u_1) \end{cases} \end{aligned}$$

We can see that because of $s_5(u_0)$, we cannot map the CQ in $\mathcal{Q}_1$ into all the CQ that are in $\mathcal{Q}_2$ and thus the answers to $\mathcal{Q}_2$ are not included in the answers to $\mathcal{Q}_1$. This example illustrates a case where the maximally sound $\mathcal{M}$-translation cannot be finite.

Now, we propose a solution to approximate a maximally sound $\mathcal{M}$-translation. Our approach is presented in Algorithm 17, which takes as input a UCQ$^{\neq,\mathbf{C}}$ $\mathcal{Q}_{\mathcal{S}}$, a mapping

$\mathcal{M}$, and an integer k serving as a depth limit.

The algorithm works in two main steps. First, it computes the maximum recovery of $\mathcal{M}$ without equalities, denoted by $\mathcal{M}'$. Then, it applies the disjunctive mapping rewriting $\mathcal{M}'$-rewriting$^{\sqsubseteq_{\mathcal{M}}}_{\vee}(\mathcal{Q}_{\mathcal{S}}, k)$ up to a depth of k, allowing us to compute an approximation of the maximally sound $\mathcal{M}$-translation in the form of a UCQ$^{\neq,\mathbf{C}}$.

The value of k can be adjusted to control the trade-off between precision and computational complexity. A higher value for k leads to an approximation that is closer to a maximally sound $\mathcal{M}$-translation but at the potential cost of increased computational time. Thanks to Theorem V.25, the algorithm ensures that the returned UCQ is a sound $\mathcal{M}$-translation and that the closer k is to its maximum value, the closer the result is to a maximally sound one.

Algorithm 17: Approximation of a Maximally Sound $\mathcal{M}$-translation

Input: A UCQ$^{\neq,\mathbf{C}}$ $\mathcal{Q}_{\mathcal{S}}$, a mapping $\mathcal{M}$, an integer k
Output: A UCQ UCQ$^{\neq,\mathbf{C}}$ that is a sound $\mathcal{M}$-translation
$\mathcal{M}' \leftarrow$ MaximumRecoveryWithoutEqualities($\mathcal{M}$) ; `// Algorithm 9`
$\mathcal{Q}_{\mathcal{O}} \leftarrow \mathcal{M}'$-rewriting$^{\sqsubseteq_{\mathcal{M}}}_{\vee}(\mathcal{Q}_{\mathcal{S}}, k)$; `// Limit the rewriting to k steps`
return $\mathcal{Q}_{\mathcal{O}}$;

Finally, we propose a solution to always compute a finite maximally sound $\mathcal{M}$-translation. Our approach is described in Algorithm 18, where we use the class of $R^{\vee}_{\neq,\mathbf{C}}$ rules (that is disjunctive rules with $\neq$ and $\mathbf{C}$ in the body) to form the desired translation.

The algorithm first computes a Maximum Recovery without equalities from the given mapping $\mathcal{M}$ (using Algorithm 9). Then, it constructs the desired rule set by adding queries from the UCQ to the Maximum Recovery. The result is an $R^{\vee}_{\neq,\mathbf{C}}$ rule set that provides a maximally sound $\mathcal{M}$-translation.

This solution presents a slight improvment of the algorithm in [Cima et al., 2022] that computes a $R^{\vee}_{\mathbf{K},\neq}$-Maximally Sound $\mathcal{M}$-translation (where $R^{\vee}_{\mathbf{K},\neq}$ is a set of disjunctive rules that can contain the operator $\mathbf{K}$ and the predicate $\neq$ in their body) of a UCQ (see Algorithm 20 in the appendix D.2). Our method does not require the creation of a new kind of inverse mapping, but instead uses the existing concept of maximum recovery. Moreover, our approach forgoes the use of the epistemic modal operator $\mathbf{K}$ and instead employs only the special predicate $\mathbf{C}$ to test if a term is a constant. This simplification offers a more practical and implementable solution. To evaluate such a rule set, we only need to use the disjunctive mapping chase, that is always finite. The answers will be the tuples in the atoms with the special predicate *Ans*.

Algorithm 18: Compute $R^{\vee}_{\neq,\mathbf{C}}$-Maximally Sound $\mathcal{M}$-translation

Input: A mapping $\mathcal{M}$, UCQ $\mathcal{Q}_\mathcal{S} = \{q^1_\mathcal{S}, \ldots, q^n_\mathcal{S}\}$ over $V_\mathcal{S}$
Output: $R^{\vee}_{\neq,\mathbf{C}}$ rule set $\mathcal{R}$
$\mathcal{R} \leftarrow$ MaximumRecoveryWithoutEqualities($\mathcal{M}$) ; // Algorithm 9
for $i = 1$ *to* n **do**
  $\mathcal{R} \leftarrow \mathcal{R} \cup \{q^i_\mathcal{S}(\mathbf{x}_i) \rightarrow \text{Ans}(\mathbf{x}_i)\}$;
end
return $\mathcal{R}$;

Alghough this algorithm always halts and outputs a solution, its interest in practice might be questioned. Indeed, the target query class of the translation is not the same as the source class and it is even not in the class of first-order queries, which means that (1) we can doubt the aid it provides in terms of understanding for a user accustomed to relational queries, and (2) it will not help to translate classical constraints as introduced in Chapter II.

V.2.3.2 Maximally sound Σ-translation

To compute maximally sound Σ-translations, a possible technique is to compose the rules of the ontology with the mapping to create a new mapping that allows to "forget" part of the ontology. A simple version of this technique was introduced for DL-lite$_\mathcal{R}$ ontologies and GAV mappings under the name $\mathcal{T}$-mapping [Rodriguez-Muro et al., 2013,Sequeda et al., 2014]. In [Buron et al., 2020], a similar technique is used for RDFS ontologies and GLAV mappings. In a nutshell, the heads of the mapping assertions are saturated with the axioms of $\mathcal{R}_\mathcal{O}$ that are Datalog, i.e., do not introduce fresh variables. Then, the Datalog rules of $\mathcal{R}_\mathcal{O}$ can be ignored when a query $\mathcal{Q}_\mathcal{O}$ is rewritten with $\mathcal{R}_\mathcal{O}$. When $\mathcal{R}_\mathcal{O}$ is RDFS, the whole query rewriting step with $\mathcal{R}_\mathcal{O}$ can be skipped. However, this works only for simple Datalog rules (where a rule body is restricted to an atom on variables that occur only once). Note that, as presented in Section IV.2.2, this technique is also used to compute maximally sound Σ translations with a DL-Lite$_{RDFS}$ ontology in [Cima et al., 2022].

This idea of compiling the ontology into the mappings has been generalized in [Buron et al., 2021]. In that paper they define the class of *parallelisable* existential rules, which ensure the following property: for any parallelisable (finite) rule set $\mathcal{R}$, for any instance I, there is a finite rule set $\mathcal{R}^\star$ (that can be obtained by composing rules of $\mathcal{R}$) such that $chase(I, \mathcal{R})$ is equivalent to $\text{chase}_1(I, \mathcal{R}^\star)$, i.e., $\text{saturation}_1(I, \mathcal{R}^\star)$.

Hence, when $\mathcal{M} \cup \mathcal{R}_\mathcal{O}$ is parallelisable, one can replace the two sets by a single set of mappings, say $\mathcal{M}^\star$, and replace the KBDM specification $\Sigma = (V_\mathcal{O}, \mathcal{R}_\mathcal{O}, V_\mathcal{S}, \mathcal{M})$ by $\Sigma' = (V_\mathcal{O}, \emptyset, V_\mathcal{S}, \mathcal{M}^\star)$.

Thus, when $\mathcal{M} \cup \mathcal{R}_\mathcal{O}$ is parallelisable, we can compute a Σ-translation by composing $\mathcal{R}_\mathcal{O}$ with $\mathcal{M}$ to obtain a mapping $\mathcal{M}^\star$ and then compute a $\mathcal{M}^\star$-translation. We assume that we have an algorithm *Compile* that composes the rules $\mathcal{R}_\mathcal{O}$ with $\mathcal{M}$ to create a new mapping $\mathcal{M}^\star$ equivalent to $\mathcal{M} \cup \mathcal{R}_\mathcal{O}$ where equivalence is with respect to possible input

databases, that is, for all D, $I_{(D,\mathcal{M}^\star)} \equiv chase(I_{(D,\mathcal{M})}, \mathcal{R}_\mathcal{O})$.

Algorithm 19 computes a Σ-translation when $\mathcal{M} \cup \mathcal{R}_\mathcal{O}$ is parallelisable.

Algorithm 19: Maximally Sound Σ-translation for parallelisable rules

Input: KBDM specification $\Sigma = (V_\mathcal{O}, \mathcal{R}_\mathcal{O}, V_\mathcal{S}, \mathcal{M})$ with $\mathcal{M} \cup \mathcal{R}_\mathcal{O}$ parallelisable, a UCQ $\mathcal{Q}_\mathcal{S}$

Output: Maximally sound Σ-translation of the input query

$\mathcal{M}^\star \leftarrow \text{Compile}(\mathcal{M}, \mathcal{R}_\mathcal{O})$; // Compose $\mathcal{R}_\mathcal{O}$ with $\mathcal{M}$ to create $\mathcal{M}^\star$ equivalent to $\mathcal{M} \cup \mathcal{R}_\mathcal{O}$

$\mathcal{Q}_\mathcal{O} \leftarrow \text{MaximallySoundMInverseTranslation}(\mathcal{M}^\star, \mathcal{Q}_\mathcal{S})$; // Compute a maximally sound $\mathcal{M}^\star$-translation

return $\mathcal{Q}_\mathcal{O}$;

V.3 Summary

Let us briefly review the content of this chapter and highlight our main contributions.

- **$\mathcal{O}$-to-$\mathcal{S}$-translation:** Using classical rewriting techniques, a perfect $\mathcal{M}^{-1}$-translation can always be computed (Theorem V.1). This can be extended in the obvious way to a perfect Σ^{-1}-translation if the rules are FUS (Proposition V.8). Our original contribution here is to show that, for given KBDM specification Σ and CQ q, even when we know that a perfect Σ^{-1}-translation of q exists, it is computationally infeasible to produce this translation (Theorem V.3). Note that our proof relies on an unpublished result by David Carral.
- **$\mathcal{S}$-to-$\mathcal{O}$-translation:**
  - We have stated in Theorem V.18 that the class $\text{UCQ}^{\mathbf{C},\neq}$ is able to express any minimally complete ($\mathcal{M}$ or Σ)-translation of a UCQ (and $\text{UCQ}^{\mathbf{C},\neq}$).
  - We found that a maximum recovery is exactly the notion required to compute $\mathcal{S}$-to-$\mathcal{O}$-translations.
  - In our framework, a complete $\mathcal{M}/\Sigma$-translation does not always exist. We give syntactic conditions that characterise when there is one (Theorem V.11) and when there is one that is not the UCQ containing the empty CQ (Theorem V.13). Note that this is independent from the target query language.
  - In contrast, a sound $\mathcal{M}/\Sigma$-translation always exists, but it may be trivial. We show that it is decidable to check if a nontrivial sound $\mathcal{M}$-translation exists (Theorem V.22).
  - We provide an algorithm that computes a minimally complete $\mathcal{M}/\Sigma$-translation of a $\text{UCQ}^{\mathbf{C},\neq}$, when a complete translation exists (Theorems V.14 and V.18). The property of being minimally complete is independent from the target query language.

- We show that when a maximally sound $\mathcal{M}$-translation of a UCQ$^{\mathbf{C},\neq}$ exists, it can be obtained by its mapping-rewriting with a maximum recovery of $\mathcal{M}$ (Theorems V.21 and V.25). Note that a maximum recovery is a disjunctive mapping, as studied in Chapter III. Two fundamental problems remain open: first, is the problem of determining whether a pair $(\mathcal{Q}, \mathcal{M})$ has a maximally sound $\mathcal{M}$-translation decidable? We conjecture that it is not, since determining whether $\mathcal{Q}$ has a finite mapping-rewriting with a disjunctive mapping is an undecidable problem. Second, we have no algorithm that halts when a (finite) maximally sound $\mathcal{M}$-translation exists. However, by limiting the number steps of the mapping rewriting algorithm (Section III.5.2), we can still approximate a maximally sound $\mathcal{M}$-translation: the more steps allowed, the closer we get to a maximally sound $\mathcal{M}$-translation (Theorem V.25 and Algorithm 17).

- We then consider maximally sound Σ-translations in the case of parallelisable rules. When the union of the mapping and the rule set of a KBDM specification is parallelisable, these can be composed to create a mapping equivalent to this union. This allows us to apply the techniques defined for the $\mathcal{M}$-translation (Algorithm 19).

Finally, the following tables indicate whether translations are computable or not, for the different kinds of translations and input/target query classes studied in this chapter. Sometimes computability is known up to a restriction. The parentheses in the notation of target query classes indicate that, when the input class is a subclass, the result remains in this subclass: UCQ($^{\mathbf{C},\neq}$) means that the result is a UCQ if the input is a UCQ, and (U)CQ means that the result is a CQ if the input is a CQ.

Ontology	Source	Translation	Computable
UCQ$^{\mathbf{C},\neq}$	UCQ($^{\mathbf{C},\neq}$)	perfect	yes

Table V.1: $\mathcal{M}^{-1}$-translation

Ontology	Source	Translation	Restriction	Computable
UCQ$^{\mathbf{C},\neq}$	UCQ($^{\mathbf{C},\neq}$)	perfect	$\mathcal{R}_{\mathcal{O}}$ FUS	yes
UCQ$^{\mathbf{C},\neq}$	UCQ($^{\mathbf{C},\neq}$)	perfect	A perfect translation exists	no

Table V.2: Σ^{-1}-translation

Source	Ontology	Translation	Computable
UCQ	(U)CQ	minimally complete UCQ[4]	yes
$\text{UCQ}^{\mathbf{C},\neq}$	$\text{UCQ}^{\mathbf{C},\neq}$	minimally complete	yes
$\text{UCQ}^{\mathbf{C},\neq}$	set of $\text{CQ}^{\mathbf{C},\neq}$	maximally sound	no
$\text{UCQ}^{\mathbf{C},\neq}$	$\text{UCQ}^{\mathbf{C},\neq}$	sound	yes
$\text{UCQ}^{\mathbf{C},\neq}$	$\text{UCQ}(^{\mathbf{C},\neq})$	perfect	yes

Table V.3: $\mathcal{M}$-translation

Source	Ontology	Translation	Restriction	Computable
UCQ	(U)CQ	minimally complete UCQ[4]		yes
$\text{UCQ}^{\mathbf{C},\neq}$	$\text{UCQ}^{\mathbf{C},\neq}$	minimally complete		yes
$\text{UCQ}^{\mathbf{C},\neq}$	set of $\text{CQ}^{\mathbf{C},\neq}$	maximally sound	parallelisable	no
$\text{UCQ}^{\mathbf{C},\neq}$	$\text{UCQ}^{\mathbf{C},\neq}$	sound	parallelisable	yes
$\text{UCQ}^{\mathbf{C},\neq}$	$\text{UCQ}(^{\mathbf{C},\neq})$	perfect	$\mathcal{R}_{\mathcal{O}}$ FUS	yes

Table V.4: Σ-translation

[4] from [Cima et al., 2019]

VI - Translation of constraints

In this chapter, we apply the constraint translation framework defined in Section II.4 to several classes of constraints commonly used in database theory and KR. To achieve this, we rely on the techniques of UCQ translation defined in Chapter V. In Section VI.1, we present the different types of constraints that are considered. Section VI.2 includes a review of existing literature on the translation of constraints. Then, we study the translation of constraints that are equivalence-stable (Definition II.9). We consider the translation of negative constraints (Section VI.3), a restricted kind of equality-generating dependency (Section VI.4) and a restricted kind of (disjunctive) tuple-generating dependency (Section VI.5).

VI.1 Considered classes of constraints

In this section, we delve into the specific classes of constraints that we will consider. These are negative constraints (denoted $\mathbb{C}^{-}$), Tuple-Generating Dependencies (TGD) (denoted $\mathbb{C}^{TGD}$), Equality-Generating Dependencies (EGD) (denoted $\mathbb{C}^{=}$), and lastly, Disjunctive Tuple-Generating Dependencies (DTGD) (denoted $\mathbb{C}^{DTGD}$).

In Chapter VI, we will present results about the following translations: $\mathbb{C}^{-}$-to-$\mathbb{C}^{-}$, $\mathbb{C}^{(D)TGD}$-to-$\mathbb{C}^{(D)TGD}$, and $\mathbb{C}^{=}$-to-$\mathbb{C}^{=}$.

Intuitively, a negative constraint specifies some contradiction, something that should not happen in the data or facts. This type of constraint is also known as *denial* constraints in database theory.

Definition VI.1 (Negative constraint)
A negative constraint C is a closed formula of the form

$$\forall \mathbf{x}(B[\mathbf{x}] \rightarrow \bot)$$

where the body of C, $B[\mathbf{x}]$, is a conjunction of atoms.

Note that, in examples, we leave universal quantifiers implicit, just as we do for rules.

Example VI.1: Negative constraint

Let $C^{-} = cat(x) \wedge dog(x) \rightarrow \bot$. This negative constraint specifies that an individual cannot be both a cat and a dog. Equivalently, we can write $\forall x\ (cat(x) \rightarrow \neg dog(x))$ (a cat cannot be a dog) or $\forall x\ (dog(x) \rightarrow \neg cat(x))$ (a dog cannot be a cat).

Intuitively, the following types of constraints specify mandatory information that should be in the data or facts.

Definition VI.2 (Tuple-Generating Dependency)
A *Tuple-Generating Dependency (TGD)* is a conjunctive rule (Def. I.10) that has the semantics of a constraint.

Example VI.2: Tuple-Generating Dependency

Let $C^+ = pet(x) \rightarrow \exists y\ hasName(x, y)$. This constraint states that, in our base, each pet must have a name.

The class of Tuple-Generating Dependencies has several important subclasses well studied in database theory. One of them is the class of *inclusion dependencies (ID)*, which are TGDs of the form $p(\mathbf{x}, \mathbf{y}) \rightarrow q(\mathbf{x}, \mathbf{y})$, with $\mathbf{x} \neq \emptyset$ and such that there is no repeated variable in the head and in the body. Intuitively, an ID describes the inclusion of some positions in the relation used in the body to some other positions of the relation used in the head. Inclusion dependencies can be used in particular to describe class hierarchies and *foreign keys*.

Example VI.3: Inclusion dependency

Let $C_1^+ = pet(x) \rightarrow animal(x)$ and $C_2^+ = hasName(x, y) \rightarrow pet(x)$. The constraint C_1^+ states that every pet is an animal (the class of pets is included in the class of animals) and C_2^+ describes that the first position of *hasName* must come from the first position of *pet*.

Another important class of positive constraints is the class of Equality-Generating Dependencies, which enforce that some values must be equal.

Definition VI.3 (Equality Generating Dependency)
An *Equality-Generating Dependency (EGD)* is a rule of the following form:

$$\forall x \forall y \forall \mathbf{z}(B[x, y, \mathbf{z}] \rightarrow x = y)$$

Example VI.4: Equality Generating Dependency

Let $C^= = hasName(x, y) \wedge hasName(x, z) \rightarrow y = z$. This constraint states that an individual in the base cannot have two names.

One common class of EGD is the class of functional dependencies. A *functional dependency (FD)* is an EGD of the form $p(\mathbf{x}, \mathbf{y}) \wedge p(\mathbf{x}, \mathbf{y}) \rightarrow y = z$ with $y \in \mathbf{y}$ and $z \in \mathbf{z}$. A common FD in databases is the *key dependency*.

The last class we consider is a generalisation of TGDs that allows for disjunction in the head.

Definition VI.4 (Disjunctive Tuple-Generating Dependency)
A Disjunctive Tuple-Generating Dependency (DTGD) is a closed formula of the form:

$$\forall \mathbf{x} \forall \mathbf{y}(B[\mathbf{x}, \mathbf{y}] \rightarrow \bigvee_{i=1}^{n} \exists \mathbf{z_i}\ H_i[\mathbf{x_i}, \mathbf{z_i}]) \text{ with } \mathbf{x_i} \subseteq \mathbf{x}$$

where $n \geq 1$, B and H_i are non-empty conjunctions of atoms with $\mathtt{vars}(B) = \mathbf{x} \cup \mathbf{y}$ and $\mathtt{vars}(H_i) = \mathbf{x_i} \cup \mathbf{z_i}$; furthermore $\mathbf{x} = \bigcup_{i=1}^{n} \mathbf{x_i}$.

Note that DTGDs are identical to the disjunctive existential rules (Definition III.1) that we will consider in Chapter III, but they have the semantics of constraints.

VI.2 Related work

In this section, we give some connections to related work on constraint translation.

In many studies, the database schema includes constraints, hence only the database instances that satisfy these constraints are considered. Indeed, we can assume that a source satisfies its own constraints.

Definition VI.5 (Legal instance)
A database D is a *legal instance* for a KBDM specification $\Sigma = (V_{\mathcal{O}}, \mathcal{R}_{\mathcal{O}}, V_{\mathcal{S}}, \mathcal{M})$ and a set of source constraints $\mathcal{C}_{\mathcal{S}}$ if $D \models_1 \mathcal{C}_{\mathcal{S}}$.

As illustrated in Section II.6.4, [Console et al., 2013] gives techniques of constraint translation in order to produce constraints that allow to optimise query rewriting. The idea is that we want constraints that will always be satisfied when considering only legal databases: these constraints can then help us to discard some rules when rewriting queries. This motivates the notion of valid constraints.

Definition VI.6 (Valid constraints)
For a given KBDM specification $\Sigma = (V_{\mathcal{O}}, \mathcal{R}_{\mathcal{O}}, V_{\mathcal{S}}, \mathcal{M})$, a set of constraints $\mathcal{C}_{\mathcal{O}}$ on $V_{\mathcal{O}}$ is said to be *valid* if for every legal instance D, $I_{(D,\mathcal{M})} \models_1 \mathcal{C}_{\mathcal{O}}$;

There is a direct link with the notion of the preservation of the satisfaction in our framework.

Theorem VI.5: Valid constraints & preservation of the satisfaction

Let $\Sigma = (V_{\mathcal{O}}, \mathcal{R}_{\mathcal{O}}, V_{\mathcal{S}}, \mathcal{M})$ be a KBDM specification, and $\mathcal{C}_{\mathcal{S}}$ be a constraint sets on $V_{\mathcal{S}}$. A constraint sets $\mathcal{C}_{\mathcal{O}}$ over $V_{\mathcal{O}}$ preserves the satisfaction of $\mathcal{C}_{\mathcal{S}}$ with respect to $\mathcal{M}$ if and only if it is valid with respect to $\mathcal{M}$.

Proof. ($\Rightarrow$) Since $\mathcal{C}_{\mathcal{O}}$ preserves the satisfaction of $\mathcal{C}_{\mathcal{S}}$ with respect to $\mathcal{M}$, we have, for each database D over $V_{\mathcal{S}}$, $D \models_1 \mathcal{C}_{\mathcal{S}}$ implies $I_{(D,\mathcal{M})} \models_1 \mathcal{C}_{\mathcal{O}}$. In other words, $\mathcal{C}_{\mathcal{O}}$ is satisfied for every legal database, which is the definition of valid constraints.

($\Leftarrow$) Since $\mathcal{C}_{\mathcal{O}}$ is valid, we have for all legal databases D, $I_{(D,\mathcal{M})} \models_1 \mathcal{C}_{\mathcal{O}}$. Since a database is legal, we have $D \models_1 \mathcal{C}_{\mathcal{S}}$ implies $I_{(D,\mathcal{M})} \models_1 \mathcal{C}_{\mathcal{O}}$, which is the definition of preservation of satisfaction. ■

The source constraints considered in [Console et al., 2013] are (foreign) key dependencies? and the mapping is GAV. The target class of constraints is the set of axioms expressible in a DL-Lite dialect (namely, DL-LITE$_{\mathcal{A}}$). They propose an algorithm to translate source constraints into valid constraints; their translation preserves satisfaction but not maximally.

Another related work is [Nikolaou et al., 2019]. This work considers axioms expressed in DL-Lite$_\mathcal{R}$ or in Datalog, and constraints expressed in different subclasses of TGDs. It defines a semantics for constraints in an OBDA framework and studies two problems, respectively called source-to-target and target-to-source implication (defined next: Problem VI.6. VI.7).

The semantics they give to constraints is not exactly the same as in this dissertation. They also select a unique interpretation on which satisfaction of constraints is checked, but this interpretation is the minimal model of the mapping and the ontology after skolemization (i.e., existential variables in rules heads are replaced by skolem functions whose arguments are the frontier variables). If we restrict the considered constraints to equivalence-stable constraints, the constraints are satisfied by a universal model of the mapping and ontology if and only if they are satisfied by the minimal model of the skolemized mapping and ontology.

Problem VI.6: Source-to-target constraint implication

Let $\Sigma = (V_\mathcal{O}, \mathcal{R}_\mathcal{O}, V_\mathcal{S}, \mathcal{M})$ be an OBDA specification with $\mathcal{M}$ and $\mathcal{R}_\mathcal{O}$ containing only skolemized rules, and $\mathcal{R}_\mathcal{O}$ is composed of DL-LITE$_\mathcal{R}$ axioms. We say that a constraint set $\mathcal{C}_\mathcal{S}$ defined on $V_\mathcal{S}$ implies a constraint $C_\mathcal{O}$ defined over $V_\mathcal{O}$ with respect to $\mathcal{M}$ and $\mathcal{R}_\mathcal{O}$ if for every database D and minimal model M of Σ, it holds that $M \models_1 C_\mathcal{O}$ whenever $D \models_1 \mathcal{C}_\mathcal{S}$.

