Efficient reasoning on large-scale heterogeneous data

Maxime Buron

INSTITUT POLYTECHNIQUE DE PARIS

NNT : 2020IPPAX061

Thèse de doctorat

Raisonnement efficace sur des grands graphes hétérogènes

Thèse de doctorat de l'Institut Polytechnique de Paris
préparée à l'Ecole polytechnique

École doctorale n°626 Ecole Doctorale de l'Institut Polytechnique de Paris (ED IP Paris)
Spécialité de doctorat : Informatique

Thèse présentée et soutenue à Palaiseau, le 7 octobre 2020, par

MAXIME BURON

Composition du Jury :

Maurizio Lenzerini Professeur, Université de Rome	Rapporteur
Marie-Christine Rousset Professeur, Université de Grenoble	Rapporteur (Présidente)
Meghyn Bienvenu Chargé de recherche, CNRS	Examinateur
Alin Deutsch Professeur, Université de Californie, San Diego	Examinateur
François Goasdoué Professeur, Université de Rennes 1	Co-directeur
Ioana Manolescu Directeur de recherche, INRIA	Directeur
Marie-Laure Mugnier Professeur, Université de Montpellier	Co-directeur

Résumé

Le Web Sémantique a pour but de rendre les données du web compréhensible par les machines et de lier les bases de données ainsi formées, par des liens similaires aux hyperliens présents entre les pages du web. Le modèle de graphe RDF a été adopté pour représenter de telles données, qui décrivent les choses apparaissant sur le web et les liens, qu'elles partagent. Les capacités de flexibilité de RDF permettent aux graphes RDF d'intégrer des données très hétérogènes en une base de connaissances unifiées. Le vocabulaire utilisé dans les graphes RDF pour les types de lies et les types de choses, ainsi que les contraintes existantes entre ces types sont définies dans une ontologie. RDF Schema (RDFS) spécifie une ontologie d'un graphe RDF comme une partie de ce graphe. Les ontologies sont habituellement définies avec une sémantique, qui permet de déduire de nouvelles données en utilisant celles contenues originalement dans la base de connaissance. Pour extraire des informations d'une base de connaissance, nous l'interrogeons à l'aide de requêtes. SPARQL a été introduit comme le language de requête standard pour RDF, le coeur de ce langage est composé par les requêtes Basic Graph Pattern. Dans cette thèse, nous avons étudié des méthodes pour répondre à des requêtes BGP générales sur les graphes RDF en prenant en compte les connaissances introduites par une ontologie RDFS. Nous avons proposé des méthodes efficaces pour deux configurations distinctes; une où le graphe que nous souhaitons requêter est stocké dans une base de données et une deuxième, où le graphe intègre des connaissances issues de sources de données hétérogènes.

La première partie se focalise sur comment la grande partie de l'expressivité du raisonnement RDFS peut être pris en compte lors de la réponse à une requête. Premièrement, nous avons étudié une partition de l'ensemble des règles de déduction RDFS en deux sous-ensembles, qui définissent la sémantique des ontologies RDFS. Cette partition vient avec des conditions impliquant un raisonnement séparé, i.e., n'induisant que des connaissances ontologiques avec un sous-ensemble des règles et seulement des connaissances en dehors de l'ontologie pour l'autre sous-ensemble. Nous avons montré que sous ces conditions certaines capacités de "metamodeling" de RDF(S) sont encore utilisables, c'est à dire que l'on peut définir des données a propos du vocabulaires définie dans l'ontologie d'un graphe. Deuxièmement, en exploitant la partition des règles de déduction RDFS, nous avons défini un algorithme de reformulation de requête BGP, qui est correcte et complet, incorporant le raisonnement permis par une ontologie dans une requête. Ainsi, le problème de réponse à une requête en prenant en compte le raisonnement RDFS est

réduit au problème d'évaluation d'une requête dans une base de données RDF sans raisonnement. Finalement, nous avons introduit une nouvelle structure de stockage pour les données RDF, en combinant deux structures connues, ce qui améliore l'évaluation de requête dans les bases de données RDF, particulièrement pour certaines requêtes BGP générales.

Dans la deuxième partie, nous considérons le problème d'interrogation, par des requêtes BGP, de sources de données hétérogènes intégrées en un graphe RDF. Nous introduisons un cadre d'intégration de données sous des contraintes ontologiques RDFS, utilisant une spécification d'intégration basée sur des mappings Global-Local-As-View, rarement considérée jusqu'ici dans la littérature. Nous présentons plusieurs stratégies de réponse à des requêtes, qui, soit matérialisent les données en un graphe RDF, soit laissent ce graphe virtuel. Ces stratégies diffèrent sur quand et comment le raisonnement RDFS est supporté. Nous avons implémenté ces stratégies dans une plate-forme et mené des expérimentations qui démontrent l'intérêt particulier d'une des stratégies basée sur la saturation des mappings. Finalement, nous montrons que cette dernière technique peut être étendue au delà des règles de déduction RDFS au raisonnement défini par un sous-ensemble des règles existentielles.

Abstract

The Semantic Web aims to make the data on the web machine-comprehensible and to link together the databases thus formed in a similar way the hyperlinks link the web pages. The RDF graph data model has been adopted to represent such data, which describe the things appearing on the web and the links they share. The RDF flexibility abilities allow RDF graphs to integrate very heterogeneous data in a unified knowledge base. The vocabulary used in RDF graphs for the type of links and the type of things and the constraint existing between them are defined is an ontology. RDF Schema (RDFS in short) specifies an ontology of a graph as a part of it. Ontologies usually come with a semantic which enables to entail new data through a reasoning process from the data originally specified in the knowledge base. To extract information from a knowledge base, we query it. SPARQL have been introduced as the standard query language for RDF, its core is formed by the Basic Graph Pattern queries. In this thesis, we have investigated some methods to answer general BGP queries on RDF graphs by taking into account the knowledge induced by an RDFS ontology. We have proposed some efficient solutions for two distinct settings; one where the graph we want to query is stored into a database and a second where the queried graph integrates the knowledge from heterogeneous data sources.

The first part focuses on how to take into account a large part of the expressivity of RDFS reasoning in efficient query answering techniques. First, we study a partition of the RDFS entailment rule set into two subsets, which both define the semantic of RDFS ontologies. It comes with the conditions ensuring split reasoning, i.e., inferring only ontological knowledge with one rule subset, and only knowledge outside the ontology with the other subset. And we show that under these conditions some metamodeling capabilities of RDF(S) are still usable, which means that it allows to define some data about the vocabulary defined in an RDFS ontology. Second, exploiting the partition of RDFS entailment rules, we introduce a sound and complete query reformulation algorithm for BGP queries, incorporating the reasoning enabled by the ontology into the query. Hence, the problem of query answering under RDFS reasoning is reduced to a query evaluation in a RDF database with no reasoning. Finally, we introduce a novel RDF storage layout, which combines two well-known layouts, that improves query evaluation performance in RDF databases, especially for some general BGP queries. We also observe an improvement of the performance using a saturation-based query answering methods, where the entailed knowledge is stored with the other knowledge in a pre-processing step.

In a second part, we have studied BGP query answering methods on heterogeneous data sources through a RDF graph with an RDFS ontology. It takes in the general context of data integration, where several data sources are integrated under an unified vocabulary. The RDF data model has gained wide acceptance for modeling and sharing heterogeneous data of various domains, notably in the context of linked data. Following the Ontology-Based Data Access paradigm, we introduce a framework of data integration under an RDFS ontology, using the Global-Local-As-View mappings, which is a very expressive mapping language. GLAV mappings enable a certain kind of value invention which increases the amount of information accessible through the integration e.g., to state the existence of some data whose values are not known in the sources. We present several query answering strategies, which may materialize the integrated RDF graph into an RDF database or leave it virtual, and differ on how and when RDFS reasoning is handled. We implement these strategies in a platform, in order to conduct experiments, which demonstrate the particular interest of one of the strategies based on mapping saturation. Finally, we show that mapping saturation can be extended to reasoning defined by a subset of existential rules.

Contents

List of Figures

List of Tables

Remerciements

Maintenant que la soutenance est passée, je peux calmement repenser à ces trois dernières années et remercier les personnes, qui m'ont aidées, guidées et inspirées. En premier lieu, je souhaite remercier Ioana, François et Marie-Laure pour l'encadrement de qualité que vous m'avez procuré. Malgré la distance géographique et une pandémie, vous avez toujours été là et avez manifesté de l'intérêt pour moi. C'était très précieux.

Merci à Maurizio Lenzerini et à Marie-Christine Rousset pour avoir accepté de rapporter ce manuscript et pour vos retours très positifs. Je souhaite aussi remercier Meghyn Bienvenu et Alin Deutsch d'avoir chacun trouvé le temps pour être examinateur. Finalement, je souhaite dire merci à Fabian Suchanek, qui a presque pu être examinateur, et dont les courses de master m'ont beaucoup motivé à faire une thèse dans ce domaine.

Merci aussi à toute l'équipe Cedar, que ce soit les anciens ou les nouveaux membres, d'avoir partagé vos histoires et votre joie de vivre. Je pense notamment à Khaled, Mirjana, Pawel et Tayeb, formant le noyau dur du groupe, qui mangeait au Magnan et avec qui les discussions ont été riches et rafraîchissantes. Merci aussi à Michaël pour le super encadrement de stage. Je souhaite remercier Luciano, Enhui et Khaled pour m'avoir accompagné au cours l'aventure du doctorat ; vous allez réussir, j'en suis sûr. Je repense aussi à Alexandre et Félix, avec qui j'aurais aimé passer plus de temps. Merci aux nouveaux permanents Angelos pour les conseils et Oana pour les échanges sur l'écologie. Je souhaite remercier les assistantes Maëva et Hanadi, qui m'ont beaucoup aidé à organiser mes déplacements, tâche pour laquelle je suis très mauvais.

Au cours de ma thèse, j'ai aussi eu le plaisir de rencontrer de nombreuses personnes dans des réunions, des visites ou des conférences. Je tiens à les remercier toutes. Pour les accueils chaleureux à Montpellier, merci à Guillaume, Olivier, Federico et aux Michels et à Lannion, merci à toute l'équipe, j'étais heureux de revoir Ludivine et Cheikh Brahim. Je souhaite aussi remercier chaleureusement les participants et organisateurs des projets Icoda, CQFD et WebClaimExplain.

Et pour finir, je remercie tous ceux qui me soutiennent depuis longtemps. Premièrement, ma famille, qui a toujours eu une confiance aveugle dans mes études, maman, papa, Mathilde, mes grands-mères et mon grand-père, qui nous a quitté au cours de ma première année de thèse et à qui je dédie ce manuscrit. Enfin, mes amis, notamment mes nouveaux et anciens collocs, Florian, Thibault, Ted, Arthur, Fabien, Baptiste, Thich, Ted, Jérémy,

Bob et Capucine - grand merci à toi pour le soutien moral - et ceux qui vivent plus loin : Mélodie, Jason, Bastien et Justine, vous me manquez.

Chapter

1 Introduction

Semantic web

In 2001, Tim Berners-Lee published a visionary article introducing the *semantic web*, with the goal of making all the data on the web machine-comprehensible. The semantic web relies on the RDF data model, defined in the Resource Description Framework specification [Lassila and Swick, 1999], which allows to make statements about resource identifiers (Universal Resource Identifiers, or URIs in short) [RFC, a]. These statements take the form of *triples*. For instance, a triple may state "Berlin is located in Germany" by gathering three URIs, which respectively identify the city of Berlin, the property of being geographically located and Germany. Through triples, the RDF data model allows to describe links that exist between resources, which together form RDF graphs.

The self-describing characteristic of the RDF data model eases publications of and contributions to a variety of graphs available online. Common resources are used to link these graphs together building a large web of heterogeneous data. The Linked Open Data cloud[1] displays such a network of graphs, where we can find, for instance, data from the Wikipedia encyclopedia in the DBpedia graph[2], data from MusicBrainz[3], a collaborative platform with music information, and data about many other topics like government, life science, etc.

The most shared resources across graphs, are those that constitute the *schema* of graphs. These resources are the *classes*, which allow to type resources, e.g., "Berlin is of type capital" states that Berlin is a country's capital, and the *properties*, which define links between resources, such as "is located in" mentioned above in "Berlin is located in Germany".

The RDF Schema (RDFS, in short) W3C specification [Lassila and Swick, 1999] specifies a schema vocabulary for defining constraints on properties and classes. These constraints are specified using triples as well, such as, for instance, the constraint "the capital class is a subclass of the city class". In a graph, such triples form an *ontology*. The semantics of an RDF graph, in which some triples form its ontology, is given by the set of RDFS entailment rules, which enable to infer new triples, called the *implicit triples* of

[1] `https://lod-cloud.net`
[2] `https://dbpedia.org`
[3] `http://musicbrainz.org`

this graph, as opposed to its explicit triples.

Other languages, mainly based on first-order logic, allow representing ontological knowledge and reasoning about it. Notably, Description Logics [Baader et al., 2003] (DLs) are a prominent family of languages designed for this purpose. Some members of this family, with less expressivity, have been specially tailored to access data, such as the DL-Lite family [Calvanese et al., 2007]. Let us also mention existential rules, a more recent language, which generalizes many DLs used to access data (see e.g.,[Calì et al., 2009, Mugnier and Thomazo, 2014] for introductions). Most DLs, even lightweight ones, are able to represent more complex constraints than RDFS. They adopt a representation of classes and properties based on, respectively, unary and binary relations. This choice of representation naturally isolates properties and classes from individuals.

On the other hand, an RDF graph is able to assert properties of its schema resources. For instance, a graph can state that "the class of capitals is related to the class of countries". Expressing such statements is not possible in classical description logics. We say that RDF graphs are capable of *metamodeling*.

The SPARQL language [Prud'hommeaux and Seaborne, 2004], introduced in 2004, is the standard query language for RDF. It relies on Basic Graph Patterns (BGP) and its core fragment is that of BGP queries, which play the same role as conjunctive queries in relational databases. For instance, a BGP query may ask is "what is both located in Germany and instance of the class of capitals ?", where the properties and classes mentioned by the query are URIs. However, general BGP queries also allow to interrogate the graph schema, as in the query "which classes does Berlin belong to ?", or RDFS constraints, as in the query "which are the subclasses of the class of cities?". Such BGP queries are called *non-relational* and allow a user to explore the schema of a graph, which can be useful for instance to formulate further queries using different facets defined by the graph schema.

Efficient answering of expressive queries

We distinguish *query answering*, which consists of answering queries by taking into account not only the explicit triples of an RDF graph, but also the implicit triples entailed from the graph by RDFS entailment rules; in contrast, *query evaluation* only considers the explicit triples of a graph, just like a DBMS considers the data explicitely stored in the database. In our example, the query "is Berlin a city ?" is answered positively when the implicit triples are taken into account, since Berlin belongs to the capitals class, which is a subclass of cities class. However, the evaluation of same query would yield false as an answer. Considering ontological constraints and associated inferred triples greatly increases the usefulness of the queried graph, as it allows a user to pose a query using the vocabulary he/she is familiar with, and still get complete answers.

However, taking the ontological knowledge into account in query answering comes at a price: query answering incurs more effort than query evaluation. Two classical approaches exist to solve the query answering problem, both of which include a *reasoning* step that reduces query answering to query evaluation. The first approach, called *graph saturation* consists to compute all the implicit triples and store these triples together with the explicit ones. This can be done in a preprocessing step, as it does not depend on the query, followed at query time by the evaluation of the query on the saturated graph, which returns the complete answers set. The second approach performs the reasoning step at query time, starting with a *query reformulation* step, which incorporates the ontological knowledge into the query, followed by the evaluation of the reformulated query on the

original graph. While the saturation-based approach requires time and space to compute and store the saturated graph, it leads to good performance at query time. However, the saturation has to be updated when the graph changes, hence it is not adapted to dynamic settings. On the other hand, the reformulation-based approach is not impacted by graph changes, but the reasoning step occurs at query time and the rewritten query is typically harder to evaluate. These two approaches have been developed for several ontological formalisms. Depending on the expressivity of the formalism, classical saturation or/and query reformulation may not terminate in all cases.

An aspect with an important impact on query evaluation performance is the way in which data is stored (and read from) a stable storage. In an RDF database, the set of persistent data structures which, together, hold the data comprised in an RDF graph is usually called *storage layout*. The evaluation of a BGP query in RDF databases requires translating the query into a description of work (usually called *query plan*), which involves reading the data from the storage layout.

In this thesis, we consider both reformulation-based and saturation-based query answering approaches. We propose a *new query reformulation algorithm* for general BGP queries using RDFS entailment rules. As the evaluation of general BGP queries, and all the more reformulated queries, challenges existing RDF storage layouts, we propose a *new storage layout* with a corresponding translation from BGP queries to logical query plans, that improves query evaluation performance in both query answering approaches, especially for non-relational BGP queries.

Ontology-based integration of heterogeneous data sources

The development of data management systems has quickly lead to the need to *integrate* several databases under a single schema, or unified view. The RDF data model has gained wide acceptance for modeling and sharing heterogeneous data of various domains, notably in the context of linked data.

An *integration system* describes the relation between the schemas of data sources, called *local schemas* and a *global schema*, used to model integrated data [Doan et al., 2012].

Traditional query evaluation approaches in data integration systems fall into two classes: *materialization* [Jarke, 2003], where all data source content is migrated in a single centralized store, and *mediation* [Wiederhold, 1992], where data remains in its local store and queries expressed on the global schema are processed by single module called *mediator*.

The simplest option for describing an integration system is to define each element of the global schema as a view over the local schemas; this is known as Global-As-View (GAV in short). In a GAV system, a query over the global schema is easily transformed into a query over the local schemas, by *unfolding* each global schema relation, i.e., replacing it with its definition. Given the simplicity of unfolding, the complexity of query evaluation with the mediation-based approach is due to query evaluation across the different sources.

In contrast, in a Local-As-View (LAV) integration system, elements of the local schemas are defined as views over the global one. Mediation-based query evaluation in a LAV system requires *rewriting the query with the views* describing the local sources [Halevy, 2001], which can be a challenging task for mediator.

GLAV (Global-Local-As-View) data integration [Friedman et al., 1999] generalizes both GAV and LAV. In GLAV scenarios, a query q_1 over one or several local schemas is associated to a query over the global schema q_2, both having the same answer variables; the pair (q_1, q_2) is commonly called a *mapping*. The semantics is that for each answer

of q_1, the integration system exposes the data comprised in a corresponding answer of q_2. GLAV maximizes the *integration expressive power*, in the following sense: unlike LAV, a GLAV mapping may expose only part of a given source's data, and may combine data from several sources; unlike GAV, a GLAV mapping may include joins or complex expressions over the global schema. Moreover, GLAV mappings enable a certain kind of *value invention* which *increases the amount of information accessible through the integration system*, e.g., to state *the existence* of some data whose *values* are not known in the sources.

Ontology-Based Data Access (OBDA) has recently emerged as a data integration paradigm based on ontologies, which allow to represent domain knowledge that is exploited when accessing data, relying on knowledge representation and reasoning techniques [Poggi et al., 2008]. It distinguishes between a conceptual level defined by the ontology, which plays the role of the global schema, and a data level defined by data sources, both levels being connected by mappings. Currently available OBDA systems usually combine a DL ontology, a relational database and GAV mappings.

Our work follows the OBDA vision, although it implements it in a different setting, namely RDFS instead of description logics, GLAV mappings instead of GAV mappings, and heterogeneous data sources that may not be relational. We tackle the problem of general BGP query answering in such integration systems, by comparing several query answering strategies, based on a data materialization or mediating approach concerning integration, and saturation or query reformulation techniques concerning ontological reasoning, and proposing new strategies.

Organization of the thesis

The thesis is organized as follows:

Chapter 2 This chapter introduces basic notions about RDF graphs, RDFS ontologies, RDF entailment rules, including the subset of standard RDFS entailment rules, and Basic Graph Pattern (BGP) queries. We define the problem of query answering in RDF graphs with respect to RDF entailment rules. The second part of the chapter reviews notions and results about data integration systems, based on GAV (with query unfolding), LAV (with view-based query rewriting) and GLAV mappings.

Chapter 3 This chapter presents our work about query answering on RDF graphs with general BGP queries and standard RDFS entailment rules. First, we situate our work in relation to the state of the art. Second, we study a partition of the RDFS entailment rule set into two subsets, as well as the conditions ensuring split reasoning, i.e., inferring only ontology triples with one rule subset, and only triples outside the ontology with the other subset. This partition allows us to introduce a sound and complete query reformulation algorithm. Finally, we investigate a new storage layout for RDF graphs, which combines two well-known layouts. Our experiments show that the combined layout avoids the performance pitfalls of the two others, especially for non-relational BGP queries.

Chapter 4 This chapter presents our work on heterogeneous data source integration in the RDF data model, whereas we study BGP query answering. We start by classifying existing works about mediation-based query answering, and point out that among these, few have been devoted to integration systems using GLAV mappings. We introduce our framework, RDF integration systems, with the associated query answering problem. We

present several query answering strategies, including two novel ones, following either the materialization or the mediation approach, and differring on how and when RDFS reasoning is incorporated. These strategies are implemented in Obi-Wan, our RDF integration system, which allowed us to experimentally compare them. Finally, we extend the theoretical framework for query answering to a subset of existential rules.

Chapter 5 This chapter concludes this thesis and indicates some possible extensions of this work.

So far, this work has led to the following publications:

- [Buron et al., 2018] **Rewriting-Based Query Answering for Semantic Data Integration Systems (Informal publication)** Maxime Buron, François Goasdoué, Ioana Manolescu, Marie-Laure Mugnier, in *34ème Conférence sur la Gestion de Données – Principes, Technologies et Applications (BDA)*, the database conference of the French community, 2018
- [Buron et al., 2019] **Reformulation-Based Query Answering for RDF Graphs with RDFS Ontologies.** Maxime Buron, François Goasdoué, Ioana Manolescu, Marie-Laure Mugnier, in *Extended Semantic Web Conference (ESWC)*, 2019
- [Buron et al., 2020b] **Ontology-Based RDF Integration of Heterogeneous Data.** Maxime Buron, François Goasdoué, Ioana Manolescu, Marie-Laure Mugnier, in *Extended Database Technologies (EDBT)* conference, 2020
- [Buron et al., 2020c] **Obi-Wan: Ontology-Based RDF Integration of Heterogeneous Data** Maxime Buron, François Goasdoué, Ioana Manolescu, Marie-Laure Mugnier, to appear as a Demonstration in the *Very Large Databases (VLDB)* conference, 2020 (also accepted for presentation at BDA 2020)
- [Buron et al., 2020a] **Revisiting RDF storage layouts for efficient query answering** Maxime Buron, François Goasdoué, Ioana Manolescu, Tayeb Merabti, Marie-Laure Mugnier, Technical report, 2020

CHAPTER

2 Preliminaries

This chapter introduces the formalisms and techniques on which this thesis relies. In Section 2.1, we briefly present RDF, the Resource Description Framework, and basic graph pattern queries, the core fragment of SPARQL, the query language for the semantic web. In Section 2.2, we recall a formal framework defining data integration systems and the use of view-based query rewriting in this context.

2.1 RDF data model and SPARQL query language

We present the basics of the *RDF graph data model* (Section 2.1.1), how they can be enriched with ontological knowledge using *RDF Schema* (Section 2.1.2), how *RDF entailment* can be used to make explicit the implicit information RDF graphs encode (Section 2.1.3), and finally, how they can be queried using the widely-considered *SPARQL Basic Graph Pattern queries* (Section 2.1.4), a.k.a. SPARQL conjunctive queries.

2.1.1 RDF graphs

RDF is a W3C recommendation published first in 2004 [Res, 2004], and revised in 2014 [RDF, 2014a]. It defines abstract model of an RDF graph based on three types of values: *IRIs* (Internationalized Resource Identifiers), *literals* and *blank nodes*.

An IRI is a compact sequence of characters used to identify an abstract or a physical resource. The IRI standard [RFC, b] extends the previous Uniform Resource Identifier standard [RFC, a], which was limited to ASCII characters only. For example, any URL, like `https://starwars.com/databank/luke-skywalker` is a valid IRI representing the Star Wars character, Luke Skywalker. In the following, we will denote the set of all IRIS by $\mathscr{I}$. To simplify IRIs, RDF allows defining *namespaces* within an RDF graph. A namespace is an abbreviation of the prefix of a IRI. For example, if we define the namespace *sw* as an abbreviation of the IRI prefix `https://starwars.com/databank/`, then sw:luke-skywalker is equivalent to the full abovementioned IRI. The standard allows specifying a default namespace, for which there is no need to specify a prefix. When the default namespace is understood, a simple IRI such as :luke-skywalker can be understood to mean the full-length URI above. For readability, we will shorten it in our further examples into :Luke.

A *literal* is a string that represents a value. For example, a literal can represent a name, e.g., the string "Luke", or a number, e.g., "5". RDF enables to group values into *data types* like *Integer*, but in this thesis, for simplicity, we will only consider *plain literals* without data type; similarly, we will ignore the possible language tags attached to literals, e.g., "@fr", "@en". Hence, the literal set is the set of all strings; it will be denoted by $\mathscr{L}$.

A *blank node* represents an anonymous resource, either a literal or an IRI. We will represent blank nodes using the prefix _:. For example, _:*b* is a blank node having the name *b*. The name of a blank node is a local identifier within the graph at hand. Hence blank nodes can be assimilated to database labeled nulls [Abiteboul et al., 1995, Goasdoué et al., 2013]. The set of blank nodes will be denoted by $\mathscr{B}$.

Using the above sets of resources $\mathscr{I}, \mathscr{L}, \mathscr{B}$, we can formalize the RDF triples which compose an RDF graph:

Definition 2.1 (RDF triple)**.** *An* RDF triple *(or* triple *in short) is a triple of RDF resources* $(\mathsf{s}, \mathsf{p}, \mathsf{o})$ *belonging to the product:*

$$(\mathscr{I} \cup \mathscr{B}) \times \mathscr{I} \times (\mathscr{L} \cup \mathscr{I} \cup \mathscr{B}).$$

The resource s *is the* subject *of this triple,* p *is its* property *and* o *is its* object.

For example, the triple (:Luke, :firstName, "Luke") states that Luke's first name is Luke, while the triple (:Luke, :pilotOf, _:b_s) states that Luke is the pilot of something, represented the blank node _:b_s.

Definition 2.2 (RDF graph)**.** *An* RDF graph *is a set of RDF triples. Considering an RDF graph G, we denote by* Val(*G*) *the set of all values (IRIs, blank nodes and literals) occurring in G, and by* Bl(*G*) *its set of blank nodes.*

Example 2.1 (Sample RDF graph)**.** *Let us consider a first RDF graph* G_1 *stating that Luke uses something represented by the blank node* _:b_d *and both Luke and the thing represented by* _:b_d *are pilot of a thing represented by* _:b_s*.*

:Luke	:uses	_:b_d
:Luke	:pilotOf	_:b_s
_:b_d	:pilotOf	_:b_s

In this example, _:b_d and _:b_s may represent distinct entities (e.g., a droid and a starship, respectively) or the same entity. The notion of *homomorphism between RDF graphs* allows characterizing whether an RDF graph *simply entails*, i.e., is more specific than or subsumed by, another.

Definition 2.3 (RDF graph homomorphism)**.** *Let G and G′ be two RDF graphs. A* homomorphism *from G to G′ is a substitution* φ *of* Bl(*G*) *by* Val(*G*)*, and is the identity for the other G values (IRIs and literals), such that* $\varphi(G) \subseteq G'$*, where* $\varphi(G) = \{(\varphi(s), \varphi(p), \varphi(o)) \mid (s, p, o) \in G\}$.

From now on, we write $G' \models^{\varphi} G$ to state that G' simply entails G, as witnessed by the homomorphism φ from G to G'.