Problem VI.7: Target-to-source constraint implication

Let $\Sigma = (V_\mathcal{O}, \mathcal{R}_\mathcal{O}, V_\mathcal{S}, \mathcal{M})$ be an OBDA specification with $\mathcal{M}$ and $\mathcal{R}_\mathcal{O}$ containing only skolemized rules, and $\mathcal{R}_\mathcal{O}$ is composed of DL-LITE$_\mathcal{R}$ axioms. We say that a constraint $\mathcal{C}_\mathcal{O}$ defined on $V_\mathcal{O}$ implies a constraint set $C_\mathcal{S}$ defined over $V_\mathcal{S}$ with respect to $\mathcal{M}$ and $\mathcal{R}_\mathcal{O}$ if for every database D and minimal model M of Σ, it holds that $D \models_1 C_\mathcal{S}$ whenever $M \models_1 \mathcal{C}_\mathcal{O}$.

Constraint implication is tightly related to our notion of constraint satisfaction preservation, again when considering equivalence-stable constraints, as stated next.

Theorem VI.8: Constraint implication & preservation of satisfaction

Let $C_\mathcal{S}$ be a constraint over $V_\mathcal{S}$, a set of constraints $\mathcal{C}_\mathcal{S}$ over $V_\mathcal{S}$, an equivalence-stable constraint $C_\mathcal{O}$ over $V_\mathcal{O}$ and a set of equivalence-stable constraints $\mathcal{C}_\mathcal{O}$ over $V_\mathcal{O}$. We have:

1. $\mathcal{C}_\mathcal{S}$ implies $C_\mathcal{O}$ if and only if $C_\mathcal{O}$ preserves the satisfaction of $\mathcal{C}_\mathcal{S}$;
2. $\mathcal{C}_\mathcal{O}$ implies $C_\mathcal{S}$ if and only if $C_\mathcal{S}$ preserves the satisfaction of $\mathcal{C}_\mathcal{O}$.

Proof. 1. The definition of "$\mathcal{C}_\mathcal{S}$ implies $C_\mathcal{O}$" is "it holds that $D \models_1 C_\mathcal{S}$ whenever $M \models_1 \mathcal{C}_\mathcal{O}$", which, when considering equivalence-stable constraints, is equivalent to "it

holds that $D \models_1 C_{\mathcal{S}}$ whenever $I_{(D,\Sigma)} \models_1 \mathcal{C}_{\mathcal{O}}$", which is exactly the preservation of the satisfaction of $\mathcal{C}_{\mathcal{S}}$ by $C_{\mathcal{O}}$.

2. The definition of "$\mathcal{C}_{\mathcal{O}}$ implies $C_{\mathcal{S}}$" is "it holds that $M \models_1 \mathcal{C}_{\mathcal{O}}$ whenever $D \models_1 \mathcal{C}_{\mathcal{S}}$", which, when considering equivalence-stable constraints, is equivalent to "it holds that $I_{(D,\Sigma)} \models_1 \mathcal{C}_{\mathcal{O}}$ whenever $D \models_1 C_{\mathcal{S}}$", which is exactly the preservation of the satisfaction of $\mathcal{C}_{\mathcal{O}}$ by $C_{\mathcal{S}}$. ■

The above work also determines the complexity of the problem of implication. Table VI.1 summarises the complexity of checking if a target constraint (i.e. a constraint over $V_{\mathcal{O}}$) is implied by a source constraint (i.e., a constraint over $V_{\mathcal{S}}$), depending on several classes of constraints and rules. Table VI.2 does the same for the complexity of checking if a source constraint is implied by a target constraint. As a corollary of Theorem VI.8, these complexities also apply to the problem of checking whether a source (resp. ontological) constraint preserves the satisfaction of an ontological (resp. source) constraint.

Ontology	**Target constraint**	**Source constraints**	**Complexity**
DL-LITE$_{\mathcal{R}}$	DTGD	Frontier-guarded TGDs[1]	2EXPTIME-COMPLETE[2]
		Full TGDs[3]	EXPTIME-COMPLETE[2]
		Linear TGDs[4]	PSPACE-COMPLETE[2]
		FDs	Π_2^p-COMPLETE[5]
Datalog rules	full TGD[3]	FDs	Undecidable[5]

For all the lines, the mapping is GAV.

Table VI.1: Source-to-target constraint implication complexities [Nikolaou et al., 2019]

Mapping	**Source constraint**	**Target constraints**	**Complexity**
Fr. guarded[1]	Fr. guarded DTGD[1]	Fr. guarded DTGDs[1]	3EXPTIME
Trivial[6]	ID	Guarded TGDs[7]	2EXPTIME-HARD
LAV	Linear TGD[4]	Full[3] & linear[4] TGDs	2EXPTIME-HARD
Fr. guarded[1]	Fr. guarded DTGD[1]	IDs	2EXPTIME-COMPLETE
Fr. guarded[1]	Full TGD[3]	Full TGDs[3]	Undecidable[8]

For all the lines, the ontology is full DL-LITE$_{\mathcal{R}}$, and the mapping is GAV.

Table VI.2: Target-to-source constraint implication complexities [Nikolaou et al., 2019]

[1] A (D)TGD / rule is frontier-guarded if an atom in the body contains all the frontier variables.
[2] Hardness holds even for a trivial mapping[6] and an empty ontology.
[3] A full TGD does not have existential variables, i.e., it is a Datalog rule seen as a constraint.

VI.3 Negative constraints

In this section, we present how to translate negative constraints and which type of constraints is produced for each type of translation.

How to translate negative constraints We have a direct link with the translation of UCQs: one can reduce the translation of a negative constraint to a translation of a Boolean CQ.

Theorem VI.9

Let I be an instance, C^- be a negative constraint , and $q = \text{body}(C^-)$ be a Boolean CQ . $I \models_1 C^-$ if and only if $I \not\models q$.

Proof. ($\Rightarrow$) If $I \models_1 C^-$, then we know that there is no homomorphism from $\text{body}(C^-)$ to I. Therefore, there is no homomorphism from q to I: $I \not\models q$.

($\Leftarrow$) If $I \not\models q$, then there is no homomorphism from $q = \text{body}(C^-)$ to I. Thus, $I \models_1 C^-$. ■

In the following, we call *CQ associated with a negative constraint* C^-, which we denote q^{C^-}, the Boolean CQ $\text{body}(C^-)$.

A corollary of Theorem VI.9 is that negative constraints are equivalence-stable (Definition II.9).

Corollary VI.10

Every negative constraint C^- is equivalence-stable.

Proof. It is a direct consequence of Theorem VI.9: since for each pair of equivalent instances (I_1, I_2), we have $q^{C^-}(I_1) = q^{C^-}(I_2)$, then we have $I_1 \models_1 C^-$ if and only if $I_2 \models_1 C^-$. ■

If we consider a set of negative constraints as a whole instead of translating negative constraints one by one, we may get a "more faithful" translation, as illustrated in Example VI.11. Therefore, it seems relevant to consider the translation from a set of negative constraints to another set of negative constraints.

[4] A (D)TGD is linear if its body is atomic.
[5] Hold even for a trivial mapping[6] and an empty ontology.
[6] A mapping is trivial if it is injective and contains only rules of the form $p(\mathbf{x}) \rightarrow q(\mathbf{x})$.
[7] A TGD is guarded if one atom of its body contains all the variables of the body.
[8] Hold even if the mapping is LAV.

Example VI.11: Translation of a set of negative constraints

Let $\mathcal{C}^-_{\mathcal{S}} = \{p(x) \rightarrow \bot, q(x) \rightarrow \bot\}$ be a set of negative constraints over $V_{\mathcal{S}}$ and a mapping $\mathcal{M} = \{q(x) \rightarrow t(x), p(x) \rightarrow t(x)\}$.
There exists no perfect $\mathcal{M}$-translation of any constraint in $\mathcal{C}^-_{\mathcal{S}}$ (and each translation that preserves satisfaction is trivial). However, $\mathcal{C}^-_{\mathcal{O}} = \{t(x) \rightarrow \bot\}$ is a perfect $\mathcal{M}$-translation of the entire set $\mathcal{C}^-_{\mathcal{S}}$.

We call *UCQ associated with a set of negative constraints* $\mathcal{C}^-$, the UCQ composed of the CQs associated with the negative constraints in $\mathcal{C}^-$, that is, $\mathcal{Q}^{\mathcal{C}^-} = \{q^{C^-} \mid C^- \in \mathcal{C}^-\}$.

We can generalise the result of Theorem VI.9 to a set of negative constraints.

Theorem VI.12

Let us consider a set of negative constraints $\mathcal{C}^-$. For any instance I, $I \models_1 \mathcal{C}^-$ if and only if $I \not\models_1 \mathcal{Q}^{\mathcal{C}^-}$.

Proof. ($\Rightarrow$) If $I \models_1 \mathcal{C}^-$, then, for each negative constraint $C^- \in \mathcal{C}^-$, $I \models_1 C^-$. Thus, by Theorem VI.9, we know that $I \not\models_1 q^{C^-}$. Therefore, there exists no CQ $q \in \mathcal{Q}^{\mathcal{C}^-}$ such that $I \models_1 q$ which implies $I \not\models_1 \mathcal{Q}^{\mathcal{C}^-}$.

($\Leftarrow$) The arguments are exactly the same but in the other direction. ■

A corollary of Theorem VI.9 is that sets of negative constraints are equivalence-stable as well.

Corollary VI.13

Every set of negative constraints $\mathcal{C}^-$ is equivalence-stable.

Proof. It is a direct consequence of Theorem VI.12: since for each pair of equivalent instances (I_1, I_2), we have $\mathcal{Q}^{\mathcal{C}^-}(I_1) = \mathcal{Q}^{\mathcal{C}^-}(I_2)$, then we have $I_1 \models_1 \mathcal{C}^-$ if and only if $I_2 \models_1 \mathcal{C}^-$. ■

Corollary VI.13 allows one to use the techniques defined in Chapter V since the semantics of negative constraints is the same on all the universal models (hence any of them can be chosen).

As we saw in Chapter V, there does not always exist a perfect translation of a (U)CQ. The following proposition clarifies the link between minimally complete/maximally sound translation of a CQ and translation of a negative constraint that maximally preserves satisfaction/minimally preserves violation.

Theorem VI.14

Let $\mathcal{C}^-_{\mathcal{S}}$ be a set of negative constraints on a vocabulary $V_{\mathcal{S}}$ and $\mathcal{C}^-_{\mathcal{O}}$ be a set of negative constraints on a vocabulary $V_{\mathcal{O}}$ disjoint from $V_{\mathcal{S}}$. For $X \in \{\mathcal{M}, \Sigma\}$, we have that:

1. $\mathcal{C}^-_{\mathcal{O}}$ maximally preserves satisfaction of $\mathcal{C}^-_{\mathcal{S}}$ wrt X if and only if $\mathcal{Q}^{\mathcal{C}^-_{\mathcal{O}}}$ is a maximally sound X-translation of $\mathcal{Q}^{\mathcal{C}^-_{\mathcal{S}}}$;
2. $\mathcal{C}^-_{\mathcal{S}}$ minimally preserves violation of $\mathcal{C}^-_{\mathcal{O}}$ wrt X if and only if $\mathcal{Q}^{\mathcal{C}^-_{\mathcal{S}}}$ is a minimally complete X^{-1}-translation of $\mathcal{Q}^{\mathcal{C}^-_{\mathcal{O}}}$;
3. $\mathcal{C}^-_{\mathcal{O}}$ minimally preserves violation of $\mathcal{C}^-_{\mathcal{S}}$ wrt X if and only if $\mathcal{Q}^{\mathcal{C}^-_{\mathcal{O}}}$ is a minimally complete X-translation of $\mathcal{Q}^{\mathcal{C}^-_{\mathcal{S}}}$;
4. $\mathcal{C}^-_{\mathcal{S}}$ maximally preserves satisfaction of $\mathcal{C}^-_{\mathcal{O}}$ wrt X if and only if $\mathcal{Q}^{\mathcal{C}^-_{\mathcal{S}}}$ is a maximally sound X^{-1}-translation of $\mathcal{Q}^{\mathcal{C}^-_{\mathcal{O}}}$;

Proof. First, note that, by definition, (1) and (2) are equivalent, and (3) and (4) are also equivalent. So, we prove them together.

(1) & (2) ($\Rightarrow$) By definition of maximally preserving the satisfaction, we have that for each instance D over $\mathcal{S}$, if $D \models_1 \mathcal{C}^-_{\mathcal{S}}$ implies $I_{(D,\Sigma)} \models_1 \mathcal{C}^-_{\mathcal{O}}$ and there is no strictly stronger constraint that preserves satisfaction. By Theorem VI.12, we know that "$D \models_1 \mathcal{C}^-_{\mathcal{S}}$ implies $I_{(D,\Sigma)} \models_1 \mathcal{C}^-_{\mathcal{O}}$" is equivalent to $D \not\models_1 \mathcal{Q}^{\mathcal{C}^-_{\mathcal{S}}}$ implies $I_{(D,\Sigma)} \not\models_1 \mathcal{Q}^{\mathcal{C}^-_{\mathcal{O}}}$ whose contrapositive is $I_{(D,\Sigma)} \models_1 \mathcal{Q}^{\mathcal{C}^-_{\mathcal{O}}}$ implies $D \models_1 \mathcal{Q}^{\mathcal{C}^-_{\mathcal{S}}}$. Then, since $I_{(D,\Sigma)}$ is a universal model, we have $\mathcal{Q}^{\mathcal{C}^-_{\mathcal{O}}}$ is a sound translation of $\mathcal{Q}^{\mathcal{C}^-_{\mathcal{S}}}$. Also, we know that there exists no constraint $\mathcal{C}^-$ that preserves satisfaction of $\mathcal{C}^-_{\mathcal{S}}$ that is strictly stronger than $\mathcal{C}^-_{\mathcal{O}}$. That is, there is no translation $\mathcal{C}^-$ that preserves satisfaction such that there exists a database D_2 with $I_{(D_2,\Sigma)} \not\models_1 \mathcal{C}^-$ and $I_{(D_2,\Sigma)} \models_1 \mathcal{C}^-_{\mathcal{O}}$. By Theorem VI.12, there is no $\mathcal{Q}^{\mathcal{C}^-}$ such that there exists a database D_2 with $I_{(D_2,\Sigma)} \models \mathcal{Q}^{\mathcal{C}^-}$ and $I_{(D_2,\Sigma)} \not\models \mathcal{Q}^{\mathcal{C}^-_{\mathcal{O}}}$, that is, $\mathcal{Q}^{\mathcal{C}^-_{\mathcal{O}}} \sqsubset_\Sigma \mathcal{Q}^{\mathcal{C}^-}$. We can conclude that $\mathcal{Q}^{\mathcal{C}^-_{\mathcal{O}}}$ is a maximally sound translation of $\mathcal{Q}^{\mathcal{C}^-_{\mathcal{S}}}$.

($\Leftarrow$) By definition of a maximally sound X-translation, we have that for each instance I over $\mathcal{S}$, if $I_{(D,\Sigma)} \models \mathcal{Q}^{\mathcal{C}^-_{\mathcal{O}}}$ implies $D \models \mathcal{Q}^{\mathcal{C}^-_{\mathcal{S}}}$, and there is no other sound X-translation that brings more answers. By Theorem VI.12, we know that if "$I_{(D,\Sigma)} \models \mathcal{Q}^{\mathcal{C}^-_{\mathcal{O}}}$ implies $D \models \mathcal{Q}^{\mathcal{C}^-_{\mathcal{S}}}$" then $I_{(D,\Sigma)} \not\models_1 \mathcal{C}^-_{\mathcal{O}}$ implies $D \not\models_1 \mathcal{C}^-_{\mathcal{S}}$ whose contrapositive is $D \models_1 \mathcal{C}^-_{\mathcal{S}}$ implies $I_{(D,\Sigma)} \models_1 \mathcal{C}^-_{\mathcal{O}}$, which the definition of preserving the satisfaction. Also, we know that there exists no UCQ $\mathcal{Q}$ that is a sound translation of $\mathcal{Q}^{\mathcal{C}^-_{\mathcal{S}}}$ such that $\mathcal{Q}^{\mathcal{C}^-_{\mathcal{O}}} \sqsubseteq_\Sigma \mathcal{Q}$. That is, there is no sound translation $\mathcal{Q}$ such that there exists a database D_2 with $I_{(D_2,\Sigma)} \not\models_1 \mathcal{Q}$ and $I_{(D_2,\Sigma)} \models_1 \mathcal{Q}^{\mathcal{C}^-_{\mathcal{O}}}$. By Theorem VI.12, there exists no $\mathcal{C}^-$ (with $\mathcal{Q}^{\mathcal{C}^-} = \mathcal{Q}$) such that there exists an instance $I_{(D_2,\Sigma)}$ with $I_{(D_2,\Sigma)} \models_1 \mathcal{C}^-$ and $I_{(D_2,\Sigma)} \not\models_1 \mathcal{C}^-_{\mathcal{O}}$. We can conclude that $\mathcal{C}^-_{\mathcal{O}}$ is a translation of $\mathcal{C}^-_{\mathcal{S}}$ that maximally preserves its satisfaction.

(3) & (4) The proof is the same as before but with the notions of minimally preserves violation and minimally complete translation. ■

Thanks to Theorem VI.14, all we need to translate negative constraints is to use the techniques of translation presented in chapter V.

Example VI.15: Translation of negative constraints

Take again Example II.3: let $\Sigma = (V_{\mathcal{O}}, \mathcal{R}_{\mathcal{O}}, V_{\mathcal{S}}, \mathcal{M})$ be a KBDM specification defined as below:

$$\mathcal{R}_{\mathcal{O}} = \{r(x,y) \to \exists z.p(y,z)\},$$
$$\mathcal{M} = \begin{cases} s_1(x,y) & \to p(x,y), \\ s_2(x,x) & \to \exists z.r(x,z), \\ s_3(x,y) & \to p(x,y) \end{cases}$$

We take the CQs of Example II.3 and transform them into negative constraints (i.e., a CQ q yields $q \to \bot$):

$\mathcal{O}$-to-$\mathcal{S}$-translation:

- The set of negative constraints $\mathcal{C}_{\mathcal{O}} = \{p(u,v) \to \bot\}$ can be translated into:
  - an $\mathcal{M}^{-1}$-translation that preserves satisfaction:
    $\mathcal{C}^1_{\mathcal{O}\to\mathcal{S}} = \{s_1(u,v) \to \bot\}$
  - a perfect $\mathcal{M}^{-1}$-translation:
    $\mathcal{C}^2_{\mathcal{O}\to\mathcal{S}} = \{s_1(u,v) \to \bot, s_3(u,v) \to \bot\}$
  - a perfect Σ^{-1}-translation:
    $\mathcal{C}^3_{\mathcal{O}\to\mathcal{S}} = \{s_1(u,v) \to \bot, s_3(u,v) \to \bot, s_2(u,u) \to \bot\}$

$\mathcal{S}$-to-$\mathcal{O}$-translation:

- The set of negative constraints $\mathcal{C}^1_{\mathcal{S}} = \{s_1(u,v) \to \bot\}$ can be translated into:
  - an $\mathcal{M}$-translation that minimally preserves violation:
    $\mathcal{C}^1_{\mathcal{S}\to\mathcal{O}} = \{p(u,v) \to \bot\}$
    (which is also a Σ-translation that minimally preserves violation), but the only translation preserving satisfaction is $\mathcal{Q}^2_{\mathcal{S}\to\mathcal{O}} = \{\}$, which equates to $\bot$.
- The set of negative constraints $\mathcal{C}^2_{\mathcal{S}} = \{s_2(u,v) \to \bot\}$ can be translated into:
  - an $\mathcal{M}$-translation that maximally preserves satisfaction:
    $\mathcal{C}^2_{\mathcal{S}\to\mathcal{O}} = \{r(u,w) \to \bot\}$

Class of the result of the translation In the following table (Table VI.3), we summarise the types of constraints produced by using the different techniques of UCQ-translation presented in Chapter V. We use $\mathbb{C}^-$ to denote the class of negative constraints (that is the class from which we translate) and $\mathbb{C}^-_{\mathbf{C},\neq}$ to denote negative constraints that can contain the predicates $\neq$ and $\mathbf{C}$ presented in Section IV.1 (with the same restrictions as in $\text{UCQ}^{\mathbf{C},\neq}$).

UCQ-translation type	Result	Constraint translation type	Restriction
Perfect $\mathcal{M}^{-1}$	$\mathbb{C}^-$	Perfect $\mathcal{M}^{-1}$	
Perfect Σ^{-1}	$\mathbb{C}^-$	Perfect Σ^{-1}	$\mathcal{R}_\mathcal{O}$ FUS
UCQ-minimally complete $\mathcal{M}$	$\mathbb{C}^-$	$\mathbb{C}^-$-Minimally preserve violation wrt $\mathcal{M}$	
Minimally complete $\mathcal{M}$	$\mathbb{C}^-_{\mathbf{C},\neq}$	Minimally preserve violation wrt $\mathcal{M}$	
Sound $\mathcal{M}$	$\mathbb{C}^-_{\mathbf{C},\neq}$	preserve satisfaction wrt $\mathcal{M}$	
Perfect $\mathcal{M}$	$\mathbb{C}^-_{\mathbf{C},\neq}$	Perfect $\mathcal{M}$	
UCQ-minimally complete Σ	$\mathbb{C}^-$	$\mathbb{C}^-$-minimally preserve violation wrt Σ	
Minimally complete Σ	$\mathbb{C}^-_{\mathbf{C},\neq}$	Minimally preserve violation wrt Σ	
Sound Σ	$\mathbb{C}^-_{\mathbf{C},\neq}$	Preserve satisfaction wrt Σ	Parallelisable
Perfect Σ	$\mathbb{C}^-_{\mathbf{C},\neq}$	Perfect Σ	$\mathcal{R}_\mathcal{O}$ FUS

Table VI.3: Result of the translation with different techniques of UCQ-translation

Stability by selection. The stability by selection of negative constraints can be characterised thanks to the characterisation of the stability by selection of UCQs.

Corollary VI.16

Let $\Sigma = (V_\mathcal{O}, \mathcal{R}_\mathcal{O}, V_\mathcal{S}, \mathcal{M})$ be a KBDM specifcation.

1. *A negative constraint* $C_\mathcal{S} = B \rightarrow \bot$ over $V_\mathcal{S}$ is *stable by selection* if and only if there exists a piece unifier μ such that μ unifies the whole of B with rules bodies in $\mathcal{M}$.
2. *A set of negative constraints* $\mathcal{C}_\mathcal{S}$ over $V_\mathcal{S}$ is *stable by selection* if and only if every negative constraint $C_\mathcal{S} \in \mathcal{C}_\mathcal{S}$ is stable by selection.

Proof. 1. It is a corollary of Proposition II.15.

2. It is a corollary of (1) and Proposition II.17. ■

Existence of an "informative" translation. We saw in chapter V that a translation of a UCQ could be not informative. That is the case when the only complete translations

of a UCQ are UCQs containing an empty CQ or when the only sound translation is the empty UCQ. Here, we extend this to negative constraints.

Corollary VI.17

Let $\Sigma = (V_{\mathcal{O}}, \mathcal{R}_{\mathcal{O}}, V_{\mathcal{S}}, \mathcal{M})$ be a KBDM specifcation.

1. *A set of negative constraints* $\mathcal{C}^-_{\mathcal{S}}$ over $V_{\mathcal{S}}$ is an *always-true* set of constraints if and only if $\mathcal{Q}^{\mathcal{C}^-_{\mathcal{S}}}$ is the empty UCQ.
2. *A set of negative constraints* $\mathcal{C}^-_{\mathcal{S}}$ over $V_{\mathcal{S}}$ is an *always-false* set of constraints if and only if $\mathcal{Q}^{\mathcal{C}^-_{\mathcal{S}}}$ contains the empty CQ.

Proof. It is a corollary of Theorem VI.14. ■

VI.4 Equality generating dependencies

In the following, we use the special predicates $\mathbf{C}$ and $\neq$ introduced in Section IV.1.

We propose translation techniques for a restricted kind of EGDs only. These are EGDs where x_1 and x_2 can only be constants, i.e., EGDs of the form:

$$B[x_1, x_2, \mathbf{y}] \wedge \mathbf{C}(x_1) \wedge \mathbf{C}(x_2) \rightarrow x_1 = x_2$$

Note that this restriction does not cause any loss of generality when considering constraints defined on the source schema, since we consider only ground instances.

This kind of constraint can be reformulated as a negative constraint:

$$B[x_1, x_2, \mathbf{y}] \wedge \mathbf{C}(x_1) \wedge \mathbf{C}(x_2) \wedge x_1 \neq x_2 \rightarrow \bot$$

Note that the terms that appear in the $\neq$-atom also appear in a $\mathbf{C}$-atom: thus, the body of this negative constraint is a $\mathrm{CQ}^{\mathbf{C},\neq}$, which was one of the classes of CQs treated in Chapter V.

Subsequently, the properties of the translation defined for negative constraints can be applied to such EGDs. The idea is that we can translate them as negative constraints to obtain another set of negative constraints. These negative constraints will still contain the $\neq$-atoms since they cannot be rewritten. This allows us to transform these negative constraints into EGDs, and so we obtain a set of EGDs as a result of the translation.

The following corollary is a direct consequence of the fact that we can reduce this kind of EGDs to negative constraints.

Corollary VI.18

An EGD of the form $C^{=} = B[x_1, x_2, \mathbf{y}] \wedge \mathbf{C}(x_1) \wedge \mathbf{C}(x_2) \rightarrow x_1 = x_2$ is equivalence-stable.

Proof. Since $C^{=} \equiv B[x_1, x_2, \mathbf{y}] \wedge \mathbf{C}(x_1) \wedge \mathbf{C}(x_2) \wedge x_1 \neq x_2 \rightarrow \bot$, which is a negative constraint, we can conclude from Corollary VI.13 that $C^{=}$ is equivalence-stable. ■

Note that some techniques to translate the UCQs from Chapter V can introduce more inequalities in the translation. Negative constraints with several inequalities in their body can be transformed into an equivalent set of EGDs that still contain inequalities in their bodies or into disjunctive EGDs (that is, EGDs whose head can be a disjunction of equalities).

Example VI.19

Take again Example IV.6 with slight modifications: let $V_{\mathcal{S}} = \{A(\cdot,\cdot), B(\cdot), D(\cdot,\cdot)\}$ and $V_{\mathcal{O}} = \{S(\cdot,\cdot), T(\cdot,\cdot), U(\cdot)\}$, and consider the mapping $\mathcal{M}$ from $V_{\mathcal{S}}$ to $V_{\mathcal{O}}$ with the rules:

$$\begin{aligned} A(x,y) &\rightarrow S(x,y), \\ B(x) &\rightarrow S(x,x), \\ A(x,x) &\rightarrow U(x), \\ D(x,y) &\rightarrow T(x,y). \end{aligned}$$

Let $\mathcal{C}^{=}_{\mathcal{S}} = \{A(x,y) \wedge D(x,z) \wedge \mathbf{C}[y,z] \rightarrow y = z\}$ be a set of EGDs. It is equivalent to the set of negative constraints $\mathcal{C}^{-}_{\mathcal{S}} = \{A(x,y) \wedge D(x,z) \wedge y \neq z \wedge \mathbf{C}[y,z] \rightarrow \bot\}$.
A perfect $\mathcal{M}$-translation of $\mathcal{C}^{-}_{\mathcal{S}}$ is:

$$\mathcal{C}^{-}_{\mathcal{O}} = \begin{cases} S(x,y) \wedge T(x,z) \wedge y \neq z \wedge x \neq y \wedge \mathbf{C}[x,y,z] & \rightarrow \bot \\ U(x) \wedge T(x,z) \wedge y \neq z \wedge \mathbf{C}[x,z] & \rightarrow \bot \end{cases}$$

Note that we have two inequalities in the first negative constraint of $\mathcal{C}^{-}_{\mathcal{O}}$. We can move one into the head to obtain an equivalent set of constraints that are EGDs that can possibly contain inequalities in their body. The following set is a perfect $\mathcal{M}$-translation of $\mathcal{C}^{=}_{\mathcal{S}}$ in the class of EGDs with inequalities:

$$\mathcal{C}^{=}_{\mathcal{O}} = \begin{cases} S(x,y) \wedge T(x,z) \wedge x \neq y \wedge \mathbf{C}[x,y,z] & \rightarrow y = z \\ U(x) \wedge T(x,z) \wedge \mathbf{C}[y,z] & \rightarrow y = z \end{cases}$$

The other possibility is to move both inequalities into the head, and we obtain disjunctive EGDs:

$$\mathcal{C}^{=,\vee}_{\mathcal{O}} = \begin{cases} S(x,y) \wedge T(x,z) \wedge \mathbf{C}[x,y,z] & \rightarrow y = z \vee x = y \\ U(x) \wedge T(x,z) \wedge \mathbf{C}[y,z] & \rightarrow y = z \end{cases}$$

Remark VI.1
One may think that we could avoid to reduce to negative constraints to do the translation. Indeed, since the inequality put in the body is never rewritten, we could keep the equality in the head and rewrite only the body, the result would be the same. But this does not work in the general case for sets of EGDs, since there are cases where we have to translate several constraints at the same time. See Example VI.20.

Example VI.20

Let $V_S = \{A(\cdot,\cdot), B(\cdot,\cdot)\}$ and $V_O = \{S(\cdot,\cdot)\}$, and consider the mapping $\mathcal{M}$ from V_S to V_O with the rules:

$$A(x,y) \rightarrow S(x,y),$$
$$B(x,y) \rightarrow S(x,y).$$

Let $\mathcal{C}_S^= = \{A(x,y) \wedge \mathbf{C}[x,y] \rightarrow x = y,\ B(x,y) \wedge \mathbf{C}[x,y] \rightarrow x = y\}$ be a set of EGDs. The only constraint that preserves the satisfaction of these two EGDs, when considered separately, is the always-true constraint. In fact, let $\mathcal{C}_O^= = \{S(x,y) \wedge \mathbf{C}[x,y] \rightarrow x = y\}$. The set $\mathcal{C}_O^=$ preserves violation of both constraints in $\mathcal{C}_S^=$, but not their satisfaction when considered independently. But it is a perfect rewriting of the whole set.
Let us reduce the EGDs in $\mathcal{C}^=$ to negative constraints:

$$\mathcal{C}_S^- = \begin{cases} A(x,y) \wedge x \neq y \wedge \mathbf{C}[x,y] & \rightarrow \bot, \\ B(x,y) \wedge x \neq y \wedge \mathbf{C}[x,y] & \rightarrow \bot \end{cases}$$

We can do a perfect $\mathcal{M}$-translation of $\mathcal{C}_S^-$:

$$\mathcal{C}_O^- = \{S(x,y) \wedge x \neq y \wedge \mathbf{C}[x,y] \rightarrow \bot\}$$

Finally, we transform it into the set of EGDs $\mathcal{C}_O^=$ given above.
We can see that this reduction allows us to translate a set of EGDs.

Class of the result of the translation In the following table (Table VI.4), we summarise the types of constraints produced by using the different techniques of UCQ-translation presented in Chapter V. We use $\mathbb{C}_{\mathbf{C}}^=$ to denote the restricted class of EGDs we consider, and $\mathbb{C}_{\mathbf{C},\neq}^=$ to denote EGDs that can contain the predicates $\neq$ and $\mathbf{C}$ presented in Section IV.1 (with the same restrictions as in $\text{UCQ}^{\mathbf{C},\neq}$).

VI.5 (Disjunctive) tuple generating dependencies

The translation of (D)TGDs is much more difficult than the kinds of constraints seen previously. This comes from the fact that they are not monotone, that is, when we add atoms to an instance, a violated constraint may become satisfied, which is not the case

UCQ-translation type	Result	Constraint translation type	Restriction
Perfect $\mathcal{M}^{-1}$	$\mathbb{C}^{=}_{\mathbf{C}}$	Perfect $\mathcal{M}^{-1}$	
Perfect Σ^{-1}	$\mathbb{C}^{=}_{\mathbf{C}}$	Perfect Σ^{-1}	$\mathcal{R}_{\mathcal{O}}$ FUS
UCQ-minimally complete $\mathcal{M}$	$\mathbb{C}^{=}_{\mathbf{C}}$	$\mathbb{C}^{=}_{\mathbf{C}}$-Minimally preserve violation wrt $\mathcal{M}$	
Minimally complete $\mathcal{M}$	$\mathbb{C}^{=}_{\mathbf{C},\neq}$	Minimally preserve violation wrt $\mathcal{M}$	
Sound $\mathcal{M}$	$\mathbb{C}^{=}_{\mathbf{C},\neq}$	Preserve satisfaction wrt $\mathcal{M}$	
Perfect $\mathcal{M}$	$\mathbb{C}^{=}_{\mathbf{C},\neq}$	Perfect $\mathcal{M}$	
UCQ-minimally complete Σ	$\mathbb{C}^{=}_{\mathbf{C}}$	$\mathbb{C}^{=}_{\mathbf{C}}$-minimally preserve violation wrt Σ	
Minimally complete Σ	$\mathbb{C}^{=}_{\mathbf{C},\neq}$	Minimally preserve violation wrt Σ	
Sound Σ	$\mathbb{C}^{=}_{\mathbf{C},\neq}$	Preserve satisfaction wrt Σ	Parallelisable
Perfect Σ	$\mathbb{C}^{=}_{\mathbf{C},\neq}$	Perfect Σ	$\mathcal{R}_{\mathcal{O}}$ FUS

Table VI.4: Result of the translation with different techniques of UCQ-translation

for the two previous kinds of constraints. Obviously, we cannot have a simple reduction to the translation of one UCQ, since UCQs are monotone queries.

Example VI.21: Non-monotonicity of TGDs

Let $C^+ = p(x) \rightarrow q(x)$ be simple TGD and $I = p(a)$ be an instance.
Obviously, $I \not\models_1 C^+$ since $q(a) \notin I$.
However, for $I' = I \cup \{q(a)\}$, we have $I' \models_1 C^+$.

One problem that arises is about existential variables that occur in the mapping or the rules. Indeed, one could naively think that we could translate the body and head as a Boolean CQ and a Boolean UCQ, respectively. However, as seen in Example VI.22, it is not that simple.

Example VI.22: Naive way to translate DTGDs

Let $\Sigma = (V_{\mathcal{O}}, \mathcal{R}_{\mathcal{O}}, V_{\mathcal{S}}, \mathcal{M})$ be a KBDM specification where:

$$\mathcal{M} = \begin{cases} p(x) & \rightarrow \exists y\ r(x,y), \\ s(x) & \rightarrow t(x) \end{cases}$$

Let $C^+ = r(x,y) \rightarrow t(y)$ be a TGD. Then we could translate the body and head of C^+ as if they were Boolean CQs. We obtain the perfect $\mathcal{M}^{-1}$-translations $\exists x\ p(x)$ and $\exists y\ s(y)$ respectively for the body and head of C^+. Therefore, we could think that $C^+_{\mathcal{S}} = p(x) \rightarrow \exists y\ s(y)$ is a perfect $\mathcal{M}^{-1}$-translation of C^+. But this is not the case. Let $D = \{p(a), s(b)\}$. Clearly, $D \models_1 C^+_{\mathcal{S}}$. But $I_{(D,\mathcal{M})} = \{t(b),\ r(a, y_0)\}$ and thus $I_{(D,\mathcal{M})} \not\models_1 C^+$. This proves that $C^+_{\mathcal{S}}$ is not a perfect $\mathcal{M}$-translation of C^+.

As can be seen in Example VI.22, the problem is that in the translation process of the body and head of the TGD, we lose a frontier variable because it is unified with an existential variable of a rule during the translation. In the following, we consider a restricted kind of (D)TGDs that consists of enforcing that frontier variables are necessarily mapped to constants.

Also, to simplify the presentation of the following, we restrict ourselves to DTGDs with a head where all the disjuncts share exactly the same frontier variables. More formally:

$$B[\mathbf{x},\mathbf{y}] \wedge \mathbf{C}[\mathbf{x}] \rightarrow \bigvee_{i=1}^{n} \exists \mathbf{z_i}.H_i[\mathbf{x},\mathbf{z_i}]$$

The goal of this restriction is to allow us to represent this DTGD as an inclusion of the answers of a CQ on a particular instance into the answers of a UCQ on the same instance (Theorem VI.23).

Theorem VI.23

Let $C^+ = B[\mathbf{x},\mathbf{y}] \wedge \mathbf{C}[\mathbf{x}] \rightarrow \bigvee_{i=1}^{n} \exists \mathbf{z_i}.H_i[\mathbf{x},\mathbf{z_i}]$ be a restricted DTGD.
Then, for any instance I, $I \models_1 C^+$ if and only if $q(I) \subseteq \mathcal{Q}(I)$, where q and $\mathcal{Q}$ are the queries defined as follows:

- $q(\mathbf{x}) = \exists \mathbf{y}.B[\mathbf{x},\mathbf{y}]$;
- $\mathcal{Q}(\mathbf{x}) = \bigvee_{i=1}^{n} q_i(\mathbf{x})$ with $q_i(\mathbf{x}) = \exists \mathbf{z_i}.H_i[\mathbf{x},\mathbf{z_i}]$.

Proof. ($\Rightarrow$) Assume for any instance I, $I \models_1 C^+$. For each homomorphism h that maps $B[\mathbf{x},\mathbf{y}] \wedge \mathbf{C}[\mathbf{x}]$ into I, we know that:

- $h(\mathbf{x})$ is a tuple of constants;
- there exists $1 \geq i \geq n$ and an extension h_i of h such that $h_i(H_i[\mathbf{x},\mathbf{z_i}]) \subseteq I$.

First, note that we have $h(B[\mathbf{x},\mathbf{y}] \wedge \mathbf{C}[\mathbf{x}]) \subseteq I$ if and only if $h(\mathbf{x})$ is an answer to q on I. Indeed, q is composed of the body of C^+, without $\mathbf{C}$-atoms, but the answer variables must be sent into constants, and the answer variables are exactly the frontier variables of the rule.

Then, we have h_i which is a homomorphism from $H_i[\mathbf{x},\mathbf{z_i}]$ to I and since $h_i(\mathbf{x}) = h(\mathbf{x})$, $h(\mathbf{x})$ is also an answer to $q_i \subseteq \mathcal{Q}$ and thus to $\mathcal{Q}$. We can conclude that, for each $h(\mathbf{x})$ that is an answer to q, it is also an answer to $\mathcal{Q}$.

($\Leftarrow$) Assume for any instance I, $q(I) \subseteq \mathcal{Q}(I)$. Then, when there exists a homomorphism h that maps $B[\mathbf{x},\mathbf{y}]$ to I, we have that there also exists a homomorphism h_i from $q_i = \exists z_i.H[\mathbf{x},\mathbf{z_i}] \in \mathcal{Q}$ to I such that $h(\mathbf{x}) = h_i(\mathbf{x})$.

To satisfy C^+, either we need that there exists no homomorphism from its body to I, but in this case, there is no answer to q. Or, if there exists a homomorphism from the body to I, we have to find an extension that maps its head to I. Take h previously defined: it is a homomorphism from the body of C^+ into I if and only if it is a homomorphism that maps q to I (it effectively maps the variables in $\mathbf{C}$-atoms to constants because it maps the answer variables of q to constants).

Then, let $h'_i = h + h_i$. We can see that it is an extension of h to the disjunct $H_i[\mathbf{x}, \mathbf{z_i}]$ of the head of C^+ such that it maps it to I. Thus, we can conclude that, when $q(I) \subseteq \mathcal{Q}(I)$, the constraint is satisfied on I. ■

In the following, given a DTGD $C^+ = B[\mathbf{x}, \mathbf{y}] \wedge \mathbf{C}[\mathbf{x}] \rightarrow \bigvee_{i=1}^{n} \exists \mathbf{z_i}.H_i[\mathbf{x}, \mathbf{z_i}]$, we denote the *CQ associated to the body of* C^+ by $q^{C^+_B}(\mathbf{x}) = \exists \mathbf{y}.B[\mathbf{x}, \mathbf{y}]$ and the *UCQ associated to the head of* C^+ as $\mathcal{Q}^{C^+_H}(\mathbf{x}) = \bigvee_{i=1}^{n} \exists \mathbf{z_i}.H_i[\mathbf{x}, \mathbf{z_i}]$.

Note that we could remove the restriction on the frontier variables (that is, all the frontier variables must appear in all the disjuncts in the head). In fact, a DTGD $C^+ = B \rightarrow H_1 \vee \cdots \vee H_n$ can be seen as a disjunction of TGDs: $C^+ \equiv (C^+_1 = B \rightarrow H_1) \vee \cdots \vee (C^+_n = B \rightarrow H_n)$. Thus, we could apply the previous Theorem on each TGD C^+_i in the disjunction. And so, we have that C^+ is satisfied by an instance I if and only if there is C^+_i such that $q^{C^+_{i,B}}(I) \subseteq \mathcal{Q}^{C^+_{i,H}}(I)$.

A corollary of Theorem VI.23 is that the restricted DTGDs we consider are equivalence-stable.

Corollary VI.24

A DTGD of the form $C^+ = B[\mathbf{x}, \mathbf{y}] \wedge \mathbf{C}[\mathbf{x}] \rightarrow \bigvee_{i=1}^{n} \exists \mathbf{z_i}.H_i[\mathbf{x}, \mathbf{z_i}]$ is equivalence-stable.

Proof. It is a direct consequence of Theorem VI.23: since for each pair of equivalent instances (I_1, I_2), we have $q^{C^+_B}(I_1) = q^{C^+_B}(I_2)$ and $\mathcal{Q}^{C^+_H}(I_1) = \mathcal{Q}^{C^+_H}(I_2)$, then we have $I_1 \models_1 C^+$ if and only if $I_2 \models_1 C^+$. ■

Note that, in contrast to the previous kinds of constraint, we do not consider translation from a set of constraints but only from one constraint, since the method of translation we provide works only for an isolated (D)TGD.

To translate a DTGD C^+, the rough idea is that we will translate $q^{C^+_B}(\mathbf{x})$ and $\mathcal{Q}^{C^+_H}(\mathbf{x})$ using the techniques defined in Chapter V. But we need to determine which type of UCQ translation (sound or complete) allows us to obtain a translation of C^+ that preserves its satisfaction or its violation. Intuitively, to preserve satisfaction, we should have a translation that does not add answers to $q^{C^+_B}(\mathbf{x})$ (sound translation) and keep all answers to $\mathcal{Q}^{C^+_H}(\mathbf{x})$ (complete translation): this would guarantee that the inclusion of answers will still hold. And conversely, for the preservation of the violation: we should do a complete translation of $q^{C^+_B}(\mathbf{x})$ and sound translation of $\mathcal{Q}^{C^+_H}(\mathbf{x})$.

Let $\Sigma = (V_\mathcal{O}, \mathcal{R}_\mathcal{O}, V_\mathcal{S}, \mathcal{M})$ be a KBDM specification. In the two theorems that follow (Theorems VI.25 and VI.27), we abstract away from the direction of the translation to simplify the statements of the theorems and the proofs. The vocabularies V_1 and V_2 can be $V_\mathcal{S}$ and $V_\mathcal{O}$ or $V_\mathcal{O}$ and $V_\mathcal{S}$ respectively. For an instance over a vocabulary $V_1 = V_\mathcal{S}$ and a database D on $\mathcal{S}$, we say that $I_{(D,\Sigma)}$ is the *associated instance* over $V_2 = V_\mathcal{O}$. Conversely, if $V_1 = V_\mathcal{O}$, the *instance associated* with $I_{(D,\Sigma)}$ on $V_2 = V_\mathcal{S}$ is D. In the proofs, we will simply refer to the associated instance without specifying the direction, as the proofs are exactly the same in both directions.

Theorem VI.25

Let $\Sigma = (V_\mathcal{O}, \mathcal{R}_\mathcal{O}, V_\mathcal{S}, \mathcal{M})$ be a KBDM specification and C^+ be a DTGD over a vocabulary V_1, a vocabulary V_2 distinct from V_1, a maximally sound translation $\mathcal{Q}^B_{V_2}$ of $q^{C^+_B}$ over V_2 and a minimally complete translation $\mathcal{Q}^H_{V_2}$ of $\mathcal{Q}^{C^+_H}$ over V_2. Then, $\mathcal{C}^+_{V_2} = \{\mathtt{body}(q^B_{V_2}) \wedge C[\mathtt{ansVars}(q^B_{V_2})] \rightarrow \mathtt{body}(\mathcal{Q}^H_{V_2}) \mid q^B_{V_2} \in \mathcal{Q}^B_{V_2}\}$ is a DTGD over V_2 that maximally preserves satisfaction of C^+.

Proof. First, we demonstrate that $\mathcal{C}^+_{V_2}$ is a translation of C^+ that preserves its satisfaction. Since $\mathcal{Q}^B_{V_2}$ is a sound translation of $q^{C^+_B}$, then, for every instance I on V_1 and instance J associated with I over V_2, we have $\mathcal{Q}^B_{V_2}(J) \subseteq q^{C^+_B}(I)$. Similarly, since $\mathcal{Q}^H_{V_2}$ is a complete translation of $q^{C^+_H}$, for every instance I over V_1 and instance J associated with I over V_2, we have $\mathcal{Q}^{C^+_H}(I) \subseteq \mathcal{Q}^H_{V_2}(J)$. Since the satisfaction of C^+ is equivalent to $q(I) \subseteq \mathcal{Q}(I)$ for every I, for every associated instance J over V_2, we have $\mathcal{Q}^B_{V_2}(J) \subseteq q(I) \subseteq \mathcal{Q}(I) \subseteq \mathcal{Q}^H_{V_2}(J)$. This implies that, when C^+ is satisfied, $\mathcal{C}^+_{V_2}$ is also satisfied.

In addition, $\mathcal{C}^+_{V_2}$ is maximal (in this class). Assume that this is not the case. Then, either there exists another sound translation $\mathcal{Q}^{B'}_{V_2}$ such that, for every instance J over V_2, $\mathcal{Q}^{B'}_{V_2}(J) \subseteq \mathcal{Q}^B_{V_2}(J)$, or there exists another complete translation $\mathcal{Q}^{H'}_{V_2}$ such that for every instance J over V_2 $\mathcal{Q}^H_{V_2}(J) \subseteq \mathcal{Q}^{H'}_{V_2}(J)$. In the first case, that would mean that $\mathcal{Q}^B_{V_2}$ is not a maximally sound translation and in the latter case that $\mathcal{Q}^H_{V_2}(J)$ is not a minimally complete translation. This leads to a contradiction. ■

Example VI.26 illustrates Theorem VI.25.

Example VI.26

Let $\Sigma = (V_\mathcal{O}, \mathcal{R}_\mathcal{O}, V_\mathcal{S}, \mathcal{M})$ be a KBDM specification where, as in Example II.3:

$$\mathcal{M} = \begin{cases} s_1(x,y) & \rightarrow p(x,y), \\ s_2(x,x) & \rightarrow \exists z.r(x,z), \\ s_3(x,y) & \rightarrow p(x,y) \end{cases}$$

Consider the TGD $C^+ = s_2(u,v) \wedge \mathbf{C}(u) \rightarrow \exists w.s_1(u,w)$ on $V_\mathcal{S}$.

Its satisfaction on an instance D over V_S is equivalent to $q^{C_B^+}(D) \subseteq \mathcal{Q}^{C_H^+}(D)$ where $q^{C_B^+}(u) = \exists v.s_2(u,v)$ and $\mathcal{Q}^{C_H^+}(u) = \exists w.s_1(u,w)$.
We have $\mathcal{Q}^{C_B^+}_{\mathcal{O}}(u) = \exists z.r(u,z)$ which is a maximally sound $\mathcal{M}$-translation of $q^{C_B^+}$ and $\mathcal{Q}^{C_H^+}_{\mathcal{O}}(u) = \exists w.p(u,w)$ is a minimally complete $\mathcal{M}$-translation of $\mathcal{Q}^{C_H^+}$.
Thus, the TGD $r(u,z) \wedge \mathbf{C}(u) \rightarrow \exists w.p(u,w)$ is a $\mathcal{M}$-translation of C^+ that maximally preserves its satisfaction.

Theorem VI.27

Let $\Sigma = (V_{\mathcal{O}}, \mathcal{R}_{\mathcal{O}}, V_S, \mathcal{M})$ be a KBDM specification and C^+ be a DTGD over a vocabulary V_1, a vocabulary V_2 distinct from V_1, a minimally complete translation $\mathcal{Q}^B_{V_2}$ of $q^{C_B^+}$ over V_2 and a maximally sound translation $\mathcal{Q}^H_{V_2}$ of $\mathcal{Q}^{C_H^+}$ over V_2. Then, $\mathcal{C}^+_{V_2} = \{\mathsf{body}(q^B_{V_2}) \wedge C[\mathsf{ansVars}(q^B_{V_2})] \rightarrow \mathsf{body}(\mathcal{Q}^H_{V_2}) \mid q^B_{V_2} \in \mathcal{Q}^B_{V_2}\}$ is a DTGD over V_2 that minimally preserves violation of C^+.

Proof. The proof uses exactly the same ideas as in the proof of Theorem VI.25. ■

Example VI.28 illustrates Theorem VI.27.

Example VI.28

Take again the KBDM specification $\Sigma = (V_{\mathcal{O}}, \mathcal{R}_{\mathcal{O}}, V_S, \mathcal{M})$ from Example VI.26 where:

$$\mathcal{M} = \begin{cases} s_1(x,y) & \rightarrow p(x,y), \\ s_2(x,x) & \rightarrow \exists z.r(x,z), \\ s_3(x,y) & \rightarrow p(x,y) \end{cases}$$

Consider the TGD $C^+ = s_1(u,w) \wedge \mathbf{C}(u) \rightarrow \exists v.s_2(u,v)$ (we exchanged the body and the head compared to Example VI.26) on V_S.
Its satisfaction on an instance D over V_S is equivalent to $q^{C_B^+}(D) \subseteq \mathcal{Q}^{C_H^+}(D)$ where $q^{C_B^+}(u) = \exists w.s_1(u,w)$ and $\mathcal{Q}^{C_H^+}(u) = \exists v.s_2(u,v)$.
We have $\mathcal{Q}^{C_B^+}_{\mathcal{O}}(u) = \exists w.p(u,w)$ which is a minimally complete $\mathcal{M}$-translation of $q^{C_B^+}$ and $\mathcal{Q}^{C_H^+}_{\mathcal{O}}(u) = \exists z.r(u,z)$ is a maximally sound $\mathcal{M}$-translation of $\mathcal{Q}^{C_H^+}$.
Thus, the TGD $p(u,w) \wedge \mathbf{C}(u) \rightarrow \exists z.r(u,z)$ is a $\mathcal{M}$-translation of C^+ that minimally preserves its violation.

Note that in Theorem VI.25 and Theorem VI.27, the constraints in $\mathcal{C}^+_{V_2}$ are not necessarily in exactly the same class as C^+. Indeed, depending on the translation techniques used to translate the body and head, if C^+ is a TGD, it is not guaranteed that $\mathcal{C}^+_{V_2}$ will be a set of TGDs, it can be a set of DTGDs. Moreover, $\mathcal{C}^+_{V_2}$ can contain special predicates $\mathbf{C}$ and $\neq$, including in the head, even when C^+ does not contain these predicates.

Class of the result of the translation. In table VI.5, we summarise the types of constraints produced by using the different UCQ-translation techniques presented in Chapter V. We use $\mathbb{C}^{(D)TGD}_{B(\mathbf{C})}$ to denote the restricted class of (D)TGDs we consider. We denote by $B([special\ predicates])$ and $H([special\ predicates])$ the special predicates that appear in the body and in the head of a DTGD, respectively. For example, $\mathbb{C}^{DTGD}_{B(\mathbf{C},\neq),H(\mathbf{C})}$ denotes the class of DTGDs that can contain $\mathbf{C}$ and $\neq$ in the body and $\mathbf{C}$ in the head. We recall that the predicates $\neq$ and $\mathbf{C}$ were presented in Section IV.1 (we use them with the same restrictions as in $\mathrm{UCQ}^{\mathbf{C},\neq}$).