Example 2.2 (RDF graph homomorphism)**.** *Let us consider the graph* G_1 *from Example 2.1 and the novel graph* G_2 *stating that Luke uses and is a pilot a self-driven spaceship (pilot of itself):*

Schema triples	Notation
Subclass	$(\mathsf{s}, \prec_{sc}, \mathsf{o})$
Subproperty	$(\mathsf{s}, \prec_{sp}, \mathsf{o})$
Domain typing	$(\mathsf{s}, \hookleftarrow_{d}, \mathsf{o})$
Range typing	$(\mathsf{s}, \hookrightarrow_{r}, \mathsf{o})$
Data triples	Notation
Class fact	$(\mathsf{s}, \tau, \mathsf{o})$
Property fact	$(\mathsf{s}, \mathsf{p}, \mathsf{o})$ s.t. $\mathsf{p} \notin \{\tau, \prec_{sc}, \prec_{sp}, \hookleftarrow_{d}, \hookrightarrow_{r}\}$

Table 2.1: RDF triples.

:Luke :pilotOf :spaceship1
:Luke :uses :spaceship1
:spaceship1 :pilotOf :spaceship1

There exists a homomorphism φ *from* G_1 *to* G_2 *defined by the resource mapping set* $\{\_{:}b_d \mapsto$:spaceship1, $\_{:}b_s \mapsto$:spaceship1$\}$. *Thus,* G_2 *simply entails* G_1*, which we note* $G_2 \models^{\varphi} G_1$.

We remark that if a graph G is included in a graph G', then G' simply entails G.

2.1.2 RDF Schema

RDF Schema (RDFS), which is part of the RDF standard [RDF, 2014c], introduces the notion of *classes*, which are groups of resources; classes are themselves resources. This standard also defines two namespaces: rdf and rdfs. The property rdf:type is used to type a resource, i.e., to express that a resource is an instance of a class. For example, the triple (:Luke, rdf:type, :Person) states that Luke is an instance of the class Person.

RDFS also defines four properties used to state constraints or relationships between classes and properties:

- The property rdfs:subClassOf (abbreviated $\prec_{sc}$) is used to specify that a class is a subclass (specialization) of another;
- rdfs:subPropertyOf (abbreviated $\prec_{sp}$) allows to state that a property is a subproperty (specialization) of another;
- The properties rdfs:domain and rdfs:range (abbreviated $\hookleftarrow_{d}$ and $\hookrightarrow_{r}$ respectively) specify that resources appearing as the first (respectively, the second) argument of a property have a certain type.

The IRIs $\tau, \prec_{sc}, \prec_{sp}, \hookleftarrow_{d}$ and $\hookrightarrow_{r}$ are called the *built-in properties*. Table 2.1 sums up the short notations we adopt for these properties. We call *RDFS properties* the built-in properties except τ. A triple in which the property is an RDFS property is called a *schema triple* or, more precisely, an *RDFS triple*.

The *RDFS ontology* of a graph is the set of its RDFS triples:

Definition 2.4 (RDFS Ontology)**.** *The* RDFS ontology of an RDF graph G *is the set of schema triples contained in* G*. A graph is called* an RDFS ontology *if it contains only schema triples.*

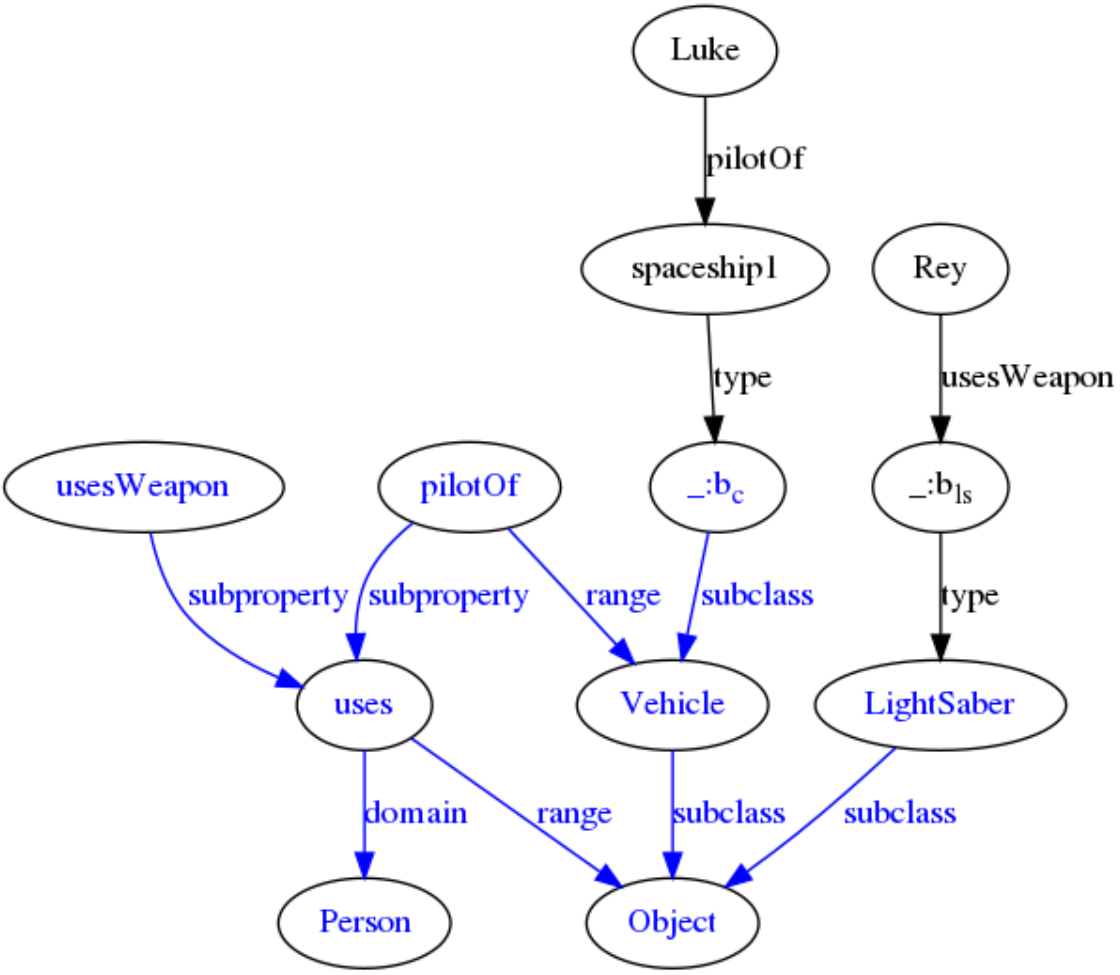


Figure 2.1: A graph G_{ex}, with its ontology O_{ex} highlighted in blue.

Example 2.3 (RDFS Ontology)**.** *The following set of triples forms an RDFS ontology, denoted by* O_{ex}*:*

:LightSaber	rdfs:subClassOf	:Object
:Vehicle	rdfs:subClassOf	:Object
$\_:b_c$	rdfs:subClassOf	:Vehicle
:uses	rdfs:domain	:Person
:uses	rdfs:range	:Object
:usesWeapon	rdfs:subPropertyOf	:uses
:pilotOf	rdfs:subPropertyOf	:uses
:pilotOf	rdfs:range	:Vehicle

O_{ex} states that: vehicles and light sabers are objects; there is an unknown subclass of vehicle represented by the blank node $\_:b_c$; if x uses y, then x is a person and y is an object; using something as a weapon and being the pilot of something are specific ways of using something; something that is piloted is a vehicle.

Figure 2.1 shows a sample RDF graph G_{ex} whose ontology is O_{ex}.

Within an RDF graph, we distinguish **data** triples from **schema** ones. The former are triples $(\mathtt{s}, \mathtt{p}, \mathtt{o})$ where $\mathtt{p}$ is *not* an RDFS property, i.e., they describe data (they either type a resource by a class, state a relationship between two resources, or state the value of a data property of a resource), while the latter express RDFS constraints.

Rule [RDF, 2014b]	Entailment rule
rdfs5	$(p_1, \prec_{sp}, p_2), (p_2, \prec_{sp}, p_3) \rightarrow (p_1, \prec_{sp}, p_3)$
rdfs11	$(c_1, \prec_{sc}, c_2), (c_2, \prec_{sc}, c_3) \rightarrow (c_1, \prec_{sc}, c_3)$
ext1	$(p, \hookleftarrow_d, c_1), (c_1, \prec_{sc}, c_2) \rightarrow (p, \hookleftarrow_d, c_2)$
ext2	$(p, \hookrightarrow_r, c_1), (c_1, \prec_{sc}, c_2) \rightarrow (p, \hookrightarrow_r, c_2)$
ext3	$(p, \prec_{sp}, p_1), (p_1, \hookleftarrow_d, o) \rightarrow (p, \hookleftarrow_d, o)$
ext4	$(p, \prec_{sp}, p_1), (p_1, \hookrightarrow_r, o) \rightarrow (p, \hookrightarrow_r, o)$
rdfs2	$(p, \hookleftarrow_d, c), (s, p, o) \rightarrow (s, \tau, c)$
rdfs3	$(p, \hookrightarrow_r, c), (s, p, o) \rightarrow (o, \tau, c)$
rdfs7	$(p_1, \prec_{sp}, p_2), (s, p_1, o) \rightarrow (s, p_2, o)$
rdfs9	$(c_1, \prec_{sc}, c_2), (s, \tau, c_1) \rightarrow (s, \tau, c_2)$

Table 2.2: The set of RDF entailment rules $\mathcal{R}_{\text{RDFS}}$.

2.1.3 RDF entailment rules

The semantics of an RDF graph is given by the **explicit triples** it contains, as well as the **implicit triples** that can be derived from it using *RDF entailment rules*.

We assume given a set of variables $\mathcal{V}$ disjoint from the RDF resources $\mathcal{I} \cup \mathcal{L} \cup \mathcal{B}$. A *triple pattern* is a triple of values belonging to:

$$(\mathcal{I} \cup \mathcal{B} \cup \mathcal{V}) \times (\mathcal{I} \cup \mathcal{V}) \times (\mathcal{L} \cup \mathcal{I} \cup \mathcal{B} \cup \mathcal{V}).$$

A *basic graph pattern* (BGP) is a set of triple patterns. It generalizes the notion of RDF graph by also allowing variables in the subject, property and object positions.

For a BGP P, we note Var(P) the set of variables occurring in P and Bl(P) its set of blank nodes.

Definition 2.5 (BGP to RDF graph homomorphism)**.** *A* homomorphism *from a BGP P to an RDF graph G is a substitution φ of* Bl(P) ∪ Var(P) *by* Val(G) *and is the identity elsewhere, such that $\varphi(P) \subseteq G$ with $\varphi(P) = \{(\varphi(s), \varphi(p), \varphi(o)) \mid (s, p, o) \in P\}$.*

We write $G \models^{\varphi} P$ to state that φ is a homomorphism from P to G.

Note that blank nodes are processed as variables in the definition of homomorphism.

Below, we define a syntax for RDF entailment rules based on basic graph patterns. Should we translate these rules into first-order logic, using a single ternary predicate to denote triples, we would obtain specific tuple-generating dependencies (TGDs) [Abiteboul et al., 1995] or existential rules, e.g., [Mugnier and Thomazo, 2014]. In particular, these rules allow one to assert the existence of unknown entities, thanks to existentially quantified variables in the head of TGDs / existential rules. However, the set of built-in RDF entailment rules that we consider next do not have this feature: these rules would be logically translated into range-restricted, or datalog, rules (in which variables that occur in a rule head also occur in the rule body and are universally quantified [Abiteboul et al., 1995]). In the next chapters, we mainly work with built-in RDF entailment rules, and extend some results to more general RDF entailment rules in Section 4.6, which justifies the following definitions that go beyond built-in RDF entailment rules.

Definition 2.6 (RDF entailment rule)**.** *An* RDF entailment rule *r is of the form body(r) → head(r), where body(r) and head(r) are basic graph patterns, containing no blank node, respectively called* body *and* head *of the rule r.*

The set of built-in RDF entailment rules is defined in [RDF, 2014b]. These rules produce implicit triples by exploiting the RDFS ontology of an RDF graph. In this thesis, we consider the rule set defined in Table 2.2, denoted by $\mathcal{R}_{\text{RDFS}}$; in the table, all values except RDFS properties denote variables. For example, for Rule `rdfs2`, we have:

- body(`rdfs2`) = $\{(p, \hookleftarrow_d, c), (s, p, o)\}$
- head(`rdfs2`) = $\{(s, \tau, c)\}$

where p, c, s and o are variables. This rule specifies that the subject of a triple belongs to the domain of the triple property.

We define how RDF entailment rules *directly entail* implicit triples from explicit ones.

Definition 2.7 (Direct entailment)**.** *The* direct entailment *of an RDF graph G with a set of RDF entailment rules $\mathcal{R}$, denoted by $C_{G,\mathcal{R}}$, characterizes the set of implicit triples resulting from triggering (a.k.a. firing) the rules in $\mathcal{R}$ using the explicit triples of G only. It is defined as:*

$$C_{G,\mathcal{R}} = \{\ \varphi(head(r))^{safe} \mid \exists r \in \mathcal{R}, G \models^{\varphi} body(r) \text{ and there is no } \varphi' \text{ extension of } \varphi \text{ s.t. } G \models^{\varphi'} head(r)\}$$

where $\varphi(head(r))^{safe}$ is obtained from $\varphi(head(r))$ by replacing each variable in Var(*head*(r))\ Var(*body*(r)) *by a* fresh *blank node. Note that the condition "there is no φ'* extension *of φ s.t. $G \models^{\varphi'} body(r) \cup head(r)$" prevents the production of an obviously redundant set of triples.*

Without loss of generality, as in the RDF standard, we only consider well-formed entailed triples, i.e., from $(\mathscr{I} \cup \mathscr{B}) \times \mathscr{I} \times (\mathscr{L} \cup \mathscr{I} \cup \mathscr{B})$.

Example 2.4 (Direct entailment)**.** *Consider G_3, the graph of Example 2.1 extended with the ontology O_{ex} of Example 2.3:*

$$G_3 = \{(\text{:Rey}, \text{:usesWeapon}, \_{:}b_s)\} \cup O_{\text{ex}}$$

as well as the rule `rdfs7`*:*

$$(p_1, \prec_{sp}, p_2), (s, p_1, o) \rightarrow (s, p_2, o)$$

The rule `rdfs7` *applies to G_3, i.e.,*

$$G_3 \models^{\varphi} body(\texttt{rdfs7})$$

through the homomorphism φ defined as $\{p_1 \mapsto$:usesWeapon, $p_2 \mapsto$:uses, $s \mapsto$:Rey, $o \mapsto \_{:}b_{ls}\}$. The rules `rdfs11`*,* `ext2`*,* `ext3` *and* `ext4` *also apply to G_3. $C_{G_3,\mathcal{R}_{\text{RDFS}}}$, the direct entailment of G_3 with $\mathcal{R}_{\text{RDFS}}$, contains exactly the following triples (the rules used to derive them appear between parenthesis):*

:Rey	:uses	$\_{:}b_{ls}$	(`rdfs7`)
:usesWeapon	$\hookleftarrow_d$	:Person	(`ext3`)
:pilotOf	$\hookleftarrow_d$	:Person	(`ext3`)
:usesWeapon	$\hookrightarrow_r$	:Object	(`ext4`)
:pilotOf	$\hookrightarrow_r$	:Object	(`ext2` *and* `ext4`)
$\_{:}b_c$	$\prec_{sc}$	:Object	(`rdfs11`)

The *saturation* of an RDF graph allows materializing the semantics of an RDF graph, by iteratively augmenting this graph with the triples it directly entails using a set $\mathcal{R}$ of RDF entailment rules, until a fixpoint is reached.

We formalize this as the sequence $(G_i^{\mathcal{R}})_{i\in\mathbb{N}}$ of RDF graphs recursively defined as follows:

- $G_0^{\mathcal{R}} = G$, and
- $G_{i+1}^{\mathcal{R}} = G_i^{\mathcal{R}} \cup C_{G_i^{\mathcal{R}},\mathcal{R}}$ for $0 \leq i$.

Definition 2.8 (Saturation of RDF graph)**.** *Let G be an RDF graph, and $\mathcal{R}$ be a set of entailment rules. The* saturation *of G w.r.t $\mathcal{R}$, denoted by $G^{\mathcal{R}}$, is defined by:*

$$G^{\mathcal{R}} = \cup_{i\in\mathbb{N}} G_i^{\mathcal{R}}.$$

Example 2.5 (Saturation of G_{ex})**.** *Consider the graph G_{ex} in Figure 2.1. Its saturation w.r.t. $\mathcal{R}_{\text{RDFS}}$ is displayed in Figure 2.2, and is obtained by the second direct entailment step, hence*

$$(G_{\text{ex}})^{\mathcal{R}_{\text{RDFS}}} = G_{\text{ex}} \cup C_{G_{\text{ex}},\mathcal{R}_{\text{RDFS}}} \cup C_{G_{\text{ex}}\cup C_{G_{\text{ex}},\mathcal{R}_{\text{RDFS}}},\mathcal{R}_{\text{RDFS}}}$$

The first direct entailment can be easily found by extending the one presented in Example 2.4. The second direct entailment produces three triples: (:Luke, τ, :Person)*,* (:Rey, τ, :Person) *and* (:spaceship1, τ, :Object).

The saturation of an RDF graph by any subset of $\mathcal{R}_{\text{RDFS}}$ is finite [RDF, 2014b].

Finally, the notion of a homomorphism between RDF graphs is also used to characterize whether an RDF graph *entails* another w.r.t. a set of RDF entailment rules, i.e., in the presence of implicit triples.

Definition 2.9 (Graph entailment)**.** *An RDF graph G* entails *an RDF graph G' w.r.t. a set $\mathcal{R}$ of RDF entailment rules, noted $G \models^{\varphi}_{\mathcal{R}} G'$, whenever there is a homomorphism φ from G' to $G^{\mathcal{R}}$. From now, we will just write $G \models_{\mathcal{R}} G'$ when a particular φ is not relevant to the discussion.*

2.1.4 BGP Queries

A popular fragment of the SPARQL query language for RDF graphs is that of basic graph pattern queries, also called the SPARQL conjunctive queries. A basic graph pattern query is defined as follows:

Definition 2.10 (BGP query)**.** *A* BGP query *(BGPQ) q is of the form $q(\bar{x}) \leftarrow P$, where P is a BGP also denoted by body(q) and $\bar{x} \subseteq$ Var(P). The variables $\bar{x}$ are called the* answer variables of *q and the* arity *of q is $|\bar{x}|$. The other variables are the called* existential variables of *q.*

Example 2.6 (BGP query)**.** *Consider the following BGP query:*

$$q(x, y) \leftarrow (x, y, z), (y, \prec_{sp}, \text{:uses}), (z, \tau, \_{:}b_c)$$

It asks for the subject and the property of each triple such that the triple property is a subproperty of :uses *and the triple object has a type. Actually, there are two ways of interpreting the blank node $\_{:}b_c$: it can be seen either as an existential variable (here,"the triple object has* a *type") or as a specific entity (here, "the triple object has type $\_{:}b_c$").*

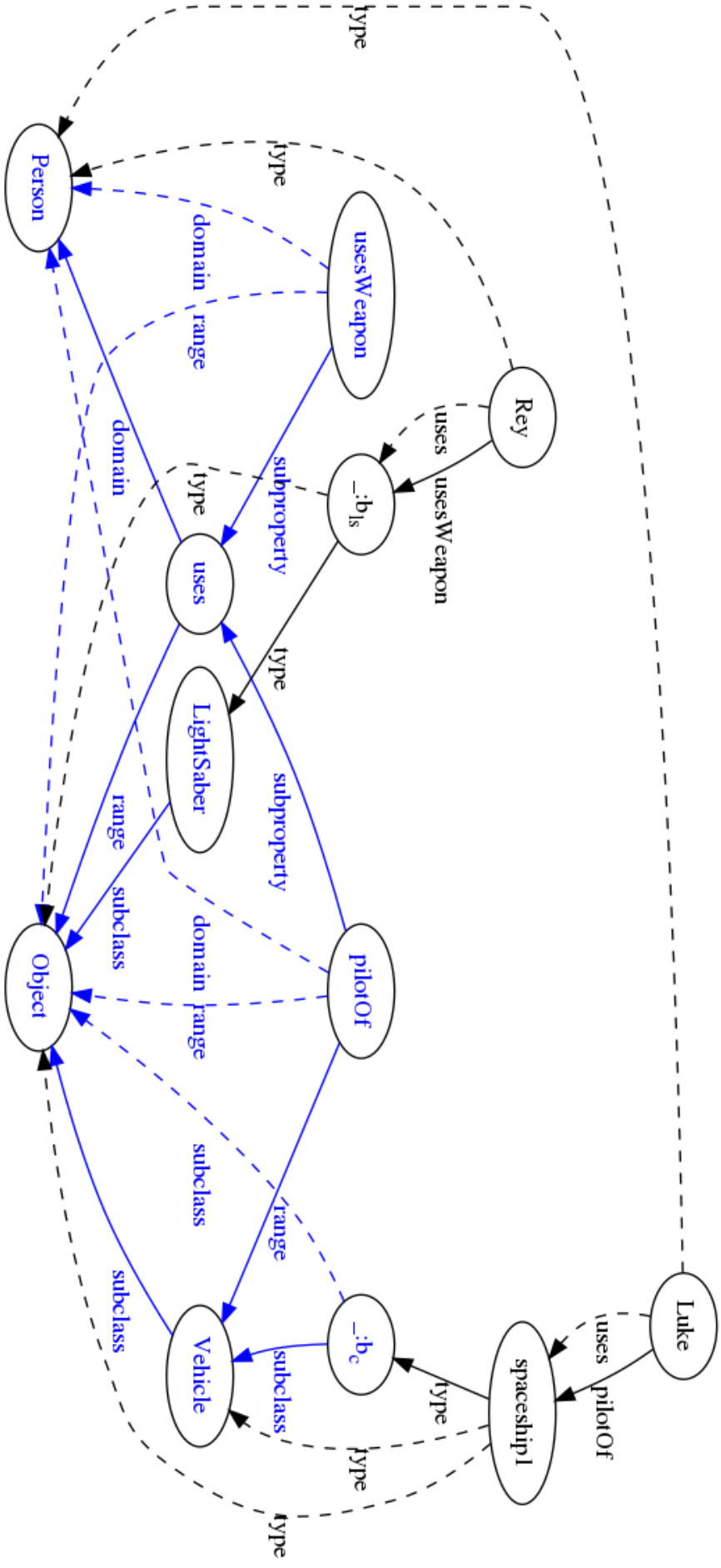


Figure 2.2: Saturation of G_{ex} w.r.t. $\mathcal{R}_{\text{RDFS}}$; ontological triples are in blue and implicit triples are dashed.

These two ways of understanding a blank node correspond to two variants of query evaluation: the standard evaluation *treats blank nodes like existential variables (as per Definition 2.5), while the* non-standard evaluation *treats them like IRIs and literals.*

Partially instantiated BGPQs

Partially instantiated BGPQs are a slight generalization of BGPQs, considered in the context of reformulation-based query answering [Goasdoué et al., 2013, Bursztyn et al., 2015]. Starting from a BGPQ q, partial instantiation replaces *some* variables with values from $\mathscr{I} \cup \mathscr{L} \cup \mathscr{B}$, as specified by a substitution σ; the partially instantiated query is denoted q_σ. Observe that when $\sigma = \emptyset$, q_σ coincides with q. Further, due to σ, and in contrast with standard BGPQs, some answer variables of q_σ can be bound:

Example 2.7 (Partially instantiated BGPQ)**.** *Consider the BGPQ asking for* who is using which kind of object*:* $q(x, y) \leftarrow (x, :\text{uses}, z), (z, \tau, y), (y, \prec_{sc}, :\text{Object})$*, and the substitution* $\sigma = \{x \mapsto :\text{Luke}\}$*. The partially instantiated BGPQ* q_σ *is:*

$$q_\sigma(:\text{Luke}, y) \leftarrow (:\text{Luke}, :\text{uses}, z), (z, \tau, y), (y, \prec_{sc}, :\text{Object})$$

BGPQ standard evaluation

The evaluation of a (partially instantiated) BGPQ is defined in terms of the homomorphisms that exist between its BGP body (Definition 2.5) and the interrogated RDF graph, i.e., in terms of the possible matches of the BGP onto the RDF graph explicit triples.

Definition 2.11 (Standard BGPQ evaluation)**.** *The* answer set to a partially instantiated BGPQ q_σ *on an RDF graph G is:*

$$q_\sigma(G) = \{\varphi(\bar{x}_\sigma) \mid G \models^\varphi body(q)_\sigma\}$$

where $\bar{x}_\sigma$ *and* $body(q)_\sigma$ *denote the result of replacing the variables in* $\bar{x}$ *and* $body(q)$*, respectively, according to* σ*.*

If $\bar{x} = \emptyset$*, q is a* Boolean *query and the answer to q is false when* $q(G) = \emptyset$ *and true when* $q(G) = \{\langle\rangle\}$*.*

We remark that the evaluation of BGPQs allows blank nodes from the RDF graph to appear in the answers. Also, with the standard evaluation, blank nodes occurring in BGPQs have the same behaviour as existential variables.

Example 2.8 (Standard BGPQ evaluation)**.** *Consider the query q from Example 2.6, and the graph* G_{ex} *in Figure 2.1. The evaluation of q on the graph* G_{ex} *yields the answer set:*

$$q(G_{ex}) = \{\langle :\text{Luke}, :\text{pilotOf}\rangle, \langle :\text{Rey}, :\text{usesWeapon}\rangle\}$$

This evaluation is derived from φ_1 *and* φ_2*, two homomorphisms from* $body(q)$ *to* G_{ex}*, defined as follows:*

- $\varphi_1 = \{x \mapsto :\text{Luke}, y \mapsto :\text{pilotOf}, z \mapsto :\text{spaceship1}\}$
- $\varphi_2 = \{x \mapsto :\text{Rey}, y \mapsto :\text{usesWeapon}, z \mapsto \_{:}b_{ls}, \_{:}b_c \mapsto :\text{LightSaber}\}$

Note that in the above standard BGPQ evaluation example, φ_2 maps the blank node $\_:b_c$ to another value. We may want to avoid this behaviour of standard query evaluation, and consider the blank node $\_:b_c$ in the query q as a value from the graph G_{ex}, i.e., a "constant" and not an existential variable; there is actually no way to write a simple query asking for the entities of type $\_:b_c$ in G_{ex}. Considering blank nodes in queries as constants is enabled by *non-standard* query evaluation.

Non-standard query evaluation

Non-standard query evaluation relies on non-standard BGP to RDF graph homomorphisms:

Definition 2.12 (Non-standard BGP to RDF graph homomorphism)**.** *A* non-standard homomorphism *from a BGP P to an RDF graph G is a substitution φ of* Var(P) *by* Val(*G*) *and is the identity elsewhere, such that for any triple $(s, p, o) \in P$, the triple $(\varphi(s), \varphi(p), \varphi(o))$ is in G.*

Based on this, we define:

Definition 2.13 (Non-standard evaluation)**.** *Let q_σ be a partially instantiated BGPQ. The* non-standard evaluation *to q_σ on an RDF graph G is:*

$$\widetilde{q_\sigma(G)} = \{\varphi(\bar{x}_\sigma) \mid G \models^{\varphi} body(q)_\sigma\}$$

where φ is a **non-standard homomorphism** *from body(q) to G.*

We remark that for a BGPQ q that contains no blank node, $\widetilde{q(G)}$ is equal to $q(G)$. We now consider the following example, where the query contains a blank node:

Example 2.9 (Non-standard evaluation)**.** *Consider again the BGPQ from the Example 2.6. Its non-standard evaluation on G_{ex} contains only one tuple:* ⟨:Luke, :pilotOf⟩.