Body translation	Head translation	Result	Restriction
Perfect $\mathcal{M}^{-1}$-translation			
perfect	perfect	$\mathbb{C}^{DTGD}_{B(\mathbf{C})}$	
Perfect Σ^{-1}-translation			
perfect	perfect	$\mathbb{C}^{DTGD}_{B(\mathbf{C})}$	$\mathcal{R}_{\mathcal{O}}$ FUS
$\mathcal{M}$-translation preserving satisfaction			
sound	UCQ-minimally complete	$\mathbb{C}^{(D)TGD}_{B(\mathbf{C},\neq)}$	
sound	minimally complete	$\mathbb{C}^{DTGD}_{B(\mathbf{C},\neq),H(\mathbf{C},\neq)}$	
$\mathcal{M}$-translation preserving violation			
UCQ-minimally complete	sound	$\mathbb{C}^{DTGD}_{B(\mathbf{C}),H(\mathbf{C},\neq)}$	
minimally complete	sound	$\mathbb{C}^{DTGD}_{B(\mathbf{C},\neq),H(\mathbf{C},\neq)}$	
Perfect $\mathcal{M}$-translation			
perfect	perfect	$\mathbb{C}^{DTGD}_{B(\mathbf{C},\neq),H(\mathbf{C},\neq)}$	
Σ-translation preserving violation			
UCQ-minimally complete	sound	$\mathbb{C}^{DTGD}_{B(\mathbf{C}),H(\mathbf{C},\neq)}$	parallelisable
minimally complete	sound	$\mathbb{C}^{DTGD}_{B(\mathbf{C},\neq),H(\mathbf{C},\neq)}$	parallelisable
Perfect Σ-translation			
perfect	perfect	$\mathbb{C}^{DTGD}_{B(\mathbf{C},\neq),H(\mathbf{C},\neq)}$	$\mathcal{R}_{\mathcal{O}}$ FUS

Table VI.5: Summary of translations of DTGD

VI.6 Summary

Let us briefly review the content of this chapter and highlight the main contributions:

1. **Related work:** We establish some links between our framework and other notions from the literature (Section VI.2).

2. **Application of the constraint framework:** we applied the translation framework presented in Chapter II on three classes of constraints, that are all equivalence-stable:

   (a) **Negative constraints:** A negative constraint can be represented as (the negation of) a Boolean CQ, and a set of negative constraints as (the negation of) a Boolean UCQ). This allows one to directly use techniques of UCQ-translation presented in Chapter V.

   (b) **EGDs:** We consider a restricted kind of EGDs where the frontier variables must be mapped to constants (which is without loss of generality for constraints over the data source). This restriction allows us to transform EGDs into negative constraints that we can translate in our framework. And so, all the results obtained for negative constraints can be applied to this restricted kind of EGDs.

   (c) **(D)TGDs:** We consider a restricted kind of (D)TGDs where the frontier variables must be mapped to constants (as for EGDs, it is without loss of generality for constraints over the source). This allows one to represent such a (D)TGD as a pair of queries such that for any instance I, the answers on I to the first query are included in the answers on I to the second query if and only if the (D)TGD is satisfied on I. This representation allows us to use again UCQ-translation techniques on both queries. Finally, we show which techniques of UCQ-translation have to be used for each query to obtain a translation that maximally preserves satisfaction or minimally preserves violation.

For each class of constraints, we provide tables that indicate the UCQ translations associated with the desired type of constraint translation, and well as the constraint class of the result.

VII - Conclusion

In this dissertion, we investigated issues related to the translation of constraints in a Knowledge-Based Data Management system. We specially focused on translations from the data level to the ontology level, as this direction is more challenging and has been less investigated.

This study involved exploring two related topics: query translation, especially in the language of UCQs, and rewriting UCQs with disjunctive rules. Exploring the first topic was expected, as CQs are a basic building block for most types of constraints. The second topic was prompted by the notion of maximum recovery studied in the field of data exchange, which appeared to be a crucial notion and takes the form of a disjunctive mapping in our context.

Our contributions are summarised at the end of each chapter. In this conclusion, we will rather focus on perspectives.

Query rewriting with disjunctive rules. We have shown that FO-rewritability (or, equivalently, UCQ-rewritability) of a set of disjunctive rules seems to cover very few rule classes. It seems more promising to study the UCQ-rewritability of a pair $(\mathcal{Q}, \mathcal{R})$. We list here some interesting open problems related to this notion:

1. Clarify the boundary between decidability and undecidability for the problem of determining whether a pair $(\mathcal{Q}, \mathcal{R})$ is UCQ-rewritable, according to specific classes of rules (and queries). In particular, UCQ-rewritability is decidable for guarded conjunctive rules and some of their generalizations [Barceló et al., 2018], does this extend to the disjunctive case?

2. We have shown that the UCQ-$\mathcal{S}$-rewritability of a pair $(\mathcal{Q}, \mathcal{M})$ is undecidable. Is it still the case for a pair $(\{Q\}, \mathcal{M})$ where Q is a CQ?

3. Our undecidability proof for UCQ-$\mathcal{S}$-rewritability exploits the fact that rewritings are restricted to predicates in $\mathcal{S}$. If we consider instead UCQ-rewritings with source-to-target rules, we know that the problem can only be simpler, as we can easily reduce UCQ-rewritability with $\mathcal{S}$-to-$\mathcal{T}$-rules to UCQ-$\mathcal{S}$-rewritability with mappings: one simply has to add a mapping rule per target predicate to give it an existence at the source level. Is the UCQ-rewritability of a pair $(\mathcal{Q}, \mathcal{R})$ decidable when $\mathcal{R}$ is a set of $\mathcal{S}$-to-$\mathcal{T}$ rules?

4. Given a pair $(\mathcal{Q}, \mathcal{R})$, our query rewriting outputs a UCQ-rewriting when one exists. However, when $\mathcal{R}$ is replaced by a disjunctive mapping $\mathcal{M}$, a pair $(\mathcal{Q}, \mathcal{M})$ may have a UCQ-$\mathcal{S}$-rewriting (where $\mathcal{S}$ is the source vocabulary) even when it has no UCQ-rewriting. Is it possible to build an algorithm that, given a pair $(\mathcal{Q}, \mathcal{M})$, outputs a UCQ-$\mathcal{S}$-rewriting for this pair when one exists?

Query translation. As pointed out in the summary of Chapter V, two fundamental problems remain open concerning maximally sound $\mathcal{S}$-to-$\mathcal{O}$ translations:

1. Is the problem of determining whether a pair $(\mathcal{Q}, \mathcal{M})$ has a maximally sound $\mathcal{M}$-translation decidable? We conjecture that it is not, since it seems close to the UCQ-$\mathcal{S}$-rewritability problem mentioned above, which we have shown to be undecidable. Note however that when UCQ-$\mathcal{S}$-rewritability is decidable for a UCQ $\mathcal{Q}$, determining whether $(\mathcal{Q}, \mathcal{M})$ has a maximally sound $\mathcal{M}$-translation is also decidable.

2. Is it possible to build an algorithm that outputs a (finite) maximally sound $\mathcal{M}$-translation when one exists? Again, this seems similar to the issue of building a UCQ-$\mathcal{S}$-rewriting when one exists (and when we can build a UCQ-$\mathcal{S}$-rewriting, we can build a maximally sound $\mathcal{M}$-translation).

Also, concerning the computation of a maximally sound Σ-translation of a UCQ, we have only dealt with the case where the union of the mapping and the rules is parallelisable: it would be interesting to handle more general FUS classes.

Constraints translation. First, the general framework itself could be further investigated. While our primary focus has been on equivalence-stable constraints, how to extend the framework to constraints that do not have this property, e.g., unrestricted EGDs and (D)TGDs, is an open question. Note, however, that EGDs and (D)TGDs on a database always fulfil the restriction, and their translations produce restricted EGDs and (D)TGDs at the ontological level. Hence, the restriction is only effective for constraints defined at the ontological level (and not: obtained by translation from the data level).

Still about the general framework, we chose to check constraint satisfaction at the ontological level on a single target instance (more precisely, in the case of equivalence-stable constraints, we consider any universel solution). Could we consider a set of target instances, i.e., a strict subset of universal solutions? How would this broaden the applicability of our framework?

Second, in the time allotted for a Ph.D. thesis (three years), we could only start to apply our framework to specific classes of constraints. Given the vast landscape of constraints in database theory, it would be useful to explore translations for other pairs of constraint classes. Furthermore, one noticeable aspect we have yet to dive into is the complexity of the core problems of interest, applied to specific classes of queries, mappings, rules and constraints.

Lastly, the practical implications of constraint translation, especially in the realms of design and optimization of KBDM systems, remain largely uncharted. This is an area for future work.

A - Proofs of Section III.3

To prove Theorem III.14, we first need the following lemmas and propositions. In the proofs, we reuse some notations and results from [Baget et al., 2011] and [König et al., 2015], which recall below.

Let $h : X \longrightarrow T$ and $h' : X' \longrightarrow T'$ be two substitutions such that, $\forall x \in X \cap X', h(v) = h'(v)$. Then we note $h + h' : X \cup X' \longrightarrow T \cup T'$ the substitution defined by: if $x \in X, (h + h')(x) = h(x)$, otherwise $(h + h')(x) = h'(x)$.

Proposition A.1: Proposition 23 in [Baget et al., 2011]

Let I be a instance, q be a CQ, $\mathbf{x} \subseteq \mathtt{vars}(q)$, $\{q_1, \ldots, q_k\}$ be a partition of the atoms of q such that
$\mathtt{vars}(q_i) \cap \mathtt{vars}(q_j) \subseteq \mathbf{x}$ for all q_i and q_j with $i \neq j$, and $h_1, \ldots, h_k$ homomorphisms from q_i to I such that $\forall t \in \mathbf{x}, \forall 1 \leq i \leq j \leq k, h_i(t) = h_j(t)$; then the substitution $h_1 + \cdots + h_k$ is a homomorphism from q to I.

Given a partition P on a set of terms, we denote by $P[t]$ the class of P that contains the term t.

Definition A.1 (Partition induced by a substitution)
A partition P on terms $\mathcal{T}$ induced by a substitution s is such that for every $t, t' \in \mathcal{T}$, if $s(t) = s(t')$ then $t' \in P[t]$ (*that is*, $P[t] = P[t']$) and P is the thinnest partition with this property. Let C be a class of P, we call *selected element* of C, which we denote t_C, the unique element of C such that $s(t_C) = t_C$.

The next three propositions are immediate.

Proposition A.2

Let $\mathcal{I}$ be a set of set of instances and $\mathcal{Q}$ be a UCQ: $\mathcal{I} \models \mathcal{Q}$ if and only if for each $I \in \mathcal{I}$, there exists a $Q \in \mathcal{Q}$ such that Q maps to I.

Proposition A.3

A partition induced by a substitution is admissible.

Proposition A.4

Let I and I' be two instances and s a substitution from I to I' such that $s(I) = I'$. Then, any substitution u_s associated with P_s, the partition induced by s, on the terms of I and I', is such that $u_s(I) = u_s(I')$.

The following lemmas A.5 and A.6 correspond to **Lemma III.12** (Point 1 and Point

2, respectively) in Chapter III. Figures A.1 and A.2 depict lemmas A.5 and A.6, respectively.

Lemma A.5

Let I_1, I_2 be two instances such that $I_1 \models I_2$ and a disjunctive rule R such that there exists a trigger (R, h_2) on I_2. Then, there exists a trigger (R, h_1) on I_1 such that $\alpha_\vee(I_1, R, h_1) \models \alpha_\vee(I_2, R, h_2)$.

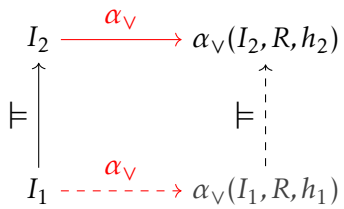


Figure A.1: Preservation of entailment by $\alpha_\vee$ (Lemma A.5)

Proof. Let $R = B \to H_1 \vee \cdots \vee H_n$ be a disjunctive rule. Since $I_1 \models I_2$, we have a homomorphism h from I_2 to I_1. Moreover, (R, h_2) being a trigger on I_2, taking $h_1 = h \circ h_2$, we have (R, h_1) being a trigger on I_1

and $\alpha_\vee(I_2, R, h_2) = \{I_2^i = I_2 \cup h_2^{+_{i\cdot 2}}(H_i) \mid 1 \leq i \leq n\}$ and $\alpha_\vee(I_1, R, h_1) = \{I_1^i = I_1 \cup (h \circ h_2)^{+_{i\cdot 1}}(H_i) \mid 1 \leq i \leq n\}$. Let us build a homomorphism h^i from I_2^i to I_1^i, for $1 \leq i \leq n$. For each i, we first consider the homomorphism h_{H_i} from $h_2^{+_{i\cdot 2}}(H_i)$ to $(h \circ h_\alpha)^{+_{i\cdot 1}}(H_i)$, defined as follows:

$\forall t \in \mathtt{vars}(h_2^{+_{i\cdot 2}}(H_i))$:

- *if* $t \in h_2(\mathtt{fr}(R))$, *then* $h_{H_i}(t) = h(t)$;
- *otherwise,* $h_{H_i}(t) = .^{+_{i\cdot 1}}((.^{+_{i\cdot 2}})^{-1}(t))$.

h and h_{H_i} satisfy the conditions of Proposition A.1 (with $\mathbf{x} = h_{H_i}(\mathtt{fr}(R))$). As a consequence, $h^i = h + h_{H_i}$ is a homomorphism from I_2^i to I_1^i.

Thus, $\alpha_\vee(I_1, R, h_1) \models \alpha_\vee(I_2, R, h_2)$. ∎

Lemma A.6

Let $\mathcal{Q}_1$ and $\mathcal{Q}_2$ be UCQs such that $\mathcal{Q}_2 \models \mathcal{Q}_1$, and let R be a disjunctive rule. Then, for any disjunctive piece-unifier $\mu_\vee^2$ of $\mathcal{Q}_2$ with R:

1. either $\beta_\vee(\mathcal{Q}_2, R, \mu_\vee^2) \models \mathcal{Q}_1$;
2. or, there is a piece-unifier $\mu_\vee^1$ of $\mathcal{Q}_1$ with R such that $\beta_\vee(\mathcal{Q}_2, R, \mu_\vee^2) \models \beta_\vee(\mathcal{Q}_1, R, \mu_\vee^1)$.

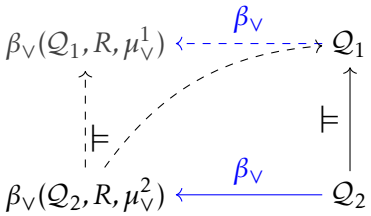


Figure A.2: Preservation of entailment by $\beta_\vee$ (Lemma A.6)

Proof. Let $R = B \to H_1 \vee \cdots \vee H_n$ be a disjunctive rule. Let $q_2^1, \ldots, q_2^n$ be the safe copies of CQs in $\mathcal{Q}_2$ of which subsets $q_2^{1'}, \ldots, q_2^{n'}$ are unified with, respectively, $H'_1, \ldots, H'_n$, subsets of, respectively, $H_1, \ldots, H_n$, to define $\mu^2_\vee = \{(q_2^{1'}, H'_1, P^2_{u_1}), \ldots, (q_2^{n'}, H'_n, P^2_{u_n})\}$ the disjunctive piece-unifier of $\mathcal{Q}_2$ with R. Let $P_{u^2_\vee} = join(\{P^2_{u_1}, \ldots, P^2_{u_n}\})$ be a partition and let $u^2_\vee$ be the substitution associated with $P_{u^2_\vee}$. Let $h_1, \ldots, h_n$ be the homomorphisms associated with each q_2^i, that map a q_1^i in $\mathcal{Q}_1$ to q_2^i (note that since each q_2^i is a safe copy of a CQ in $\mathcal{Q}_2$ then there exists a CQ q_1^i in $\mathcal{Q}_1$ that maps on it).

We consider two cases:

- Either, one of the q_1^i maps by h_i to the non-rewritten part of q_2^i, so this q_1^i maps to the CQ added to the $\mathcal{Q}_2$ by the one-step piece-rewriting, *that is*, there exists $1 \leq i \leq n$ and $q_1^i \in \mathcal{Q}_1$ such that $h_i(q_1^i) \subseteq (q_2^i \setminus q_2^{i'})$, then $u^2_\vee \circ h_i$ is a homomorphism from q_1^i to $u^2_\vee(q_2^i \setminus q_2^{i'}) \subseteq \beta_\vee(\mathcal{Q}_2, R, \mu^2_\vee)$. Thus, $\beta_\vee(\mathcal{Q}_2, R, \mu^2_\vee) \models q_1^i \models \mathcal{Q}_1$.

- Otherwise, for each $1 \leq i \leq n$, we now consider that q_1^i is a safe copy of the CQ in $\mathcal{Q}_1$ that maps to q_2^i and h_i is the homomorphism (extended by considering this safe renaming) from q_1^i to q_2^i. Let $q_1^{i'}$ be the maximal subset of q_1^i that maps to $q_2^{i'}$ by h_i, *that is*, $q_1^{i'} \subseteq q_1^i$, $h_i(q_1^{i'}) \subseteq q_2^{i'}$ and $h_i(q_1^i \setminus q_1^{i'}) \cap q_2^{i'} = \emptyset$.

  Let H''_i be the maximal subset of H'_i which is unified by $u^2_\vee$ with the subset $h_i(Q_1^{i'})$ of $Q_2^{i'}$, *that is*, $H''_i \subseteq H'_i$, $u^2_\vee(H''_i) = u^2_\vee(h_i(q_1^{i'}))$ and $u^2_\vee(H'_i \setminus H''_i) \cap u^2_\vee(h_i(q_1^{i'})) = \emptyset$. Let $P^1_{u_i}$ be the partition induced by $u^2_\vee \circ h_i$ on $\texttt{terms}(H''_i \cup q_1^{i'})$. By construction, $\mu_i^1 = (q_1^{i'}, H''_i, P^1_{u_i})$ is thus a piece-unifier between q_1^i and H_i. Since for each $1 \leq i < j \leq n$, $q_1^{i'}$ and $q_1^{j'}$ does not share any variable, then we can define $h = h_1 + \cdots + h_n$. We find that $u^2_\vee \circ h$ is a homomorphism from $q_1^{1'} \wedge \cdots \wedge q_n^{1'}$ to $u^2_\vee(H''_1 \wedge \cdots \wedge H''_n)$. Let $P_{u^1_\vee}$ be the partition induced by $u^2_\vee \circ h$ on $\texttt{terms}(q_1^{1'} \wedge \cdots \wedge q_n^{1'}) \cup \texttt{terms}(H''_1 \wedge \cdots \wedge H''_n)$: it is admissible, since it is built from a substitution (Proposition A.3). Moreover, we have $P_{u^1_\vee} = join(P_{u_1}, \ldots, P_{u_n})$ and thus $\mu^1_\vee$ is a disjunctive unifier of $\mathcal{Q}_1$ with R.

  We now prove that $\beta_\vee(\mathcal{Q}_2, R, \mu^2_\vee) \models \beta_\vee(\mathcal{Q}_1, R, \mu^1_\vee)$.

  We build a substitution s from the selected elements of the classes in $P_{u^1_\vee}$ which are variables, to the selected elements of the classes in $P_{u^2_\vee}$ as follows: for any class $C \in P_{u^1_\vee}$, if t_C is a variable of a H''_i, then $s(t_C) = u^2_\vee(t_C)$, otherwise $s(t_C) = u^2_\vee(h(t))$

(t occurs in a $q_i^{1'}$). Note that for any term t in $P_{u^1_\vee}$, we have $s(u^1_\vee(t)) = u^2_\vee(h(t))$. We now build a substitution h' from $\mathtt{vars}(\beta_\vee(\mathcal{Q}_1, R, \mu^1_\vee))$ to $\mathtt{terms}(\beta_\vee(\mathcal{Q}_2, R, \mu^2_\vee))$ by considering three cases according to the part of $\beta_\vee(\mathcal{Q}_1, R, \mu^1_\vee)$ in which the variables occurs (in a q_i^1 but not in $q_i^{1'}$, in $\mathtt{body}(R)$ but not in H_i'', or in the remaining part corresponding to the images of $\mathtt{vars}(q_1^{i'}) \cap \mathtt{vars}(q_1^i)$ by $u^1_\vee$):

- if $x \in \mathtt{vars}(q_1^i) \setminus \mathtt{vars}(q_1^{i'})$, $h'(x) = h(x)$;
- if $x \in \mathtt{vars}(\mathtt{body}(R)) \setminus \mathtt{vars}(\bigcup_{i=1}^n H_i'')$, $h'(x) = u^2_\vee(x)$;
- if $x \in u^1_\vee(\bigcup_{i=1}^n(\mathtt{vars}(q_1^{i'}) \cap \mathtt{vars}(q_1^i)))$ (or alternatively $x \in u^1_\vee(\mathtt{fr}(R) \cap \mathtt{vars}(\bigcup_{i=1}^n H_i'')))$, $h'(x) = s(x)$.

We conclude by showing that h' is a homomorphism from $\beta_\vee(\mathcal{Q}_1, R, \mu^1_\vee) = u^1_\vee(\mathtt{body}(R)) \cup \bigcup_{i=1}^n u^1_\vee(q_1^i \setminus q_1^{i'})$ to $\beta_\vee(\mathcal{Q}_2, R, \mu^2_\vee) = u^2_\vee(\mathtt{body}(R)) \cup \bigcup_{i=1}^n u^2_\vee(q_2^i \setminus q_2^{i'})$ with two points:

- $h'(u^1_\vee(\mathtt{body}(R))) = u^2_\vee(\mathtt{body}(R))$. Indeed, for any variable x of $\mathtt{body}(R)$:
  - either $x \in \mathtt{vars}(\mathtt{body}(R)) \setminus \mathtt{vars}(\bigcup_{i=1}^n H_i'')$, so $h'(u^1_\vee(x)) = h'(x) = u^2_\vee(x)$ (because $u^1_\vee$ is a substitution from $\mathtt{vars}(\bigcup_{i=1}^n(q_1^{i'} \cup H_i'')))$;
  - or $x \in \mathtt{fr}(R) \cap \mathtt{vars}(\bigcup_{i=1}^n H_i''))$, so $h'(u^1_\vee(x)) = s(u^1_\vee(x)) = u^2_\vee(h(x)) = u^2_\vee(x)$ (because h is a substitution from $\mathtt{vars}(\bigcup_{i=1}^n q_1^i)$ and recall that for any term t in $P_{u^1_\vee}$, $s(u^1_\vee(t)) = u^2_\vee(h(t)))$.
- $h'(u^1_\vee(q_i^1 \setminus q_i^{1'})) \subseteq u^2_\vee(q_2^i \setminus q_2^{i'})$ for each $1 \leq i \leq n$. In fact, we will show that $h'(u^1_\vee(q_1^i \setminus q_1^{i'})) = u^2_\vee(h(q_1^i \setminus q_1^{i'}))$ and since $h(q_1^i \setminus q_1^{i'}) \subseteq q_2^i \setminus q_2^{i'}$ we will be able to conclude. To show that $h'(u^1_\vee(q_1^i \setminus q_1^{i'})) = u^2_\vee(h(q_1^i \setminus q_1^{i'}))$, just see that for any $x \in \mathtt{vars}(q_1^i \setminus q_1^{i'})$:
  - either $x \in (\mathtt{vars}(q_1^{i'}) \cap \mathtt{vars}(q_1^i))$, then $h'(u^1_\vee(x)) = s(u^1_\vee(x)) = u^2_\vee(h(x))$;
  - or $x \in (\mathtt{vars}(q_1^i) \setminus \mathtt{vars}(q_1^{i'}))$, then $h'(u^1_\vee(x)) = h'(x) = h(x) = u^2_\vee(h(x))$ (because $u^1_\vee$ is a substitution from $\mathtt{vars}(\bigcup_{i=1}^n(q_1^{i'} \cup H_i''))$ and $u^2_\vee$ is a substitution from variables of $\bigcup_{i=1}^n(q_2^{i'} \cup H_i')$ and $h(x) \notin \mathtt{vars}(\bigcup_{i=1}^n(q_2^{i'} \cup H_i')))$.

■

The following lemmas A.7 and A.8 correspond to **Lemma III.13** (Point 1 and Point 2, respectively) in Chapter III. Figures A.3 and A.4 depict lemmas A.7 and A.8, respectively.

Lemma A.7

Let I be an instance, a disjunctive rule R, a trigger (R, h) on I and let $\mathcal{Q}$ be the UCQ $\alpha_\vee(I, R, h)$. Then there exists a disjunctive piece-unifier $\mu_\vee$ of $\mathcal{Q}$ with R such that $I \models \beta_\vee(\mathcal{Q}, R, \mu_\vee)$.

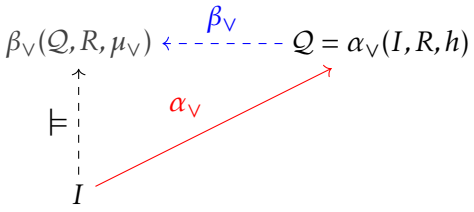


Figure A.3: Corresponding application of $\beta_\vee$ to the UCQ obtained by $\alpha_\vee$ is entailed by the original factbase (Lemma A.7)

Proof. Let $\mathcal{Q} = \alpha_\vee(I, R, h) = \{q_i = I \cup h^{+_i}(\text{head}_i(R)) \mid 1 \leq i \leq n\}$ be a UCQ. We build $\mu_\vee = \{\mu_1, \ldots, \mu_n\}$ a disjunctive piece-unifier as follows: for $1 \leq i \leq n$, $\mu_i = (q'_i, \text{head}_i(R), P_{u_i})$ with $q'_i = \rho_i \circ h^{+_i}(\text{head}_i(R))$ (ρ_i being a safe renaming of q_i) and P_{u_i} the partition induced by $\rho_i \circ h^{+_i}$ on $\texttt{terms}(q'_i) \cup \texttt{terms}(\text{head}_i(R))$.

First, we show that each μ_i is a piece-unifier of $\rho_i(q_i)$ with $\text{body}(R) \to \text{head}_i(R)$:

- $q'_i \subseteq \rho_i(q_i)$ because $h^{+_i}(\text{head}_i(R)) \subseteq q_i$ and $q'_i = \rho_i(h^{+_i}(\text{head}_i(R))$;
- P_{u_i} the partition induced by $\rho_i \circ h^{+_i}$ is admissible (thanks to Proposition A.3);
- any u_i associated with P_{u_i} is such that $u_i(\text{head}_i(R)) = u_i(q'_i)$ (thanks to Proposition A.4);
- for each existential variable z from $\text{head}_i(R)$ we have $P_{u_i}[z] = \{z, \rho_i \circ h^{+_i}(z)\}$ and $\rho_i \circ h^{+_i}(z)$ is not a separating variable because z is safely renamed twice, first by $.^{+_i}$ and secondly by ρ_i.

Then, we show that the partition $P_{u_\vee} = join(\{P_{u_1}, \ldots, P_{u_n}\})$ is admissible.

Since each P_{u_i} is admissible, the non-admissibility of their join would be only due to a variable that appears in two classes with different constants from two partitions. The only variables that can be shared between two partitions of a set of piece-unifiers built from safe copies of CQs are the frontier variables of the considered disjunctive rule. But if a frontier variable shared by two $\text{head}_i(R)$ is mapped to a constant, then it is mapped to the same constant because each P_{u_i} is induced by $\rho_i \circ h^{+_i}$ and only h can send a variable to a constant.

$\mu_\vee$ is therefore a disjunctive piece-unifier of $\mathcal{Q}$ with R. Let $u_\vee$ be a substitution associated with $P_{u_\vee}$.

Let $I' = \beta_\vee(\mathcal{Q}, R, \mu_\vee) = u_\vee(B) \cup \bigcup_{1 \leq i \leq n} u_\vee(\rho_i(q_i) \setminus q'_i) = u_\vee(B) \cup \bigcup_{1 \leq i \leq n} u_\vee(\rho_i(I \cup h^{+_i}(\text{head}_i(R))) \setminus (\rho_i \circ h^{+_i})(\text{head}_i(R))) \subseteq u_\vee(B) \cup \bigcup_{1 \leq i \leq n} u_\vee(\rho_i(I))$ (note that this inclusion is not a simple equality because $I \cap h^{+_i}(\text{head}_i(R))$ can be nonempty).

We just have to observe that $\rho_1^{-1} + \cdots + \rho_n^{-1} + h$ is a homomorphism from $u_\vee(B) \cup \bigcup_{1 \leq i \leq n} u_\vee(\rho_i(I))$ to I:

- $(\rho_1^{-1} + \cdots + \rho_n^{-1} + h)(u_\vee(\rho_i(I))) = I$, indeed:
  - If I contains only constants, it is straightforward;
  - If I contains some variables, then they were renamed in $\rho_i(I)$. $u_\vee$ can only map variables into two distinct sets of terms:
    - Assume that a variable of $\rho_i(I)$ is mapped to a variable in $\texttt{terms}(q'_i)$. Then, ρ_i^{-1} allows to recover the initial variable that was in I (because no variable of I can be in the same class of P_{u_i} as an existential variable of R and the other variables come from the application of ρ_i);
    - Otherwise, assume it is mapped to a variable in $\texttt{terms}(\texttt{head}_i(R))$. Then, h allows us to recover the initial variable in I (since these variables can only appear in q_i through the frontier variables of R thanks to the application of h on $\texttt{head}_i(R)$).
- $(\rho_1^{-1} + \cdots + \rho_n^{-1} + h)(u_\vee(B)) = h(B) \subseteq I$, indeed, by a similar reasoning:
  - Assume that a variable in B is sent by $u_\vee$ to a variable in $\texttt{terms}(q'_i)$, then ρ_i^{-1} allows to recover the variable in I to which h maps this variable from B;
  - Assume it is mapped by $u_\vee$ to a variable in $\texttt{terms}(\texttt{head}_i(R))$, then we simply have a variable in the domain of h since it can only be a frontier variable. ■

Since $I' \subseteq u_\vee(B) \cup \bigcup_{1 \leq i \leq n} u_\vee(\rho_i(I))$, it follows that $\rho_1^{-1} + \cdots + \rho_n^{-1} + h$ maps I' to I.

Lemma A.8

Let $\mathcal{Q}$ be a UCQ, R be a disjunctive rule, $\mu_\vee$ be a disjunctive piece-unifier of $\mathcal{Q}$ with R and I be the instance $\beta_\vee(\mathcal{Q}, R, \mu_\vee)$. Then, there exists a trigger (R, h) on I such that $\alpha_\vee(I, R, h) \models \mathcal{Q}$.

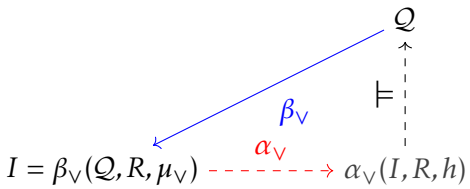


Figure A.4: Corresponding application of $\alpha_\vee$ to the CQ obtained by $\beta_\vee$ entails the original UCQ (Lemma A.8)

Proof. Let $\mu_\vee = \{\mu_1, \ldots, \mu_n\}$ be a disjunctive unifier, and let $u_\vee$ be a substitution associated with $join(\{P_{u_1}, \ldots, P_{u_n}\})$ with each P_{u_i} being the partition in each μ_i. $(R, u_\vee)$

is a trigger on $I = \beta_\vee(\mathcal{Q}, R, \mu_\vee)$ since $u_\vee(\text{body}(R)) \subseteq I$. Let $\mathcal{I}' = \alpha_\vee(I, R, u_\vee) = \{I'_i = I \cup u_\vee^{+_i}(\text{head}_i(R)) \mid 1 \leq i \leq n\}$.

To prove that $\mathcal{I}' \models \mathcal{Q}$, we will show that for each I'_i, the CQ q_i that is a safe copy of a CQ in $\mathcal{Q}$ and was unified by μ_i with $\text{head}_i(R)$ maps to I'_i by the homomorphism $u_\vee^{+_i}$. Then, let ρ_i be the renaming substitution that produced q_i from a CQ $Q \in \mathcal{Q}$, we will have $u_\vee^{+_i} \circ \rho_i$ is a homomorphism from this Q to I'_i. Thus, by Proposition A.2, we can conclude that $\mathcal{I}' \models \mathcal{Q}$.

Let us now show that $u_\vee^{+_i}$ maps q_i to I'_i:

- $u_\vee$ maps $q_i \setminus q'_i$ into $u_\vee(q_i \setminus q'_i) \subseteq I \subseteq I'_i$ and since $u_\vee^{+_i}$ is an extension of $u_\vee$ to the existential variables of R, $u_\vee^{+_i}(q_i \setminus q'_i) = u_\vee(q_i \setminus q'_i)$, so $u_\vee^{+_i}$ maps $q_i \setminus q'_i$ into I'_i.
- $u_\vee^{+_i}$ maps q'_i to $u_\vee^{+_i}(\text{head}_i(R)) \subseteq I'_i$ because first $u_\vee$ unifies q'_i and $H'_i \subseteq \text{head}_i(R)$, *that is*, $u_\vee(q'_i) = u_\vee(H'_i)$, and second $.^{+_i}$ maps $u_\vee(\text{head}_i(R))$ to $u_\vee^{+_i}(\text{head}_i(R))$;

Lemma A.9: Backward-forward Lemma

Let I be an instance, $\mathcal{Q}$ be a UCQ and R be a disjunctive rule. For any disjunctive piece-unifier $\mu_\vee$ of $\mathcal{Q}$ with R, if $I \models \beta_\vee(\mathcal{Q}, R, \mu_\vee)$ then there is a trigger (R, h) on I such that $\alpha_\vee(I, R, h) \models \mathcal{Q}$.

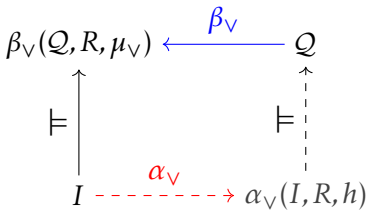


Figure A.5: Correspondences between $\beta_\vee$ (in blue) and $\alpha_\vee$ (in red) - Lemma A.9

Proof. Thanks to Lemma A.8, we know that there is a trigger (R, h) on $I_2 = \beta_\vee(\mathcal{Q}, R, \mu_\vee)$ such that $\alpha_\vee(I_2, R, h) \models \mathcal{Q}$. Then, from Lemma A.5, we know that if $I \models I_2$, then we have $\alpha_\vee(I, R, h) \models \alpha_\vee(I_2, R, h)$. Thus $\alpha_\vee(I, R, h) \models \mathcal{Q}$, which we wanted to prove. ■

Lemma A.10: Forward-backward Lemma

Given any trigger (R, h) on I, if $\alpha_\vee(I, R, h) \models \mathcal{Q}$ then either $I \models \mathcal{Q}$ or there is a disjunctive piece-unifier $\mu_\vee$ of $\mathcal{Q}$ with R, such that $I \models \beta_\vee(\mathcal{Q}, R, \mu_\vee)$.

Proof. Thanks to Lemma A.7, we know that there is a disjunctive piece-unifier $\mu_\vee^2$ of $\mathcal{Q}_2 = \alpha_\vee(I, R, h)$ with R such that $I \models \beta_\vee(\mathcal{Q}_2, R, \mu_\vee^2)$. Then, from Lemma A.6, we know

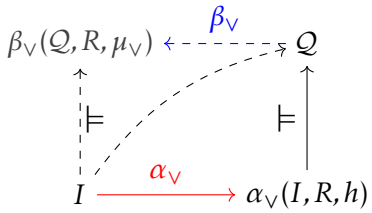


Figure A.6: Correspondences between β_V (in blue) and α_V (in red) - Lemma A.10

that if $\mathcal{Q}_2 \models \mathcal{Q}$, either $\beta_V(\mathcal{Q}_2, R, \mu_V^2) \models \mathcal{Q}$ or there exists μ_V such that $\beta_V(\mathcal{Q}_2, R, \mu_V^2) \models \beta_V(\mathcal{Q}, R, \mu_V)$. Since $I \models \beta_V(\mathcal{Q}_2, R, \mu_V^2)$, we either have $I \models \mathcal{Q}$ or $I \models \beta_V(\mathcal{Q}, R, \mu_V)$, which was what we wanted to prove. ■

Corollary A.11: Corollary of Lemma A.10

Let $\mathcal{I}$ be a set of instances, $I \in \mathcal{I}$, R a disjunctive rule and (R, h) a trigger on I. Let $\mathcal{I}_1$ be the set of instances obtained by the immediate derivation of $(\mathcal{I}, R)$ by the trigger (R, h), *that is*, $\mathcal{I}_1 = \mathcal{I} \setminus \{I\} \cup \alpha_V(I, R, h)$. Then, if $\mathcal{I}_1 \models \mathcal{Q}$, either $\mathcal{I} \models \mathcal{Q}$ or there exists a unifier μ_V of $\mathcal{Q}$ with R such that $\mathcal{I} \models \{\beta_V(\mathcal{Q}, R, \mu_V)\} \cup \mathcal{Q}$.

Proof. Since $\mathcal{I} \setminus \{I\} \models \mathcal{Q}$, we just have to prove that either $I \models \mathcal{Q}$ or $I \models \beta_V(\mathcal{Q}, R, \mu_V)$, which is exactly Lemma A.10. ■

We extend the notion of disjunctive chase result to any derivation tree or derivation sequence. So we call *derivation tree result* the set of instances $res(\mathcal{T}) = \{ \bigcup_{v \in nodes(\gamma)} \lambda(v) \mid \gamma \in \Gamma(\mathcal{T})\}$ where $\mathcal{T}$ is any derivation tree and λ its labelling function. Also, we call *derivation sequence result* the set of instances $res(\mathcal{D}) = \mathcal{I}_n$ where $\mathcal{I}_n$ is the last set of instances in the derivation $\mathcal{D}$.

Note that if $\mathcal{T}$ is finite, we have $res(\mathcal{T}) \equiv res(\mathcal{D})$ for any derivation $\mathcal{D}$ that we can assign to $\mathcal{T}$. Indeed, for each finite sequence $\mathcal{D}_n$ of length n, $\mathcal{I}_n$ corresponds exactly to the labels of the leaves of a derivation tree built from the same trigger applications: hence, $\mathcal{I}_n$ is isomorphic to $res(\mathcal{T})$.

Lemma A.12

Let q be a CQ, $\mathcal{T} = (V, E, \lambda)$ be a derivation tree, γ be a branch of $\mathcal{T}$ and $I_\gamma \in res(\mathcal{T})$ the set of facts associated with γ, *that is*, $I_\gamma = \bigcup_{v \in nodes(\gamma)} \lambda(v)$. If a homomorphism h maps q to I_γ , then there is a vertex $v \in \gamma$ such that $\lambda(v) \models q$.

Proof. To each atom of I_γ, we give a rank that corresponds to the depth[1] of the vertex of

[1] The depth of a vertex v is defined as the length of the path from the root to v.

γ where it was produced. Since $h(q)$ is finite, let k be the maximum rank of the atoms in $h(q)$. Let v be the vertex at depth k in γ, we have $h(q) \subseteq \lambda(v)$, so $\lambda(v) \models q$. ■

Theorem A.13

Let $\mathcal{Q}$ be a UCQ, a set of disjunctive rules $\mathcal{R}$ and a instance I. Then $\mathsf{chase}(I,\mathcal{R}) \models \mathcal{Q}$ if and only if there exists a finite derivation tree $\mathcal{T}$ of $(I,\mathcal{R})$ such that $res(\mathcal{T}) \models \mathcal{Q}$.

Proof. ($\Leftarrow$) We only need to extend the derivation tree $\mathcal{T}$, in a fair way, to add what is missing in the tree. Indeed, by the definition of the result of a derivation tree / disjunctive chase, each instance of $\mathsf{chase}(I,\mathcal{R})$ includes at least one instance of $res(\mathcal{T})$, so $\mathsf{chase}(I,\mathcal{R}) \models res(\mathcal{T})$ and thus $\mathsf{chase}(I,\mathcal{R}) \models \mathcal{Q}$.

($\Rightarrow$) Let $\mathcal{T}_C = (V,E,\lambda)$ be the fair derivation tree used to define $\mathsf{chase}(I,\mathcal{R})$, *that is*, $\mathsf{chase}(I,\mathcal{R}) = \{I_\gamma = \bigcup_{v \in nodes(\gamma)} \lambda(v) \mid \gamma \in \Gamma(\mathcal{T}_C)\}$. For each $I_\gamma \in \mathsf{chase}(I,\mathcal{R})$, let $\mathcal{Q}_\gamma \subseteq \mathcal{Q}$ be the set of CQs that maps to I_γ ($\mathcal{Q}_\gamma$ contains at least one CQ, cf. Proposition A.2). By Lemma A.12, for each CQ in $\mathcal{Q}_\gamma$ there is a vertex $v \in \gamma$ such that $\lambda(v) \models q_\gamma$. In each branch γ, we select v_γ the highest of these vertices in γ.

These selected vertices are called terminal vertices. We build the subtree $\mathcal{T}'$ of $\mathcal{T}_C$ by removing from $\mathcal{T}_C$ all vertices that are the successors of a terminal vertex. Thus, every branch of $\mathcal{T}'$ is finite. We show that (1) $\mathcal{T}'$ is still a derivation tree and (2) it is finite.

1. By construction, each node in $\mathcal{T}'$ is either a terminal node (in which case it is a leaf), or we did not erase any of its children (and so its children still correspond to the result of applying a trigger). Thus, $\mathcal{T}'$ is still a derivation tree.

2. Since each rule is finite, each node in a derivation tree has a finite number of children (it is locally finite). According to König's infinity lemma [Kőnig, 1927], "an infinite, locally finite rooted tree has an infinite branch". Its contrapositive is "a locally finite rooted tree without infinite branch is finite". Thus, $\mathcal{T}'$ is finite. ■

Corollary A.14: Corollary of Theorem A.13

$I,\mathcal{R} \models \mathcal{Q}$ if and only if there exists a finite derivation tree $\mathcal{T}$ of $(I,\mathcal{R})$ such that $res(\mathcal{T}) \models \mathcal{Q}$. Equivalently, $I,\mathcal{R} \models \mathcal{Q}$ if and only if there exists a derivation $\mathcal{D}$ from I with $\mathcal{R}$ such that $res(\mathcal{D}) \models \mathcal{Q}$.

We can finally prove Theorem III.14.

Theorem III.14: Let $\mathcal{R}$ be a set of disjunctive rules and $\mathcal{Q}$ be a UCQ. Then, for any instance I, holds $I,\mathcal{R} \models \mathcal{Q}$ if and only if there is a disjunctive piece-rewriting $\mathcal{Q}'$ of $\mathcal{Q}$ such that $I \models \mathcal{Q}'$.

Proof of Theorem III.14. We show that there exists a derivation of $(\{I\}, \mathcal{R})$ leading to an $\mathcal{I}_i$ such that $\mathcal{I}_i \models \mathcal{Q}$ if and only if there exists a piece-rewriting $\mathcal{Q}'$ of $\mathcal{Q}$ with $\mathcal{R}$ such that $I \models \mathcal{Q}'$.

($\Rightarrow$) We prove the first direction by induction on the number of rule applications in a derivation sequence $\mathcal{D}$ such that $\mathcal{Q}$ maps to $res(\mathcal{D})$ (such a tree / derivation exists: see Corollary A.14).

At rank 0, the property is trivially true taking $\mathcal{Q}' = \mathcal{Q}$. Let us assume that it is true at rank n. Let $\mathcal{D} = (\mathcal{I}_0 = \{I\}) \xrightarrow{t_1} \ldots \xrightarrow{t_n} \mathcal{I}_n \xrightarrow{(R,h)} \mathcal{I}_{n+1}$ be a derivation with $\mathcal{Q}$ that maps to $\mathcal{I}_{n+1}$. By using the Corollary A.11, we have either:

1. $\mathcal{I}_n \models \mathcal{Q}$;
2. or there exists $\mu_\vee$ such that $\mathcal{I}_n \models \{\beta_\vee(\mathcal{Q}, R, \mu_\vee)\} \cup \mathcal{Q}$.

In both cases, we have a UCQ that maps to $\mathcal{I}_n$. Let us name it $\mathcal{Q}_n$. By the induction hypothesis, there exists a piece-rewriting $\mathcal{Q}'$ of $\mathcal{Q}_n$ such that $I \models \mathcal{Q}'$. By definition, $\mathcal{Q}_n$ is a one-step piece-rewriting of $\mathcal{Q}$, and thus $\mathcal{Q}'$ is also a piece-rewriting of $\mathcal{Q}$.

($\Leftarrow$) We prove the opposite direction by induction on the length of the rewriting sequence producing $\mathcal{Q}'$ from $\mathcal{Q}$ and relying upon Lemma A.9. The property is trivially true at rank 0 taking $\mathcal{I}_0 = \{I\}$. Let us assume that it is true at rank n. Assume that $\mathcal{Q}_{n+1}$ is obtained from $\mathcal{Q}$ by a rewriting sequence $\mathcal{Q} = \mathcal{Q}_0, \mathcal{Q}_1, \ldots, \mathcal{Q}_n, \mathcal{Q}_{n+1} = \beta_\vee(\mathcal{Q}_n, R, \mu_\vee) \cup \mathcal{Q}_n$ of length $n+1$, and $I \models \mathcal{Q}_{n+1}$. So, there is a CQ q in $\mathcal{Q}_{n+1}$ such that $I \models q$. We have two cases:

1. $q \in \mathcal{Q}_n$: then, by induction hypothesis, there exists $\mathcal{I}_i$ such that $\mathcal{I}_i \models q$, thus $\mathcal{I}_i \models \mathcal{Q}_n$ and also $\mathcal{I}_i \models \mathcal{Q}_{n+1}$.
2. $q = \beta_\vee(\mathcal{Q}_n, R, \mu_\vee)$: then, by Lemma A.9, there exists $\mathcal{I}_1 = \alpha_\vee(I, R, h)$ such that $\mathcal{I}_1 \models \mathcal{Q}_n$. So, we have that for each $I_m \in \mathcal{I}_1$, $I_m \models \mathcal{Q}_n$. And by the induction hypothesis, we see that for each I_m, there exists a derivation of $(\{I_m\}, \mathcal{R})$ leading to a $\mathcal{I}_m$ such that $\mathcal{I}_m \models \mathcal{Q}$ and thus we have a derivation from $\mathcal{I}_1$ that produces $\mathcal{I}_i$ (that is the union of all $\mathcal{I}_m$) such that $\mathcal{I}_i \models \mathcal{Q}$. ■

B - Proofs of Section III.5

Recall that *disjunctive mapping rewritability* is the following problem: Given a set of disjunctive $\mathcal{S}$-to-$\mathcal{T}$-rules $\mathcal{M}$ and a UCQ $\mathcal{Q}$ on $\mathcal{T}$, does the pair $(\mathcal{Q},\mathcal{M})$ admit a UCQ-$\mathcal{S}$-rewriting?

Theorem III.23: Disjunctive mapping rewritability is undecidable.

To prove it, we build a reduction from the following problem: Given a Boolean CQ q and a set of (conjunctive) datalog rules $\mathcal{R}$, does the pair $(q,\mathcal{R})$ admit a UCQ-rewriting? This problem is undecidable, which follows from the undecidability of determining whether a datalog program is uniformly bounded [Gaifman et al., 1993]. Indeed, a datalog program $\mathcal{R}$ is uniformly bounded if and only if the pair $(q,\mathcal{R})$ is UCQ-rewritable for any full atomic query q, that is, in which all the variables are answer variables. Since there is a finite number of non-isomorphic atomic CQs to consider, it follows that determining if a pair $(q,\mathcal{R})$ is UCQ-rewritable for q an atomic CQ is also undecidable. In turn, this problem can be reduced to the problem of determining whether a pair $(q',\mathcal{R})$ is UCQ-rewritable for q' a Boolean CQ. To build q', we just add to q an atom with special predicate *answer* which contains all the variables of q. This ensures that the answer variables are properly considered when comparing two generated CQs.

W.l.o.g. we assume that datalog rules have no constants (and an atomic head).

Our reduction translates each instance $(q,\mathcal{R})$ of the conjunctive datalog UCQ-rewriting problem, defined on a set of predicates $\mathcal{P}$, into an instance $(\mathcal{Q}^{Q,\mathcal{R}},\mathcal{M}^{Q,\mathcal{R}})$ of the disjunctive mapping rewritability problem, defined on a pair of predicats sets $(\mathcal{S},\mathcal{T})$ such that:

- $\mathcal{S} = \mathcal{P} \cup \{T\}$, where T is a fresh unary predicate,
- $\mathcal{T}$ is the union of: (1) a set of predicates in bijection with $\mathcal{S}$, where each predicate is topped with a hat (e.g. $\hat{p}$ is obtained from p), and (2) a set of fresh predicates in bijection with $\mathcal{R}$, where we denote by p_{R_i} the predicate associated with the rule R_i; the arity of each p_{R_i} is $|\mathtt{fr}(R_i)|$.

We also denote by $T[\mathbf{x}]$ the conjunction of atoms $T(x_i)$ for each $x_i \in \mathbf{x}$, i.e. $T[\mathbf{x}] = T(x_1) \wedge \cdots \wedge T(x_n)$ where $|\mathbf{x}| = n$. Similarly, $\hat{T}[\mathbf{x}] = \hat{T}(x_1) \wedge \cdots \wedge \hat{T}(x_n)$. Let Q be any CQ (or set of atoms) on $\mathcal{S}$, we denote by $\hat{Q}$ the CQ (or set of atoms) Q whose predicates have all been renamed with a hat, $\hat{Q}$ is therefore on $\mathcal{T}$. Let Q be any CQ, we denote by Q^T the CQ Q completed with a T atom on each term. Then, $\hat{Q}^T$ is the CQ obtained from $\hat{Q}$ by adding its T atoms. Finally, $\widehat{Q^T}$ is obtained from Q^T by substituting each predicate p (including T) by $\hat{p}$.

Definition of the reduction Let $q = \exists \mathbf{x_q}\ B_q[\mathbf{x_q}]$ be a CQ and a datalog rule set $\mathcal{R} = \{R_1,\ldots,R_n\}$ with each $R_i = B_i[\mathbf{x_i},\mathbf{y_i}] \rightarrow H_i[\mathbf{x_i}]$. We define the UCQ $\mathcal{Q}^{q,\mathcal{R}}$ and the disjunctive datalog mapping $\mathcal{M}^{q,\mathcal{R}}$ associated with Q and $\mathcal{R}$ as follows:

- $\mathcal{Q}^{q,\mathcal{R}} = \{q_q\} \cup \mathcal{Q}_\mathcal{R}$ with
  $q_Q = \exists \mathbf{x_q}.\hat{B}_q[\mathbf{x_q}] \wedge \hat{T}[\mathbf{x_q}]$, i.e., $q_Q = \widehat{q^T}$,
  $\mathcal{Q}_\mathcal{R} = \{q_{R_i} = \exists \mathbf{x_i}, \mathbf{y_i}.\hat{B}_i[\mathbf{x_i}, \mathbf{y_i}] \wedge p_{R_i}(\mathbf{x_i}) \wedge \hat{T}[\mathbf{x_i}, \mathbf{y_i}] \mid R_i \in \mathcal{R}\}$
- $\mathcal{M}^{q,\mathcal{R}} = \mathcal{M}_\mathcal{R} \cup \mathcal{M}_{trans}$ with
  $\mathcal{M}_\mathcal{R} = \{m_{R_i} = T[\mathbf{x_i}] \rightarrow p_{R_i}(\mathbf{x_i}) \vee \hat{H}_i(\mathbf{x_i}) \mid R_i \in \mathcal{R}\}$
  $\mathcal{M}_{trans} = \{p(\mathbf{x}) \rightarrow \hat{p}(\mathbf{x}) \mid p \in \mathcal{S}\}$

Let us comment on the reduction. The UCQ $\mathcal{Q}^{q,\mathcal{R}}$ is built from Q and, for every $R_i \in \mathcal{R}$, a CQ q_{R_i}. Each q_{R_i} is composed of the conjunction of $\mathsf{body}(R_i)$ and a special atom $p_{R_i}(\mathbf{x_i})$, where p_{R_i} is a fresh predicate associated with R_i and $\mathbf{x_i}$ is the frontier of R_i. The idea is that $p_{R_i}(\mathbf{x_i})$ will be unifiable (and thus erasable) only with a corresponding mapping assertion m_{R_i}, which moreover enforces to have a CQ containing an atom unifiable with $\mathsf{head}(R_i)$. Then, for each term t in a CQ, one adds a unary atom $T(t)$. The set of rules $\mathcal{M}^{q,\mathcal{R}}$ is built by creating, for each rule R_i, a disjunctive rule m_{R_i} with a body that contains a $T(x)$ atom for each frontier variable x of R_i, and a head with the special atom associated with R_i as first disjunct, and $\mathsf{head}(R_i)$ as second disjunct. Finally, the predicates from $\mathcal{S}$ of each atom in $\mathcal{Q}^{q,\mathcal{R}}$ or in the head of disjunctive rules in $\mathcal{M}^{q,\mathcal{R}}$ are turned into target predicates (i.e., renamed with a "hat"), and a set of atomic $\mathcal{S}$-to-$\mathcal{T}$ rules $\mathcal{M}_{trans}$ is added to translate the source predicates into target predicates.