In this thesis, we adopt non-standard BGPQ evaluation by default, hence the blank node in queries should always be considered as constants and not as existential variables. It is always possible to transform a query q used with standard evaluation into a query q' used with non-standard evaluation in such a way that their evaluation coincide, i.e., $\widetilde{q'(G)}$ is equal to $q(G)$ for any G. The transformation simply replaces each blank node in q with fresh existential variable. For simplicity, we will not use the notation $\widetilde{\ }$ anymore in the sequel of the text, i.e., $q(G)$ will denote the non-standard evaluation of q on G. We will also use the expression "query evaluation" instead of "non-standard evaluation", except when we want to emphasize the evaluation behaviour w.r.t. blank nodes in the query.

Minimisation of a union of BGPQs

We also consider unions of BGPQs, denoted by UBGPQs, in which all the BGPQs have the same arity. We extend this notion to unions of partially instantiated BGPQs, in which the BGPQs are associated with a (possibly empty) substitution. The evaluation of a UBGPQ on a graph G is the union of the evaluations of its BGPQs on G.

A union of BGPQs may contain some redundant BGPQs, in the sense that some BGPQ may be contained in another:

Definition 2.14 (BGPQ Containment)**.** *Given two BGPQs q_1 and q_2, we say that q_1 is* contained *in q_2, when for each RDF graph G, $q_1(G) \subseteq q_2(G)$.*

So, in a union of BGPQs, removing a BGPQ contained in another BGPQ of the union does not change the evaluation of the union on any RDF graph. We say that a UBGPQ is *minimal*, when no such removal is possible:

Definition 2.15 (Minimal UBGPQs)**.** *A union of BGP queries is* minimal*, when none of its BGPQs is contained in another.*

The notion of BGP to RDF (non-standard) homomorphism is naturally extended to that of a BGP to BGP (non-standard) homomorphism, by simply allowing variables in its range. Then a homomorphism h from a BGPQ $q_1(\bar{x}) \leftarrow P_1$ to a BGPQ $q_2(\bar{y}) \leftarrow P_2$ with $|\bar{x}| = |\bar{y}|$ is a homomorphism h from P_1 to P_2 such that $h(\bar{x}) = \bar{y}$. This notion is in turn extended to partially instantiated BGPQs q'_1 and q'_2, respectively obtained from q_1 and q_2 by substitutions σ_1 and σ_2: a homomorphism h from q'_1 to q'_2 is a homomorphism from $\sigma_1(P_1)$ to $\sigma_2(P_2)$ such that $h(\sigma_1(\bar{x})) = \sigma_2(\bar{y})$. It is well-known that q_1 is contained in q_2 if and only if there is a homomorphism h from q_2 and q_1, and this result extends to partially instantiated BGPQs.

Hence, an algorithm to produce minimal UBGPQs can be built on an algorithm checking the existence of a BGPQ to BGPQ homomorphism as follows: while there are q_1 and q_2 two distinct queries in the union such that q_1 is contained in q_2, remove q_1 from the union.

2.1.5 Query answering

The answers of a BGPQ on a graph w.r.t. a set of RDF entailment rule is defined as the evaluation (Definition 2.11) of the query on the saturation of the graph by the RDF entailment rules at hand:

Definition 2.16 (BGPQ answers)**.** *The* set of answers of a BGPQ *q on an RDF graph G w.r.t. a RDF entailment rule set $\mathcal{R}$, is denoted $q(G, \mathcal{R})$ and is defined by:*

$$q(G, \mathcal{R}) = q(G^{\mathcal{R}})$$

With general RDF entailment rules, the problem of query answering is not decidable [Baget et al., 2011]. In contrast, if one uses the rule set $\mathcal{R}_{\text{RDFS}}$ shown in Table 2.2, the query answering problem is decidable, since the saturation of a graph is always finite.

There are two main techniques for solving the query answering problem; one based on graph saturation and another based on query reformulation. Of course, these techniques only work under certain conditions on the query, the rules and the graph.

The **saturation-based query answering** technique is based on Definition 2.16. It assumes that the RDF graph is stored in a database system, capable of evaluating BGPQs. The query answering process is then is divided into two steps:

1. **Graph saturation.** The saturation $G^{\mathcal{R}}$ of the RDF graph G w.r.t. the RDF entailment rules $\mathcal{R}$ is computed and stored in the database system.

2. **Query evaluation.** Every incoming query q is evaluated on the saturation of the graph by the database system, i.e., the system computes $q(G^{\mathcal{R}})$.

The **reformulation-based query answering** technique is also composed of two steps:

1. **Query reformulation.** Every incoming query q is reformulated into a reformulated query Q, typically a union of BGPQs, using the ontology O of G and the rules $\mathcal{R}$. Hence, the reformulation step is performed without accessing data triples of G,

2. **Reformulation evaluation** The evaluation of Q on the graph G is performed by a database system. It is guaranteed to return the answers of q on G w.r.t. $\mathcal{R}$.

A reformulation Q of a query q w.r.t. an ontology O and a set of RDF entailment rules $\mathcal{R}$ is such that for any RDF graph G with ontology O, the reformulation-based technique for query answering is *sound*, i.e., $Q(G) \subseteq q(G^{\mathcal{R}})$, and *complete*, i.e., $q(G^{\mathcal{R}}) \subseteq Q(G)$.

2.2 Data integration

The aim of data integration [Doan et al., 2012, Lenzerini, 2002, Abiteboul et al., 2011] is to provide a uniform interface to a multitude of data sources. This interface is a *global schema* (or *mediated schema*), i.e., a set of relations, on which a user poses queries. The relations of the global schema are connected by semantic *mappings* with the data sources. In particular, we will consider the case when the mappings are specified through *view definitions* [Halevy, 2000, Halevy, 2001].

There are two main approaches for answering queries on the global schema. The first one, called the *warehousing* approach, materializes the global schema data using data provided by the sources. Hence it reduces the problem of query answering to standard database query evaluation on the materialization. In contrast, in the *mediation-based approach*, data remains in the sources; the query on the global schema is rewritten into (sub-)queries on the sources, whose results may be processed and assembled within a *mediator query engine*. In this thesis, we mostly investigate the mediator approach.

Below, we first recall in Section 2.2.1 the theory of data integration, while Section 2.2.2 presents the Global As View integration framework and Section 2.2.3, the Local As View framework. Section 2.2.4 introduces the natural generalization of these frameworks: the Global Local As View integration.

2.2.1 Theory of data integration

This presentation of data integration systems mainly follows the definitions presented in [Lenzerini, 2002]. We begin with the definition of a data integration system:

Definition 2.17 (Data Integration System)**.** *A* data integration system $\mathcal{I}$ *is triple* $\langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ *where:*

- $\mathcal{G}$ *is the* global schema
- $\mathcal{S}$ *is the* source schema
- $\mathcal{M}$ *is the* set of mappings *between* $\mathcal{S}$ *and* $\mathcal{G}$*, such that each mapping is a formula of the form:*

$$\forall \bar{x}\ q_{\mathcal{S}}(\bar{x}) \rightsquigarrow q_{\mathcal{G}}(\bar{x})$$

where $q_{\mathcal{S}}$ *(resp.* $q_{\mathcal{G}}$*) is a query belonging to* $\mathcal{L}_{\mathcal{M},\mathcal{S}}$*, the* source query language for mappings *(resp.* $\mathcal{L}_{\mathcal{M},\mathcal{G}}$*, the* global query language for mappings*) and is called the* body (resp. head) of the mapping. *The logical symbol* $\rightsquigarrow$ *may have different interpretations as we discuss shortly below.*

We recall some common notions of *mappings interpretations*:

Definition 2.18 (Mapping interpretations)**.** *Let m be a mapping $q_{\mathcal{S}}(\bar{x}) \rightsquigarrow q_{\mathcal{G}}(\bar{x})$ between $\mathcal{S}$ a source schema and $\mathcal{G}$ a global schema, and D (resp. B) an instance of $\mathcal{S}$ (resp. $\mathcal{G}$). We say that D and B* satisfy the mapping m, depending on its interpretation*:*

- *when m is* sound*, $\rightsquigarrow$ is interpreted as $\rightarrow$ (logical implication), i.e.,*

$$q_{\mathcal{S}}(D) \subseteq q_{\mathcal{G}}(B);$$

- *when m is* complete*, $\rightsquigarrow$ is interpreted as $\leftarrow$, i.e.,*

$$q_{\mathcal{S}}(D) \supseteq q_{\mathcal{G}}(B);$$

- *when m is* exact*, m is both sound and complete i.e., $\rightsquigarrow$ is interpreted as $\leftrightarrow$, or,*

$$q_{\mathcal{S}}(D) = q_{\mathcal{G}}(B).$$

In the data integration literature, assuming that the mappings are sound (but possibly not exact) is often denoted as the *Open World Assumption* (OWA); it is natural when a large, possibly dynamic set of sources are integrated. In contrast, considering that all mappings are exact is known as the *Closed Word Assumption* (CWA). This setting is natural, for instance, when we query a database of relations together with a set of views materialized from the same relations.

Definition 2.19 (Integration System Instance)**.** *Given D an instance of the source schema of the data integration $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$, we say that B is a* global instance of $\mathcal{I}$ w.r.t. D*, if it holds that:*

- *B is an instance of the global schema $\mathcal{G}$,*
- *D and B satisfy each mapping of $\mathcal{M}$.*

Given an instance of the source schema, we are interested in the *certain answers* i.e., those that are sure to be part of the query result for all global instances.

Definition 2.20 (Certain Answers)**.** *Let q be a query on a data integration system $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ and D an instance of $\mathcal{S}$. We define the* certain answers *of q on $\mathcal{I}$ w.r.t. D as the set of tuples which are answers of q on every global instance of $\mathcal{I}$ w.r.t. D. We denote these certain answers by $q(\mathcal{I}, D)$.*

As mentioned above, the (certain) answers to a query can be computed either by materializing the global instance or by rewriting the query posed on the global schema into a query posed on the source schema, in which case we rely on:

Definition 2.21 (Query Containment and Equivalence)**.** *Let $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ be a data integration system and q_1, q_2 be two queries, each being either in terms of $\mathcal{S}$ or $\mathcal{G}$. We say that q_1* is contained in *q_2 w.r.t. $\mathcal{I}$, when for all D instance of $\mathcal{S}$, it holds that:*

$$q_1(D, \mathcal{I}) \subseteq q_2(D, \mathcal{I})$$

where for $i \in \{1, 2\}$, $q_i(D, \mathcal{I})$ denotes the answers of q_i on D, if q_i is in terms of $\mathcal{S}$, or the certain answers of q_i on $\mathcal{I}$ w.r.t. D, otherwise.

The queries q_1 and q_2 are equivalent *when q_1 is contained in q_2 and q_2 is contained in q_1.*

A rewriting of a query q on the global schema, is a query q_r on the source schema, which is contained in q. Ideally, the rewriting q_r should be equivalent to q; however, depending on the integration system, such a query may not exist. Thus, we are interested in the rewritings that are maximal in a language of query rewritings:

Definition 2.22 (Maximally contained rewriting)**.** *Let $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ be a data integration system and $\mathcal{L}_r$ be a query language on $\mathcal{S}$, called the* rewriting language*. Given a query q posed in terms of $\mathcal{G}$, a query q_r in $\mathcal{L}_r$ is a* maximally contained rewriting *of q in $\mathcal{I}$ w.r.t. $\mathcal{L}_r$, when:*

- *q_r is contained in q w.r.t. $\mathcal{I}$,*
- *for all q'_r in $\mathcal{L}_r$ such that q'_r is contained in q, it holds that q'_r is contained in q_r.*

In the next two sections, we study the two cases of data integration systems whose mappings are seen as *views*, namely:

- *Global As View* data integration systems, where the global schema is relational and the global query language for mappings $\mathcal{L}_{\mathcal{M},\mathcal{G}}$ is restricted to atomic queries on the global schema (Section 2.2.2),
- *Local As View* data integration systems, where the source schema is relational and the source query language for mappings $\mathcal{L}_{\mathcal{M},\mathcal{S}}$ is restricted to atomic queries on the source schema (Section 2.2.3).

In both cases, **we make the Open World Assumption, in other words we assume that the mappings are sound.**

2.2.2 Global As View data integration

In Global As View integration systems, the global schema $\mathcal{G}$ is relational and consists of a set of atoms $V_1, V_2, \ldots, V_n$ termed *views*. The mapping set $\mathcal{M}$ contains view definitions having the following form:

$$q_{\mathcal{S}}(\bar{x}_i) \rightsquigarrow V_i(\bar{x}_i)$$

with $1 \leq i \leq n$. We slightly change the mapping notation from Definition 2.17 to use a view name instead of the a query name in the mapping head, without any consequence on the semantics. Notice that several mappings can have the same view name in their head.

Definition 2.23 (Conjunctive unfolding)**.** *Let q be a conjunctive query on the views $V_1, V_2, \ldots, V_n$, such that*

$$q(\bar{x}) \leftarrow \bigwedge_{1 \leq j \leq m} V^j(\bar{u}_j)$$

where for all $1 \leq j \leq m$, V^j is a view among $V_1, \ldots, V_n$ and $\bar{u}_j$ is a tuple of constants and variables, with the same arity as V^j. An unfolding of q *is a query defined by:*

$$q_r(\bar{x}) \leftarrow \bigwedge_{1 \leq j \leq m} \sigma_j(body(q^j_{\mathcal{S}}))$$

where σ_j is the substitution defined by $\bar{x}_j \mapsto \bar{u}_j$ (the other variables being replaced with fresh variables), such that $\mathcal{M}$ contains the following mappings, for $1 \leq j \leq m$:

$$q^j_{\mathcal{S}}(\bar{x}_j) \rightsquigarrow V^j(\bar{x}_j).$$

In the above, $\mathcal{L}_{\mathcal{M},\mathcal{S}}$, the source query language of mappings, is left unconstrained on purpose; this allows, for instance, to have queries in $q_{\mathcal{S}}$ that are not defined in a relational language. For the purpose of this section, we will rather consider that $\mathcal{S}$ is a relational schema, and the queries $q_{\mathcal{S}}$ are FOL queries. Thus, a conjunctive unfolding is also a FOL query.

In GAV integration systems under OWA, finding the certain answers of a conjunctive query is easily solved using query unfolding:

Property 2.1 ([Abiteboul et al., 2011])**.** *Under the Open World Assumption i.e., mappings in $\mathcal{M}$ are sound, let q be a conjunctive query on the views $V_1, V_2, \ldots, V_n$. Then q_r, the union of the unfoldings of q, is a maximally contained rewriting of q in the integration system w.r.t. FOL queries. Further, q_r computes the certain answers i.e., for all D instance of the views, $q_r(D) = q(D, \mathcal{I})$.*

Equivalently, the canonical extensions of the views can be materialized to get the certain answers on GAV integration systems under OWA.

Definition 2.24 (Canonical Extensions)**.** *Given an instance D of the source schema, we define $\mathcal{E}_D$ the* canonical extensions of the views of $\mathcal{I}$ *w.r.t. D by:*

$$\mathcal{E}_D = \{V_i(\bar{a}) \mid \bar{a} \in q_{\mathcal{S}}(D), 1 \leq i \leq n \text{ and } q_{\mathcal{S}}(\bar{x}) \rightsquigarrow V_i(\bar{x}) \in \mathcal{M}\}$$

Property 2.2 ([Abiteboul et al., 2011])**.** *Under OWA, given an instance D of the source schema and a conjunctive query q over the views, the following holds:*

$$q(D, \mathcal{I}) = q(\mathcal{E}_D).$$

2.2.3 Local As View data integration

In Local As View integration systems, the source schema $\mathcal{S}$ is relational and consists of a set of views $V_1, V_2, \ldots, V_n$. Further, each *LAV mapping* from $\mathcal{M}$ is of the form:

$$V_i(\bar{x}_i) \rightsquigarrow q_{\mathcal{G}}(\bar{x}_i).$$

Instances of the source schema are commonly called *extensions of the views*. Below, we will only consider sound LAV mappings (OWA). In the literature, such mappings have also been called view definitions. Below, we will use the notation $V_i(\bar{x}_i) \rightarrow q_{\mathcal{G}}(\bar{x}_i)$, to emphasize that $\rightsquigarrow$ corresponds to logical implication.

Example 2.10 (LAV integration system)**.** *We describe $\mathcal{I}_{ex}$, a sample LAV integration system. Its source schema $\mathcal{S}$ contains three relations* Emp(eID, name, dID), Dept(dID, cID, country), Salary(eID,amount)*, where* eID*,* dID *and* cID *are respectively identifiers for employees, departments and companies. $\mathcal{I}_{ex}$ has two sound mappings (each $q_{\mathcal{G}}$ is replaced by its body, to simplify the notation):*

$$V_1(eID, name, country) \rightarrow Emp(eID, name, dID), Dept(dID, \text{“}IBM\text{”}, country)$$

V_1 provides the names of IBM employees and where they work, and

$$V_2(eID, amount) \rightarrow Emp(eID, name, \text{“}R\&D\text{”}), Salary(eID, amount),$$

V_2 contains the salaries of employees in R&D departments. As a consequence of the Open World Assumption, no single view is expected to bring all *the information specified by its associated query. For instance, V_1 describes* some *IBM employees, but there may be other such employees absent from V_1. Similarly, V_2 may or may not contain information about employees present in V_1 .*

The following example shows that it is not always possible to find an equivalent rewriting of query in a LAV integration system.

Example 2.11. *Consider the query:*

$$q(n, a) \leftarrow Emp(e, n, d), Dept(d, c, \text{“}France''), Salary(e, a)$$

on the integration system $\mathcal{I}_{ex}$ above. The query does not have an equivalent rewriting using V_1 and V_2, because V_1 only provides IBM employees working in France, while V_2 only has salaries of employees of R&D departments. A maximally contained rewriting (Definition 2.22) brings all the query answers that can be obtained through the given set of views; the rewriting may be not be equivalent to q (but just contained in q). In our example, the query:

$$q_r(n, a) \leftarrow V_1(e, n, \text{"}France\text{"}), V_2(e, a)$$

is a maximally contained rewriting of q in $\mathcal{I}_{ex}$ w.r.t. conjunctive queries on the views; it returns employees of French IBM R&D departments with their salary, clearly a subset of q's answers.

A remarkable result [Abiteboul and Duschka, 1998] holds for *(unions of) conjunctive queries ((U)CQs)*:

Theorem 2.1 (Conjunctive LAV Rewriting)**.** *Let $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ be a LAV integration system such that $\mathcal{G}$ is a relational schema, $\mathcal{L}_{\mathcal{M},\mathcal{G}}$ is the set of conjunctive queries on $\mathcal{G}$, and $\mathcal{L}_r$ is the set of unions of conjunctive queries on the views in $\mathcal{S}$. Let q be a union of conjunctive queries on the views. If q_r is a maximally contained rewriting of q in $\mathcal{I}$ w.r.t. $\mathcal{L}_r$, then for all D instances of $\mathcal{S}$, $q_r(D)$ is equal to $q(D, \mathcal{I})$.*

Computing a maximally contained rewriting in a LAV integration system is more complicated than in a GAV integration system. A notable algorithm is Minicon, proposed in [Pottinger and Halevy, 2001], which can be used to compute such rewritings of conjunctive queries. Its running time is $O(|q|k|\mathcal{M}|)^{|q|}$, where $|q|$ is the number of atoms in the conjunctive query, k is the maximal number of atoms in mapping heads, and $|\mathcal{M}|$ is the number of mappings.

2.2.4 Global Local As View data integration

We have so far presented the assumptions which enable computing the certain answers of a query through rewriting in GAV and LAV integration systems. We now present the natural generalization of these integration frameworks: *global-local-as-view (GLAV) integration systems* [Friedman et al., 1999]. A GLAV integration system $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ is such that $\mathcal{G}$ is a relational schema and $\mathcal{M}$ contains only *GLAV mappings* of the form $q_{\mathcal{S}}(\bar{x}) \rightarrow q_{\mathcal{G}}(\bar{x})$, where $q_{\mathcal{G}}$ is a conjunctive query.

Given a GLAV integration system $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$, we can define a GAV integration system $\mathcal{I}_{GAV} = \langle \mathcal{V}, \mathcal{S}, \mathcal{M}_{GAV} \rangle$, called *the GAV integration from $\mathcal{I}$* and a LAV integration system $\mathcal{I}_{LAV} = \langle \mathcal{G}, \mathcal{V}, \mathcal{M}_{LAV} \rangle$, called *the LAV integration from $\mathcal{I}$*, such that for each mapping $m = q_{\mathcal{S}}(\bar{x}) \rightarrow q_{\mathcal{G}}(\bar{x})$ in $\mathcal{M}$, there exists a unique view name V_m of arity $|\bar{x}|$ in the schema $\mathcal{V}$, and:

- $\mathcal{M}_{GAV}$ contains the mappings $q_{\mathcal{S}}(\bar{x}) \rightarrow V_m(\bar{x})$;
- $\mathcal{M}_{LAV}$ contains the mappings $V_m(\bar{x}) \rightarrow q_{\mathcal{G}}(\bar{x})$.

Intuitively, the integration system $\mathcal{I}_{GAV}$ defines how to populate the extensions of the views in $\mathcal{I}_{LAV}$. Together, they imitate the GLAV integration system $\mathcal{I}$. Using the Property 2.2, we obtain:

Property 2.3. *Let D be a instance of the source schema $\mathcal{S}$ and q be a conjunctive query on the global schema $\mathcal{G}$. It holds that:*

$$q(D, \mathcal{I}) = q(\mathcal{E}_D, \mathcal{I}_{LAV})$$

where $\mathcal{E}_D$ is the canonical extensions of views of $\mathcal{I}_{GAV}$ w.r.t. D.

Example 2.12. *Assume a source schema $\mathcal{S}$ holds the relations* Person(eID, name) *and* Contract(eID, dID, country) *with information about people and about work contracts at IBM, and a global schema $\mathcal{G}$ holds the relations* Emp(eID, name, dID) *and* Dept(dID, cID, country) *with employees and departments. We define m, a GLAV mappings from $\mathcal{S}$ to $\mathcal{G}$, by:*

$$m = Person(e, n), Contract(e, d, c) \rightarrow Emp(e, n, d'), Dept(d', "IBM", c),$$

where d' is an existential variable of the head of m representing a department; in other words, this mapping hides *the department provided by the source from the global schema. Using this mapping, we define the GLAV integration system $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$, with $\mathcal{M} = \{m\}$.*

Using a view name $V_m(eID, name, country)$ and the following mappings:

$$m_{GAV} = Person(eID, name), Contract(eID, dID, country) \rightarrow V_m(eID, name, country),$$

$$m_{LAV} = V_m(eID, name, country) \rightarrow Emp(eID, name, dept'), Dept(dept', "IBM", country),$$

we obtain the GAV system from $\mathcal{I}$:

$$\mathcal{I}_{GAV} = \langle \{V_m\}, \mathcal{S}, \{m_{GAV}\} \rangle$$

and the LAV system from $\mathcal{I}$:

$$\mathcal{I}_{LAV} = \langle \mathcal{G}, \{V_m\}, \{m\} \rangle$$

(See Figure 2.3).

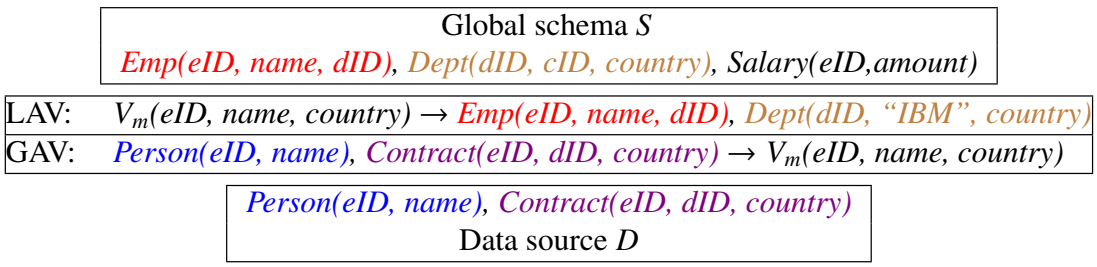


Figure 2.3: Example of GLAV mapping.

Importantly, unlike GAV mappings, GLAV ones do not require all variables of $q_{\mathcal{G}}$ to be answer variables; this is not the case of $dept'$ in the head of m above. This makes integration more powerful. For example, suppose that ⟨1,"John Doe","France"⟩ *is contained in V_m according to $\mathcal{I}_{GAV}$. Then in $\mathcal{I}_{LAV}$, V_m exposes this tuple in the global schema $\mathcal{G}$ as: Emp(1,"John Doe",x), $Dept$(x,"IBM","France"), stating that John Doe works for a*

department x located in France. Here, x is an existential variable (called "labeled null" in [Abiteboul et al., 1995]); the GLAV mapping states the existence *of such a department in the global schema, even if its identifier is unknown (because it is not provided by V_m). Therefore, John Doe is a certain answer to a query asking for all employees in IBM departments, based on the above GLAV mapping. This answer cannot be found using GAV mappings.*

Query rewriting techniques from LAV and GAV integration systems are composed to obtain one for GLAV systems. To obtain answers in a GLAV integration system $\mathcal{I}$, a rewriting in the LAV integration performed within $\mathcal{I}$ needs to be *unfolded* in the GAV integration from $\mathcal{I}$ (Definition 2.23), replacing every occurence of a view symbol V_m with the body of the mapping query defining the view in $\mathcal{I}_{GAV}$. Evaluating the resulting query (potentially over different data sources) computes the answers.

Property 2.4. *Let q be a conjunctive query on $\mathcal{G}$ and q_r^{LAV} a maximally contained rewriting of q in $\mathcal{I}_{LAV}$ w.r.t. the set of unions of conjunctive queries on the view names in $\{V_m \mid m \in \mathcal{M}\}$. Let q_r be the union of the unfoldings in $\mathcal{I}_{GAV}$ of each conjunctive query in the union of q_r^{LAV}. Then:*

- *q_r is a maximally contained rewriting in $\mathcal{I}$ w.r.t. the set of UCQs on $\mathcal{S}$;*
- *q_r computes the certain answers of q in $\mathcal{I}$ i.e., for all D instance of the source schema $\mathcal{S}$:*

$$q(D, \mathcal{I}) = q_r(D)$$

Proof. We first prove the second point (q_r computes the certain answers of q in $\mathcal{I}$). For all D instance of $\mathcal{S}$, we have:

$$\begin{aligned} q(D, \mathcal{I}) &= q(\mathcal{E}_D, \mathcal{I}_{LAV}) \quad \text{(Property 2.3)} \\ &= q_r^{LAV}(\mathcal{E}_D) \quad \text{(Theorem 2.1)} \\ &= q_r(D) \quad \text{(Property 2.2)} \end{aligned}$$

The first point is a direct consequence of the second. □

2.3 Summary

In the course of this chapter, we have first introduced the definitions of RDF graph, RDFS ontology, with $\mathcal{R}_{\text{RDFS}}$ the RDFS entailment rule set, BGP query evaluation and answering. These definitions will be intensively used in the next chapters, in which the main topic is the problem of BGP query answering on RDF graphs w.r.t. $\mathcal{R}_{\text{RDFS}}$. Secondly, we have presented some results about query answering in data integration systems. They will be used in Chapter 4 for data integration using an RDF global schema with RDFS constraints.