Note that there is a natural bijection (up to variable renaming) between the CQs defined on $\mathcal{P}$ and the CQs defined on $\mathcal{S}$ that have a T-atom on each term: to Q_w on $\mathcal{P}$ we assign the CQ Q_w^T composed of Q_w (or any CQ isomorphic to Q_w) completed by T-atoms on each term.

Then the correctness of the reduction is proved thanks to three lemmas:

- We prove in Lemma B.2 that for any CQ Q_w belonging to a piece-rewriting of $\{q\}$ with $\mathcal{R}$, a CQ isomorphic to Q_w^T belongs to a piece-rewriting of $\mathcal{Q}^{q,\mathcal{R}}$ with $\mathcal{M}^{q,\mathcal{R}}$.
- We prove in Lemma B.1 that any CQ Q_S belonging to an $\mathcal{S}$-rewriting of $\mathcal{Q}^{q,\mathcal{R}}$ with $\mathcal{M}^{q,\mathcal{R}}$ is of the form $Q_S = (Q_\mathcal{P})^T$ where $Q_\mathcal{P}$ is a set of atoms on $\mathcal{P}$.
- We prove in Lemma B.5 that for any CQ of the form $(Q_\mathcal{P})^T$, with $Q_\mathcal{P}$ on $\mathcal{P}$, belonging to a piece-rewriting of $\mathcal{Q}^{q,\mathcal{R}}$ with $\mathcal{M}^{q,\mathcal{R}}$, a CQ isomorphic to $Q_\mathcal{P}$ belongs to a piece-rewriting of $\{q\}$ with $\mathcal{R}^\star$, where $\mathcal{R}^\star$ is the reflexive and transitive closure of $\mathcal{R}$ by unfolding. This lemma is established by showing that any CQ in a piece-rewriting of $\mathcal{Q}^{q,\mathcal{R}}$ with $\mathcal{M}^{q,\mathcal{R}}$ "corresponds" either to a rule from $\mathcal{R}^\star$, or to a piece-rewriting of $\{q\}$ with $\mathcal{R}^\star$.

Furthermore, the proof implicitly uses the following observations:

1. Let $\mathcal{Q}$ be a finite rewriting of $\{Q\}$ with $\mathcal{R}$. Then there is a piece-rewriting $\mathcal{Q}_i$ of $\{Q\}$ with $\mathcal{R}$ such that $\mathcal{Q} \models \mathcal{Q}_i$.
   Proof: For every complete rewriting $\mathcal{Q}'$ of $\{Q\}$ with $\mathcal{R}$, we have $\mathcal{Q} \models \mathcal{Q}'$ (indeed, let

M be a model of $\mathcal{Q}$ and h be a witnessing homomorphism from a CQ Q' in $\mathcal{Q}$ to M. Let $I = h(Q')$ be an instance. Since $I \models \mathcal{Q}$ and $\mathcal{Q}$ is sound, we have $\mathcal{R}, I \models Q$, hence $I \models \mathcal{Q}'$ because $\mathcal{Q}'$ is complete. Hence, M is a model of $\mathcal{Q}'$). Since piece-rewriting is a complete procedure, there is a complete set of CQs $\mathcal{Q}_i$ produced by a possibly infinite sequence of piece-rewritings. Then, $\mathcal{Q} \models \mathcal{Q}_i$. This means that for each CQ $Q' \in \mathcal{Q}$, there is a CQ $Q_j \in \mathcal{Q}_i$ such that $Q' \models Q_j$. We can restrict $\mathcal{Q}_i$ to these Q_j while keeping the entailment from $\mathcal{Q}$.

2. Let $\mathcal{Q}$ be a UCQ-rewriting of $\{Q\}$ with $\mathcal{R}$. Then there is a complete piece-rewriting $\mathcal{Q}_i$ of $\{Q\}$ with $\mathcal{R}$ such that $\mathcal{Q}_i \equiv \mathcal{Q}$.
   Proof: Let $\mathcal{Q}_i$ be a complete set of CQs obtained by a possibly infinite sequence of piece-rewritings of $\{Q\}$ with $\mathcal{R}$. As previously, we consider a model M of $\mathcal{Q}_i$ and I a (finite)-witnessing subset of M. Since $\mathcal{Q}_i$ is sound, we have $\mathcal{R}, I \models Q$, hence $I \models \mathcal{Q}$ because $\mathcal{Q}$ is complete. Hence, M is a model of $\mathcal{Q}$ and $\mathcal{Q}_i \models \mathcal{Q}$. We do the same reasoning by considering a model of $\mathcal{Q}$ to conclude that $\mathcal{Q} \models \mathcal{Q}_i$. We can restrict $\mathcal{Q}_i$ to an equivalent finite subset because $\mathcal{Q}$ is finite and equivalent to $\mathcal{Q}_i$ (see e.g., Theorem 1 in [König et al., 2015]).

Proof of Theorem III.23. We prove that there exists a UCQ-rewriting of $(\{Q\}, \mathcal{R})$ iff there exists a UCQ-$\mathcal{S}$-rewriting of $(\mathcal{Q}^{q,\mathcal{R}}, \mathcal{M}^{q,\mathcal{R}})$.

($\Rightarrow$) Let $\mathcal{Q}$ be a UCQ-rewriting of $(\{Q\}, \mathcal{R})$. Then there exists a piece-rewriting $\mathcal{Q}_i$ of $\{Q\}$ with $\mathcal{R}$ such that $\mathcal{Q}_i \equiv \mathcal{Q}$. By Lemma B.2, there is a piece-rewriting $\mathcal{Q}_j$ of $\mathcal{Q}^{q,\mathcal{R}}$ with $\mathcal{M}^{q,\mathcal{R}}$ that contains a subset of CQs in natural bijection with those in $\mathcal{Q}_i$. Let $\mathcal{Q}_j^{\mathcal{S}}$ be the subset of $\mathcal{Q}_j$ that contains only the CQs on $\mathcal{S}$. $\mathcal{Q}_j^{\mathcal{S}}$ is a finite $\mathcal{S}$-rewriting of $(\mathcal{Q}^{q,\mathcal{R}}, \mathcal{M}^{q,\mathcal{R}})$.

Suppose $\mathcal{Q}_j^{\mathcal{S}}$ is not a complete $\mathcal{S}$-rewriting. Then, by Lemma B.1, there is a CQ $(Q_{\mathcal{P}})^T$ which belongs to an $\mathcal{S}$-rewriting of $(\mathcal{Q}^{q,\mathcal{R}}, \mathcal{M}^{q,\mathcal{R}})$ but which is not more specific than any of the CQs in $\mathcal{Q}_j^{\mathcal{S}}$. Then there is a CQ $(Q'_{\mathcal{P}})^T$ that belongs to a piece-rewriting $\mathcal{Q}'_j$ of $\mathcal{Q}^{q,\mathcal{R}}$ with $\mathcal{M}^{q,\mathcal{R}}$ such that $(Q_{\mathcal{P}})^T \models (Q'_{\mathcal{P}})^T$. Then by Lemma B.5, $Q'_{\mathcal{P}}$ is isomorphic to a CQ belonging to a piece-rewriting of $\{q\}$ with $\mathcal{R}^\star$, hence to a rewriting of $(\{Q\}, \mathcal{R})$. Since $\mathcal{Q}_i$ is a UCQ-rewriting, there is a Q_c in $\mathcal{Q}_i$ such that $Q'_{\mathcal{P}} \models Q_c$. Hence, $(Q'_{\mathcal{P}})^T \models (Q_c)^T$ (and thus $(Q_{\mathcal{P}})^T \models (Q_c)^T$) and, since $(Q_c)^T$ belongs to $\mathcal{Q}_j^{\mathcal{S}}$, this contradicts the assumption that $(Q_{\mathcal{P}})^T$ is not more specific than a CQ in $\mathcal{Q}_j^{\mathcal{S}}$.

($\Leftarrow$) Let $\mathcal{Q}^{\mathcal{S}}$ be a UCQ-$\mathcal{S}$-rewriting of $(\mathcal{Q}^{q,\mathcal{R}}, \mathcal{M}^{q,\mathcal{R}})$. Then there exists a piece-rewriting $\mathcal{Q}_i$ of $\mathcal{Q}^{q,\mathcal{R}}$ with $\mathcal{M}^{q,\mathcal{R}}$ such that $\mathcal{Q}^{\mathcal{S}} \models \mathcal{Q}_i$ (i.e., for each CQ Q' in $\mathcal{Q}^{\mathcal{S}}$, there is a CQ Q'' in $\mathcal{Q}_i$ such that $Q' \models Q''$).

Consider $\mathcal{Q}_i^{\mathcal{S}}$ the subset of $\mathcal{Q}_i$ that contains only the CQs on $\mathcal{S}$. We still have $\mathcal{Q}^{\mathcal{S}} \models \mathcal{Q}_i^{\mathcal{S}}$. Since $\mathcal{Q}^{\mathcal{S}}$ is complete w.r.t. $\mathcal{S}$, so is $\mathcal{Q}_i^{\mathcal{S}}$. Thus $\mathcal{Q}^{\mathcal{S}} \equiv \mathcal{Q}_i^{\mathcal{S}}$. By Lemma B.1, any CQ in $\mathcal{Q}_i^{\mathcal{S}}$ is of the form $(Q_{\mathcal{P}})^T$ as required in Lemma B.5. So, by Lemma B.5, there is a piece-rewriting $\mathcal{Q}_j$ of $\{q\}$ with $\mathcal{R}^\star$ that contains all the CQs in natural bijection with those in $\mathcal{Q}_i^{\mathcal{S}}$. So $\mathcal{Q}_j$ is a finite rewriting of $\{q\}$ with $\mathcal{R}^\star$. Since $\mathcal{R}^\star \equiv \mathcal{R}$ (see also Proposition B.4), it is also a finite rewriting of $\{q\}$ with $\mathcal{R}$.

Suppose $\mathcal{Q}_j$ is not complete. Then there is a CQ $Q_\mathcal{P}$ that belongs to a rewriting of $(\{q\},\mathcal{R})$ and is not more specific than any of the CQs in $\mathcal{Q}_j$. Then there is a CQ $Q'_\mathcal{P}$ that belongs to a piece-rewriting of $\{q\}$ with $\mathcal{R}$ such that $Q_\mathcal{P} \models Q'_\mathcal{P}$. Then by Lemma B.2, $(Q'_P)^T$ is isomorphic to a CQ belonging to a piece-rewriting of $\mathcal{Q}^{q,\mathcal{R}}$ with $\mathcal{M}^{q,\mathcal{R}}$. Since $\mathcal{Q}_i^\mathcal{S}$ is complete w.r.t. $\mathcal{S}$, there is a $(Q_c)^T$ in $\mathcal{Q}_i^\mathcal{S}$ such that $(Q'_\mathcal{P})^T \models (Q_c)^T$. We also have $Q'_\mathcal{P} \models Q_c$ (hence $Q_\mathcal{P} \models Q_c$) and, since Q_c belongs to $\mathcal{Q}_j$, this contradicts the assumption that $Q_\mathcal{P}$ is not more specific than a CQ in $\mathcal{Q}_j$. ■

Proofs of the three lemmas

We first point out the following.

- Thanks to the mapping assertions $\mathcal{M}_{trans}$, we can always "remove the hats" from any predicate (except the p_{R_i} special predicates) in any CQ Q_w belonging to a piece-rewriting of $\mathcal{Q}^{q,\mathcal{R}}$ with $\mathcal{M}^{q,\mathcal{R}}$; we just have to extend the rewriting sequence by using the rules in $\mathcal{M}_{trans}$. Moreover, if Q_w does not contain any special atom $p_{R_i}(\mathbf{x_i})$, this extended rewriting is on $\mathcal{S}$, hence belongs to an $\mathcal{S}$-rewriting of $\mathcal{Q}^{q,\mathcal{R}}$ with $\mathcal{M}^{q,\mathcal{R}}$.

- Another property of any CQ Q_w belonging to a piece-rewriting of $\mathcal{Q}^{q,\mathcal{R}}$ with $\mathcal{M}^{q,\mathcal{R}}$ is that each of its terms appears in a T or $\hat{T}$ atom. Indeed, since the CQs in $\mathcal{Q}^{q,\mathcal{R}}$ have a $\hat{T}$ atom for each term and all the variables of the rules in $\mathcal{M}^{q,\mathcal{R}}$ are frontier variables, no rewriting step introduces a new term without a $\hat{T}$, and the only rule that can rewrite a $\hat{T}$ atom replaces it with a T atom.

As an immediate consequence of the previous observations, we have the following lemmas.

Lemma B.1

Let Q be a CQ, $\mathcal{R}$ be a set of datalog rules and $Q_\mathcal{S}$ be a CQ belonging to an $\mathcal{S}$-rewriting of $\mathcal{Q}^{q,\mathcal{R}}$ with $\mathcal{M}^{q,\mathcal{R}}$. Then, $Q_\mathcal{S}$ is of form $(Q_\mathcal{P})^T$, where $Q_\mathcal{P}$ is a CQ on $\mathcal{P}$.

Lemma B.2

Let Q be a CQ, $\mathcal{R}$ be a set of datalog rules and Q_w be a CQ belonging to a piece-rewriting of Q with $\mathcal{R}$. Q_w^T is isomorphic to a CQ belonging to a piece-rewriting of $\mathcal{Q}^{q,\mathcal{R}}$ with $\mathcal{M}^{q,\mathcal{R}}$.

This lemma can be proved by induction thanks to the following proposition.

Proposition B.3

Let Q be a CQ, $\mathcal{R}$ be a set of datalog rules, $R_i \in \mathcal{R}$, and let $Q_w = \beta(Q, R_i, \mu)$ be a CQ where μ is a piece-unifier of Q with R_i. Let Q_{R_i} and m_{R_i} be, respectively, the CQ and $\mathcal{S}$-to-$\mathcal{T}$ rule associated with R_i as defined in the reduction. There exists a disjunctive piece-unifier $\mu_\vee$ of $\{Q_{R_i}, \hat{Q}^T\}$ with m_{R_i} such that $\hat{Q}_w^T$ is isomorphic to a CQ belonging to a piece-rewriting of $\{\beta_\vee(\{Q_{R_i}, \hat{Q}^T\}, m_{R_i}, \mu_\vee)\}$ with $\{T(x) \to \hat{T}(x)\}$.

Proof. Let $R_i = B_i[\mathbf{x_i}, \mathbf{y_i}] \to H_i(\mathbf{x_i})$ be a conjunctive rule. Since μ is a piece-unifier of Q with R_i, there is at least one atom with predicate H_i in Q, i.e., $Q = \exists \mathbf{u}, \mathbf{v}.H_i(\mathbf{u}) \wedge D[\mathbf{u}, \mathbf{v}]$ where D is any conjunction of atoms, and $Q_w = \exists \mathbf{u}, \mathbf{v}, \mathbf{y_i}.B_i[\mathbf{u}, \mathbf{y_i}] \wedge D[\mathbf{u}, \mathbf{v}]$. By the reduction, we obtain $q_{R_i} = \exists \mathbf{x_i}, \mathbf{y_i}.\hat{B}_i[\mathbf{x_i}, \mathbf{y_i}] \wedge p_{R_i}(\mathbf{x_i}) \wedge \hat{T}[\mathbf{x_i}, \mathbf{y_i}]$ and $m_{R_i} = T[\mathbf{x_i}] \to p_{R_i}(\mathbf{x_i}) \vee \hat{H}_i(\mathbf{x_i})$. We consider $\mu_\vee = \{\mu_{p_{R_i}}, \hat{\mu}\}$ where $\mu_{p_{R_i}}$ is the piece-unifier unifying $p_{R_i}(\mathbf{x_i}) \in Q_{R_i}$ and $\text{head}_1(m_{R_i})$, and $\hat{\mu}$ is the piece-unifier "isomorphic" to μ between $\hat{Q}$ and $\text{head}_2(m_{R_i})$. More formally, given $\mu = (Q', H', P_u)$ and $\hat{Q}^s$ a safe copy of $\hat{Q}$, $\hat{\mu} = ((\hat{Q}')^s, \hat{H}', P_u^s)$ where $.^s$ is the renaming function of the variables of $\hat{Q}$. Clearly, the join of the partitions in $\mu_\vee$ is admissible since there is no constant. Since $\hat{Q} \subseteq \hat{Q}^T$, $\hat{\mu}$ is a piece-unifier of $\hat{Q}^T$ with the rule $T[\mathbf{x_i}] \to \hat{H}_i(\mathbf{x_i})$, associated with $\text{head}_2(m_{R_i})$. Let u and $u_{\mu_\vee}$ be the substitutions associated with μ and $\mu_\vee$, respectively. Then;

$\beta_\vee(\{q_{R_i}, \hat{q}^T\}, m_{R_i}, \mu_\vee) = u_{\mu_\vee}(T[\mathbf{x_i}] \cup \hat{B}_i[\mathbf{x_i}, \mathbf{y_i}]^s \cup T[\mathbf{x_i}, \mathbf{y_i}]^s \cup (\hat{q}^T \setminus \hat{q}')^s) = u_{\mu_\vee}(\hat{B}_i[\mathbf{x_i}, \mathbf{y_i}]^s \cup (\hat{q} \setminus \hat{q}')^s)^T$

which is isomorphic to $u(\hat{B}_i[\mathbf{x_i}, \mathbf{y_i}] \cup (\hat{q} \setminus \hat{q}'))^T = \hat{q}_w^T$.

Since the partition associated with $\mu_{p_{R_i}}$ does not merge any frontier variables from m_{R_i}, no classes of $\hat{\mu}$ are merged in the join of the partitions of $\mu_{p_{R_i}}$ and $\hat{\mu}$. Hence, the joined partition is in bijection with P_u^s, and thus with P_u. As a consequence, $u(\hat{B}_i[\mathbf{x_i}, \mathbf{y_i}] \cup (\hat{q} \setminus \hat{q}'))^T$ is isomorphic to $u_{\mu_\vee}(\hat{B}_i[\mathbf{x_i}, \mathbf{y_i}]^s \cup (\hat{q} \setminus \hat{q}')^s)^T$. ■

Proof of Lemma B.2. By induction on the length k of the rewriting sequence from $\{Q\}$ producing $\mathcal{Q}_k$ in which Q_w is generated, we first prove that $(\hat{Q}_w)^T$ is isomorphic to a CQ belonging to a piece-rewriting of $\mathcal{Q}^{q,\mathcal{R}}$ with $\mathcal{M}^{q,\mathcal{R}}$:

- (k=0) $Q_w = Q$; since $Q_Q = \widehat{Q^T} \in \mathcal{Q}^{q,\mathcal{R}}$, we can produce $\hat{Q}^T = \hat{Q}_w^T$ by a rewriting sequence using the rule $T(x) \to \hat{T}(x)$.
- (k+1) Let $\mathcal{Q}_{k+1} = \mathcal{Q}_k \cup \{Q_w\}$. Assume Q_w is generated in $\mathcal{Q}_{k+1}$ by a piece-unifier of $Q_k \in \mathcal{Q}_k$ with $R \in \mathcal{R}$. By induction hypothesis, $(\hat{Q}_k)^T$ is isomorphic to a CQ belonging to a piece-rewriting of $\mathcal{Q}^{q,\mathcal{R}}$ with $\mathcal{M}^{q,\mathcal{R}}$. Then, by Proposition B.3, $(\hat{Q}_w)^T$ is isomorphic to a CQ belonging to a piece-rewriting of $\{\beta_\vee(\{Q_R, (\hat{Q}_k)^T\}, m_R, \mu_\vee)\}$ with $\{T(x) \to \hat{T}(x)\}$, hence a piece-rewriting of $\mathcal{Q}^{q,\mathcal{R}}$ with $\mathcal{M}^{q,\mathcal{R}}$.

Finally, we can "remove the hats" from any CQ belonging to a piece-rewriting of $\mathcal{Q}^{q,\mathcal{R}}$ with $\mathcal{M}^{q,\mathcal{R}}$. We just have to extend the rewriting sequence by some rewriting steps with $\mathcal{M}_{trans}$. Thus, since $(\hat{Q}_w)^T$ is isomorphic to a CQ belonging to a piece-rewriting of $\mathcal{Q}^{q,\mathcal{R}}$ with $\mathcal{M}^{q,\mathcal{R}}$, so is Q_w^T. ■

Next, we denote by $\mathcal{R}^\star$ the set of all the rules that can be obtained by composing rules from a datalog rule set $\mathcal{R}$. Composing two datalog rules is also known as "unfolding a rule by another". Given two datalog rules $R_1 = B_1 \to H_1$ and $R_2 = B_2 \to H_2$, and a (most general) classical unifier u of an atom A in B_2 with the atom in H_1, the *unfolding* of R_2 by R_1 is the rule $R_2 \circ R_1 = u(B_1) \cup u(B_2 \setminus \{A\}) \to u(H_2)$. Starting from $\mathcal{R}$, one can build $\mathcal{R}^\star$ by repeatedly unfolding a rule from $\mathcal{R}^\star$ by a rule from $\mathcal{R}$, until a fixpoint is reached (if any). Clearly, $R_1, R_2 \models R_2 \circ R_1$. Hence, $\mathcal{R}^\star$ is logically equivalent to $\mathcal{R}$.

Proposition B.4

Let Q be a CQ and $\mathcal{R}$ be a set of rules. Any UCQ $\mathcal{Q}$ is a complete rewriting of Q with $\mathcal{R}$ iff it is a complete rewriting of Q with $\mathcal{R}^\star$.

Proof. For all instance I and CQ Q, one has $I, \mathcal{R} \models Q$ iff $I, \mathcal{R}^\star \models Q$. Let $\mathcal{Q}$ be a complete rewriting of Q with $\mathcal{R}$. Then, for all I, $I \models \mathcal{Q}$ iff $I, \mathcal{R} \models Q$ iff $I, \mathcal{R}^\star \models Q$, thus $\mathcal{Q}$ is a complete rewriting of Q with $\mathcal{R}^\star$. Similarly, any complete rewriting $\mathcal{Q}$ of Q with $\mathcal{R}^\star$ is a complete rewriting of Q with $\mathcal{R}$. ■

Lemma B.5

Let Q be a CQ, $\mathcal{R}$ be a set of datalog rules and $Q_\mathcal{P}$ be a CQ on $\mathcal{P}$ such that $(Q_\mathcal{P})^T$ belongs to a piece-rewriting of $\mathcal{Q}^{q,\mathcal{R}}$ through $\mathcal{M}^{q,\mathcal{R}}$. Then, $Q_\mathcal{P}$ is isomorphic to a CQ belonging to a piece-rewriting of Q with $\mathcal{R}^\star$.

To prove the lemma, we first prove some properties of the piece-rewritings of $\mathcal{Q}^{q,\mathcal{R}}$ with $\mathcal{M}^{q,\mathcal{R}}$.

Proposition B.6

Let $\mathcal{Q}^w$ be a piece-rewriting of $\mathcal{Q}^{q,\mathcal{R}}$ with $\mathcal{M}^{q,\mathcal{R}}$, $\mathcal{Q}^w$ can be partitioned into two sets: $\mathcal{Q}^w_M$ the subset of CQs without any p_{R_i}-atom, and $\mathcal{Q}^w_P$ the subset of CQs with exactly one p_{R_i}-atom. Furthermore, $\mathcal{Q}^w_P$ is a rewriting of $\mathcal{Q}_\mathcal{R}$ (the subset of $\mathcal{Q}^{q,\mathcal{R}}$ containing only the queries associated with the rules from $\mathcal{R}$), and any $\mathcal{S}$-rewriting of $\mathcal{Q}^w$ with $\mathcal{M}_{trans}$ is a rewriting of $\mathcal{Q}^w_M$ with $\mathcal{M}_{trans}$.

Proof. We first show that any CQ in $\mathcal{Q}^w$ contains at most one p_{R_i} atom:

- it is the case for $\mathcal{Q}^{q,\mathcal{R}}$;
- piece-rewriting with a renaming mapping assertion in $\mathcal{M}_{trans}$ does not add a p_{R_i} atom;
- piece-rewriting with a disjunctive mapping assertion m_{R_i} removes a p_{R_i} atom (and does not add one), thus there remains at most one p_{R_j} atom in the produced query.

Thus $\mathcal{Q}^w_M + \mathcal{Q}^w_P = \mathcal{Q}^w$.

When we use a CQ without atom p_{R_i} in a piece-rewriting step, the produced query does not have such an atom either. So, we only have to consider the queries in $\mathcal{Q}_\mathcal{R}$ to generate $\mathcal{Q}^w_P$.