CHAPTER

3 RDF query answering

This chapter is devoted to query answering techniques on RDF graphs with general BGP queries. In Section 3.1, we situate our work with respect to the state of the art. Then, we present two main contributions. First, a sound and complete query reformulation algorithm, which exploits a partition of RDFS entailment rules into assertion and constraint rules (Section 3.2). Second, a novel RDF storage layout, namely TCP, which combines the two well-known T and CP layouts (Section 3.3). For both contributions, we detail the experiments carried out to assess our theoretical and algorithmic results. The material presented here comes from [Buron et al., 2019, Buron et al., 2020a], whereas Section 3.2.1 and Section 3.2.8 are new.

3.1 Motivation and state of the art

In this section, we first compare RDFS to description logics, a prominent family of ontological languages, and emphasize the metamodeling capabilites of RDFS, which are generally not provided by description logics. Then we present a short state of the art on BGPQ answering based on query reformulation and on storage layouts for RDF graphs.

3.1.1 RDF representations

Most representation and reasoning formalisms used to describe knowledge bases are based on first-order logic (FOL). However, FOL entailment is undecidable in general, hence many formalisms correspond to restricted fragments of FOL, for which entailment is decidable. Introduced in the 80's, Description Logics (DLs) [Baader et al., 2007] are a family of languages specially designed to represent and reason with ontological knowledge. They mostly correspond to decidable fragments of FOL and each specific DL achieves a specific tradeoff between expressivity and tractability of reasoning. Given a vocabulary composed of *concept (or class) names* and *role (or property) names*, a set of constructors associated with a specific DL allows one to define complex concepts and roles. Then, a DL knowledge base (KB) is composed of a *TBox* (the schema-level, or terminological / ontological knowledge), which is a set of axioms, in the form of inclusions between possibly complex concepts (respectively roles), and an *ABox* (the assertion-level, or factual knowledge), which is a set of facts asserting that an individual is an instance

of a concept or connecting two individuals by some role. Even if DLs have their own syntax, a DL knowledge base (KB) can be seen as a FOL theory (i.e., a set of FOL sentences). Concept and roles names are translated into unary and binary predicates, respectively, and individuals are translated into constants. Then a TBox is composed of formulas that can be seen as "entailment rules", more precisely formulas of the form $\forall \mathbf{x}\ (body[\mathbf{x}] \rightarrow head[\mathbf{x}])$, where $\mathbf{x}$ is a free variable (respectively a pair of free variables) in *body* and *head* for a concept inclusion (respectively for a role inclusion), and *body* and *head* are the FOL translation of concepts (respectively roles) involved in the inclusion.

Whereas there is a strict separation between concept names, role names and individuals in DLs, RDF does not enforce such constraint: an IRI may act as a class, a property and an individual in the same graph (such a property is called *punning* in OWL 2, the second version of the Web Ontology Language). Hence, in RDF, there is no strict boundary between the schema and the assertion levels of a knowledge base: schema triples and data triples are *first of all triples*. Moreover, differently from DLs, the RDF/RDFS specification allows to represent unknown values (blank nodes) in both data and schema triples. From a FOL perspective, an RDF graph is a set of atoms built on a single ternary predicate expressing the notion of triple, whose arguments are constants (IRIs or literals) or existentially quantified variables (blank nodes). Note that we will use a similar translation to transform BGPs into CQs in Section 4.3.

The factual assertions contained in an ABox are naturally translated into RDF data triples, as shown in the following table:

Assertion	**DL**	**RDF**
Concept/Class	$C(a)$	$(:a, \tau, :C)$
Role/Property	$R(s, o)$	$(:s, :R, :o)$

In this table, we first remark that individuals in DLs are translated into IRIs (for simplicity, we ignore here the distinction between literals and IRIs). Concept names and role names are also translated into pairwise distinct IRIs, which are also distinct from the IRIs used for the individuals. Because of the strict separation between concepts, roles and individuals in DLs (and, less importantly, the possible presence of blank nodes in RDF triples), the inverse translation (from RDF data triples to ABoxes) is not always possible.

Intuitively, in a knowledge base, *metamodeling* consists of expressing assertions on elements of its "schema", which are then seen as individuals that can be described. Hence, metamodeling in DLs would require that the concepts and roles, which are elements of the schema of a DL knowledge base, are treated as individuals. As already mentioned, concept and role names are strictly separated from individuals, therefore DL knowledge bases do not allow for metamodeling. Even if a TBox expresses constraints on roles and concepts, these constraints are not assertions, in the sense that they cannot be queried. On the contrary, RDF graphs have inherent metamodeling capabilities. First, we have seen that RDFS constraints already use a form of metamodeling. For example, the triples $(:Luke, \tau, :Jedi)$ and $(:Jedi, \prec_{sc}, :Person)$ require metamodeling capabilities, since :Jedi appears as a class in the first triple (so :Jedi is part of the schema) and as subject in the second.

We distinguish between two levels of *metamodeling* in an RDF graph (inspired by [Giacomo et al., 2011]):

- *Domain metamodeling* is the ability for user-defined IRIs (as opposed to RDF(S) built-in IRIs) to act, in the same graph, (i) as a property or a class in some triples, and (ii) as the subject or the object in other triples. This ability of domain metamodeling is crucial to express RDFS triples as exemplified above, to create *metaclasses*, i.e., classes of classes, e.g., (:Jedi, τ, :SWCharacterCategory) and to assert *properties of classes or properties*, e.g., (:Jedi, :createdBy, "*George Lucas*").

- *Full metamodeling* is the above ability for all IRIs, i.e., extended to RDF(S) built-in IRIs.

We would like to emphasize that the metamodeling capabilities of RDF graphs have a genuine interest if the knowledge they allow to express can be queried. This justifies the study of **general BGP queries**, which allow to take full advantage of them.

The full metamodeling capabilites are exploited by the RDFS standard itself, as the next example shows:

Example 3.1 (Metamodeling in RDFS)**.** *The standard RDF semantics [RDF, 2014b] includes the following triples:*

rdf:type	rdf:type	rdf:Property
rdf:type	rdfs:range	rdfs:Class
rdfs:subClassOf	rdfs:range	rdfs:Class
rdfs:subClassOf	rdfs:domain	rdfs:Class

where the built-in IRIs rdfs:Class *and* rdf:Property *represent respectively the class of all the classes and the class of all the properties. The first triple defines the property* rdf:type *as an instance of the class* rdfs:Property. *The three other triples define some schema constraints for the built-in properties* rdf:type *and* rdfs:subClassOf.

DLs and RDF have two different ways of expressing ontological knowledge and reasonning on it. In DLs, ontological knowledge is expressed in the TBox as a set of axioms, and each axiom can be translated into a specific "entailment rule" in FOL. In RDFS, the ontology is made of assertions in the form of schema triples, while the set of entailment rules ($\mathcal{R}_{\text{RDFS}}$, Figure 2.2) remains fixed.

We recall that RDFS ontologies allow to express four kinds of constraints (subclass, subproperty, domain and range). Such knowledge is at the core of most of DLs, for instance the lightweight DL-Lite$_{\mathcal{R}}$ [Calvanese et al., 2007], also called DL-Lite$^{\mathcal{H}}_{core}$ in [Artale et al., 2009]. The four RDFS constraints can be translated into the following DL entailment rules, provided that the sets of names used for classes (C) and properties (R) are disjoint:

Constraints	**RDFS constraint**	**DL entailment rules**
Subclass	$(:C_1, \prec_{sc}, :C_2)$	$C_1(x) \rightarrow C_2(x)$
Subproperty	$(:R_1, \prec_{sp}, :R_2)$	$R_1(x, y) \rightarrow R_2(x, y)$
Domain	$(:R, \hookleftarrow_d, :C)$	$R(x, y) \rightarrow C(x)$
Range	$(:R, \hookrightarrow_r, :C)$	$R(x, y) \rightarrow C(y)$

Of course, most DLs allow to express richer constraints, by defining complex concepts and/or roles.

In this chapter, we study **RDFS reasoning on RDF graphs with full metamodeling capabilities**. Although the expressivity of RDFS is limited to simple constraints, we will see that efficiently supporting RDFS reasoning with full metamodeling presents some difficulties.

Finally, let us mention that the extension of DLs to support domain metamodeling was studied, notably in [Giacomo et al., 2011], where it is shown that domain metamodeling can be added to expressive DLs, while keeping decidability of conjunctive query answering, and even without complexity increase, like for $\mathcal{SHIQ}$, the core OWL 2. More generally, the OWL language provides an RDF syntax for expressing DLs [OWL, a, OWL, b] and reuses the vocabulary of RDFS. The richest fragment, namely OWL-Full, supports full metamodeling, but entailment is undecidable [Motik, 2005].

3.1.2 Query answering techniques

As outlined in the preceding chapter (Section 2.1.5), two main techniques for answering BGPQs on RDF graphs have been investigated in the literature, based on the *saturation* of graphs by the rules, or on the *reformulation* of the query by the rules. Both of them reduce query answering to query evaluation.

Saturation-based query answering

The saturation-based query answering technique directly follows from the definition of query answers (Section 2.1.4 for BGPQs). Indeed, it trivially follows from Definition 2.16 that $q(G, \mathcal{R}) = q(G^{\mathcal{R}})$,i.e., query answering reduces to query evaluation on the *saturated* RDF graph. Saturation-based query answering is typically fast, because it only requires query evaluation, which can be efficiently performed by a data management engine. However, saturation takes time to be computed, requires extra space to be stored, and must be recomputed or maintained (e.g., [Broekstra and Kampman, 2003, Bishop et al., 2011, Goasdoué et al., 2013]) upon updates.

Most RDF data management systems rely on saturation-based query answering. They either allow computing graph saturation, e.g., Jena[1] and RDFox [Nenov et al., 2015], or simply assume that RDF graphs have been saturated before being stored, e.g., DB2RDF[2].

Reformulation-based query answering

The reformulation-based query answering technique also reduces query answering to query evaluation, however, the reasoning needed to ensure complete answer sets is performed on the query instead of the RDF graph. A given query q, asked on an RDF graph G w.r.t. a set $\mathcal{R}$ of entailment rules, is *reformulated* into a query q' such that $q(G, \mathcal{R}) = q'(G)$ holds.

In the literature, reformulation-based query answering techniques have been proposed for diverse fragments of the SPARQL, a W3C's recommendation [SPA, 2013]. The BGP queries (Definition 2.10) form the core of SPARQL. In this thesis, we will mainly work with *BGPQs*, partially instantiated BGPQs and unions of them, without any restriction. The ability of BGPQs to query jointly schema and data triples is a key feature of SPARQL. Some works consider query reformulation for sub-languages of BGPQs, which weaken or even suppress this ability: by forbidding triple patterns with an RDFS property, we obtain the *BGPQ-data* fragment; more restrictive, the *BGPQ-CQ* fragment also forbids triple patterns in which a variable appears in a property or class position i.e., respectively, triples of the form $(\_, y, \_)$ and $(\_, \tau, z)$, where y and z are variables.

[1] `https://jena.apache.org/documentation/inference/`

[2] `https://www.ibm.com/support/knowledgecenter/SSEPGG_11.1.0/com.ibm.swg.im.dbclient.rdf.doc/doc/c0059661.html`

Note that BGPQ-data still allows to query some domain metamodeling, e.g., one can query for "all classes containing, at least, an instance together with their creators" using $q(y, z) \leftarrow (x, \tau, y), (y, :\text{createdBy}, z)$.

Meta modeling	Queries			
	BGPQ-CQ	**BGPQ-data**	**BGPQ**	**SPARQL**
DL	[Adjiman et al., 2007, Calvanese et al., 2007] [Rodriguez-Muro et al., 2013] [Urbani et al., 2014] [Lanti et al., 2017]	[Bischof et al., 2014]		[Kontchakov et al., 2014]
DMeta/data		[Goasdoué et al., 2011] [Goasdoué et al., 2013] [Bursztyn et al., 2015]		[Arenas et al., 2009]
DMeta			[Kaoudi et al., 2008] [Urbani et al., 2011] [Pinto et al., 2011]	
FMeta			Section 3.2	

Table 3.1: Reformulation-based query answering related work.

In Table 3.1, we classify work about reformulation-based query answering according to the supported metamodeling abilities of graphs (rows) and the considered SPARQL fragment (columns), and illustrate this classification with notable papers. In rows, we consider increasing metamodeling capabilities: DL means graphs without metamodeling capabilities, DMeta and FMeta stand for domain and full metamodeling, respectively, and DMeta/data corresponds to domain modeling with the constraint that queries must match data triples of the graph. In columns, we consider increasingly richer query languages, namely BGPQ-CQ, BGPQ-data, BGPQ, SPARQL.

Let us start with the DL row. The papers [Adjiman et al., 2007, Calvanese et al., 2007, Rodriguez-Muro et al., 2013, Lanti et al., 2017] consider the BGPQ-CQ query fragment, where query reformulation can be computed using algorithms akin to classical backward chaining, and results in a union of BGPQs (UBGPQs in short). Queries in the BGPQ-data fragment for OWL 2 QL knowledge bases can be reformulated as schema-agnostic reformulations in a large fragment of SPARQL, which notably allows for property paths [Bischof et al., 2014]. In [Kontchakov et al., 2014], SPARQL queries on both the ABox and the TBox of DL knowledge bases represented in OWL can be reformulated into SPARQL queries (which use data triple patterns only). Note however that the SPARQL queries considered in this article are built using three pairwise disjoint sets of variables, which must only match, respectively, the concepts, the roles and the instances of a DL knowledge base. Hence, such queries do not allow to fully query a knowledge base supporting metamodeling. Finally, [Urbani et al., 2014] studies a *hybrid* approach for query answering with BGPQ-CQs, where some one-triple queries are chosen for materialization and reused during reformulation-based query answering.

Concerning the second row (DMeta/data), graphs containing domain metamodeling from which data triples are queried using BGPQ-data have been studied in [Goasdoué et al., 2011, Goasdoué et al., 2013]. In these works, the reformulation is a union of *partially instantiated* BGPQs, since the variables in class and property position may be instantiated during the reformulation process. In this chapter, we use the same target language for reformulation, however with general BGPQs. Reformulation-based query answering is well-suited to frequently updated RDF graphs, because it uses the queried RDF graph at query time (and not its saturation). However, reformulated queries

tend to be more complex than the original ones, thus costly to evaluate. To mitigate this, [Bursztyn et al., 2015] provides an *optimized reformulation framework* in which an incoming BGPQ is reformulated into a *join of unions of BGPQs (JUBGPQ in short)*. This approach being based on a database-style *cost model*, JUBGPQ reformulations can be very efficiently evaluated. In [Arenas et al., 2009], SPARQL queries are reformulated into *nested* SPARQL queries, which allow for nested regular expressions in the property position of query triples. These reformulations provide sound and complete query answering on restricted RDF graphs with RDFS ontologies: these graphs *must not contain blank nodes*. While such nested reformulations are more compact, the queries we produce are more usable in practice, since their evaluation can be delegated to any off-the-shelf RDBMS, or to an RDF engine even if this engine is unaware of reasoning; moreover, we do not impose restrictions on RDF graphs.

Concerning the third row (DMeta), graphs with domain metamodeling queried by one-triple BGPQs have been considered for RDFS reasoning [Kaoudi et al., 2008], and more powerful reasoning [Urbani et al., 2011]. A reformulation algorithm is proposed in [Pinto et al., 2011] on DL-Lite$_R$ knowledge base extensions for domain metamodeling based on [Giacomo et al., 2011].

In Section 3.2, we introduce a **reformulation algorithm for general BGPQs on RDF graphs with full metamodeling**, up to natural conditions corresponding to the notion of an RDFS ontology.

In practice, some commercial RDF data management systems use reformulation-based query answering but the systems we have tested return incomplete answer sets in the RDF setting we consider[3], e.g., AllegroGraph[4] and Stardog [5] miss answers because they cannot evaluate triples with a variable property on the schema, while Virtuoso[6] only exploits subclass and subproperty constraints, but not domain and range ones. Our experiments concerning these systems are reported in the Appendix (Section A.1.2).

3.1.3 RDF storage layouts

In Section 3.3, we consider the problem of efficiently querying an **RDF database**, which stores RDF graphs in a persistent way (e.g., on a disk) and allows multiple users to query these graphs, including updating them. As previously, we focus on answering **general BGPQs** on a graph, which allow for variables in any subject, property, or object position of triple patterns.

Answering a query on an RDF database requires *translating* it into a description of work that the execution engine must perform; without loss of generality, we call this work description a *query plan*, as is common in the database literature. Specifically, we distinguish a *logical plan* specifying the operations to use to answer the query, from a *physical (executable) plan*, derived from the logical one with the help of statistics and cost parameters characterizing the data (size, value frequencies etc.) and the execution environment (hardware etc.)

Both plans start by accessing some data from the store and continue with various other processing steps (e.g., filtering, combining multiple inputs etc.). The set of persistent data structures that hold the data of an RDF graph in the database are called *stor-*

[3] See discussion at
`https://team.inria.fr/cedar/rdfs-reasoning-experiments/`.

[4] `https://franz.com/agraph/support/documentation/current/reasoner-tutorial.html`

[5] `https://www.stardog.com/docs/#_owl_rule_reasoning`

[6] `http://docs.openlinksw.com/virtuoso/rdfsparqlruleimpl`

age layout. When a set of frequent BGPQs are known in advance, they can be used to design a workload-aware layout, which optimizes data access for these queries, e.g., [Bornea et al., 2013, Goasdoué et al., 2011, Pham et al., 2015]. Lacking a known query set, a workload-unaware layout is used, with the two most popular being the following:

- **T (triple)** that stores all triples together as a single (s, p, o) collection of *subject, predicate, object* tuples, e.g., [Broekstra et al., 2002, Wilkinson et al., 2003, Neumann and Weikum, 2010];
- **CP (class and property)** that separates the data for each property and class, i.e., as (*i*) a collection of (s, o) pairs for every property p, and (*ii*) a collection of all the resources s that have a given type c in the graph, e.g., [Abadi et al., 2007].

Indexes can be added to both the T and the CP layouts.

In our work, we focus on **translating BGPQs** into **logical plans** on workload-unaware layouts. We target logical plans for generality, since physical plans strongly depend on the RDF database implementation, the presence of indexes etc.; these decisions are best left to the optimization and execution layer, and we do not study them here. However, *the choice of the logical plan can massively impact the space of alternatives (physical plans)* considered for the query, and thus the query answering performance. In particular, we translate our plans in SQL (if the RDF database has a relational back-end) or SPARQL (if a native RDF engine evaluates them), which enables to retain all benefits of system-specific optimization.

In Section 3.3, we introduce a novel workload-unaware layout, namely the **TCP layout**, which combines the data structures of both T and CP layouts. We argue that this new layout is particularly interesting for answering general BGPQs, as well as unions of BGPQs resulting from query reformulation. To assess the behaviour of the new TCP layout, we carry out experiments and compare the reformulation-based and saturation-based query answering performances on the three layouts (i.e., T, CP and TCP).

Prior works such as [Abadi et al., 2007, Bornea et al., 2013], [Neumann and Weikum, 2010, Pham et al., 2015, Sidirourgos et al., 2008] only considered saturation-based query answering. While [Abadi et al., 2007] advocated the CP layout, [Sidirourgos et al., 2008] nuanced the analysis: in a row store, they show that proper indexing (such as we used here) can significantly improve performance using T, while graphs containing many distinct properties may hurt CP performance. The optimized T layout of [Weiss et al., 2008], indexed on all (s, p, o) permutations, has been used for reformulation-based query answering in [Goasdoué et al., 2011, Goasdoué et al., 2013, Bursztyn et al., 2015]. Storage was optimized based on a known workload by creating materialized views in [Goasdoué et al., 2011].

Query reformulation has also been used with the CP layout in [Buron et al., 2019, Bursztyn et al., 2016]. Both [Bursztyn et al., 2015] for T and [Bursztyn et al., 2016] for CP explored how to express a reformulated query as a join of several subqueries, so as to minimize the evaluation cost through the RDBMS. A simplified version of TCP is briefly mentioned in [Chebotko et al., 2009], which studies generic SPARQL-to-SQL translation functions, as an example of possible relational layout. However, [Chebotko et al., 2009] does not consider the performance impact of this layout; nor do they consider RDFS entailment.

Optimized storage layouts [Bornea et al., 2013, Pham et al., 2015], [Wilkinson et al., 2003] or indexes [Udrea et al., 2007, Atre et al., 2010] have been investigated to limit joins by storing e.g., the values of several properties for a given sub-

Rule	Entailment rule
rdfs2	$(p, \hookleftarrow_d, c), (s, p, o) \rightarrow (s, \tau, c)$
rdfs3	$(p, \hookrightarrow_r, c), (s, p, o) \rightarrow (o, \tau, c)$
rdfs7	$(p_1, \prec_{sp}, p_2), (s, p_1, o) \rightarrow (s, p_2, o)$
rdfs9	$(c_1, \prec_{sc}, c_2), (s, \tau, c_1) \rightarrow (s, \tau, c_2)$

(a) Assertion rules $\mathcal{R}_a$.

Rule	Entailment rule
rdfs5	$(p_1, \prec_{sp}, p_2), (p_2, \prec_{sp}, p_3) \rightarrow (p_1, \prec_{sp}, p_3)$
rdfs11	$(c_1, \prec_{sc}, c_2), (c_2, \prec_{sc}, c_3) \rightarrow (c_1, \prec_{sc}, c_3)$
ext1	$(p, \hookleftarrow_d, c_1), (c_1, \prec_{sc}, c_2) \rightarrow (p, \hookleftarrow_d, c_2)$
ext2	$(p, \hookrightarrow_r, c_1), (c_1, \prec_{sc}, c_2) \rightarrow (p, \hookrightarrow_r, c_2)$
ext3	$(p, \prec_{sp}, p_1), (p_1, \hookleftarrow_d, o) \rightarrow (p, \hookleftarrow_d, o)$
ext4	$(p, \prec_{sp}, p_1), (p_1, \hookrightarrow_r, o) \rightarrow (p, \hookrightarrow_r, o)$

(b) Constraint rules $\mathcal{R}_c$.

Figure 3.1: The partition of the RDFS entailment rule set $\mathcal{R}_{\text{RDFS}}$

.

ject together. They allow translating several BGPQ triples into a single table (or index) access.

3.2 Complete RDFS query reformulation

In this section, we first study in more detail the properties of RDFS entailment rules, according to a partition of $\mathcal{R}_{\text{RDFS}}$ into two subsets: the set $\mathcal{R}_a$ of assertion rules and the set $\mathcal{R}_c$ of constraint rules. These subsets are defined in Figure 3.1. Then, we introduce a new query reformulation algorithm, which is sound and complete for the set of rules $\mathcal{R}_c$. This algorithm can be used in conjunction with an existing query reformulation algorithm for $\mathcal{R}_a$ [Goasdoué et al., 2013]. Together, they form a sound and complete reformulation algorithm for the whole $\mathcal{R}_{\text{RDFS}}$ rule set.

3.2.1 Preliminaries: RDFS ontology and $\mathcal{R}_{\text{RDFS}}$ rule set properties

To study the behaviour of $\mathcal{R}_{\text{RDFS}}$ applied to an RDF graph, let us consider its partition into $\mathcal{R}_a$, the subset of *assertion rules*, and $\mathcal{R}_c$, the subset of *constraint rules* (see Figure 3.1). At first glance, one may have the intuition that assertion rules entail only data triples, while constraint rules entail only schema triples. However, we will see that this intuition is false about assertion rules (and holds true about constraint rules). Hence, we will introduce some restrictions on RDF graphs under which this intuition is verified. This will yield some interesting properties of reasoning, which we will exloit in our reformulation technique.

First, we observe that *constraint rules* from $\mathcal{R}_c$ can only be applied on schema triples and entail only schema triples.

Property 3.1 (Constraint rule properties)**.** *Let G be an RDF graph, with O its RDFS ontology, and r be a rule in $\mathcal{R}_c$, such that $G \models^{\varphi} body(r)$. Then it holds that:*

1. $\varphi(body(r)) \subseteq O$

2. *$\varphi(head(r))$ is a set of schema triples.*

Second, we observe that the body of each *assertion rule* from $\mathcal{R}_a$ is composed of two triple patterns and at least one of them has an RDFS property. The rule head contains one triple pattern with the property τ, except for the rule `rdfs7`, which has a variable in property position. Without restrictions on the RDF graphs, a rule from $\mathcal{R}_a$ can be applied using either schema and data triples together, or schema triples only, and it may entail either a data triple or a schema triple, as illustrated by the next example:

Example 3.2 (Possible reasoning with $\mathcal{R}_a$ on unrestricted RDF graphs)**.** *We illustrate the different cases of assertion rule applications, according the kind of triples used in input and entailed.*

1. *Schema and data triples used together to entail data triples have been exposed in the Example 2.4.*

2. *Schema and data triples used together to entail schema triples. Let the graph* $G = \{(:\text{PeopleSubGroupOf}, \prec_{sp}, \prec_{sc}), (:\text{Jedi}, :\text{PeopleSubGroupOf}, :\text{Person})\}$*; we can apply the rule* `rdfs7` *on G to entail the schema triple:* $(:\text{Jedi}, \prec_{sc}, :\text{Person})$.

3. *Schema triples only to entail data triples. Consider the graph* $G = \{(\hookleftarrow_d, \hookrightarrow_r, \text{rdfs:Class}), (:\text{pilotOf}, \hookleftarrow_d, :\text{Person})\}$*, we can apply the rule* `rdfs3` *on G to entail the data triple:* $(:\text{Person}, \tau, \text{rdfs:Class})$.

4. *Schema triples only to entail schema triples. Consider the graph* $G = \{(\prec_{sp}, \prec_{sp}, \prec_{sc}), (:\text{uses}, \prec_{sp}, :\text{pilotOf})\}$*, we can apply the rule* `rdfs7` *on G to entail the data triple:* $(:\text{uses}, \prec_{sc}, :\text{pilotOf})$.

In the following, we introduce some restrictions on RDF graphs in order to avoid schema triples to be entailed by rules in $\mathcal{R}_a$ (cases 2 and 4 of Example 3.2). We first define such graphs:

Definition 3.1 ($\mathcal{R}_a$-compliant graph)**.** *An RDF graph G is* $\mathcal{R}_a$-compliant*, if, for all* $r \in \mathcal{R}_a$ *and homomorphism* φ *from* $body(r)$ *to* $G^{\mathcal{R}_{\text{RDFS}}}$*,* $\varphi(head(r))$ *is a data triple.*

Hence, when a graph G is $\mathcal{R}_a$-compliant, G and $G^{\mathcal{R}_a}$ have the same ontology.

Among the rules in $\mathcal{R}_a$, only `rdfs7` can entail a schema triple, which motivates the following syntactic restriction on property $\prec_{sp}$:

Definition 3.2 (Restriction on $\prec_{sp}$)**.** *An RDF graph G fulfills the* restriction on $\prec_{sp}$*, if no triple in G with property* $\prec_{sp}$ *has an object which is an RDFS property (*$\prec_{sc}$*,* $\prec_{sp}$*,* $\hookleftarrow_d$ *or* $\hookrightarrow_r$*).*

It is easy to check that when a graph fulfills the restriction on $\prec_{sp}$, it also is $\mathcal{R}_a$-compliant. However, applications of rules in $\mathcal{R}_a$ may still use only schema triples, even if the graph satisfies the property of being $\mathcal{R}_a$-compliant or the stronger restriction on $\prec_{sp}$, see e.g., case 3 of Example 3.2. We define the more demanding notion of *graph with split reasoning* to exclude this possibility:

Definition 3.3 (Graph with split reasoning)**.** *An RDF graph G with an ontology O is a* graph with split reasoning *if*

- *G is* $\mathcal{R}_a$*-compliant, and*

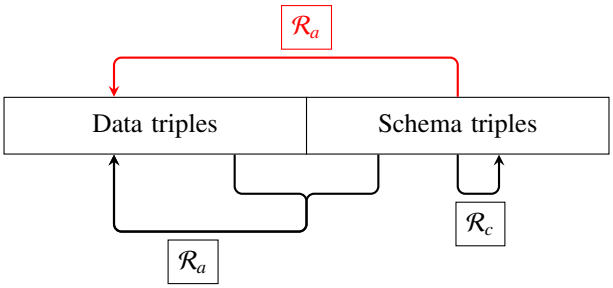


Figure 3.2: Kinds of triples involved and produced by RDFS rules on $\mathcal{R}_a$-compliant graphs. The red arrow is avoided on graphs with split reasoning.