Since p_{R_i} predicates do not belong to $\mathcal{S}$ and no rule in $\mathcal{M}_{trans}$ allows to rewrite a p_{R_i}-atom, only the queries in $\mathcal{Q}^w_M$ can generate queries on $\mathcal{S}$ using $\mathcal{M}_{trans}$. ■

Definition B.1 (Reverse function)
Let $\mathcal{Q}^w$ be any rewriting of $\mathcal{Q}^{q,\mathcal{R}}$ with $\mathcal{M}^{q,\mathcal{R}}$. We define a "reverse" function, noted *reverse*, from $\mathcal{Q}^w$ to a set of CQs plus a set of conjunctive datalog rules, both on $\mathcal{P}$, as follows:

- for any $Q \in \mathcal{Q}^w_M$, $reverse(Q) = Q_r$ where Q_r is the query obtained from Q by removing the "hats" on the predicates, then deleting the T atoms;
- for any $Q \in \mathcal{Q}^w_P$, let $Q = (\exists \mathbf{x}, \mathbf{y}. p_{R_i}(\mathbf{x}) \wedge C[\mathbf{x}, \mathbf{y}])$. Note that C is a conjunction without any p_{R_j}-atom. Then: $reverse(Q) = C_r[\mathbf{x}, \mathbf{y}] \rightarrow H_i(\mathbf{x})$ where C_r is the conjunction obtained from C by removing the "hats" on the predicates, then deleting the T atoms, and $H_i(\mathbf{x})$ is obtained from the head of $R_i \in \mathcal{R}$ by substituting each frontier variable with the corresponding term in $p_{R_i}(\mathbf{x})$.

Proposition B.7

Let $\mathcal{Q}^w$ be a piece-rewriting of $\mathcal{Q}^{q,\mathcal{R}}$ with $\mathcal{M}^{q,\mathcal{R}}$. For any $Q_w \in \mathcal{Q}^w_P$, $reverse(Q) \in \mathcal{R}^\star$.

Proof. By induction on the length k of the sequence of piece-rewriting steps generating $\mathcal{Q}^w$:

- ($k = 0$) Recall that $\mathcal{Q}^w_P = \mathcal{Q}^{q,\mathcal{R}}_P = \mathcal{Q}_\mathcal{R}$. Now, observe that for each query $Q_{R_i} \in \mathcal{Q}_\mathcal{R}$, we have $reverse(Q_{R_i}) = R_i$ which belongs to $\mathcal{R}$.
- ($k + 1$) Any query $Q_w \in \mathcal{Q}^w_P$ is either obtained in at most k piece-rewriting steps and thus $reverse(Q_w) \in \mathcal{R}^k$ by induction hypothesis, or there are two cases (by Proposition B.6):
  - Q_w is generated by a piece-rewriting step from a CQ Q_k with a p_{R_i}-atom and a rule in $\mathcal{M}_{trans}$. Then $reverse(Q_w) = reverse(Q_k)$ and since Q_k is generated in at most k piece-rewriting steps, by the induction hypothesis $reverse(Q_k) \in \mathcal{R}^\star$.
  - Q_w is generated by a piece-rewriting step from two queries $Q_1 = (\exists \mathbf{x}_1, \mathbf{y}_1. p_{R_1}(\mathbf{x}_1) \wedge \hat{C}_1[\mathbf{x}_1, \mathbf{y}_1])$ and $Q_2 = (\exists \mathbf{x}_2, \mathbf{y}_2. p_{R_2}(\mathbf{x}_2) \wedge \hat{C}_2[\mathbf{x}_2, \mathbf{y}_2])$ with a disjunctive rule having one of the two special predicates p_{R_1} or p_{R_2}. Assume that the rule is $m_{R_1} = T[\mathbf{x}] \rightarrow p_{R_1}(\mathbf{x}) \vee \hat{H}_1(\mathbf{x})$ associated with R_1. Let $\mu_\vee =$

$\{\mu_1 = (p_{R_1}(\mathbf{x}), p_{R_1}(\mathbf{x_1}), P_1),\ \mu_2 = (\hat{C}'_2, \hat{H}'_1, P_2)\}$ be the disjunctive piece-unifier that has produced $Q_w = u_{\mu_\vee}(T \wedge \hat{C}_1 \wedge p_{R_2} \wedge (\hat{C}_2 \setminus \hat{C}'_2))$. Then, $reverse(Q_w) = u_{\mu_\vee}(C_1 \wedge (C_2 \setminus C'_2)) \rightarrow u_{\mu_\vee}(H_2)$.

By definition, $reverse(Q_1) = C_1 \rightarrow H_1$ and $reverse(Q_2) = C_2 \rightarrow H_2$. Let $.^s$ be the safe renaming of Q_2 used in μ_2. Thus we see that $\mu'_2 = (C'_2, H'_1, (P_2)^{s^{-1}})$ is a piece-unifier between C_2, the body of $reverse(Q_2)$, and H_1, the head of $reverse(Q_1)$. It follows that $reverse(Q_2) \circ reverse(Q_1) = u_{\mu'_2}(C_1 \wedge (C_2 \setminus C'_2)) \rightarrow u_{\mu'_2}(H_2)$.

By the induction hypothesis, $reverse(Q_1)$ and $reverse(Q_2)$ belong to $R^\star$, so $reverse(Q_2) \circ reverse(Q_1)$ belongs to $R^\star$. Since $reverse(Q_w)$ is isomorphic to $reverse(Q_2) \circ reverse(Q_1)$, it belongs to $R^\star$. ■

Proposition B.8

Let $\mathcal{Q}^w$ be a piece-rewriting of $\mathcal{Q}^{q,\mathcal{R}}$ with $\mathcal{M}^{q,\mathcal{R}}$. For any $Q_w \in \mathcal{Q}^w_M$, $reverse(Q_w)$ is isomorphic to a CQ that belongs to a piece-rewrite of $\{q\}$ with $\mathcal{R}^\star$.

Proof. By induction on the length k of the sequence of piece-rewriting steps generating $\mathcal{Q}^w$:

- $(k = 0)$ $\mathcal{Q}^w_M = \mathcal{Q}^{q,\mathcal{R}}_M = \{Q_Q\}$ and $reverse(Q_Q) = Q$.
- $(k + 1)$ Any query $Q_w \in \mathcal{Q}^w_M$ is either obtained in at most k piece-rewriting steps, hence, by induction hypothesis, $reverse(Q_w)$ is isomorphic to a CQ belonging to a piece-rewriting of Q with $\mathcal{R}^\star$, or there are two cases:
  - Q_w is generated by a piece-rewriting step from a CQ Q_k without p_{R_i} atom and a rule in $\mathcal{M}_{trans}$. Then $reverse(Q_w) = reverse(Q_k)$ and, since Q_k is generated in at most k piece-rewriting steps, by the induction hypothesis $reverse(Q_k)$ is isomorphic to a CQ belonging to a piece-rewriting of Q with $\mathcal{R}^\star$.
  - Q_w is generated by a (disjunctive) piece-rewriting step from a CQ $Q_m = (\hat{C} \wedge \hat{T})^T \in \mathcal{Q}^w_M$, a CQ $Q_R = (\hat{B}_R \wedge p_{R_i})^T$ and the rule $m_{R_i} = p_{R_i} \vee \hat{H}_i$. Let $\mu_\vee = \{\mu_i = (p_{R_i}, p_{R_i}, P_i),\ \mu_2 = (\hat{C}', \hat{H}'_i, P_2)\}$ be the disjunctive piece-unifier that has produced Q_w. Therefore, we have $Q_w = \beta_\vee(\{Q_m, Q_R\}, m_{R_i}, \mu_\vee) = u_{\mu_\vee}(\hat{B}_R \wedge (\hat{C} \setminus \hat{C}') \wedge \hat{T})^T$, hence $reverse(Q_w) = u_{\mu_\vee}(B_R \wedge (C \setminus C'))$.

    By Proposition B.7, $reverse(Q_R) = (B_R \rightarrow H_i) \in \mathcal{R}^\star$, and by the induction hypothesis, $reverse(Q_m) = C$ is isomorphic to a CQ belonging to a piece-rewriting of Q with $\mathcal{R}^\star$. Let μ'_2 be obtained from μ_2 by replacing each predicate $\hat{p}$ with p (that is, removing the hats). Then, μ'_2 is a piece-unifier of $reverse(Q_m)$ with $reverse(Q_R)$ (up to a bijective variable renaming) and $\beta(reverse(Q_m), reverse(Q_R), \mu'_2) = u_{\mu'_2}(B_R \wedge (C \setminus C'))$.

With the same arguments about the join of the partitions of $\mu_\vee$ and μ'_2 as at the end of the proof of Proposition B.3, we conclude that $u_{\mu'_2}(B_R \wedge (C \setminus C'))$ is isomorphic to $u_{\mu_\vee}(B_R \wedge (C \setminus C'))$. Thus, $reverse(Q_w)$ is isomorphic to a piece-rewriting of $\{q\}$ with $\mathcal{R}^\star$. ■

Proof of Lemma B.5. Assume $(q_\mathcal{P})^T$ belongs to a piece-rewriting of $\mathcal{Q}^{q,\mathcal{R}}$ with $\mathcal{M}^{q,\mathcal{R}}$.

Since $(q_\mathcal{P})^T$ is on $\mathcal{S}$, it belongs to $\mathcal{Q}^w_M$. Hence, by Proposition B.8, $reverse((q_\mathcal{P})^T) = q_\mathcal{P}$ is isomorphic to a CQ belonging to a piece-rewriting of $\{q\}$ with $\mathcal{R}^\star$. ■

C - Complexity

C.1 Common complexity classes

In this dissertation, we mention various complexity classes for decision problems. Although not all necessary technical concepts are defined here, we present the class definitions, ordered by increasing complexity.

Definition C.1 (AC_0)
A problem is part of the AC_0 class if it can be solved by a limited depth Boolean circuit, using a polynomial number of AND and OR gates.

Definition C.2 (NLogSpace (NL))
A problem belongs to NL if it can be resolved by a non-deterministic Turing machine that only uses a working tape of logarithmic space relative to the input.

Definition C.3 (Polynomial Time (PTime))
A problem is classified as P if it can be solved by a deterministic Turing machine running in polynomial time with respect to the input.

Definition C.4 (Non-deterministic Polynomial Time (NP) / coNP)
A problem is categorized as NP if it can be solved by a nondeterministic Turing machine running in polynomial time relative to the input. A problem is in coNP if it complement is in NP.

Definition C.5 (Polynomial Space (PSpace))
A problem is solved in PSpace if it can be solved by a Turing machine that only uses a tape of polynomial space in relation to the input. Note that it has been demonstrated that non-deterministic polynomial space is equivalent to deterministic polynomial space.

In the following definitions, an oracle for a problem is a hypothetical device that is capable of solving that problem instantly. When a Turing machine is augmented with an oracle, it can consult the oracle at any time and use the answer provided by the oracle.

Definition C.6 (Σ_n^P)
A problem is classified as Σ_n^P if it can be solved by a nondeterministic Turing machine running in polynomial time augmented by an oracle for a problem that is complete for Σ_{n-1}^P, where $\Sigma_0^P = \mathsf{P}$.

Definition C.7 (Π_n^P)
A problem falls under Π_n^P if it is the complement of a problem that can be solved by a nondeterministic Turing machine running in polynomial time augmented by an oracle for a problem that is complete for Σ_{n-1}^P, where $\Sigma_0^P = \mathsf{P}$.

Definition C.8 (Δ_n^P)
A problem is part of Δ_n^P if it can be solved by a deterministic Turing machine running in polynomial time with an oracle augmented by a problem that is complete for Σ_{n-1}^P, where $\Sigma_0^P = \mathsf{P}$.

The classes Σ_n^P, Π_n^P, and Δ_n^P are part of a hierarchy of complexity classes known as the polynomial hierarchy. Each subsequent class in this hierarchy represents problems that are intuitively harder than the previous classes. However, all these problems are solvable in polynomial time if given an oracle for a problem from the appropriate lower class.

Definition C.9 (Exponential Time (ExpTime))
A problem belongs to ExpTime if it can be solved by a deterministic Turing machine operating in simple exponential time relative to the input.

Definition C.10 (Double Exponential Time (2ExpTime))
A problem is solved in 2ExpTime if it can be solved by a deterministic Turing machine running in double exponential time relative to the input.

Definition C.11 (Triple Exponential Time (3ExpTime))
A problem is solved in 3ExpTime if it can be solved by a deterministic Turing machine running in triple exponential time relative to the input.

In these definitions, double and triple exponential time refer to the time complexities 2^{2^n} and $2^{2^{2^n}}$, respectively, where n is the input size.

A problem P is considered hard for a given complexity class C if any problem instance from C can be transformed into an instance of P through a process called a reduction. This reduction must be an "adapted" reduction. This typically means that the reduction itself can be computed in polynomial time, although for certain lower complexity classes (like PTime and below), the reduction must use only logarithmic space.

In detail, the idea of a reduction is a way of formalising the intuition that one problem is at least as hard as another problem. If we can efficiently transform any instance of problem A into an instance of problem B, then problem B is at least as hard as problem A. If B can be solved, then A can also be solved by transforming it into an instance of B and solving that. This transformation process is what we refer to as a reduction.

Finally, a problem P is considered complete for a given complexity class C if it belongs to the class C and is hard for C. In other words, P is as hard as the hardest problems in C. If there is a way to solve P efficiently (in polynomial time, for instance), then every problem in C can be solved efficiently. Conversely, if P cannot be solved efficiently, then no problem in C can be solved efficiently. This is why complete problems are very important in the study of computational complexity: they represent the upper bound of difficulty within their complexity class.

C.2 Types of Computational Complexity

In the context of computational complexity, we often consider the cost of a problem in terms of two different aspects: the size of the input data and the size or complexity of the problem's expression or query. This leads to three notions of complexity: data complexity, expression complexity, and combined complexity [Abiteboul et al., 1995].

Definition C.12 (Data Complexity)
The data complexity of a problem is the computational complexity of solving the problem as a function of the size of the input data, keeping the expression or query fixed.

Definition C.13 (Expression Complexity)
The expression complexity of a problem is the computational complexity of solving the problem as a function of the size of the expression or query, keeping the input data fixed.

Definition C.14 (Combined Complexity)
The combined complexity of a problem is the computational complexity of solving the problem as a function of the size of both the input data and the expression or query.

These concepts are particularly useful in areas such as database theory, where we often differentiate between the complexity of a query (expression complexity), the complexity of processing a database (data complexity), and the complexity of the overall task (combined complexity). In many practical scenarios, we are often more interested in data complexity because the query can be considered fixed and small, while the database can be very large. On the contrary, when developing query optimisation techniques or considering the impact of changes to the query, one might be more interested in expression complexity.

D - Supplementary algorithms

D.1 Main steps of the algorithm TargetRewriting

Here are the main steps of the algorithm *TargetRewriting* from [Pérez, 2011] - for more details and explanations, see Lemmas 6.2.3, 6.2.6, and A.2.3 in [Pérez, 2011]):

1. **Input**:
   - A mapping $\mathcal{M}$ composed of rules where the inequalities are allowed in the body.
   - A query $\mathcal{Q}_\mathcal{S}(\mathbf{x})$ in UCQ$^{\neq}$ over $V_\mathcal{S}$ that is target-rewritable under $\mathcal{M}$ (that is, we *know* that there exists a perfect $\mathcal{M}^{-1}$-translation of $\mathcal{Q}_\mathcal{S}(\mathbf{x})$).
2. **Converting to a Monotone Query over** $\mathcal{C}_\mathcal{M}$:
   (a) Let $\mathcal{C}_\mathcal{M} = \{\mathcal{M}\text{–}\texttt{rewriting}(H[\mathbf{x}]) \mid B[\mathbf{x}] \rightarrow H[\mathbf{x}] \in \mathcal{M}\}$
   (b) Transform $\mathcal{Q}_\mathcal{S}(\mathbf{x})$ to obtain a logically equivalent formula which is a monotone query over $\mathcal{C}_\mathcal{M}$, that is, we obtain a query only composed of conjunctions of conjunctive queries in $\mathcal{C}_\mathcal{M}$ (according to the dissertation, it is always possible):
      i. Break down $\mathcal{Q}_\mathcal{S}(\mathbf{x})$ into a finite disjunction of formulas, following these constraints[1]:
         - Separate $\mathbf{x}$ into two disjoint sets $\mathbf{x}'$ and $\mathbf{x}''$.
         - Form $\theta(\mathbf{x}', \mathbf{x}'')$ as a conjunction of equalities respecting the inequalities in $\mathcal{Q}_\mathcal{S}(\mathbf{x})$ and such that for every $u \in \mathbf{x}''$ we have that $\theta(\mathbf{x}', \mathbf{x}'')$ contains an equality $u = v$ with $v \in \mathbf{x}'$.
         - Define $\gamma(\mathbf{x}', \mathbf{y})$ as a subset of the conjuncts of $\mathcal{Q}_\mathcal{S}(\mathbf{x})$, with replacements according to $\theta(\mathbf{x}', \mathbf{x}'')$.
         - Form $\delta(\mathbf{x}', \mathbf{y})$ as a conjunction of inequalities for each pair of distinct variables in $(\mathbf{x}', \mathbf{y})$.
      ii. The result is a disjunction with disjuncts of the following form:

$$\exists \mathbf{y} \, (\gamma(\mathbf{x}', \mathbf{y}) \wedge \delta(\mathbf{x}', \mathbf{y}) \wedge \theta(\mathbf{x}', \mathbf{x}''))$$

3. **Build disjuncts of the translation**:
   For every disjunct $\exists \mathbf{y} \, (\gamma(\mathbf{x}', \mathbf{y}) \wedge \delta(\mathbf{x}', \mathbf{y}) \wedge \theta(\mathbf{x}', \mathbf{x}''))$:
   (a) For each conjunction $\alpha(\mathbf{u}) \in \gamma(\mathbf{x}', \mathbf{y})$ such that $\alpha(\mathbf{u}) \in \mathcal{C}_\mathcal{M}$, we know that it is the rewriting of a head $H[\mathbf{x}]$: add $H[\mathbf{u}]$ to $\gamma^\star(\mathbf{x}', \mathbf{y}')$[2].

[1] The previous work does not give more details about how to do that.
[2] This part looks very similar to a chase of γ with $\mathcal{M}$.

(b) Include the corresponding disjunct in the translation $\mathcal{Q}_{\mathcal{O}}(\mathbf{x})$:

$$\exists \mathbf{y}' \left(\gamma^{\star}(\mathbf{x}', \mathbf{y}') \wedge \delta(\mathbf{x}', \mathbf{y}') \wedge \mathbf{C}(\mathbf{x}', \mathbf{y}') \wedge \theta(\mathbf{x}', \mathbf{x}'') \right)$$

4. **Output**:

   - A query $\mathcal{Q}_{\mathcal{O}}(\mathbf{x})$ in $\text{UCQ}^{\neq,\mathbf{C}}$ over $V_{\mathcal{O}}$ that is a $\mathcal{M}^{-1}$-translation of $\mathcal{Q}_{\mathcal{S}}(\mathbf{x})$, such that, if an inequality $x \neq y$ occurs in one of its CQs, we also have in it the atoms $\mathbf{C}(x)$ and $\mathbf{C}(y)$.

D.2 Algorithm that computes a $R^{\vee}_{\mathbf{K},\neq}$-Maximally Sound Σ-translation

This section contains algorithms from [Cima et al., 2022]. Algorithm 20 is the algorithm in [Cima et al., 2022] that computes a $R^{\vee}_{\mathbf{K},\neq}$-Maximally Sound Σ-translation of a UCQ when considering an ontology with DL-LITE$_{RDFS}$.

Algorithm 20: Compute $R^{\vee}_{\mathbf{K},\neq}$-Maximally Sound Σ-translation

Input: OBDM specification $\Sigma = (\mathcal{O}, \mathcal{S}, \mathcal{M})$ where $\mathcal{O}$ is a DL-LITE$_{RDFS}$ ontology, UCQ $\mathcal{Q}_{\mathcal{S}} = \{q^1_{\mathcal{S}}, \ldots, q^n_{\mathcal{S}}\}$ over $V_{\mathcal{S}}$
Output: $R^{\vee}_{\mathbf{K},\neq}$ rule program $\mathcal{Q}_{\mathcal{O}}$

```
M' ← InvMap(Σ) ;                                   // Algorithm 21
for i = 1 to n do
    M' ← M' ∪ {q^i_S(x_i) → Ans(x_i)};
end
Q_O ← rename(M');
return Q_O;
function rename(M');
foreach predicate s ∈ V_S do
    Create a fresh copy ŝ of predicate s;
    foreach rule in m ∈ M' do
        Replace the predicate s by ŝ in m;
    end
end
return M';
```

We give below the main steps of the algorithm InvMap (Algorithm 21).

The InvMap algorithm is an algorithm that computes the inverse of a given GLAV mapping $\mathcal{M}$ with respect to an ontology $\mathcal{O}$ in DL-LITE$_{RDFS}$. The algorithm works with a KBDM specification and outputs a set of $R^{\vee}_{\mathbf{K},\neq}$ rules. Intuitively, the main steps are:

1. **Preprocessing**: Merge the ontology with the mapping and then transform the original GLAV mapping into a saturated form using the Saturate phase (Algo-

rithm 23), which is not a chase, but a phase where we add inequalities in the bodies of the rules.

2. **Split**: For each mapping assertion, compute all possible splittings around variable sets to break down the relational atoms into connected components (Algorithm 24).

3. **Compute Generators and calculate supports**: For each split component, compute the generators with respect to the given mapping (Algorithm 25). Derive the supports for the generators, introducing necessary restrictions and equalities (Definition D.3). The supports act as the structural framework connecting different generators.

4. **Formulate $R^{\vee}_{\mathbf{K},\neq}$ Rules**: Based on the supports and generators, form an $R^{\vee}_{\mathbf{K},\neq}$ rule.

Algorithm 21: Inversion of mapping to obtain a $\mathrm{R}^{\vee}_{\mathbf{K}}$-mapping

Input: KBDM specification $\Sigma = (\mathcal{O}, \mathcal{S}, \mathcal{M})$ with $\mathcal{O}$ a DL-LITE$_{RDFS}$ ontology
Output: Set R of $\mathrm{R}^{\vee}_{\mathbf{K}}$ rules
$\mathcal{M}' \leftarrow \mathrm{merge}(\Sigma)$; // Algorithm 22
$\mathcal{M}' \leftarrow \mathrm{SaturateM}(\mathcal{M}')$; // Algorithm 23
$\mathcal{R} \leftarrow \emptyset$;
foreach $m : \phi(\mathbf{x},\mathbf{y}) \wedge \xi(\mathbf{x}) \rightarrow \exists \mathbf{z}.\psi(\mathbf{x},\mathbf{z}) \in \mathcal{M}'$ **do**
  $\Gamma \leftarrow \emptyset$;
  foreach $(U, V) \in \mathit{Partitions}(\mathbf{z})$ **do**
    Let $S_\psi \leftarrow \mathrm{Split}(\psi, U)$; // Algorithm 24
    // See Algorithm 25 for Gens
    foreach $G \in \mathit{Gens}(S_\psi, \mathcal{M}', V \cup \mathbf{x})$ **do**
      // See Definition D.2 for $\eta_G^{[\mathbf{x}]}$
      if $(x, x') \notin \eta_G^{[\mathbf{x}]}$ *for each* $x, x' \in \mathbf{x}$ *s.t.* $x \neq x'$ **then**
        Let $\sigma^{\mathbf{x}}_G$ be the $\mathbf{x}$-support of G ; // Definition D.3
        $\Gamma \leftarrow \Gamma \cup \{\sigma^G_{\mathbf{x}}\}$;
      end
    end
  end
  $\mathrm{inv}(m) \leftarrow (\mathbf{K}(\exists \mathbf{z}.\psi(\mathbf{x},\mathbf{z}) \wedge \xi(\mathbf{x})) \rightarrow \bigvee_{\gamma \in \Gamma} \gamma)$;
  $\mathcal{R} \leftarrow \mathcal{R} \cup \{\mathrm{inv}(m)\}$;
end
return $\mathcal{R}$;

Definition D.1 (Relation η_G)
Let $G = \langle g_1, \ldots, g_n \rangle$ be a tuple in $\mathrm{Gens}(A, \mathcal{M}, V)$ with $g_i = \langle h_i, m_i \rangle$ for each $i = 1, \ldots, n$, and a tuple of renamings $\langle \rho_1, \ldots, \rho_n \rangle$ for the variables of $\mathcal{M}$ whose images are pairwise

Algorithm 22: Merging GLAV Mapping with DL-LITE$_{\text{RDFS}}$ Ontology

Input: OBDM specification $\Sigma = (\mathcal{O}, \mathcal{S}, \mathcal{M})$, with $\mathcal{O}$ a DL-LITE$_{\text{RDFS}}$ ontology
Output: Equivalent mapping $\mathcal{M}'$
$\mathcal{M}' \leftarrow \emptyset$;
foreach $m = \forall \mathbf{x}.\forall \mathbf{y}.\phi(\mathbf{x},\mathbf{y}) \rightarrow \exists \mathbf{z}.\psi(\mathbf{x},\mathbf{z}) \in \mathcal{M}$ **do**
  $\mathcal{M}' \leftarrow \mathcal{M}' \cup \{\forall \mathbf{x}.\forall \mathbf{y}.\phi(\mathbf{x},\mathbf{y}) \rightarrow \exists \mathbf{z}.\mathtt{chase}(\psi(\mathbf{x},\mathbf{z})), \mathcal{R}_{\mathcal{O}})\}$;
end
foreach *n-ary predicate* $s \in V_{\mathcal{S}}$ **do**
  $\mathcal{M}' \leftarrow \mathcal{M}' \cup \{\forall x_1,\ldots,x_n.s(x_1,x_2,\ldots,x_n) \rightarrow \top(x_1) \wedge \top(x_2) \wedge \ldots \wedge \top(x_n)\}$;
end
return $\mathcal{M}'$;

disjoint. Define $\beta_i = \rho_i(\text{r-body}(m_i))$ for $i = 1,\ldots,n$, and $\sigma_G = \bigvee_i \rho_i(\text{r-body}(m_i))$. Intuitively, each β_i represents a set of relational source atoms that is needed to generate a homomorphic image of $h_i(A_i)$ using m_i.

In order to connect together the different $h_i(A_i)$ to form an image of A, we define the binary relation η_G over $\text{Var}(\sigma_G)$ as follows:

$$\eta_G = \left\{\Big(\rho_i(h_i(v)), \rho_j(h_j(v))\Big) \mid i,j = 1,\ldots,n,\, v \in V\right\}$$

These relations impose restrictions on the variables of σ_G, taking the form of equalities, to ensure the connectivity of the image of A.