- *for all $r \in \mathcal{R}_a$ and homomorphism φ such that $G^{\mathcal{R}_{\text{RDFS}}} \models^{\varphi} body(r)$, $\varphi(body(r))$ contains at least a data triple.*

We call these graphs "with split reasoning", because intuitively schema reasoning is separated from data reasoning, i.e., constraint rules (resp. assertion rules) are applied on and entails only schema (resp. data) triples. Moreover, assertion rules use data and schema triples, but there are able to entail all data triples with the schema triples from G's ontology, without requiring schema triples entailed by constraint rules (see next Theorem 3.1). Graphs with split reasoning do not have the complete capabilities of full metamodeling enabled by the RDF standards (see Example 3.1). The case 3 of Example 3.2 shows a graph that is not with split reasoning and which contains the RDF standards triple $(\hookleftarrow_d, \hookrightarrow_r, \text{rdfs:Class})$. Forbidding the capabilities of full metamodeling of RDFS ontologies is a syntactical way to obtain graphs with split reasoning, it is the purpose of *first-order restriction*:

Definition 3.4 (First-order restriction)**.** *We say that an RDF graph G satisfies the* first-order (FO) restriction*, if in its RDFS ontology, built-in properties (i.e., $\prec_{sc}, \prec_{sp}, \hookleftarrow_d, \hookrightarrow_r, \tau$) do not occur in subjects or objects of triples. Such ontology is called an* FO ontology.

The FO restriction on graphs are quite natural, since it ensures that ontologies do not define new constraints on the built-in properties.

Property 3.2. *When an RDF graph satisfies the first-order restriction, it is a graph with split reasoning.*

We also remark that graphs satisfying the first-order restriction are still capable of expressing domain metamodeling. Note that the restriction on $\prec_{sp}$ is incomparable with the first-order restriction and is not sufficient to ensure split reasoning. This is illustrated by case 3 of Example 3.2, where G fulfills the restriction on $\prec_{sp}$ but does not ensure split reasoning.

Finally, we sum up the previous results in the Figure 3.2, where the arrows are labeled by rule sets. Each arrow starts from the kind of triples (data and/or schema) on which the rules from the corresponding rule set can be applied and points to the kind of the triples entailed by these rules. All arrows are possible on $\mathcal{R}_a$-compliant graphs (and for graphs under restriction on $\prec_{sp}$), but the red arrow is impossible for graphs with split reasoning (and under first-order restriction).

The following theorem makes explicit a property implicitely used in the technical report [Goasdoué et al., 2012] (associated with the paper [Goasdoué et al., 2013]), where

RDF graphs have FO-ontologies, and states it for the larger class of RDF graphs with split reasoning.

Theorem 3.1. *Let G be an RDF graph with split reasoning, then $G^{\mathcal{R}_{\text{RDFS}}}$ and $G^{\mathcal{R}_a}$ have the same set of data triples.*

Note that the previous theorem requires the split reasoning property. It does not hold in general for $\mathcal{R}_a$-compliant graphs, as illustrated by the next example:

Example 3.3. *Let G be the following graph, which is $\mathcal{R}_a$-compliant (because it fulfills the restriction on $\prec_{sp}$) :*
$G = \{(\prec_{sc}, \prec_{sp}, \text{:includedIn}), (\text{:StarShip}, \prec_{sc}, \text{:Vehicle}), (\text{:Vehicle}, \prec_{sc}, \text{:Object})\}$,
then $G^{\mathcal{R}_{\text{RDFS}}}$, the saturation of G w.r.t. $\mathcal{R}_{\text{RDFS}}$, contains these additional triples:

:StarShip	:includedIn	:Vehicle
:Vehicle	:includedIn	:Object
:StarShip	$\prec_{sc}$	:Object
:StarShip	:includedIn	:Object

So the data triple (:StarShip, :includedIn, :Object) *belongs to $G^{\mathcal{R}_{\text{RDFS}}}$, but it does not belong to $G^{\mathcal{R}_a}$, since $\mathcal{R}_a$ does not define the transitivity of the property $\prec_{sc}$. It is a counter-example to the applicability of Theorem 3.1 to $\mathcal{R}_a$-compliant graphs.*

Using the previous results, we show that the saturation process w.r.t. $\mathcal{R}_{\text{RDFS}}$ on $\mathcal{R}_a$-compliant graphs can be divided into a first step of saturation w.r.t. $\mathcal{R}_c$, and a second step of saturation w.r.t. $\mathcal{R}_a$. **If the graph is more specifically with split reasoning, then the saturation steps can also be reversed**.

Theorem 3.2 (Partitioning reasoning)**.** *Let G be an $\mathcal{R}_a$-compliant graph, then the following holds:*

$$G^{\mathcal{R}_c \cup \mathcal{R}_a} = (G^{\mathcal{R}_c})^{\mathcal{R}_a} \tag{3.1}$$

Moreover, if G is with split reasoning, then the following holds:

$$G^{\mathcal{R}_c \cup \mathcal{R}_a} = (G^{\mathcal{R}_a})^{\mathcal{R}_c} \tag{3.2}$$

In the next sections of this chapter, **we will assume that all RDF graphs are with split reasoning**. This will allow us to exploit the equality 3.2: since the saturation of such graph can be performed using first $\mathcal{R}_a$ then $\mathcal{R}_c$, query reformulation can follow the opposite order, using first $\mathcal{R}_c$ then $\mathcal{R}_a$. In Section 3.2.8 we will relax this assumption by considering query answering on $\mathcal{R}_a$-compliant graphs.

3.2.2 Overview of the query reformulation technique

Let us first notice that the body of any BGPQ q can be divided into three disjoint subsets of triples (s, p, o), according to the nature of property p:

- the set b_c of RDFS triples where p is a built-in RDFS property ($\prec_{sc}$, $\prec_{sp}$, $\hookleftarrow_d$, $\hookrightarrow_r$);
- the set b_d of data triples where p is τ or a user-defined property;
- and the set b_v where p is a variable.

We denote by q_c, q_d and q_v the subqueries respectively associated with these subsets. If b_v is not empty, q can be reformulated as a union of BGPQs, say Q, composed of all BGPQs that can be obtained from q by substituting some (possibly none) variables occurring in a property position in q_v with one of the four built-in RDFS properties. Intuitively, each BGPQ in Q corresponds to a choice on how triple patterns with a variable in property position will be mapped to a graph: those substituted here will be mapped to schema triples, while the others will later be mapped to data triples. We assume this preprocessing step to simplify the explanations, even if in practice it may not be performed. Then, the answers to any BGPQ $q' \in Q$ can be computed in two steps:

1. compute the answers to the subquery q'_c, i.e., with body restricted to the RDFS triples; if q'_c has no answer, neither has q'. Otherwise, each answer to q'_c defines a (partial) instantiation σ of the variables in q'.

2. compute the data-level answers to each partially instantiated query $(q'_{d,v})_\sigma$, where $q'_{d,v}$ is the subquery with body $b'_d \cup b'_v$, and return the union of all the obtained answers.

To summarize, Step 1 computes answers to RDFS triples, which allows one to produce a set of partially instantiated queries that no longer contain RDFS triples. Hence, these queries can then be answered using RDF data only, which is the purpose of Step 2. Our two-step query reformulation follows this decomposition.

It furthermore considers the partition of the set $\mathcal{R}_{\text{RDFS}}$ into $\mathcal{R}_c$ and $\mathcal{R}_a$ studied in the preceding section (Section 3.2.1). Indeed, following Theorem 3.1, the reason of this decomposition is that query answering remains complete if, on the one hand, only $\mathcal{R}_c$ is considered to answer queries made of RDFS triples (Step 1: for any graph G, $q'_c(G, \mathcal{R}) = q'_c(G, \mathcal{R}_c)$), and, on the other hand, only $\mathcal{R}_a$ is considered to answer queries on data triples only (as already observed in [Goasdoué et al., 2013]).

Our query reformulation technique does not directly work with the entailment rules as backward-chaining (and related techniques in ontology-mediated query answering) would do. With ontologies expressed in the DL-Lite family, a CQ q is usually reformulated into a union of queries Q, using the axioms of the ontology (the TBox), but independently from any data [Poggi et al., 2008, Calvanese et al., 2007]. In the more general framework of existential rules, query reformulation can be sketchily described as follows, using a refomulation operator [König et al., 2013]: at each step of the reformulation process, the reformulation operator unifies part of a query q with (part of) a rule head, and produces a new query from q by replacing the unified part of q with the body of the rule (after application of the unifier). Expressed in our framework, the property to be satisfied by reformulation is the following, for any query q and RDF graph G (including the ontology):

$$q(G, \mathcal{R}) = Q(G)$$

However, the above techniques cannot be applied with RDF entailment rules because for instance, rules `rdfs5` and `rdfs11` (Figure 3.1), which express the transitivity of the subclass and subproperty relations, would lead to an infinite process, since no assumption can be made on the ontology, as it is part of the RDF graph. Note that the two previously mentioned rules are not the only ones that would lead to an infinite reformulation process, as illustrated by the next example:

Example 3.4. *Consider the constraint rule* `ext1`*:*

$$(p, \hookleftarrow_d, c_1), (c_1, \prec_{sc}, c_2) \rightarrow (p, \hookleftarrow_d, c_2)$$

and the query

$$q(x, y) \leftarrow (x, \hookrightarrow_d, y)$$

Applying a classical reformulation operator to q using `ext1` *leads to the new query:*

$$q_1(x, y) \leftarrow (x, \hookrightarrow_d, y_1), (y_1, \prec_{sc}, y)$$

The same process can be applied on the query, since its body contains a triple with the property $\hookrightarrow_d$*. So, we can endlessly apply this process on the new queries, leading to the following sequence of queries:*

$$n \geq 1,\ q_n(x, y) \leftarrow (x, \hookrightarrow_d, y_n), (y_n, \prec_{sc}, y_{n-1}), \ldots, (y_2, \prec_{sc}, y_1), (y_1, \prec_{sc}, y)$$

We notice that no query of this sequence is contained into another (in the classic database sense), so there is no reason to stop the reformulation process. Similarly, an infinite reformulation process could be obtained from the assertion rule `rdfs9` *and the query* $q(x, y) \leftarrow (x, \tau, y)$*.*

Finally, note that classical reformulation techniques can be used when queries are restricted to the BGPQ-data or BGPQ-CQ fragments, which implies that only $\mathcal{R}_a$ reasoning is needed: in this case, RDFS entailment rules can be replaced by logical rules built from ontology triples (see the discussion in Section 3.1.1), which ensure the termination of reformulation.

Instead of using directly $\mathcal{R}_{\text{RDFS}}$, a set of so-called *reformulation rules* is specifically associated with $\mathcal{R}_c$, and similarly for $\mathcal{R}_a$ (these sets are introduced in Section 3.2.3 and Section 3.2.5 respectively), using the ontology O of the queried graphs. We can now outline the two-step query reformulation algorithm.

Step 1. Reformulation w.r.t. $\mathcal{R}_c$: Given a fixed ontology O, the input BGPQ q is first reformulated into a union Q_c of partially instantiated BGPQs, using the set of reformulation rules associated with $\mathcal{R}_c$ (see Figure 3.3). This reformulation step is sound and complete for query answering w.r.t. $\mathcal{R}_c$, i.e., for any graph G of ontology O:

$$q(G, \mathcal{R}_c) = Q_c(G)$$

Step 2. Reformulation w.r.t. $\mathcal{R}_a$: As explained above, Q_c consists of queries that do not contain RDFS triples. It is given as input to the query reformulation algorithm of [Goasdoué et al., 2013], which relies on a set of reformulation rules associated with $\mathcal{R}_a$ to output a union $Q_{c,a}$ of partially instantiated BGPQs. This reformulation step being sound and complete for query answering on the data triples of an RDF graph, we obtain the soundness and completeness of the two-step reformulation (next Theorem 3.5):

$$q(G, \mathcal{R}_c \cup \mathcal{R}_a) = Q_c(G, \mathcal{R}_a) = Q_{c,a}(G)$$

3.2.3 Reformulation rules associated with $\mathcal{R}_c$

We now detail reformulation rules associated with $\mathcal{R}_c$, see Figure 3.3. Each reformulation rule is of the form $\frac{input}{output}$, where the input is composed of a triple from a partially instantiated query q_σ and a triple from O and the output is a new query obtained from q_σ by instantiating a variable, removing the input triple, or replacing it by one or two triples.

$$\frac{(s,v,o) \in q_\sigma}{q_{\sigma \cup \{v \to \prec_{sc}\}}}, \frac{(s,v,o) \in q_\sigma}{q_{\sigma \cup \{v \to \prec_{sp}\}}}, \frac{(s,v,o) \in q_\sigma}{q_{\sigma \cup \{v \to \hookleftarrow_d\}}}, \frac{(s,v,o) \in q_\sigma}{q_{\sigma \cup \{v \to \hookrightarrow_r\}}} \tag{3.3}$$

$$\frac{(\mathsf{s},\mathsf{p},\mathsf{o}) \in q_\sigma, (\mathsf{s},\mathsf{p},\mathsf{o}) \in O}{q_\sigma[(\mathsf{s},\mathsf{p},\mathsf{o})/-]} \tag{3.4}$$

$$\frac{(v_1, \hookleftarrow, v_2) \in q_\sigma, (\mathsf{p}, \hookleftarrow, \mathsf{c}) \in O}{q_\sigma[(v_1, \hookleftarrow, v_2)/(v_1, \prec_{sp}, \mathsf{p}), (\mathsf{c}, \prec_{sc}, v_2)]} \tag{3.5}$$

$$\frac{(v_1, \hookleftarrow, v_2) \in q_\sigma, (\mathsf{p}, \hookleftarrow, \mathsf{c}) \in O}{q_{\sigma \cup \{v_1 \to \mathsf{p}\}}} \tag{3.6}$$

$$\frac{(v_1, \hookleftarrow, v_2) \in q_\sigma, (\mathsf{p}, \hookleftarrow, \mathsf{c}) \in O}{q_{\sigma \cup \{v_2 \to \mathsf{c}\}}} \tag{3.7}$$

$$\frac{(v, \hookleftarrow, \mathsf{c}) \in q_\sigma, (\mathsf{p}, \hookleftarrow, \mathsf{c}) \in O}{q_{\sigma \cup \{v \to \mathsf{p}\}}} \tag{3.8}$$

$$\frac{(\mathsf{p}, \hookleftarrow, v) \in q_\sigma, (\mathsf{p}, \hookleftarrow, \mathsf{c}) \in O}{q_{\sigma \cup \{v \to \mathsf{c}\}}} \tag{3.9}$$

$$\frac{(v, \hookleftarrow, \mathsf{c}) \in q_\sigma, (\mathsf{p}, \hookleftarrow, \mathsf{c}) \in O}{q_\sigma[(v, \hookleftarrow, \mathsf{c})/(v, \prec_{sp}, \mathsf{p})]} \tag{3.10}$$

$$\frac{(\mathsf{p}, \hookleftarrow, v) \in q_\sigma, (\mathsf{p}, \hookleftarrow, \mathsf{c}) \in O}{q_\sigma[(\mathsf{p}, \hookleftarrow, v)/(\mathsf{c}, \prec_{sc}, v)]} \tag{3.11}$$

$$\frac{(s, \hookleftarrow, \mathsf{c}_1) \in q_\sigma, (\mathsf{c}, \prec_{sc}, \mathsf{c}_1) \in O, \mathsf{c} \neq \mathsf{c}_1}{q_\sigma[(s, \hookleftarrow, \mathsf{c}_1)/(s, \hookleftarrow, \mathsf{c})]} \tag{3.12}$$

$$\frac{(\mathsf{p}, \hookleftarrow, o) \in q_\sigma, (\mathsf{p}, \prec_{sp}, \mathsf{p}_1) \in O, \mathsf{p} \neq \mathsf{p}_1}{q_\sigma[(\mathsf{p}, \hookleftarrow, o)/(\mathsf{p}_1, \hookleftarrow, o)]} \tag{3.13}$$

$$\frac{(v_1, \prec, v_2) \in q_\sigma, (\mathsf{c}_1, \prec, \mathsf{c}_2) \in O}{q_{\sigma \cup \{v_1 \to \mathsf{c}_1\}}} \tag{3.14}$$

$$\frac{(v, \prec, \mathsf{c}_2) \in q_\sigma, (\mathsf{c}_1, \prec, \mathsf{c}_2) \in O}{q_{\sigma \cup \{v \to \mathsf{c}_1\}}} \tag{3.15}$$

$$\frac{(\mathsf{c}_1, \prec, v) \in q_\sigma, (\mathsf{c}_1, \prec, \mathsf{c}_2) \in O}{q_{\sigma \cup \{v \to \mathsf{c}_2\}}} \tag{3.16}$$

$$\frac{(\mathsf{c}_1, \prec, o) \in q_\sigma, (\mathsf{c}_1, \prec, \mathsf{c}_2) \in O, \mathsf{c}_1 \neq \mathsf{c}_2}{q_\sigma[(\mathsf{c}_1, \prec, o)/(\mathsf{c}_2, \prec, o)]} \tag{3.17}$$

$$\frac{(s, \prec, \mathsf{c}_2) \in q_\sigma, (\mathsf{c}_1, \prec, \mathsf{c}_2) \in O, \mathsf{c}_1 \neq \mathsf{c}_2}{q_\sigma[(s, \prec, \mathsf{c}_2)/(s, \prec, \mathsf{c}_1)]} \tag{3.18}$$

Figure 3.3: Reformulation rules for $\mathcal{R}_c$ using a partially instantiated query q_σ w.r.t. an RDFS ontology O. For compactness, we factorize similar rules, using the symbol $\hookleftarrow$ to denote either $\hookleftarrow_d$ or $\hookrightarrow_r$, and $\prec$ to denote either $\prec_{sc}$ or $\prec_{sp}$.

The notation *old triple/new triple(s)* means that *old triple* is replaced by *new triple(s)*. The specific case where *old triple* is simply removed is denoted by *old triple*/−. The notations for the triples themselves are the following:

- a bold character like $\mathtt{c}$, $\mathtt{p}$, $\mathtt{s}$ or $\mathtt{o}$ represents an IRI or a blank node
- a v character represents a variable of the query
- s and o characters represent either variables, IRIs or blank nodes, in subject and object positions respectively.

The four rules (3.3) substitute a variable in a property position by one of the four built-in RDFS properties. All the other rules take as input query triples of the form $(s, \mathtt{p}, o)$, where $\mathtt{p}$ is a built-in RDFS property. Rule (3.4) simply removes from q_σ an (instantiated) input triple found in O.

Query triples with a domain ($\hookleftarrow_d$) or range property ($\hookrightarrow_r$) are processed by Rules (3.5)-(3.13). Given a triple $(\mathtt{p}, \hookleftarrow, \mathtt{c})$ in O (where $\hookleftarrow$ stands for $\hookleftarrow_d$ or $\hookrightarrow_r$), Rule (3.5) replaces a query triple of the form $(v_1, \hookleftarrow, v_2)$ by two triples $(v_1, \prec_{sp}, \mathtt{p})$ and $(\mathtt{c}, \prec_{sc}, v_2)$. This rule relies on the fact that a triple $(\mathtt{p}', \hookleftarrow, \mathtt{c}')$ belongs to the saturation of the RDF graph by $\mathcal{R}_c$ if and only if $\mathtt{p}'$ is a subproperty of $\mathtt{p}$ (including $\mathtt{p} = \mathtt{p}'$) and $\mathtt{c}$ is a subclass of $\mathtt{c}'$ (including $\mathtt{c} = \mathtt{c}'$), as stated by Lemma 3.1 in the following section 3.2.4. Rules (3.10) and (3.11) are based on the same lemma: given a triple $(\mathtt{p}, \hookleftarrow, \mathtt{c})$ in O, Rule (3.10) (respectively Rule (3.11)) replaces a query triple of the form $(v, \hookleftarrow, \mathtt{c})$ (respectively $(\mathtt{p}, \hookleftarrow, v)$) by the triple $(v, \prec_{sp}, \mathtt{p})$ (respectively $(\mathtt{c}, \prec_{sc}, v)$). However, we do not assume that the ontology ensures the reflexivity of the subclass and subproperty relations, hence Rules (3.6)-(3.9), whose sole purpose is to deal with the cases $\mathtt{c} = \mathtt{c}'$ and $\mathtt{p} = \mathtt{p}'$ in the lemma. Should the ontology contain axiomatic triples ensuring the reflexivity of subclass and subproperty, these four rules would be useless. Note that a natural candidate rule to deal with the case where $\mathtt{c} \neq \mathtt{c}'$ and $\mathtt{p} \neq \mathtt{p}'$ in the lemma would have been the following:

$$\frac{(\mathtt{p}', \hookleftarrow, \mathtt{c}') \in q_\sigma, (\mathtt{p}, \hookleftarrow, \mathtt{c}) \in O}{q_{\sigma[(\mathtt{p}', \hookleftarrow, \mathtt{c}')/(\mathtt{p}', \prec_{sp}, \mathtt{p}), (\mathtt{c}, \prec_{sc}, \mathtt{c}')]}} \tag{3.19}$$

However, such a rule is flawed: it would blindly consider all triples $(\mathtt{p}, \hookleftarrow, \mathtt{c})$ from O, which causes a combinatorial explosion. Instead, we propose Rules (3.12) and (3.13), which use $\mathtt{p}'$ and $\mathtt{c}'$ as guides to replace $(\mathtt{p}', \hookleftarrow, \mathtt{c}')$ by other domain / range triples based on the subproperty-chains from $\mathtt{p}'$ and the subclass-chains to $\mathtt{c}'$.

Query triples with a subclass ($\prec_{sc}$) or subproperty ($\prec_{sp}$) property are processed by Rules (3.14)-(3.18). Rules (3.14), (3.15), (3.16) instantiate a variable using an ontology triple of the form $(\mathtt{c}_1, \prec, \mathtt{c}_2)$. In Rule (3.14), which considers a query triple with two variables and instantiates one of these variables, we arbitrarily chose to instantiate the first variable. The two last rules (3.17) and (3.18) allow to go up or down in the class and property hierarchies.

3.2.4 Reformulation algorithm associated with $\mathcal{R}_c$

The reformulation algorithm itself, denoted by Reformulate$_c$, is presented in Algorithm 1. The set of queries to be explored (named *toExplore*) initially contains q. Exploring a query consists of generating all new queries that can be obtained from it by applying a reformulation rule (lines 7–9). Newly generated queries are put in the set

named *produced*. The algorithm proceeds in a breadth-first manner, exploring at each step the queries that have been generated at the previous step. When no new query can be generated at a step, the algorithm stops, otherwise the next step will explore the newly generated queries (line 11). Note the use of a set named *explored*, which contains all explored queries; the purpose of this set is to avoid infinite generation of the same queries when the subclass or subproperty hierarchy contains cycles (other than loops), otherwise it is useless. Importantly, not all explored queries are returned in the resulting set, but only those that no longer contain RDFS triples (lines 5–6). Indeed, on the one hand RDFS triples that contain variables are instantiated by the rules in all possible ways using the ontology, and, on the other hand, instantiated triples that belong to the ontology are removed (by Rule (3.4)). Finally, note that a variable v in a triple of the form (s, v, o) is replaced by a built-in RDFS property in some queries (by Rule (3.3)) and left unchanged in others as it may also be later mapped to a user-defined property in the RDF graph G.

Algorithm 1: Reformulate$_c$.

Input : BGPQ q and ontology O
Output: the reformulation of q with the rules from Fig. 1

```
1  result ← ∅; toExplore ← {q}; explored ← ∅
2  while toExplore ≠ ∅ do
3      produced ← ∅
4      for each q_σ ∈ toExplore do
5          if q_σ does not contain any RDFS triple then
6              result ← result ∪ {q_σ}
7          for each RDFS triple t in q_σ do
8              for each q'_σ obtained by applying a reformulation rule to t do
9                  produced ← produced ∪ {q'_σ}
10         explored ← explored ∪ {q_σ}
11     toExplore ← produced \ explored
12 return result
```

A simple analysis of the reformulation rule behaviour shows that the worst-case time complexity of the algorithm Reformulate$_c$ is polynomial in the size of O and simply exponential in the size of q. More precisely:

Property 3.3. *The algorithm* Reformulate$_c$ *runs in time* $O(|Val(O)|^{6|q|})$*, where* $|q|$ *is the number of triples in the body of* q*.*

The correctness of the algorithm relies on the following lemma (already mentioned in the presentation of reformulation rules), which characterizes the saturated graph $G^{\mathcal{R}_c}$. We call $\prec_{sc}$-chain (resp. $\prec_{sp}$-chain) from s to o a possibly empty sequence of triples $(s_i, \prec_{sc}, o_i)$ (resp. $(s_i, \prec_{sp}, o_i)$) with $1 \leq i \leq n$, such that $s_1 = s$, $o_n = o$ and, for $i > 1$, $s_i = o_{i-1}$. Since we do not enforce the reflexivity of the subclass relation, a triple $(\mathsf{c}, \prec_{sc}, \mathsf{c})$ belongs to $G^{\mathcal{R}_c}$ if and only if there is a non-empty $\prec_{sc}$-chain from c to c (which includes the case $(\mathsf{c}, \prec_{sc}, \mathsf{c}) \in G$). The same holds for the subproperty relation.

Lemma 3.1. *Let* G *be an RDF graph. It holds that:*

- $(\mathsf{c}, \prec_{sc}, \mathsf{c}') \in G^{\mathcal{R}_c}$ *iff* G *contains a non-empty* $\prec_{sc}$*-chain from* c *to* c'*;*

- $(\mathtt{p}, \prec_{sp}, \mathtt{p}') \in G^{\mathcal{R}_c}$ *iff* G *contains a non-empty empty* $\prec_{sp}$*-chain from* $\mathtt{p}$ *to* $\mathtt{p}'$*;*
- $(\mathtt{p}', \hookleftarrow_d, \mathtt{c}') \in G^{\mathcal{R}_c}$ *iff* G *contains a triple* $(\mathtt{p}, \hookleftarrow_d, \mathtt{c})$*, a (possibly empty)* $\prec_{sp}$*-chain from* $\mathtt{p}'$ *to* $\mathtt{p}$ *and a (possibly empty)* $\prec_{sc}$*-chain from* $\mathtt{c}$ *to* $\mathtt{c}'$*. The case for* $(\mathtt{p}', \hookrightarrow_r, \mathtt{c}') \in G^{\mathcal{R}_c}$ *is similar (*$\hookleftarrow_d$ *is replaced by* $\hookrightarrow_r$ *in the above statement).*

All blank nodes introduced by the reformulation rules, specifically refer to unknown classes and properties they identify within the ontology at hand. This justifies the need of using non-standard query evaluation (Definition 2.13), in this thesis.