Definition D.2 (Relations $\eta^{[\mathbf{x}]}$ and $\eta_G^{[\mathbf{x}]}$)
Given a binary relation η_G over $\mathtt{vars}(\sigma_G)$ defined Definition D.1, and a set $\mathbf{x} \subseteq V$, let $\eta^{[\mathbf{x}]}$ be the relation denoted by:

$$\eta^{[\mathbf{x}]} = \{(x, \rho_i(h_i(x))) \mid i = 1,\ldots,n,\, x \in \mathbf{x}\}$$

This binary relation extends η_G with specific elements from $\mathbf{x}$.

Finally, the relation $\eta_G^{[\mathbf{x}]}$ is defined as the reflexive and transitive closure of $\eta_G \cup \eta^{[\mathbf{x}]}$:

$$\eta_G^{[\mathbf{x}]} = \text{closure}(\eta_G \cup \eta^{[\mathbf{x}]})$$

Definition D.3 ($\mathbf{x}$-support of G)
Given a tuple G, a set $\mathbf{x} \subseteq V$, and the relations η_G and $\eta_G^{[\mathbf{x}]}$ defined in Definitions D.1 and D.2, we define the $\mathbf{x}$-support as follows:

For each equivalence class E of η_G, select an element $s(E) \in E$ as representative; if E contains elements of $\mathbf{x}$, require $s(E) \in \mathbf{x}$. Given $v \in \text{Var}(\sigma_G)$, we use $v \sim_{G,\mathbf{x}}$ to denote the equivalence class of $\eta_G^{[\mathbf{x}]}$ that contains v.

Finally, define the $\mathbf{x}$-support $\sigma_G^{[\mathbf{x}]}$ of G as the formula:

$$\sigma_G^{[\mathbf{x}]} = \exists \mathbf{w}.\eta_G^{[\mathbf{x}]}(\sigma_G)$$

Algorithm 23: Saturating a GLAV Mapping

Input: GLAV mapping $\mathcal{M}$
Output: Saturated GLAV mapping
SaturatedMapping $\leftarrow \emptyset$;
foreach $m = \forall \mathbf{x}.\forall \mathbf{y}.\phi(\mathbf{x},\mathbf{y}) \rightarrow \exists \mathbf{z}.\psi(\mathbf{x},\mathbf{z}) \in \mathcal{M}$ **do**
  foreach *possible unifier* μ *between variables in* $\mathbf{x}$ **do**
    $m' \leftarrow \mu(m)$;
    foreach *pair of distinct variables* $x_1, x_2 \in \mu(\mathbf{x})$ **do**
      Add the inequality atom $x_1 \neq x_2$ in the body of m';
    end
    SaturatedMapping $\leftarrow$ SaturatedMapping $\cup \{m'\}$;
  end
end
return *SaturatedMapping*;

Algorithm 24: Splitting a set of relational atoms around a set of variables

Input: Set of relational atoms A, Set of variables U
Output: Set of subsets of A that form connected components of G^U_A
Initialize G^U_A as an empty graph with nodes representing atoms in A;
foreach *atom* $\alpha \in A$ **do**
  add the node α to G^U_A;
end
foreach *pair of nodes* $\alpha_1, \alpha_2 \in A$ **do**
  if *there exists* $z \in U$ *such that* z *occurs in both* α_1 *and* α_2 **then**
    Add edge between α_1 and α_2 in G^U_A;
  end
end
$result \leftarrow \emptyset$;
foreach *atom* $\alpha \in A$ **do**
  if $\alpha \notin result$ **then**
    $C \leftarrow \emptyset$;
    foreach *atom* $\alpha' \in A$ *reachable from* α *in* G^U_A **do**
      $C \leftarrow C \cup \{\alpha'\}$;
    end
    $result \leftarrow result \cup \{C\}$;
  end
end
return *result*;

where $\eta_G^{[\mathbf{x}]}(\sigma_G)$ is obtained from σ_G by replacing each variable $v \in \mathrm{Var}(\sigma_G)$ with $s(v \sim_{G,\mathbf{x}})$, and $\mathbf{w}$ are the variables of $\eta_G^{[\mathbf{x}]}(\sigma_G)$ not occurring in $\mathbf{x}$.

Algorithm 25: Generators of a Saturated GLAV Mapping

Input: Tuple of sets of atoms $A = (A_1, \ldots, A_n)$, Saturated GLAV mapping $\mathcal{M}$, Set of variables $V \subseteq \mathtt{vars}(A_1) \cup \ldots \cup \mathtt{vars}(A_n)$
Output: Set of tuples of V-distinguished generators G

```
G ← ∅;
foreach A_i ∈ A do
    foreach m ∈ M do
        foreach homomorphism h from A_i to head of m do
            if h(v) is a frontier variable of m for each v ∈ V and h(u) is not a frontier
              variable of m for each u ∉ V then
                Create a generator g = (h, m);
                Add g to the corresponding G_i for A_i;
            end
        end
    end
    foreach tuple G' = (g_1, ..., g_n) such that g_i ∈ G_i, for each i = 1, ..., n;
     do
        // See the definition D.4 for the meaning of V-distinguished
        if G' satisfies the conditions of being V-distinguished then
            Add G' to G;
        end
    end
end
return G;
```

Definition D.4 (V-distinguished Generator)
Let $g = (h, m)$ be a generator, where h is a homomorphism from a set of atoms A to the head of m, where m is a source-to-target rule. Let $V \subseteq \mathtt{vars}(A)$ be a set of variables. Then, the generator g is called V-distinguished if the following conditions are met:

- For each $v \in V$, $h(v)$ is a frontier variable of m.
- For each $u \notin V$, $h(u)$ is not a frontier variable of m.

Index

Bibliography

[Abiteboul et al., 1995] Abiteboul, S., Hull, R., and Vianu, V. (1995). *Foundations of Databases*. Addison-Wesley.

[Ahmetaj et al., 2018] Ahmetaj, S., Ortiz, M., and Simkus, M. (2018). Rewriting guarded existential rules into small datalog programs. In Kimelfeld, B. and Amsterdamer, Y., editors, *21st International Conference on Database Theory, ICDT 2018, March 26-29, 2018, Vienna, Austria*, volume 98 of *LIPIcs*, pages 4:1–4:24. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

[Alfonso et al., 2021] Alfonso, E. M., Chortaras, A., and Stamou, G. (2021). Ucq-rewritings for disjunctive knowledge and queries with negated atoms. *Semantic Web*, 12(4):685–709.

[Alviano et al., 2012] Alviano, M., Faber, W., Leone, N., and Manna, M. (2012). Disjunctive datalog with existential quantifiers: Semantics, decidability, and complexity issues. *Theory Pract. Log. Program.*, 12(4-5):701–718.

[Arenas et al., 2009a] Arenas, M., Pérez, J., Reutter, J. L., and Riveros, C. (2009a). Inverting schema mappings: Bridging the gap between theory and practice. *Proc. VLDB Endow.*, 2:1018–1029.

[Arenas et al., 2010] Arenas, M., Pérez, J., Reutter, J. L., and Riveros, C. (2010). Foundations of schema mapping management. In *Proceedings of the twenty-ninth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 227–238.

[Arenas et al., 2008] Arenas, M., Pérez, J., and Riveros, C. (2008). The recovery of a schema mapping: bringing exchanged data back. In *ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*.

[Arenas et al., 2009b] Arenas, M., Pérez, J., and Riveros, C. (2009b). The recovery of a schema mapping: Bringing exchanged data back. *ACM Trans. Database Syst.*, 34:22:1–22:48.

[Artale et al., 2007] Artale, A., Calvanese, D., Kontchakov, R., and Zakharyaschev, M. (2007). Dl-lite in the light of first-order logic. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence, July 22-26, 2007, Vancouver, British Columbia, Canada*, pages 361–366. AAAI Press.

[Artale et al., 2009] Artale, A., Calvanese, D., Kontchakov, R., and Zakharyaschev, M. (2009). The dl-lite family and relations. *J. Artif. Intell. Res.*, 36:1–69.

[Baader et al., 2005] Baader, F., Brandt, S., and Lutz, C. (2005). Pushing the EL envelope. In Kaelbling, L. P. and Saffiotti, A., editors, *IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK, July 30 - August 5, 2005*, pages 364–369. Professional Book Center.

[Baader et al., 2007] Baader, F., Calvanese, D., McGuinness, D. L., Nardi, D., and Patel-Schneider, P. F. (2007). The description logic handbook. In *The Description Logic Handbook*.

[Baget et al., 2011] Baget, J., Leclère, M., Mugnier, M., and Salvat, E. (2011). On rules with existential variables: Walking the decidability line. *Artif. Intell.*, 175(9-10):1620–1654.

[Baget et al., 2023] Baget, J.-F., Bisquert, P., Leclère, M., Mugnier, M.-L., Pérution-Kihli, G., Tornil, F., and Ulliana, F. (2023). Integraal: a tool for data-integration and reasoning on heterogeneous and federated sources. In *BDA 2023: 39ème Conférence sur la Gestion de Données – Principes, Technologies et Applications*, Montpellier, France. LIRMM, Inria, Univ Montpellier, CNRS, INRAE.

[Baget et al., 2009] Baget, J.-F., Leclère, M., Mugnier, M.-L., and Salvat, E. (2009). Extending Decidable Cases for Rules with Existential Variables. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence, IJCAI 2009*, pages 677–682.

[Barceló et al., 2018] Barceló, P., Berger, G., Lutz, C., and Pieris, A. (2018). First-order rewritability of frontier-guarded ontology-mediated queries. In Lang, J., editor, *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*, pages 1707–1713. ijcai.org.

[Beeri and Vardi, 1981] Beeri, C. and Vardi, M. Y. (1981). The implication problem for data dependencies. In *Automata, Languages and Programming: Eighth Colloquium Acre (Akko), Israel July 13–17, 1981 8*, pages 73–85. Springer.

[Bienvenu et al., 2014] Bienvenu, M., ten Cate, B., Lutz, C., and Wolter, F. (2014). Ontology-based data access: A study through disjunctive datalog, csp, and MMSNP. *ACM Trans. Database Syst.*, 39(4):33:1–33:44.

[Bourhis et al., 2016] Bourhis, P., Manna, M., Morak, M., and Pieris, A. (2016). Guarded-based disjunctive tuple-generating dependencies. *ACM Trans. Database Syst.*, 41(4):27:1–27:45.

[Buron et al., 2020] Buron, M., Goasdoué, F., Manolescu, I., and Mugnier, M. (2020). Ontology-based RDF integration of heterogeneous data. In Bonifati, A., Zhou, Y., Salles, M. A. V., Böhm, A., Olteanu, D., Fletcher, G. H. L., Khan, A., and Yang, B., editors, *Proceedings of the 23rd International Conference on Extending Database Technology, EDBT 2020, Copenhagen, Denmark, March 30 - April 02, 2020*, pages 299–310. OpenProceedings.org.

[Buron et al., 2021] Buron, M., Mugnier, M., and Thomazo, M. (2021). Parallelisable existential rules: a story of pieces. In Bienvenu, M., Lakemeyer, G., and Erdem, E., editors, *Proceedings of the 18th International Conference on Principles of Knowledge*

Representation and Reasoning, KR 2021, Online event, November 3-12, 2021, pages 162–173.

[Bursztyn et al., 2015] Bursztyn, D., Goasdoué, F., and Manolescu, I. (2015). Optimizing reformulation-based query answering in RDF. In Alonso, G., Geerts, F., Popa, L., Barceló, P., Teubner, J., Ugarte, M., den Bussche, J. V., and Paredaens, J., editors, *Proceedings of the 18th International Conference on Extending Database Technology, EDBT 2015, Brussels, Belgium, March 23-27, 2015*, pages 265–276. OpenProceedings.org.

[Calí et al., 2009] Calí, A., Gottlob, G., and Lukasiewicz, T. (2009). A general datalog-based framework for tractable query answering over ontologies. In *ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*.

[Calì et al., 2010] Calì, A., Gottlob, G., and Pieris, A. (2010). Advanced processing for ontological queries. *Proc. VLDB Endow.*, 3(1):554–565.

[Calvanese et al., 2017] Calvanese, D., Cogrel, B., Komla-Ebri, S., Kontchakov, R., Lanti, D., Rezk, M., Rodriguez-Muro, M., and Xiao, G. (2017). Ontop: Answering SPARQL queries over relational databases. *Semantic Web*, 8(3):471–487.

[Calvanese et al., 2011] Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Poggi, A., Rodriguez-Muro, M., Rosati, R., Ruzzi, M., and Savo, D. F. (2011). The MASTRO system for ontology-based data access. *Semantic Web*, 2(1).

[Calvanese et al., 2007a] Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., and Rosati, R. (2007a). Eql-lite: Effective firsta-order query processing in description logics. In *IJCAI*, volume 7, pages 274–279.

[Calvanese et al., 2007b] Calvanese, D., Giacomo, G. D., Lembo, D., Lenzerini, M., and Rosati, R. (2007b). Tractable reasoning and efficient query answering in description logics: The *DL-Lite* family. *J. Autom. Reason.*, 39(3):385–429.

[Carral et al., 2017] Carral, D., Dragoste, I., and Krötzsch, M. (2017). Restricted chase (non)termination for existential rules with disjunctions. In Sierra, C., editor, *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 922–928. ijcai.org.

[Carral et al., 2018] Carral, D., Krötzsch, M., Marx, M., Ozaki, A., and Rudolph, S. (2018). Preserving constraints with the stable chase. In *International Conference on Database Theory*.

[Chortaras et al., 2011] Chortaras, A., Trivela, D., and Stamou, G. (2011). Optimized query rewriting for owl 2 ql. In *CADE*.

[Cima, 2020] Cima, G. (2020). *Abstraction in ontology-based data management*. PhD thesis, Sapienza University of Rome, Italy.

[Cima et al., 2022] Cima, G., Console, M., Lenzerini, M., and Poggi, A. (2022). Monotone Abstractions in Ontology-Based Data Management. In *AAAI Conference on Artificial Intelligence*.

[Cima et al., 2019] Cima, G., Lenzerini, M., and Poggi, A. (2019). Semantic characterization of data services through ontologies. In *International Joint Conference on Artificial Intelligence*.

[Cima et al., 2020] Cima, G., Lenzerini, M., and Poggi, A. (2020). Non-monotonic ontology-based abstractions of data services. In *International Conference on Principles of Knowledge Representation and Reasoning*.

[Cima et al., 2023] Cima, G., Poggi, A., and Lenzerini, M. (2023). The notion of abstraction in ontology-based data management. *Artif. Intell.*, 323:103976.

[Console et al., 2013] Console, M., Lenzerini, M., Mancini, R., Rosati, R., and Ruzzi, M. (2013). Synthesizing extensional constraints in ontology-based data access. In *Description Logics*.

[Eiter et al., 2012] Eiter, T., Ortiz, M., Simkus, M., Tran, T.-K., and Xiao, G. (2012). Query rewriting for horn-shiq plus rules. *Proceedings of the AAAI Conference on Artificial Intelligence*.

[Fagin et al., 2011] Fagin, R., Kolaitis, P., Popa, L., and Tan, W.-c. (2011). Reverse data exchange: Coping with nulls. *ACM Trans. Database Syst.*, 36:11.

[Fagin et al., 2005] Fagin, R., Kolaitis, P. G., Miller, R. J., and Popa, L. (2005). Data exchange: semantics and query answering. *Theor. Comput. Sci.*, 336(1):89–124.

[Fagin et al., 2004] Fagin, R., Kolaitis, P. G., Popa, L., and Tan, W. C. (2004). Composing schema mappings: second-order dependencies to the rescue. *ACM Trans. Database Syst.*, 30:994–1055.

[Gaifman et al., 1993] Gaifman, H., Mairson, H. G., Sagiv, Y., and Vardi, M. Y. (1993). Undecidable optimization problems for database logic programs. *J. ACM*, 40(3):683–713.

[Gerasimova et al., 2020] Gerasimova, O., Kikot, S., Kurucz, A., Podolskii, V. V., and Zakharyaschev, M. (2020). A data complexity and rewritability tetrachotomy of ontology-mediated queries with a covering axiom. In Calvanese, D., Erdem, E., and Thielscher, M., editors, *Proceedings of the 17th International Conference on Principles of Knowledge Representation and Reasoning, KR 2020, Rhodes, Greece, September 12-18, 2020*, pages 403–413.

[Gottlob et al., 2012] Gottlob, G., Manna, M., Morak, M., and Pieris, A. (2012). On the complexity of ontological reasoning under disjunctive existential rules. In Rovan, B., Sassone, V., and Widmayer, P., editors, *Mathematical Foundations of Computer Science*

2012 - 37th International Symposium, MFCS 2012, Bratislava, Slovakia, August 27-31, 2012. Proceedings, volume 7464 of *Lecture Notes in Computer Science*, pages 1–18. Springer.

[Gottlob et al., 2011] Gottlob, G., Orsi, G., and Pieris, A. (2011). Ontological queries: Rewriting and optimization. *2011 IEEE 27th International Conference on Data Engineering*, pages 2–13.

[Gottlob and Schwentick, 2011] Gottlob, G. and Schwentick, T. (2011). Rewriting ontological queries into small nonrecursive datalog programs. *ArXiv*, abs/1106.3767.

[Grahne and Onet, 2018] Grahne, G. and Onet, A. (2018). Anatomy of the chase. *Fundamenta Informaticae*, 157(3):221–270.

[Hell and Nesetril, 2004] Hell, P. and Nesetril, J. (2004). Graphs and homomorphisms. In *Oxford lecture series in mathematics and its applications*.

[Kharlamov et al., 2017] Kharlamov, E., Hovland, D., Skjæveland, M. G., Bilidas, D., Jiménez-Ruiz, E., Xiao, G., Soylu, A., Lanti, D., Rezk, M., Zheleznyakov, D., Giese, M., Lie, H., Ioannidis, Y. E., Kotidis, Y., Koubarakis, M., and Waaler, A. (2017). Ontology based data access in statoil. *J. Web Semant.*, 44:3–36.

[Kőnig, 1927] Kőnig, D. (1927). Über eine schlussweise aus dem endlichen ins unendliche. *Acta litt. sci. Reg. Univ. Hung. Francisco-Josephinae, Sect. sci. math.*, 3(2-3):121–130.

[König, 2014] König, M. (2014). *Interrogation de grandes bases de connaissances : algorithmes de réécriture de requêtes conjonctives en présence de règles existentielles Soutenue le. (Ontology-based Query Answering)*. PhD thesis, Universté de Montpellier.

[König et al., 2015] König, M., Leclère, M., Mugnier, M., and Thomazo, M. (2015). Sound, complete and minimal ucq-rewriting for existential rules. *Semantic Web*, 6(5):451–475.

[Leclère et al., 2023] Leclère, M., Mugnier, M.-L., and P'erution-Kihli, G. (2023). Query rewriting with disjunctive existential rules and mappings. In *International Conference on Principles of Knowledge Representation and Reasoning*.

[Lembo et al., 2015] Lembo, D., Lenzerini, M., Rosati, R., Ruzzi, M., and Savo, D. F. (2015). Inconsistency-tolerant query answering in ontology-based data access. *J. Web Semant.*, 33:3–29.

[Lenzerini, 2002] Lenzerini, M. (2002). Data integration: A theoretical perspective. In Popa, L., Abiteboul, S., and Kolaitis, P. G., editors, *Proceedings of the Twenty-first ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 3-5, Madison, Wisconsin, USA*, pages 233–246. ACM.

[Lenzerini, 2018] Lenzerini, M. (2018). Managing data through the lens of an ontology. *AI Mag.*, 39(2):65–74.

[Lenzerini, 2019] Lenzerini, M. (2019). Direct and reverse rewriting in data interoperability. In *International Conference on Advanced Information Systems Engineering*.

[Lutz et al., 2018] Lutz, C., Marti, J., and Sabellek, L. (2018). Query expressibility and verification in ontology-based data access. In *International Conference on Principles of Knowledge Representation and Reasoning*.

[Lutz et al., 2009] Lutz, C., Toman, D., and Wolter, F. (2009). Conjunctive query answering in the description logic EL using a relational database system. In Boutilier, C., editor, *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*, pages 2070–2075.

[Melnik, 2004] Melnik, S. (2004). Generic model management: Concepts and algorithms. In *Generic Model Management: Concepts And Algorithms*.

[Morak, 2021] Morak, M. (2021). Sticky existential rules and disjunction are incompatible. In Bienvenu, M., Lakemeyer, G., and Erdem, E., editors, *Proceedings of the 18th International Conference on Principles of Knowledge Representation and Reasoning, KR 2021, Online event, November 3-12, 2021*, pages 691–695.

[Mugnier, 2020] Mugnier, M. (2020). Data access with horn ontologies: Where description logics meet existential rules. *Künstliche Intell.*, 34(4):475–489.

[Nikolaou et al., 2019] Nikolaou, C., Grau, B. C., Kostylev, E. V., Kaminski, M., and Horrocks, I. (2019). Satisfaction and implication of integrity constraints in ontology-based data access. In *International Joint Conference on Artificial Intelligence*.

[Pérez, 2011] Pérez, J. (2011). *Schema Mapping Management in Data Exchange Systems*. PhD thesis, Citeseer.

[Pérez, 2013] Pérez, J. (2013). The inverse of a schema mapping. In *Data Exchange, Information, and Streams*.

[Pérez-Urbina et al., 2010] Pérez-Urbina, H., Motik, B., and Horrocks, I. (2010). Tractable query answering and rewriting under description logic constraints. *J. Appl. Log.*, 8:186–209.

[Poggi et al., 2008] Poggi, A., Lembo, D., Calvanese, D., Giacomo, G. D., Lenzerini, M., and Rosati, R. (2008). Linking data to ontologies. *J. Data Semant.*, 10:133–173.

[Rodriguez-Muro et al., 2013] Rodriguez-Muro, M., Kontchakov, R., and Zakharyaschev, M. (2013). Ontology-based data access: Ontop of databases. In *ISWC*.

[Rodriguez-Muro et al., 2013] Rodriguez-Muro, M., Kontchakov, R., and Zakharyaschev, M. (2013). Query rewriting and optimisation with database dependencies in ontop. In *Description Logics*.

[Rosati, 2012] Rosati, R. (2012). Query rewriting under extensional constraints in dl-lite. In *Description Logics*.

[Rudolph and Krötzsch, 2013] Rudolph, S. and Krötzsch, M. (2013). Flag & check: data access with monadically defined queries. In *ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*.

[Sequeda et al., 2014] Sequeda, J. F., Arenas, M., and Miranker, D. P. (2014). OBDA: query rewriting or materialization? in practice, both! In *ISWC*.

[Thomazo, 2013] Thomazo, M. (2013). *Conjunctive query answering under existential rules-decidability, complexity, and algorithms*. PhD thesis, Université Montpellier II-Sciences et Techniques du Languedoc.

[Trivela et al., 2015] Trivela, D., Stoilos, G., Chortaras, A., and Stamou, G. (2015). Optimising resolution-based rewriting algorithms for dl ontologies. In *Description Logics*.

[van der Meyden, 1997] van der Meyden, R. (1997). The complexity of querying indefinite data about linearly ordered domains. *J. Comput. Syst. Sci.*, 54(1):113–135.

Abstract

The general context of this work is the issue of designing high-quality systems that integrate multiple data sources via a semantic layer encoded in a knowledge representation and reasoning language. We consider knowledge-based data management (KBDM) systems, which are structured in three layers: the data layer, which comprises the data sources, the knowledge (or ontological) layer, and the mappings between the two. Mappings and knowledge are expressed within the existential rule framework. One of the intrinsic difficulties in designing a KBDM is the need to understand the content of data sources. Data sources are often provided with typical queries and constraints, from which valuable information about their semantics can be drawn, as long as this information is made intelligible to KBDM designers. This motivates our core question: is it possible to translate data queries and constraints at the knowledge level while preserving their semantics?

The main contributions of this thesis are the following. We extend previous work on data-to-ontology query translation with new techniques for the computation of perfect, minimally complete, or maximally sound query translations. Concerning data-to-ontology constraint translation, we define a general framework and apply it to several classes of constraints. Finally, we provide a sound and complete query rewriting operator for disjunctive existential rules and disjunctive mappings, as well as undecidability results, which are of independent interest.

Keywords: Data Quality, Ontology Based Data Access, Existential Rules, Knowledge based data management, Query rewriting, Constraint translation

---

Résumé

Le contexte général de ce travail concerne le problème de la conception de systèmes de haute qualité qui intègrent plusieurs sources de données via une couche sémantique codée dans un langage de représentation et de raisonnement des connaissances. Nous considérons les systèmes de gestion de données basés sur les connaissances (KBDM), qui sont structurés en trois couches : la couche de données, qui comprend les sources de données, la couche de connaissances (ou ontologique) et les mappings entre les deux. Les mappings et les connaissances sont exprimés dans le cadre des règles existentielles. L'une des difficultés intrinsèques à la conception d'un système KBDM est la nécessité de comprendre le contenu des sources de données. Les sources de données sont souvent fournies avec des requêtes et des contraintes typiques, à partir desquelles des informations précieuses sur leur sémantique peuvent être tirées, tant que ces informations sont rendues intelligibles aux concepteurs d'un système KBDM. Cela motive notre question principale : est-il possible de traduire les requêtes et contraintes sur les données au niveau ontologique tout en préservant leur sémantique ?

Les principales contributions de cette thèse sont les suivantes. Nous étendons les travaux précédents sur la traduction de requêtes des données vers l'ontologie avec de nouvelles techniques pour le calcul de traductions de requêtes parfaites, minimalement complètes ou maximalement adéquates. Concernant la traduction de contraintes des données vers l'ontologie, nous définissons un cadre général et l'appliquons à plusieurs classes de contraintes. Enfin, nous fournissons un opérateur de réécriture de requête adéquat et complet pour les règles existentielles disjonctives et les mappings disjonctifs, ainsi que des résultats d'indécidabilité, qui sont d'intérêt indépendant.

Mots-clés : Qualité des données, Accès aux données médiatisé par une ontologie, Règles existentielles, Gestion des données basée sur des connaissances, Ré-écriture de requête, Traduction de contraintes