Theorem 3.3. *Let G be an RDF graph with ontology O and q be a BGP query. Let Q_c be the output of* $\text{Reformulate}_c(q, O)$*. Then:*

$$q(G, \mathcal{R}_c) = Q_c(G) \tag{3.20}$$

$$Q_c(G) = Q_c(G \setminus O) \tag{3.21}$$

Example 3.5. *Consider the following BGPQ asking* which people use a vehicle of a specific type and how they use it*:*

$$q(x, y) \leftarrow (x, y, z), (z, \tau, t), (y, \prec_{sp}, :\text{uses}), (t, \prec_{sc}, :\text{Vehicle})$$

Its answer set on G_{ex} *(on Figure 2.1) w.r.t.* $\mathcal{R}_{\text{RDFS}}$*, which can be computed on* $(G_{\text{ex}})^{\mathcal{R}_{\text{RDFS}}}$ *from Figure 2.2, is the following:*

$$q(G_{\text{ex}}, \mathcal{R}_{\text{RDFS}}) = \{\langle :\text{Luke}, :\text{pilotOf}\rangle\}.$$

The output of $\text{Reformulate}_c(q, \text{RDFS}(G_{\text{ex}}))$ *is:*

$$\begin{aligned} Q_c = \; & \{q'(x, :\text{pilotOf}) \leftarrow (x, :\text{pilotOf}, z), (z, \tau, \_{:}b_c), \\ & q''(x, :\text{usesWeapon}) \leftarrow (x, :\text{usesWeapon}, z), (z, \tau, \_{:}b_c)\} \end{aligned}$$

where q' and q'' can be obtained by first using Rule (3.15) on q's triple $(y, \prec_{sp}, :\text{uses})$*, which binds y to* :pilotOf *or to* :usesWeapon*, yielding two queries; then, Rule (3.15) is used again on the triple* $(t, \prec_{sc}, :\text{Vehicle})$ *of each query to bind t to* $\_{:}b_c$*. Further, these bindings produce the fully instantiated RDFS constraints* $(:\text{pilotOf}, \prec_{sp}, :\text{uses})$ *and* $(\_{:}b_C, \prec_{sc}, :\text{Vehicle})$ *in the first query, as well as* $(:\text{usesWeapon}, \prec_{sp}, :\text{uses})$ *and* $(\_{:}b_C, \prec_{sc}, :\text{Vehicle})$ *in the second query, which are removed by Rule (3.4).*

The non-standard *evaluation (Definition 2.13) of Q_c on* G_{ex}*, i.e.,* $q'(G_{\text{ex}}) \cup q''(G_{\text{ex}})$ *provides the correct answer set* $\{\langle :\text{Luke}, :\text{pilotOf}\rangle\}$*, whose only tuple results from q'. Using* standard *evaluation (Definition 2.11), the incorrect answer* $\langle :\text{Rey}, :\text{usesWeapon}\rangle$ *would also have been obtained from q'', since under standard semantics q'' asks for* who is using a weapon of some type *(this is the case of* :Rey *who is using a light saber) and not* who is using a weapon of the particular unknown type of vehicle designated by $\_{:}b_c$ in G_{ex}.

3.2.5 Reformulation with $\mathcal{R}_a$

We present in this section reformulation with $\mathcal{R}_a$, the set of assertion rules. We will reuse the algorithm introduced in [Goasdoué et al., 2013], which is based on a set of reformulation rules associated with $\mathcal{R}_a$, and inspired our own algorithm of reformulation

$$\frac{(s, v, o) \in q_\sigma}{q_{\sigma \cup \{v \to \tau\}}} \tag{3.23}$$

$$\frac{(s, v, o) \in q_\sigma, (\mathsf{p}_1, \prec_{sp}, \mathsf{p}) \in O}{q_{\sigma \cup \{v \to \mathsf{p}\}}} \tag{3.24}$$

$$\frac{(s, \tau, v) \in q_\sigma, (\mathsf{c}_1, \prec_{sc}, \mathsf{c}) \in O}{q_{\sigma \cup \{v \to \mathsf{c}\}}} \tag{3.25}$$

$$\frac{(s, \tau, v) \in q_\sigma, (\mathsf{p}, \hookleftarrow_d, \mathsf{c}) \in O}{q_{\sigma \cup \{v \to \mathsf{c}\}}} \tag{3.26}$$

$$\frac{(s, \tau, v) \in q_\sigma, (\mathsf{p}, \hookrightarrow_r, \mathsf{c}) \in O}{q_{\sigma \cup \{v \to \mathsf{c}\}}} \tag{3.27}$$

$$\frac{(s, \tau, \mathsf{c}_2) \in q_\sigma, (\mathsf{c}_1, \prec_{sc}, \mathsf{c}_2) \in O}{q_{\sigma[(s,\tau,\mathsf{c}_2)/(s,\tau,\mathsf{c}_1)]}} \tag{3.28}$$

$$\frac{(s, \tau, \mathsf{c}) \in q_\sigma, (\mathsf{p}, \hookleftarrow_d, \mathsf{c}) \in O,}{q_{\sigma[(s,\tau,\mathsf{c})/(s,\mathsf{p},v)]}} \tag{3.29}$$

$$\frac{(s, \tau, \mathsf{c}) \in q_\sigma, (\mathsf{p}, \hookrightarrow_r, \mathsf{c}) \in O}{q_{\sigma[(s,\tau,\mathsf{c})/(v,\mathsf{p},s)]}} \tag{3.30}$$

$$\frac{(s, \mathsf{p}_2, o) \in q_\sigma, (\mathsf{p}_1, \prec_{sp}, \mathsf{p}_2) \in O}{q_{\sigma[(s,\mathsf{p}_1,o)/(s,\mathsf{p}_2,o)]}} \tag{3.31}$$

Figure 3.4: Reformulation rules for $\mathcal{R}_a$ using a partially instantiated query q_σ and an ontology O.

with $\mathcal{R}_c$. However, the set of reformulation rules associated with $\mathcal{R}_a$ that we will adopt is slightly different from the original one.

In [Goasdoué et al., 2013], query answering is restricted to *instance-level queries*, i.e., queries are evaluated on the graph data triples only. As stated in Theorem 3.1, query answering on the graph data triples can be performed by considering reasoning w.r.t. $\mathcal{R}_a$ only. Hence, this previous work describes a reformulation algorithm taking as input a BGPQ q and an RDF graph G with ontology O and returning a reformulated query Q_a such that:

$$q(G^{\mathcal{R}_a} \setminus O) = Q_a(G \setminus O) \tag{3.22}$$

Note that $G^{\mathcal{R}_a} \setminus O$ is exactly the set of data triples of $G^{\mathcal{R}_a}$.

We adapt the results of [Goasdoué et al., 2013] by:

- changing the set of reformulation rules such that (i) the reformulation algorithm only requires the ontology as input, instead of the whole graph, and (ii) some redundant rules are removed,
- considering reformulation for query answering on data and schema triples together.

We consider the reformulation rules shown in Figure 3.4. The notations and semantics of the rules is the same as in Figure 3.3 (reformulation rules for $\mathcal{R}_c$). Compared to the

reformulation rules in [Goasdoué et al., 2013]), we removed four rules that instantiate query variables with values from the graph (rules (6), (8), (9) and (11) in the original set of rules). The remaining rules take ontology triples as input. In Figure 3.4, the reformulation rules from 3.23 to 3.27 bind a query variable in property or class position to a value from the ontology or to τ, while the other rules replace a query triple having a constant in class or property position with a another triple that "entails" it w.r.t. $\mathcal{R}_a$ and O. The first set of rules is necessary to fully apply the other rules during the reformulation process on query triples of the form (x, τ, y) and (x, y, z).

Finally, Reformulate$_a$, the reformulation algorithm w.r.t. $\mathcal{R}_a$, takes a BGP query and an ontology as input. It is similar to Reformulate$_c$, with the following differences: (i) the reformulation rules w.r.t. $\mathcal{R}_a$ are considered instead of the ones w.r.t. $\mathcal{R}_c$, (ii) the conditions about RDFS triples are removed, i.e., in Algorithm 1:

- the block at lines 5-6 is replaced with the line 6,
- the for loop at line 7 iterates on all triples in q_σ.

Hence, Reformulate$_a$ returns all the queries that have been explored during its execution, similarly to a classical reformulation algorithm, and contrarily to Reformulate$_c$, which only returns explored queries without RDFS triples.

Reformulate$_a$ is correct and complete w.r.t. $\mathcal{R}_a$ for $\mathcal{R}_a$-compliant graphs:

Theorem 3.4. *Let q be a BGP query and O an RDFS ontology, let Q_a denote the output of Reformulate$_a$ on q and O; then, for any $\mathcal{R}_a$-compliant graph G, with O its ontology, the following holds:*

$$q(G, \mathcal{R}_a) = Q_a(G)$$

If G is moreover with split reasoning, Equation 3.22 also holds.

The following example illustrates the behaviour of Reformulate$_a$:

Example 3.6. *Consider the query $q(x) \leftarrow (x, \tau, :\text{Vehicle})$ with the graph G_{ex} in Figure 2.1 and O_{ex} its ontology. The answer set of q on G_{ex} w.r.t. $\mathcal{R}_a$ is $q(G_{ex}^{\mathcal{R}_a}) = \{\langle :\text{spaceship1}\rangle\}$. The output of Reformulate$_a(q, O_{ex})$ is:*

$$\begin{aligned} Q_a = \; & \{q(x) \leftarrow (x, \tau, :\text{Vehicle}), \\ & q'(x) \leftarrow (x, \tau, \_{:}b_c), \\ & q''(x) \leftarrow (y, :\text{pilotOf}, x)\} \end{aligned}$$

where q' is obtained by applying the Rule (3.28) on the only triple of q and the triple $(\_{:}b_c, \prec_{sc}, :\text{Vehicle}) \in O_{ex}$ and q' is obtained by applying the Rule (3.30) on the only triple of q' and the triple $(:\text{pilotOf}, \hookrightarrow_r, :\text{Vehicle}) \in O_{ex}$. The non-standard evaluation of Q_a on G returns the correct answer, which is provided by q' and q''.

Equation 3.22 does not hold in general on $\mathcal{R}_a$-compliant graphs, since, in that case, the reformulation Q_a needs to be evaluated (in a non-standard manner) on the whole graph, i.e., including its ontology, in order to produce the complete answer set. A counter-example to this equation is:

Example 3.7. *Given G a $\mathcal{R}_a$-compliant graph containing only the RDFS triple $(\hookrightarrow_r, \hookrightarrow_r, \text{rdfs:Class})$, the rule* `rdfs3` *in $\mathcal{R}_a$ entails the triple $(\text{rdfs:Class}, \tau, \text{rdfs:Class})$, when it applies on G. Considering the query:*

$$q(x) \leftarrow (x, \tau, \text{rdfs:Class}),$$

the left part of Equation 3.22 is $q(G^{\mathcal{R}_a} \setminus G) = \langle$rdfs:Class$\rangle$*, while its right part is* $\widetilde{Q_a(\emptyset)} = \emptyset$*, where* Q_a*, the reformulation of* q *w.r.t.* $\mathcal{R}_a$ *from Reformulate*$_a$*, is:*

$$Q_a(x) = \{q(x)(x, \tau, \text{rdfs:Class}), \\ q'(x)(y, \hookrightarrow_r, x)\}.$$

3.2.6 Reformulation with $\mathcal{R}_c \cup \mathcal{R}_a$

We now present a way to combine algorithms Reformulate$_c$ and Reformulate$_a$ in order to obtain a sound and complete reformulation algorithm w.r.t. $\mathcal{R}_c \cup \mathcal{R}_a$.

The adapation of Reformulate$_a$ to an input UBGPQ instead of a BGPQ is straightforward. Hence, denoting by $Q_{c,a}$ the output of Reformulate$_a(Q_c, O)$, we obtain:

Theorem 3.5. *Let* G *be an RDF graph with split reasoning and* q *be a BGPQ without blank nodes. Let* $Q_{c,a}$ *be the reformulation of* q *by the 2-step algorithm described in Section 3.2.2. Then:*

$$q(G, \mathcal{R}_c \cup \mathcal{R}_a) = Q_{c,a}(G) = Q_{c,a}(G \setminus O)$$

Proof.

$$\begin{aligned} q(G, \mathcal{R}_c \cup \mathcal{R}_a) &= q(G^{\mathcal{R}_c \cup \mathcal{R}_a}) \quad \text{by definition} \\ &= q((G^{\mathcal{R}_a})^{\mathcal{R}_c}) \quad \text{using Theorem 3.2} \\ &= Q_c(G^{\mathcal{R}_a}) \quad \text{from Equation 3.20} \\ &\Big(= Q_{c,a}(G) \Big) \\ &= Q_c(G^{\mathcal{R}_a} \setminus O) \quad \text{from Equation 3.21} \\ &= Q_{c,a}(G \setminus O) \quad \text{from Equation 3.22} \end{aligned}$$

□

Example 3.8. *Let* q *be the following query on the graph* G_{ex} *in Figure 2.1, asking for "those who pilot an object and the type of this object":*

$$q(x, t) \leftarrow (x, \text{:pilotOf}, z), (z, \tau, t)(t, \prec_{sc}, \text{:Object})$$

Its reformulation w.r.t. $\mathcal{R}_c$ *is:*

$$Q_c = \{q_1(x, \text{:LightSaber}) \leftarrow (x, \text{:pilotOf}, z), (z, \tau, \text{:LightSaber}), \\ q_2(x, \text{:Vehicle}) \leftarrow (x, \text{:pilotOf}, z), (z, \tau, \text{:Vehicle}), \\ q_3(x, \_{:}b_c) \leftarrow (x, \text{:pilotOf}, z), (z, \tau, \_{:}b_c)\}$$

and the reformulation of Q_c *w.r.t.* $\mathcal{R}_a$ *is:*

$$Q_{c,a} = \{q_1(x, \text{:LightSaber}) \leftarrow (x, \text{:pilotOf}, z), (z, \tau, \text{:LightSaber}), \\ q_2(x, \text{:Vehicle}) \leftarrow (x, \text{:pilotOf}, z), (z, \tau, \text{:Vehicle}), \\ q'_2(x, \text{:Vehicle}) \leftarrow (x, \text{:pilotOf}, z), (z, \tau, \_{:}b_c), \\ q''_2(x, \text{:Vehicle}) \leftarrow (x, \text{:pilotOf}, z), (y, \text{:pilotOf}, z), \\ q_3(x, \_{:}b_c) \leftarrow (x, \text{:pilotOf}, z), (y, \tau, \_{:}b_c)\}$$

The expected answers of q *on* G_{ex} *w.r.t.* $\mathcal{R}_{\text{RDFS}}$ *are*

$$q(G_{ex}^{\mathcal{R}_{\text{RDFS}}}) = \{\langle \text{:Luke}, \text{:Vehicle}\rangle, \langle \text{:Luke}, \_{:}b_c\rangle\}$$

The first answer is returned by the non-standard evaluation of q'_2 and q''_2 on G_{ex}, while the second one is returned by the non-standard evaluation of q_3 on G_{ex}.

One can observe that the union of BGPQs $Q_{c,a}$ contains some redundancy. Notably, the queries q_2 and q'_2 are contained in q''_2 (Definition 2.14). Hence, the queries q_2 and q'_2 can be removed from $Q_{c,a}$ to form a minimal union of BGPQs (Definition 2.15).

Note that we have made the choice of reformulating w.r.t. $\mathcal{R}_c \cup \mathcal{R}_a$ by **first reformulating w.r.t.** $\mathcal{R}_c$, then w.r.t. $\mathcal{R}_a$. Reversing the reformulation steps would have led to a reformulation $Q_{\mathbf{a,c}}$, which has the property of being a sound and complete reformulation w.r.t. $\mathcal{R}_c \cup \mathcal{R}_a$, i.e., $q(G, \mathcal{R}_c \cup \mathcal{R}_a) = Q_{a,c}(G)$, furthermore under a weaker condition on G: G does not need to be with split reasoning, but only $\mathcal{R}_a$-compliant. The properties of this reformulation will be exposed in Section 3.2.8. In the current section, we presented the reformulation $Q_{c,a}$ instead of $Q_{a,c}$, because it has better computational properties. Indeed, starting by reformulating a query q w.r.t. $\mathcal{R}_c$ allows one to reduce the number of values by which variables of q will be instantiated during the reformulation process. This implies that less reformulations are explored by the algorithm, leading to a more efficient execution. The decrease in the number of instantiated values concerns the variables that occur in the RDFS triples of q. For instance, consider the query from Example 3.5:

$$q(x, y) \leftarrow (x, y, z), (z, \tau, t), (y, \prec_{sp}, \text{:uses}), (t, \prec_{sc}, \text{:Vehicle})$$

Reformulating q w.r.t. $\mathcal{R}_a$ leads to blindly instantiate the variable t by all the superclasses (of a class) in the graph according to the rule 3.25. Notably, t will be instantiated by :Object, violating the constraint $(t, \prec_{sc}, \text{:Vehicle})$ in the query, hence Reformulate$_a$ outputs BGPQs that will be later removed by Reformulate$_c$. Starting by reformulation w.r.t. $\mathcal{R}_c$ allows one to instantiate variables according to the RDFS constraints in the query, hence to decrease the number of instantiations.

3.2.7 Experiments

We implemented our reformulation algorithm on top of OntoSQL (`https://ontosql.inria.fr`), a Java platform providing efficient RDF storage, saturation, and query evaluation on top of an RDBMS [Bursztyn et al., 2015, Goasdoué et al., 2013]; we used Postgres v9.6. To save space, OntoSQL encodes IRIs and literals into integers, and a dictionary table which allows going from one to the other. It stores all resources of a certain type in a one-attribute table, all subject, object pairs for each data property in a table, and all schema triples in another table; the tables are indexed. Our server has a 2,7 GHz Intel Core i7 and 160 GB of RAM; it runs CentOs Linux 7.5.

We generated LUBM$^\exists$ data graphs [Lutz et al., 2013] of 10M triples and restricted the ontology to RDFS, leading to 175 triples (123 $\prec_{sc}$, 5 $\prec_{sp}$, 25 $\hookleftarrow_d$ and 22 $\hookrightarrow_r$). We devised 14 queries having from 3 to 7 triples; one has no result, while the others have a few dozen to three hundred thousand results. Each has 1 or 2 triples which match the ontology (and must be evaluated on it for correctness), including (but not limited to) the generic triple (x, y, z), which appears 7 times overall in our workload. Some of our queries are not handled through reformulation by AllegroGraph and Stardog, nor by Virtuoso (recall Section 3.1.2).

Figure 3.5 shows for each query: the size of the UBGPQ reformulation (in parenthesis after the query name on the x axis), i.e., the number of BGPQs it contains; the reformulation time (with both $\mathcal{R}_c$ and $\mathcal{R}_a$; all times are in ms); the time to translate the reformulation into SQL; the time to evaluate this SQL query; the total query answering time through

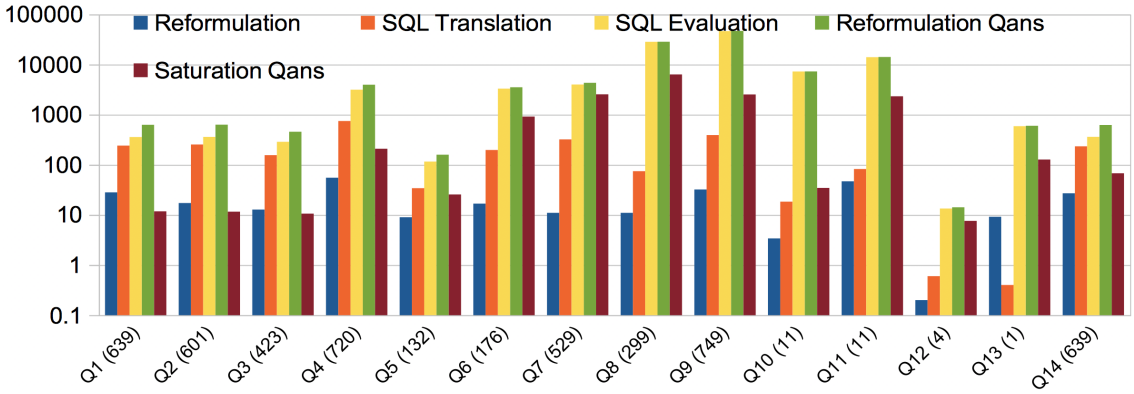


Figure 3.5: Query answering times (in ms) through reformulation and saturation.

reformulation, and (for comparison) through saturation. Note the logarithmic y axis. Details of our experiments are available in the appendix, Section A.1.2. The reformulation time is very short (0.2 ms to 55 ms). Unsurprisingly, the time to convert the reformulation into SQL is closely correlated with the reformulation size. The overhead of our approach is quite negligible, given that the answering time through reformulation is close to the SQL evaluation time.

As expected, saturation-based query answering is faster; however, saturating this graph took more than 1289 seconds, while the slowest query (Q9) took 46 seconds. As in [Goasdoué et al., 2013], we compute for each query Q a *threshold* n_Q which is the smallest number of times we need to run Q, so that saturating G and running Q n_Q times on $G^{\mathcal{R}}$ is faster than n_Q runs of Q through reformulation; intuitively, *after* n_Q *runs of* Q, *the saturation cost amortizes*. For our queries, n_Q ranged from 29 (Q9) to 9648 (Q5), which shows that saturation costs take a while to amortize. If the graph or the ontology changes, requiring maintenance of the saturated graph, reformulation may be even more competitive.

3.2.8 Reformulation for $\mathcal{R}_a$-compliant graphs

In the previous sections, we studied reformulation of BGPQ w.r.t. the $\mathcal{R}_{\text{RDFS}}$ rule set, for RDF graphs with split reasoning (Definition 3.3). In Section 3.2.1, we pointed out some interesting properties of reasoning with $\mathcal{R}_{\text{RDFS}}$ on $\mathcal{R}_a$*-compliant graphs* (Definition 3.1), a set of graphs that strictly contain those with split reasoning. In this section, we will show how to adapt our reformulation technique to cope with $\mathcal{R}_a$-compliant graphs. In these graphs, data triples may also be entailed from schema triples only, using rules in $\mathcal{R}_a$, in addition to all entailments exhibited for graphs with split reasoning, as pictured in Figure 3.2.

On $\mathcal{R}_a$-compliant graphs, only the following equation of the two in Theorem 3.2 holds:

$$G^{\mathcal{R}_c \cup \mathcal{R}_a} = (G^{\mathcal{R}_c})^{\mathcal{R}_a} \qquad \text{(3.1 revisited)}$$

This equation states that the saturation of a graph can be computed by a two-step process: a saturation w.r.t. $\mathcal{R}_c$ followed by a saturation w.r.t. $\mathcal{R}_a$. Reversing the saturation steps is possible for graphs with split reasoning and is used to prove Theorem 3.5, which states the soundness and completeness of the 2-step reformulation algorithm for graphs with split reasoning. However, this algorithm is not complete for $\mathcal{R}_a$-compliant graphs, as shown by the next example:

Example 3.9 (Incompleteness of $Q_{c,a}$)**.** *Consider the graph G containing the following ontology triples (note that O = G, i.e., G has no data triples):*

$\hookrightarrow_r$	$\hookrightarrow_r$	rdfs:Class
:pilotOf	$\hookrightarrow_r$	:Vehicle
:Vehicle	$\prec_{sc}$	:Object

Its saturation $G^{\mathcal{R}_a \cup \mathcal{R}_c}$ contains the following triples, we remark that the G only entails data by $\mathcal{R}_a$:

$\hookrightarrow_r$	$\hookrightarrow_r$	rdfs:Class	
:pilotOf	$\hookrightarrow_r$	:Vehicle	
:Vehicle	$\prec_{sc}$	:Object	
:pilotOf	$\hookrightarrow_r$	:Object	(`ext2` *on* (:pilotOf, $\hookrightarrow_r$, :Vehicle) *and* (:Vehicle, $\prec_{sc}$, :Object)
:Object	τ	rdfs:Class	(`rdfs3` *on* ($\hookrightarrow_r$, $\hookrightarrow_r$, rdfs:Class) *twice)*

Now, let the query $q(x) \leftarrow (x, \tau, \text{rdfs:Class})$. Assume its reformulation $Q_{\mathbf{c,a}}$ is computed by considering $\mathcal{R}_c$ first, which yields $Q_c = q$, then $\mathcal{R}_a$. The output of Reformulate$_a(Q_c, O)$ is the following:

$$\begin{aligned} Q_{c,a} &= q(x) \leftarrow (x, \tau, \text{rdfs:Class}) \\ &\cup \; q(x) \leftarrow (z, \hookrightarrow_r, x) \end{aligned}$$

Then, the (non-standard) evaluation of $Q_{c,a}$ on G returns the single answer :Vehicle *and misses the answer* :Object*.*

To recover completeness, still combining algorithms Reformulate$_a$ (w.r.t. $\mathcal{R}_a$) and Reformulate$_c$ (w.r.t. $\mathcal{R}_c$), we reverse their application order on a given query q, whose reformulation becomes $Q_{\mathbf{a,c}}$. In other words, given an ontology O and a BGPQ q, the reformulation $Q_{\mathbf{a,c}}$ of q is the output of Reformulate$_c$(Reformulate$_a(q, O)$, O). The next theorem states the soundness and completeness of this procedure.

Theorem 3.6. *Let G be an $\mathcal{R}_a$-compliant graph, with O its ontology, and q be a BGPQ. Then:*

$$q(G, \mathcal{R}_a \cup \mathcal{R}_c) = Q_{a,c}(G) = Q_{a,c}(G \setminus O)$$

Proof.

$$\begin{aligned} q(G, \mathcal{R}_c \cup \mathcal{R}_a) &= q(G^{\mathcal{R}_c \cup \mathcal{R}_a}) \quad \text{by definition} \\ &= q((G^{\mathcal{R}_c})^{\mathcal{R}_a}) \quad \text{from Equation 3.1} \\ &= Q_a(G^{\mathcal{R}_c}) \\ &= Q_{a,c}(G) \quad \text{from Equation 3.20} \\ &= Q_{a,c}(G \setminus O) \quad \text{from Equation 3.21} \end{aligned}$$

□

We continue Example 3.9 to show the reformulation $Q_{a,c}$ in action:

Example 3.10. *First, we compute Q_a the reformulation of q w.r.t. $\mathcal{R}_a$ (equal to $Q_{\mathbf{c,a}}$ in Example 3.9), then we reformulate it w.r.t. $\mathcal{R}_c$, which yields the following query (where $\top$ denotes the empty set):*

$$\begin{aligned} Q_{a,c} &= q(x) \leftarrow (x, \tau, \text{rdfs:Class}) \\ &\cup \; q(\text{:Vehicle}) \leftarrow \top \\ &\cup \; q(\text{:Object}) \leftarrow \top \end{aligned}$$

We easily check that the evaluation of $Q_{a,c}$ on G returns the complete answer set.

Finally, we point out that the two reformulations $Q_{c,a}$ and $Q_{a,c}$ of a query q on a graph with split reasoning are equivalent.

3.3 RDF storage layouts for efficient query answering

We now study the impact of storage layouts on the efficiency of query answering. We present BGPQ translations into logic query plans for the well-known T and CP layouts, and for the novel TCP layout (recall Section 3.1.3). These translations are considered for both classical query answering techniques, i.e., based on graph saturation and query reformulation.

Our original contributions are the following:

1. We introduce the *novel workload-unaware TCP layout*, which combines the data structures of both T and CP, and an associated algebraic translation of BGPQs into logical plans over TCP.

2. We introduce *summary-based pruning*, an optimization technique of independent interest, that reduces query answering costs on the T, CP and TCP layouts, both when using graph saturation and query reformulation.

3. We *experimentally validate the performance benefits* of the TCP layout and translation, and of summary-based pruning, on a relational and a native RDF database.

Below, after some preliminaries, Sections 3.3.2 and 3.3.3 respectively recall algebraic query translations for the T and CP layouts. We explain why naïve algebraic translation on CP leads to *poor performance*, not only for *general* BGPQs (as noted since [Sidirourgos et al., 2008]), but also for *reformulated* ones (whether general or not). This is due to *interleaved joins and unions*, which limit the optimization opportunities in the RDF database. At the same time, the T layout entails repeated self-joins of the whole triple set, degrading performance on large graphs and complex queries (this motivated introducing the CP layout [Abadi et al., 2007]). Sections 3.3.4 (translations for the TCP layout) and 3.3.5 (summary-based pruning) present our technical contributions and Section 3.3.6 details our experiments. Further details on the experiments can be found in the appendix (Section A.2).

3.3.1 Preliminaries

In the following, we will consider RDF graphs with split reasoning (Definition 3.3) i.e., allowing domain metamodeling, and we will use the following graph example:

Example 3.11 (Running example)**.** *Figure 3.6 shows a sample graph G_{ex}, describing articles and their authors; some articles are Open Access (*:OpenArt*), a subclass of which are Green Open Access ones (*:GOpenArt*).*

We will again consider reasoning w.r.t. the RDFS entailment rules $\mathcal{R}_{\text{RDFS}}$ (see Figure 3.1) and, to reformulate a query w.r.t. $\mathcal{R}_{\text{RDFS}}$, we will consider the algorithm introduced in Section 3.2.6. Further, from now on, we assume that (U)BGPQs, in particular those produced through reformulation, are *minimal* as introduced in Section 2.1.4 and illustrated at the end of Example 3.8.

:OpenArt	$\prec_{sc}$	:Article	:art1	:title	*"RDF storage"*
:GOpenArt	$\prec_{sc}$	:OpenArt	:Alice	:name	*"Alice"*
:Prof	$\prec_{sc}$	:Person	:art1	:firstAuth	:Alice
:teaches	$\hookleftarrow_{d}$	:Prof	:Alice	:teaches	:algo101
:author	$\hookrightarrow_{r}$	:Person	:art1	:author	:Bob
:firstAuth	$\prec_{sp}$	:author	:Bob	:name	*"Bob"*
			:art1	τ	:GOpenArt

Figure 3.6: Sample RDF graph G_{ex} (schema triples on left and data triples right).

Example 3.12 (Example query)**.** *Consider q the BGPQ asking* who is writing which kind of articles *defined by:*

$$q(x, y) \leftarrow (z, \text{:author}, x), (z, \tau, y), (y, \prec_{sc}, \text{:Article}).$$

Its evaluation on G_{ex} is empty. However, the answer set of q on G_{ex} w.r.t. $\mathcal{R}_{\text{RDFS}}$ is:

$$\begin{aligned} q(G_{\text{ex}}, \mathcal{R}_{\text{RDFS}}) = \; & \{\langle \text{:Alice}, \text{:GOpenArt}\rangle, \\ & \langle \text{:Alice}, \text{:OpenArt}\rangle, \\ & \langle \text{:Bob}, \text{:GOpenArt}\rangle, \\ & \langle \text{:Bob}, \text{:OpenArt}\rangle\} \end{aligned}$$

Its reformulation w.r.t. $\mathcal{R}_{\text{RDFS}}$ is:

$$\begin{aligned} Q_{c,a} = \; & q(x, \text{:OpenArt}) \leftarrow (z, \text{:author}, x), (z, \tau, \text{:OpenArt}) \\ & \cup\, q(x, \text{:OpenArt}) \leftarrow (z, \text{:firstAuth}, x), (z, \tau, \text{:OpenArt}) \\ & \cup\, q(x, \text{:OpenArt}) \leftarrow (z, \text{:author}, x), (z, \tau, \text{:GOpenArt}) \\ & \cup\, q(x, \text{:OpenArt}) \leftarrow (z, \text{:firstAuth}, x), (z, \tau, \text{:GOpenArt}) \\ & \cup\, q(x, \text{:GOpenArt}) \leftarrow (z, \text{:author}, x), (z, \tau, \text{:GOpenArt}) \\ & \cup\, q(x, \text{:GOpenArt}) \leftarrow (z, \text{:firstAuth}, x), (z, \tau, \text{:GOpenArt}) \end{aligned}$$

It can be checked that $Q_{c,a}(G_{\text{ex}}) = q(G_{\text{ex}}, \mathcal{R}_{\text{RDFS}})$.

3.3.2 BGPQ answering on the T layout

Let $t(S, P, O)$ be the table storing the triples of a graph G for the T layout.

Saturation-based query answering

The saturation $G^{\mathcal{R}_{\text{RDFS}}}$ of G is stored in the table t. The algebraic translation of a BGPQ $q(\bar{x}) \leftarrow t_1, \ldots, t_n$ on the T layout is:

$$T(q) = \pi_q(\bowtie_{jcond} (\alpha_T(t_1), \ldots, \alpha_T(t_n))$$

where α_T, the *query triple translation for the T layout*, translates the i-th query triple $t_i(s_i, p_i, o_i)$ into an algebraic expression of the form $\sigma_{scond}(t)$, where t is the triple table, and *scond* is a (possibly empty) set of selections over the attributes of t; specifically, if s_i (respectively, p_i, o_i) is an IRI or a literal, *scond* contains a predicate of the form $S = s_i$ (and similarly for p_i, o_i); *jcond* is a conjunction of join predicates containing, for every variable appearing in several positions (in one or several triples) in q, an equality between the respective attributes in the $\alpha_T(t_i)$ triple translations; finally π_q is a projection on the attributes from the $\alpha(t_i)$'s corresponding to the answer variables of q, or the values to which such variables are bound in case of a partially instantiated query.

Example 3.13. *The example query translates on the T layout as:*

$$\pi_{t_1.O,t_2.O}(\bowtie_{t_1.S=t_2.S,t_2.O=t_3.S} (\sigma_{P=:\text{author}}(t), \sigma_{P=\tau}(t), \sigma_{P=\prec_{sc}\wedge O=:\text{Article}}(t)))$$

In the above, the selection $\sigma_{P=:\text{author}}(t)$ *restricts the triples from the t table to those having the attribute P equal to* :author. *Similarly,* $\sigma_{P=\tau}(t)$ *corresponds to the selection* $P = \tau$. *On the atom* t_3, α_T *applies a double selection* $\sigma_{P=\prec_{sc}\wedge O=:\text{Article}}(t)$, *since* t_3 *has only one variable in position* S. *Further,* $\bowtie_{jcond}=\bowtie_{t_1.S=t_2.S,t_2.O=t_3.S}$ joins *the three previous selections, where* $t_1.S = t_2.S$ *and* $t_2.O = t_3.S$ *respectively reflect the co-occurrences of variables z and y. The final* projection $\pi_{t_1.O,t_2.O}$ *returns the pairs of values obtained for* (x, y).

Reformulation-based query answering

Here, the graph G is stored in t (but not its saturation), and every incoming BGPQ q is reformulated into a (partially instantiated) UBGPQ $Q_{c,a} = \bigcup_{i=1}^{n} q_i$, whose algebraic translation on the T layout is the union of the algebraic translations of its (partially instantiated) BGPQs:

$$T(Q_{c,a}) = \bigcup_{i=1}^{n} T(q_i)$$

Example 3.14. *Consider again the example query q and its reformulation* $Q_{c,a}$ *shown in Example 3.12. The algebraic translation* $T(Q_{c,a})$ *is:*

$$\begin{aligned}
&\pi_{t_1.O,:\text{OpenArt}}(\bowtie_{t_1.S=t_2.S} (\sigma_{P=:\text{author}}(t), \sigma_{P=\tau\wedge O=:\text{OpenArt}}(t))\\
&\cup \pi_{t_1.O,:\text{OpenArt}}(\bowtie_{t_1.S=t_2.S} (\sigma_{P=:\text{firstAuth}}(t), \sigma_{P=\tau\wedge O=:\text{OpenArt}}(t))\\
&\cup \pi_{t_1.O,:\text{OpenArt}}(\bowtie_{t_1.S=t_2.S} (\sigma_{P=:\text{author}}(t), \sigma_{P=\tau\wedge O=:\text{GOpenArt}}(t))\\
&\cup \pi_{t_1.O,:\text{OpenArt}}(\bowtie_{t_1.S=t_2.S} (\sigma_{P=:\text{firstAuth}}(t), \sigma_{P=\tau\wedge O=:\text{GOpenArt}}(t))\\
&\cup \pi_{t_1.O,:\text{GOpenArt}}(\bowtie_{t_1.S=t_2.S} (\sigma_{P=:\text{author}}(t)), \sigma_{P=\tau\wedge O=:\text{GOpenArt}}(t))\\
&\cup \pi_{t_1.O,:\text{GOpenArt}}(\bowtie_{t_1.S=t_2.S} (\sigma_{P=:\text{firstAuth}}(t), \sigma_{P=\tau\wedge O=:\text{GOpenArt}}(t))
\end{aligned}$$

3.3.3 BGPQ answering on the CP layout

With the CP layout, an RDF graph is stored as a set of tables corresponding to classes and properties: for each class c, there is a table $t_c(S)$ that stores all subjects s of triples (s, τ, c), and for each data or schema property $p \neq \tau$, there is a table $t_p(S, O)$ that stores all subject-object pairs (s, o) for triples (s, p, o). We call any such t_c a *class table*, and t_p a *property table*. The CP layout speeds up data access for queries which *specify the class in every triple whose property is* τ and *specify the property in every triple*. However, as noted e.g., in [Sidirourgos et al., 2008], it may render the evaluation of general queries, with variables in class or property position, inefficient, as the triples they refer to may be in *any* t_c or t_p tables, respectively.

Saturation-based query answering

Assume that $G^{\mathcal{R}_{\text{RDFS}}}$ is stored using the CP layout. To obtain the answers to a BGPQ $q(\bar{x}) \leftarrow t_1, \ldots, t_n$, a first simple *naïve translation*, which can be traced back to [Abadi et al., 2007, Sidirourgos et al., 2008], is:

$$CP(q) = \pi_q(\bowtie_{jcond} (\alpha_{CP}(t_1), \cdots, \alpha_{CP}(t_n)))$$

where π_q and *jcond* are defined as for the T layout, and α_{CP}, the *query triple translation for the CP layout*, is:

$$\alpha_{CP}(t) = \begin{cases} \pi_{S,\tau,c}(\sigma_{scond}(t_c)) \text{ if } t = (s, \tau, c) \text{ with } c \notin \mathcal{V} & (1) \\ \pi_{S,p,O}(\sigma_{scond}(t_p)) \text{ if } t = (s, p, o) \text{ with } p \notin \mathcal{V} \cup \{\tau\} & (2) \\ \bigcup_{c \in C} \alpha_{CP}((s, \tau, c)) \text{ if } t = (s, \tau, x) \text{ with } x \in \mathcal{V} & (3) \\ \alpha_{CP}((s, \tau, o)) \cup \bigcup_{p \in \mathcal{P}} \alpha_{CP}((s, p, o)) \text{ if } t = (s, x, o) \text{ with } x \in \mathcal{V} & (4) \end{cases}$$

where C and $\mathcal{P}$ are, respectively, the set of classes and of properties other than τ in the queried graph, and *scond* is a (possibly empty) *conjunction* of selections, just as we defined it for α_T, but over the class and property tables instead of the triple table t.

Example 3.15. *The naive translation of the example query q on the CP layout is:*

$$\pi_{t_1.O,t_2.O}(\bowtie_{t_1.S=t_2.S,t_2.O=t_3.S} (\pi_{S,:\text{author},O}(t_{:\text{author}}), \\ \pi_{s,\tau,:\text{GOpenArt}}(t_{:\text{GOpenArt}}) \cup \pi_{s,\tau,:\text{OpenArt}}(t_{:\text{OpenArt}}) \cup \pi_{s,\tau,:\text{Article}}(t_{:\text{Article}}) \\ \cup \pi_{s,\tau,:\text{Prof}}(t_{:\text{Prof}}) \cup \pi_{s,\tau,:\text{Person}}(t_{:\text{Person}}) \cup \pi_{S,\prec_{sc},O}(\sigma_{O=:\text{Article}}(t_{\prec_{sc}})))$$

Note that in cases (3) and (4) above, α_{CP} introduces *unions under joins*, as illustrated by the previous example. This leads to **suboptimal evaluation performance** in many data management engines, which *may optimize and execute efficiently a join over several data collections, but do not attempt to reorder (commute) joins with unions*. For instance, the query $(x, :\text{a}, :\text{a}_1)$, (x, y, z), (z, τ, u), $(z, :\text{b}, :\text{b}_1)$ translates into a plan that joins (among others) the union of all the tables (for (x, y, z)) with the union of all class tables (for (z, τ, u)). Most systems execute this "literally", i.e., they build and materialize these two very large unions, which is very costly, before joining them with the first and last triple[7].

To avoid such unions under joins, we rely on the notion of instantiation, which has been used in various query answering techniques e.g., [Goasdoué et al., 2011] and [Kontchakov et al., 2014].

Query instantiation

The instantiation of a BGPQ q consists in instantiating the variables in q that must be bound to classes and properties of the queried graph, in all possible ways, which yields a (partially instantiated) UBGPQ. Given a BGPQ q and a graph G, we denote by $q^{p,G}$ (resp. $q^{c,G}$) its *property instantiation* (resp. *class instantiation*), which is the UBGPQ obtained by instantiating all its variables in property position (resp. in class position), by all combinations of properties (resp. classes) of G.

Example 3.16. *The class instantiation $q^{c,G_{ex}}$ of the example query q, where the only instantiated variable is y, is:*

$q(x, :\text{GOpenArt}) \leftarrow (z, :\text{author}, x), (z, \tau, :\text{GOpenArt}), (:\text{GOpenArt}, \prec_{sc}, :\text{Article})$
$\cup\, q(x, :\text{OpenArt}) \leftarrow (z, :\text{author}, x), (z, \tau, :\text{OpenArt}), (:\text{OpenArt}, \prec_{sc}, :\text{Article})$
$\cup\, q(x, :\text{Article}) \leftarrow (z, :\text{author}, x), (z, \tau, :\text{Article}), (:\text{Article}, \prec_{sc}, :\text{Article})$
$\cup\, q(x, :\text{Prof}) \leftarrow (z, :\text{author}, x), (z, \tau, :\text{Prof}), (:\text{Prof}, \prec_{sc}, :\text{Article})$
$\cup\, q(x, :\text{Person}) \leftarrow (z, :\text{author}, x), (z, \tau, :\text{Person}), (:\text{Person}, \prec_{sc}, :\text{Article})$

[7] We checked this on systems that disclose their query execution strategy; experiments with others who do not, confirm the same hypothesis (see Section 3.3.6).

Class and property instantiations extend from BGPQs to UBGPQs in the natural way. Given a UBGPQ of the form $Q = q^1 \cup q^2 \cdots \cup q^n$, we set:

$$Q^{p,G} = q_1^{p,G} \cup q_2^{p,G} \cdots \cup q_n^{p,G} \text{ and } Q^{c,G} = q_1^{c,G} \cup q_2^{c,G} \cdots \cup q_n^{c,G}$$

Then, the *instantiation* of Q w.r.t. a graph G is the following:

$$Q^G = (Q^{c,G})^{p,G} \cup (Q^{p,G})^{c,G}$$

We need both terms of the above union, *exactly* in the case when some variable of Q appears both in a property and in a class position. These cases are rare and easy to detect, thus in practice we only use one of the unions as soon as we detect both are not needed. Crucially, (U)BGPQ instantiation is *correct* for saturation- and reformulation-based query answering. Intuitively, this is because instantiation enumerates all possible combinations of classes and properties that query reformulation or evaluation may find in G.

We can now define the *instantiation-based query translation*. A BGPQ q is first instantiated into $q^G = \bigcup_{i=1}^{n} q^i$, then translated on the CP layout as:

$$CP(q^G) = \bigcup_{i=1}^{n} CP(q^i)$$

Importantly, because q^G does not contain any variable in class or property position, every naïve translation $CP(q^i)$ within $CP(q^G)$ avoids both (3) or (4) in the α_{CP} triple transformation function. Hence, the translation avoids the introduction of unions under joins, with their potential bad impact on performance.

Example 3.17. *Consider the query q and its instantiation $q^{G_{ex}} = q^{c,G_{ex}}$ in Example 3.16. The instantiation-based translation of q corresponds to the naïve translation of $q^{G_{ex}}$:*

$$\bigcup_{\text{u}\in\{\text{:GOpenArt,:OpenArt,:Article,:Prof,:Person}\}} \pi_{t_1.O,\text{u}}(\bowtie_{t_1.S=t_2.S,t_2.O=t_3.S} (\pi_{S,\text{:author},O}(t_{\text{:author}}), \pi_{S,\tau,\text{u}}(t_{\text{u}}), \pi_{S,\prec_{sc},O}(\sigma_{S=\text{u},O=\text{:Article}}(t_{\prec_{sc}}))$$

Reformulation-based query answering

The graph G is again stored using the CP layout (but not saturated). In this case, answering a BGPQ q starts by computing its reformulation $Q_{c,a}$ w.r.t. the ontology of G. Then, we obtain the answers $q(G, \mathcal{R}_{\text{RDFS}})$ either through $CP(Q_{c,a})$, the naïve translation of $Q_{c,a}$, or through $CP(Q_{c,a}^G)$, the instantiation-based translation of $Q_{c,a}$, i.e., the naïve translation of its instantiation $Q_{c,a}^G$; as in the previous section, the algebraic translation of a UBGPQ is defined as the union of the algebraic translations of its BGPQs. Instantiating $Q_{c,a}$ generally increases its size, but, by removing variables in class and property positions, it avoids the unions under joins introduced in cases (3) and (4) by the α_{CP} triple translation function.

Example 3.18. *Consider the query q and its reformulation $Q_{c,a}$ from Example 3.12. Here, since no variable of $Q_{c,a}$ occurs in class or property position, $CP(Q_{c,a})$ and $CP(Q_{c,a}^G)$ lead to the same algebraic expression:*

$$\pi_{t_1.O,:\text{OpenArt}}(\bowtie_{t_1.S=t_2.S} (\pi_{S,:\text{author},O}(t_{:\text{author}}), \pi_{S,\tau,:\text{OpenArt}}(t_{:\text{OpenArt}}))$$
$$\cup \pi_{t_1.O,:\text{OpenArt}}(\bowtie_{t_1.S=t_2.S} (\pi_{S,:\text{firstAuth},O}(t_{:\text{firstAuth}}), \pi_{S,\tau,:\text{OpenArt}}(t_{:\text{OpenArt}}))$$
$$\cup \pi_{t_1.O,:\text{OpenArt}}(\bowtie_{t_1.S=t_2.S} (\pi_{S,:\text{author},O}(t_{:\text{author}}), \pi_{S,\tau,:\text{GOpenArt}}(t_{:\text{GOpenArt}}))$$
$$\cup \pi_{t_1.O,:\text{OpenArt}}(\bowtie_{t_1.S=t_2.S} (\pi_{S,:\text{firstAuth},O}(t_{:\text{firstAuth}}), \pi_{S,\tau,:\text{GOpenArt}}(t_{:\text{GOpenArt}}))$$
$$\cup \pi_{t_1.O,:\text{GOpenArt}}(\bowtie_{t_1.S=t_2.S} (\pi_{S,:\text{author},O}(t_{:\text{author}}), \pi_{S,\tau,:\text{GOpenArt}}(t_{:\text{GOpenArt}}))$$
$$\cup \pi_{t_1.O,:\text{GOpenArt}}(\bowtie_{t_1.S=t_2.S} (\pi_{S,:\text{firstAuth},O}(t_{:\text{firstAuth}}), \pi_{S,\tau,:\text{GOpenArt}}(t_{:\text{GOpenArt}}))$$

3.3.4 BGPQ answering based on the TCP layout

The TCP layout combines T and CP with the aim of getting the best of both, while avoiding the performance problems they respectively entail (Sections 3.3.2 and 3.3.3). Here, an RDF graph is stored *both* in the triple table t of the T layout *and* in the t_c class and t_p property tables of the CP layout. The rationale behind this is that CP is efficient to access triples when *the data structures holding the triples we need to access are immediately clear from the query, and small*; this is the case with query triples of the form (s, τ, c) or (s, p, o) for a known class c or property p. However, with query triples of the form (s, τ, x) and (s, x, o), the CP translation introduces unions, typically executed before joins, degrading performance. Interestingly, *direct access to a potentially large share of a graph's triples is exactly what the T layout supports well* - thus our idea to combine them. As we show in the next section, this allows to significantly improve performance, at expense of some extra storage space, typically inexpensive since it is on disk.

Saturation-based query answering

Let us assume that the saturation $G^{\mathcal{R}_{\text{RDFS}}}$ of a graph G is stored in the TCP layout. The answers to a BGPQ $q \leftarrow t_1, \ldots, t_n$ are obtained through its algebraic transformation for the TCP layout:

$$TCP(q) = \pi_q(\bowtie_{jcond} (\alpha_{TCP}(t_1), \cdots, \alpha_{TCP}(t_n)))$$

where π_q and *jcond* are defined as for the T and CP layouts, and α_{TCP}, the *query triple translation for the TCP layout*, is:

$$\alpha_{TCP}(t) = \begin{cases} \alpha_{CP}(t) \text{ if } t = (s, \tau, c) \text{ or } t = (s, p, o) \text{ with } c \notin \mathcal{V}, p \notin \mathcal{V} \cup \{\tau\} \\ \alpha_T(t) \text{ otherwise, i.e., if } t = (s, \tau, x) \text{ or } t = (s, x, o) \text{ with } x \in \mathcal{V} \end{cases}$$

Importantly, α_{TCP} translates the triples that penalize the CP layout into t atoms, and never into a union: hence, α_{TCP} avoids the cases (3) and (4) of α_{CP}.

Example 3.19. *The translation of the example query for the TCP layout combines the T layout for the second triple and the CP layout for the others:*

$$\pi_{t_1.O,t_2.O}(\bowtie_{t_1.S=t_2.S,t_2.O=t_3.S} (\pi_{S,:\text{author},O}(t_{:\text{author}}), \sigma_{P=\tau}(t), \pi_{S,\prec_{sc},O}(\sigma_{O=:\text{Article}}(t_{\prec_{sc}}))))$$

Reformulation-based query answering

Again, the answers to a query q are computed by evaluating the algebraic translation of its reformulation $Q_{c,a} = \bigcup_{i=1}^{n} q^i$, but now for the TCP layout:

$$TCP(Q_{c,a}) = \bigcup_{i=1}^{n} TCP(q^i)$$

Example 3.20. *Consider the query* $q(x, y, z) \leftarrow (x, \tau, z), (x, :\text{firstAuth}, y)$ *asking for all resources with their type and first author. Its reformulation w.r.t.* G_{ex}*'s ontology is shown below :*

$$
\begin{aligned}
& q(x, y, z) \leftarrow (x, \tau, z), (x, :\text{firstAuth}, y) \\
& \cup\, q(x, y, :\text{Article}) \leftarrow (x, \tau, :\text{OpenArt}), (x, :\text{firstAuth}, y) \\
& \cup\, q(x, y, :\text{Article}) \leftarrow (x, \tau, :\text{GOpenArt}), (x, :\text{firstAuth}, y) \\
& \cup\, q(x, y, :\text{OpenArt}) \leftarrow (x, \tau, :\text{GOpenArt}), (x, :\text{firstAuth}, y) \\
Q_{c,a} = \; & \cup\, q(x, y, :\text{Person}) \leftarrow (x, \tau, :\text{Prof}), (x, :\text{firstAuth}, y) \\
& \cup\, q(x, y, :\text{Person}) \leftarrow (x, :\text{author}, u), (x, :\text{firstAuth}, y) \\
& \cup\, q(x, y, :\text{Person}) \leftarrow (x, :\text{firstAuth}, u), (x, :\text{firstAuth}, y) \\
& \cup\, q(x, y, :\text{Prof}) \leftarrow (x, :\text{teaches}, u), (x, :\text{firstAuth}, y) \\
& \cup\, q(x, y, :\text{Person}) \leftarrow (x, :\text{teaches}, u), (x, :\text{firstAuth}, y)
\end{aligned}
$$

Its algebraic translation on the TCP layout is:

$$
\begin{aligned}
& \pi_{t_1.S,t_2.O,t_1.O}(\bowtie_{t_1.S=t_2.S} (\sigma_{P=\tau}(t), \pi_{S,:\text{firstAuth},O}(t_{:\text{firstAuth}}))) \\
& \cup\, \pi_{t_1.S,t_2.O,:\text{Article}}(\bowtie_{t_1.S=t_2.S} (\pi_{S,\tau,:\text{OpenArt}}(t_{:\text{OpenArt}}), \pi_{S,:\text{firstAuth},O}(t_{:\text{firstAuth}}))) \\
& \cup\, \pi_{t_1.S,t_2.O,:\text{Article}}(\bowtie_{t_1.S=t_2.S} (\pi_{S,\tau,:\text{GOpenArt}}(t_{:\text{GOpenArt}}), \pi_{S,:\text{firstAuth},O}(t_{:\text{firstAuth}}))) \\
& \cup\, \pi_{t_1.S,t_2.O,:\text{OpenArt}}(\bowtie_{t_1.S=t_2.S} (\pi_{S,\tau,:\text{GOpenArt}}(t_{:\text{GOpenArt}}), \pi_{S,:\text{firstAuth},O}(t_{:\text{firstAuth}}))) \\
& \cup\, \pi_{t_1.S,t_2.O,:\text{Person}}(\bowtie_{t_1.S=t_2.S} (\pi_{S,\tau,:\text{Prof}}(t_{:\text{Prof}}), \pi_{S,:\text{firstAuth},O}(t_{:\text{firstAuth}}))) \\
& \cup\, \pi_{t_1.S,t_2.O,:\text{Person}}(\bowtie_{t_1.S=t_2.S} (\pi_{S,:\text{author},O}(t_{:\text{author}}), \pi_{S,:\text{firstAuth},O}(t_{:\text{firstAuth}}))) \\
& \cup\, \pi_{t_1.S,t_2.O,:\text{Person}}(\bowtie_{t_1.S=t_2.S} (\pi_{S,:\text{firstAuth},O}(t_{:\text{firstAuth}}), \pi_{S,:\text{firstAuth},O}(t_{:\text{firstAuth}}))) \\
& \cup\, \pi_{t_1.S,t_2.O,:\text{Prof}}(\bowtie_{t_1.S=t_2.S} (\pi_{S,:\text{teaches},O}(t_{:\text{teaches}}), \pi_{S,:\text{firstAuth},O}(t_{:\text{firstAuth}}))) \\
& \cup\, \pi_{t_1.S,t_2.O,:\text{Person}}(\bowtie_{t_1.S=t_2.S} (\pi_{S,:\text{teaches},O}(t_{:\text{teaches}}), \pi_{S,:\text{firstAuth},O}(t_{:\text{firstAuth}})))
\end{aligned}
$$

Above, the first union term refers to the triple table t, while the others do not.

3.3.5 Summary-based query pruning

We now introduce an optimization technique, which can be applied on *any* storage layout to reduce (U)BGPQ answering time. It allows detecting some BGPQs with an empty answer set on a graph, without evaluating them, by using a (typically much smaller) structural summary of this graph.

Given an RDF graph G and an *equivalence relation* $\equiv$ among the nodes in G, i.e., the subjects and objects of triples, an *RDF quotient summary* [Cebiric et al., 2018] is an RDF graph $G_{/\equiv}$ built as follows. A node is created in $G_{/\equiv}$ for each equivalence class among G's nodes; further, for any triple $(n_1, p, n_2) \in G$, the triple (m_1, p, m_2) appears in $G_{/\equiv}$, where m_1 and m_2 represent the equivalence class of n_1 and n_2 respectively. If there are large equivalence classes in G, summarization is a form of compression. Several types of RDF quotient summaries have been proposed [Cebiric et al., 2018]; in our experiments, we used the RDFQuotient summary construction tool [Goasdoué et al., 2020], due to its online availability and low summary construction cost (linear in the number of triples of G). An RDFQuotient summary represents each class and property node *by itself*, and consider they are not equivalent to any other G node; thus, G and any quotient summary $G_{/\equiv}$ have the *same* schema triples.

Crucially, the following property holds for q a *structural* (U)BGPQ, i.e., in which the subjects and objects of query triples are either class and property IRIs, or variables:

$$\text{if } q(G_{/\equiv}) = \emptyset \text{ then } q(G) = \emptyset$$

Intuitively, this result holds because structural queries only allow selecting subject, property and object values that are preserved through summarization (class and property IRIs). Note however that the opposite does not hold in general, i.e., $q(G_{/\equiv})$ may have results while $q(G)$ does not. We exploit this result by defining the *structural version* of a BGPQ q, denoted q^{str}, which is obtained by replacing in q the literals and the IRIs that are not class or property IRIs, by fresh variables. For example, the structural version of the query $q(x) \leftarrow (x, \tau, :\text{OpenArt}), (x, :\text{firstAuth}, :\text{Alice})$ is: $q^{\text{str}}(x) \leftarrow (x, \tau, :\text{OpenArt}), (x, :\text{firstAuth}, y)$, with :Alice replaced by y. Hence, when a summary $G_{/\equiv}$ is available, we can use it to *prune* a UBGPQ $Q = \bigcup_i q_i$ by removing from the union all the q_i terms for which $q_i^{\text{str}}(G_{/\equiv}) = \emptyset$. As explained above, this may fail to prune some q_i with no results on G, but it preserves query results:

$$Q(G) = Q^{\text{pruned}}(G)$$

where Q^{pruned} is the result of pruning Q. As our experiments will show, this generally leads to a significant reduction of query answering time.

3.3.6 Experimental evaluation

We now describe experiments comparing the query answering methods presented in the previous sections, on the T, CP and TCP layouts.

Experimental settings

We implemented the T, CP and TCP layouts in **OntoSQL** (`https://ontosql.inria.fr`), a Java platform providing efficient RDF storage and saturation- and reformulation-based query answering on top of an RDBMS [Buron et al., 2019, Bursztyn et al., 2015, Goasdoué et al., 2013] (Postgres 9.6 in these experiments). OntoSQL encodes IRIs and literals as integers, and a dictionary table allows going from one to the other; each table (t, t_p or t_c) is indexed on all the subsets of its attributes. To use OntoSQL, we *express our algebraic translations in SQL*. We checked that in Postgres query plans, the relative positions of unions and joins in the query plans chosen by the RDBMS are those from our translations; [Bursztyn et al., 2015, Bursztyn et al., 2016] showed that this holds for two other major RDBMSs. However, *the RDBMS takes all optimization decisions*, based on its cost model and statistics. To put this into perspective also with respect to native RDF engines, we ran the same experiments also on **Virtuoso** Open Source Edition 7.2, to whom we provided *SPARQL queries*, which correspond exactly to our algebraic translation on the T layout. Virtuoso also controls its optimization decisions, and has full control over its store.

For summary-based pruning, we used the **RDFQuotient** (`https://rdfquotient.inria.fr`) tool to build the "typed strong" summary [Goasdoué et al., 2020] of a graph G; this summary is denoted $G_{/\text{TS}}$. The summary groups typed nodes according to their types, and untyped nodes by exploiting the similarity of their incoming/outgoing properties (see [Goasdoué et al., 2020] for details). In general, any quotient summary could be used; a large (more detailed) summary makes pruning more accurate, but also slower since it needs to query the summary.

Hardware We used a server with 2,7 GHz Intel Core i7 processors and 160 GB of RAM, running CentOs Linux 7.5.

Graph	$\lvert G\rvert$	$\lvert G_{/\text{TS}}\rvert$	$\lvert G^{\mathcal{R}_{\text{RDFS}}}\rvert$
LUBM	100M	340	131M
DBLP	88M	290	147M

Graph	$\lvert(G^{\mathcal{R}_{\text{RDFS}}})_{/\text{TS}}\rvert$	$\lvert G\rvert_{\text{T}}$	$\lvert G\rvert_{\text{CP}}$	$\lvert G\rvert_{\text{TCP}}$	$\lvert G^{\mathcal{R}_{\text{RDFS}}}\rvert_{\text{T}}$	$\lvert G^{\mathcal{R}_{\text{RDFS}}}\rvert_{\text{CP}}$	$\lvert G^{\mathcal{R}_{\text{RDFS}}}\rvert_{\text{TCP}}$
LUBM	439	28.95	11.95	37.89	32.52	13.52	39.31
DBLP	708	26.07	16.35	35.89	38.39	22.91	52.72

Table 3.2: Graph and summary sizes (number of triples), OntoSQL database sizes (in GB), including all indexes, for the T, CP and TCP layouts.

RDF graphs We used two benchmark graphs: a **LUBM** [Guo et al., 2005] graph of 100M triples, as well as a graph of **DBLP** bibliographic data endowed with an ontology of 14 classes and 44 schema triples. Table 3.2 shows, for these graphs and their saturation, the graph and summary sizes, and the sizes of the OntoSQL databases storing them in the T, CP and TCP layout. As expected, TCP takes most space, approx. 90% of the sum of the T and CP database sizes. However, this is stable storage (e.g., disk) space; the selective data access enabled by the class and property tables, and by indexes, as well as cost-based optimization, ensure that the data loaded in memory to process a query is much smaller.

Queries We used two **diverse** sets of queries, having from 1 to 11 triples (4 on average) on LUBM, and from 2 to 9 triples (5.9 on average) on DBLP. Each query has 1 or 2 triple(s) of the form (s, τ, x) and/or (s, x, o), except Q11 and Q15 on DBLP which do not contain any. Table 3.7 shows their number of answers, and the number of BGPQs in: their instantiation (q^G), reformulation ($Q_{c,a}$), and instantiation of their reformulation ($Q^G_{c,a}$), before and after summary-based pruning. The impact of pruning ranges from none (in particular for q^G, on LUBM Q01 to Q03 and on 8 DBLP queries) to very significant (97.8% of the q^G BGPs are pruned on LUBM Q09). Details on our experiments, and code which can be used to reproduce them, are in the appendix, Section A.2.

Experiment results: query answering times

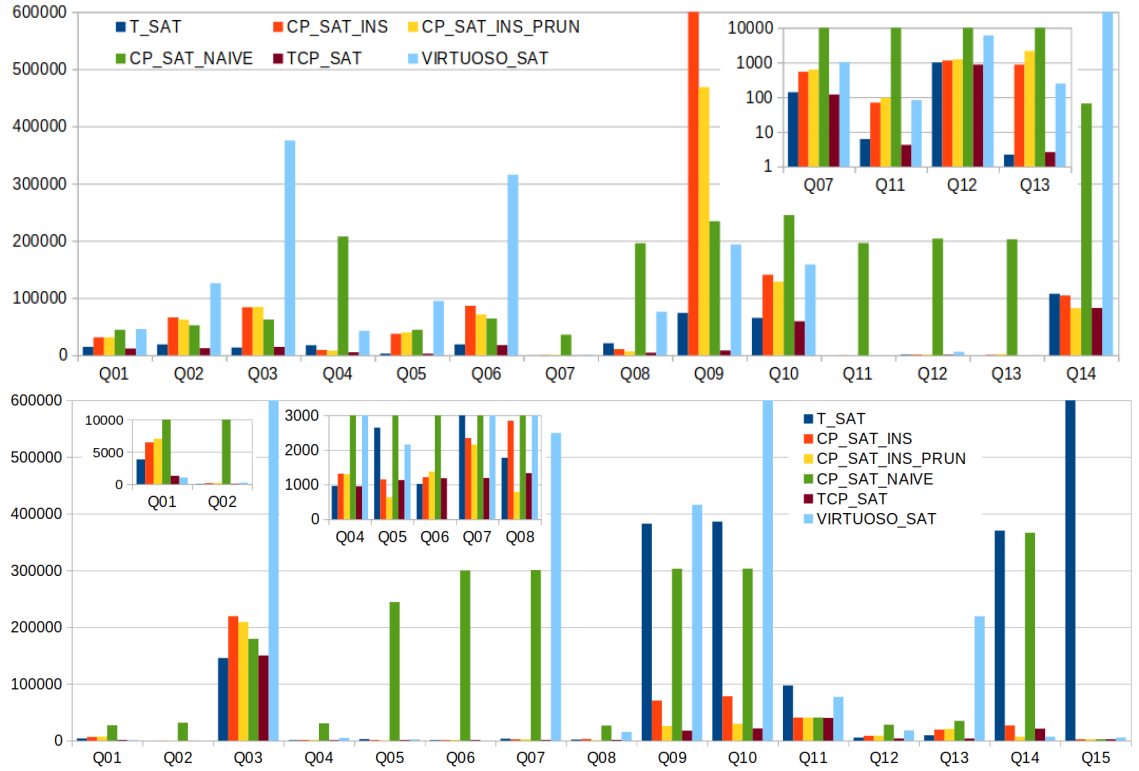
For each query, we report the average of the last five ("hot") runs out of six.

Through saturation Figure 3.8 shows the query answering times through saturation, for LUBM (top) and DBLP (bottom), with a timeout of 10 minutes; in all our graphs, executions that reached the timeout have been interrupted. Below the graphs, we show the label used in the plot for each query answering strategy, e.g., T_SAT stands for $T(q)(G^{\mathcal{R}_{\text{RDFS}}})$. For readability, some very fast queries are repeated in a "zoom" plot (the LUBM one has a logarithmic y axis).

On **LUBM** (top), we notice some very high running times for VIRTUOSO_SAT, e.g., on Q03, Q06, and a time-out on Q14. Among the SQL-based strategies, the naïve translation on CP (green bars) is slowest in 10 out of 14 queries, with large performance penalties for Q04, Q07, Q08, Q11-13. Instantiation (CP_SAT_INS, red bars) is generally faster than naïve CP. It strongly speeds up Q04, Q07, Q08, Q10-Q14; it brings a modest improvement to Q01 and Q05, but also a modest overhead for Q02, Q03, Q06. However, on the complex Q09, which has the largest q^G size, namely 3690, instantiating each of these more than doubles the answering time w.r.t. naïve CP translation (and ran until

Query	**Q01**	**Q02**	**Q03**	**Q04**	**Q05**	**Q06**	**Q07**	**Q08**	**Q09**	**Q10**	**Q11**	**Q12**	**Q13**	**Q14**
$\lvert q(G,\mathcal{R}_{\text{RDFS}})\rvert$	2.78M	0.59M	2.15M	1.72M	0.47M	2.77M	25K	3696	2003	17.35M	187	518K	857	9.85M
$\lvert q^G\rvert$	45	45	45	82	2025	45	45	82	3690	82	82	37	1369	82
$\lvert (q^G)^{\text{pruned}}\rvert$	45	45	45	67	1806	44	44	3	88	68	59	23	43	51
$\lvert Q_{c,a}\rvert$	318	106	146	68	216	80	318	9	36	88	9	27	39	1152
$\lvert (Q_{c,a})^{\text{pruned}}\rvert$	120	56	59	48	108	34	173	7	2	78	7	21	21	889
$\lvert Q^G_{c,a}\rvert$	447	149	226	68	216	121	447	9	36	169	9	63	1797	1188
$\lvert (Q^G_{c,a})^{\text{pruned}}\rvert$	206	99	135	48	108	73	299	7	2	144	7	42	50	892
Query	**Q01**	**Q02**	**Q03**	**Q04**	**Q05**	**Q06**	**Q07**	**Q08**	**Q09**	**Q10**	**Q11**	**Q12**	**Q13**	**Q14**	**Q15**
$\lvert q(G,\mathcal{R}_{\text{RDFS}})\rvert$	72K	24K	96.3M	361K	4K	10	138	42K	1.52M	3.09M	957K	414K	409K	16.3K	460K
$\lvert q^G\rvert$	18	18	18	18	50	900	900	18	900	900	1	18	324	50	1
$\lvert (q^G)^{\text{pruned}}\rvert$	18	18	18	18	3	136	136	17	85	85	1	18	324	3	1
$\lvert Q_{c,a}\rvert$	117	297	265	117	873	4257	3163	36	1500	3652	243	39	381	129	243
$\lvert (Q_{c,a})^{\text{pruned}}\rvert$	56	205	151	56	616	3089	2166	32	1184	2620	36	24	174	114	3
$\lvert Q^G_{c,a}\rvert$	252	440	408	252	1529	5927	3345	72	1	324	3	84	2088	270	243
$\lvert (Q^G_{c,a})^{\text{pruned}}\rvert$	161	331	277	161	1224	3275	2189	37	1	64	3	69	1497	117	3

Figure 3.7: Statistics of our queries on LUBM (top) and DBLP (bottom); M stands for millions and K for thousands.

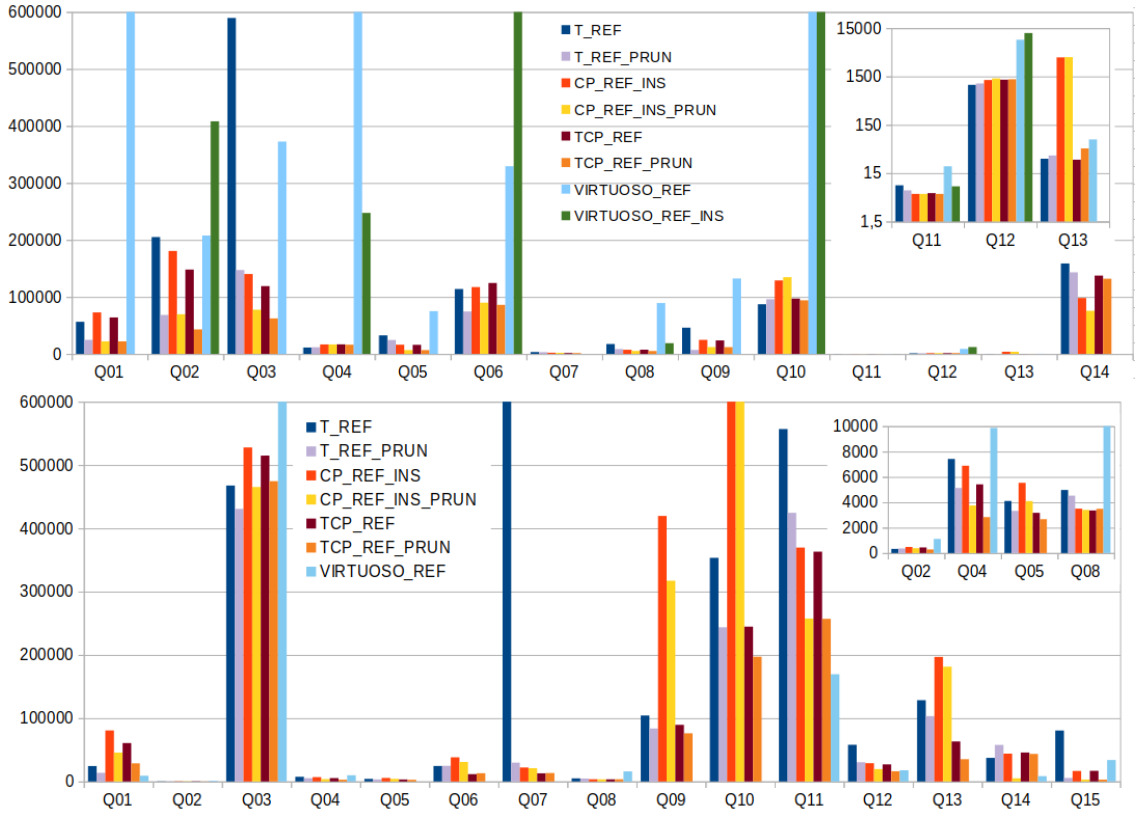


T_SAT	CP_SAT_NAIVE	CP_SAT_INS	CP_SAT_INS_PRUN	TCP_SAT	VIRTUOSO_SAT
$T(q)(G^{\mathcal{R}_{\text{RDFS}}})$	$CP(q)(G^{\mathcal{R}_{\text{RDFS}}})$	$CP(q^G)(G^{\mathcal{R}_{\text{RDFS}}})$	$CP((q^G)^{\text{pruned}})(G^{\mathcal{R}_{\text{RDFS}}})$	$TCP(q)(G^{\mathcal{R}_{\text{RDFS}}})$	$T(q)(G^{\mathcal{R}_{\text{RDFS}}})$

Figure 3.8: Query answering times (milliseconds) on LUBM (top) and DBLP (bottom), through saturation.

the timeout); pruning (yellow bars) brings it back below the timeout. T_SAT is generally faster than all executions on the CP layout, because all queries contain triples of the form (s, τ, x) and/or (s, x, o), which, as explained in Section 3.3.3, challenge CP execution. *TCP_SAT avoids the (sometimes drastic) performance problems of all CP variants, and is the fastest (by up to several orders of magnitude) on all queries but Q14*, where the CP variant with pruning is a bit faster. Virtuoso is also always slower than TCP (by up to 95×, almost two orders of magnitude).

On **DBLP** (bottom), poor performance is exhibited by Virtuoso (Q03, Q07, Q09, Q10), and on even more queries by the naïve CP strategy (green bars, Q05-Q07, Q09-Q10, Q14). T_SAT performs very badly on Q09, Q10, Q14 and Q15. These are rather large (6 to 9-triples) queries; an analysis of their plans shows significant errors in the RDBMS' estimation of join cardinality. As is well-known, join cardinality estimation errors multiply along subsequent joins; when all joins carry over a single, very large table, the negative impact of such cumulated errors can be quite strong. Historically, this observation actually motivated the introduction of the CP layout, on which join estimation errors still multiply, but usually much smaller tables are involved. Indeed, as expected, for the queries Q11 and Q15, exactly those where no triple has a variable in class and property position, naïve CP largely outperforms T_SAT (by very far for Q15). Again, we observe the robust behaviour of TCP_SAT. We conclude that through saturation, *T and CP execution each underperform on some queries, but TCP avoids all these pitfalls and is consistently very efficient*.



T_REF	T_REF_PRUN	CP_REF_INS	CP_REF_INS_PRUN
$T(Q_{c,a})(G)$	$T((Q_{c,a})^{\text{pruned}})(G)$	$CP(Q^G_{c,a})(G)$	$CP((Q^G_{c,a})^{\text{pruned}})(G)$

TCP_REF	TCP_REF_PRUN	VIRTUOSO_REF	VIRTUOSO_REF_INS
$TCP(Q_{c,a})(G)$	$TCP((Q_{c,a})^{\text{pruned}})(G)$	$T(Q_{c,a})(G)$	$T(Q^G_{c,a})(G)$

Figure 3.9: LUBM (top) and DBLP (bottom) query answering times (ms) through reformulation.

Through reformulation Figure 3.9 shows reformulation-based query answering times (note the logarithmic y axis in the zoom), again with the correspondences between the bar labels and the strategy names previously used in the paper.

On **LUBM**, among the evaluation strategies *without pruning*, TCP_REF is generally the fastest (or very close to it), with the exception of Q14, where CP_REF_INS is 1.4× faster. This query with the most results (9.85M) and a large reformulation (Table 3.7) has two atoms of the form $(x, p, z), (y, p, z)$. On CP, this leads to a large number of self-joins of the form $t_p \bowtie_o t_p$, executed very fast because loading t_p in memory once ensures the join runs completely in-memory. While the rather unusual Q14 shows a case when CP may still outperform TCP, the difference is not dramatic. *On the three layouts, pruning generally helps*: it saves, e.g., more than half of the CP answering time for Q01, Q02. In the zoomed view (shortest-running queries), pruning brings an overhead (takes more time that the query evaluation time it saves) of a few milliseconds. Among the strategies *with* pruning, TCP_REF_PRUN is the fastest, except for Q14 discussed above. As Virtuoso did not support reasoning with our rule set $\mathcal{R}_{\text{RDFS}}$ (details in the appendix, Section A.2), we gave it reformulations expressed in SPARQL; for Q07 and Q14, they failed to run, with the error *"union nesting is too deep"*. The impact of instantiation for Virtuoso is unclear; it helped for Q04, Q08 but not for Q02, Q06 etc. All missing Virtuoso bars in Figure 3.9 are *execution failures*, mostly due to large unions.

On **DBLP**, VIRTUOSO_REF failed for Q06, Q07, Q09, Q10, Q13; VIRTUOSO_REF_INS

was consistently worse, and we omitted it from the plot. The rest of the analysis is similar to the one above, except that T_REF performs very badly in a few cases (Q07, Q11). Overall, in Figure 3.9, *TCP query answering with pruning is the fastest, or very close to it, on all queries*, while all other strategies's weaknesses are exposed by one or more queries.

Experiment conclusion

We studied the performance of query answering in RDF databases through saturation and reformulation, on challenging queries that remain poorly supported: those with variables in class or property position. We have exhibited queries that lead to *poor to catastrophic performance* of query answering *on the T layout (mainly due to many self-joins on a large table)* and/or *on the CP layout (mainly due to large unions, brought by variables in class and property positions, and/or by reformulation)*. Query answering on the TCP layout is extremely *robust*; it avoids all these pitfalls by taking the best of both T and CP in reformulation and saturation-based approaches, at the expense of more storage space. As disks are getting ever cheaper[8], TCP appears to be a robust, practical layout, compatible with well-established large-scale RDF storage and query engines. For the challenging queries we study, summary-based pruning helps improve performance, in particular for the TCP layout.

We believe the TCP layout, and pruning, can be adopted with little effort, and can strongly consolidate and improve query answering performance in many RDF databases.

3.4 Summary

In this chapter, we studied query answering techniques on RDF graphs with *general* BGPQs. We argued that such queries allow a user to take advantage of RDF metamodeling capabilities. We first studied properties of built-in RDFS entailment rules, which led to define two desirable properties of RDF graphs, namely $\mathcal{R}_a$*-compliance* and the more demanding *split reasoning*. We also defined a syntactic restriction on RDF graphs, namely *FO-restriction*, which ensures the split-reasoning property. This restriction still enables forms of metamodeling and we believe that it is quite natural with respect to the notion of an ontology. We then designed a *query reformulation* algorithm for RDF graphs with split reasoning. The experiments we carried out show the effectiveness of our algorithm, in particular when we take into account the number of query runs required to amortize the graph saturation cost. Query reformulation is even more competitive when the graph changes frequently. We finally investigated the impact of *storage layout* on the efficiency of query answering with general BGPQs. We introduced the novel workload-unaware *TCP layout*, which combines the data structures of two classical layouts, namely T and CP, and experimentally assessed its interest, on a relational and a native RDF database. Besides, we introduced *summary-based pruning*, an optimization technique of independent interest, and we found that it generally reduces query answering costs on the T, CP and TCP layouts, both with graph saturation and query reformulation.

[8] E.g., `https://www.backblaze.com/blog/hard-drive-cost-per-gigabyte/`

CHAPTER

4 RDF integration of heterogeneous data sources

This chapter is devoted to the issue of querying heterogeneous data sources based on RDF as the global data model and using general BGP queries. In Section 4.1, we classify existing mediator-based approaches and situate our contributions in this landscape. In Section 4.2, we introduce our framework, namely RDF integration systems, and define the associated query answering problem. In Section 4.3, we present several query answering strategies, which may materialize global RDF graph or leave it virtual, and differ on how and when RDFS reasoning is incorporated. These strategies are implemented in OBI-WAN*, our RDF integration system, which is described in Section 4.4. Section 4.5 details our experimental evaluation. Finally, Section 4.6 extends the theoretical framework to a class of RDF entailment rules that go beyond RDFS. The material presented here has been published in [Buron et al., 2020b, Buron et al., 2020c, Buron et al., 2018], whereas Section 4.3.7 and Section 4.4 are new.*

4.1 Motivation and state of the art

The proliferation of digital data sources across all application domains brings a new urgency to the need for tools which allow to query heterogeneous data (relational, JSON, key-values, graphs etc.) in a flexible fashion. Traditional data integration systems fall into two classes: *data warehousing* [Jarke, 2003], where all data source content is materialized in a single centralized source, respectively, *mediation* [Wiederhold, 1992], where data remains in their original stores and all data can be queried through a single module called *mediator*. Data warehousing simplifies query evaluation, but requires potentially costly maintenance operations when the content of data sources changes; mediation does not suffer from these drawbacks, but requires more intricate query evaluation algorithms to distribute the work between the sources and the mediator.

Below, we classify prior mediator-based approaches according to two main dimensions, and illustrate this classification in Table 4.1. The first dimension concerns the data model and query languages at the global level of the integration system; we discuss existing systems from this perspective in Section 4.1.1. The second dimension is determined by different mapping languages used; we analyze prior work from this perspective in Section 4.1.2. Recall that mapping languages have been introduced earlier in this manuscript in Section 2.2.

4.1.1 Mediator data models and query languages

A first dimension concerns the global data model and query language provided by the mediator to its applications. From this perspective, we identify several classes of systems, outlined below.

Database-style mediators

The earliest goal of a mediator system was to mimic a single, integrated database. Thus the mediator supports one data model and its query language, e.g., relational and SQL, or XML and XPath/XQuery. More recent polystore systems support side-by-side different (data model, query language) pairs. These database-style mediators appear in the row we label **DB** in Table 4.1.

Semantic mediators

Mediators studied in knowledge representation and management research provide a view of the data sources as a set of classes and relationships, also endowed with a set of semantic constraints, or ontology. In such systems, users formulate conjunctive (relational) queries or a query in BGPQ-CQ (recall Table 3.1), a fragment of BGPQs, where querying the schema triple and using variables in property and class position is forbidden. Answering such queries involves not only evaluation over the data (as done in DB mediators), but also reasoning on the data with the help of ontologies. This mediation approach is also commonly termed *Ontology-Based Data Access* (OBDA) [Poggi et al., 2008], with ontologies expressed in Description Logics (DL, in short). Work following this approach are listed in the row we label **CQ** in Table 4.1.

Ontologies have been used to integrate relational or heterogeneous data sources in mediators with LAV views based on description logics [Levy et al., 1995] and [Abdallah et al., 2009], or their combination with Datalog in [Goasdoué et al., 2000] and [Goasdoué and Rousset, 2004]. OBDA works [Poggi et al., 2008], [Pinto et al., 2013], [Rodriguez-Muro et al., 2013] and [Lanti et al., 2017] present integration systems, also relying on Description Logics (DL), especially DL-Lite$_R$ or DL-Lite$_{\mathcal{A}}$. Semantics have been used at the integration level since e.g., [Christophides et al., 1997] for SGML and soon after for RDF [Amann and Fundulaki, 1999, Amann et al., 2000]; data is considered represented and stored in a flexible object-oriented model, thus no mappings are used.

RDF-based mediators

Lately, RDF [RDF, 2014a] has emerged a portable, self-describing formalism for modeling data. RDF is naturally suited as an integration model, thanks to its flexibility, its wide adoption in the Open Data community, its close relationship with ontology languages such as RDFS [RDF, 2014b] and OWL [OWL, a], and the presence of its associated standard SPARQL query language [SPA, 2013]. Accordingly, several mediators from the CQ group have been extended to support RDF as an integration model and SPARQL query answering. However, while SPARQL allows *querying the data together with the ontology*, e.g., "find the properties of node n, as well the classes to which the values of these properties belong", a DL-based mediation approach shares with all logic-based query languages, such as Datalog, SQL etc., the inability to do so. RDF mediators which support SPARQL but limited to querying the data only (not the ontology) appear in the row we

label **SPARQL-data** in Table 4.1. SPARQL-data is an extension of the BGPQ-data, used in Table 3.1; it additionally supports operators like FILTER, OPTIONAL etc..

In [Sequeda et al., 2014], an RDF global schema of the integration system is populated with data triples only, on which the RDFS reasoning w.r.t. an RDFS ontology is performed extended with the definition of property inverses and with support for symmetric and transitive properties. Defining transitive properties is allowed by the support of recursive queries in the query rewriting language.

Mediators supporting ontology queries

Recent RDF mediators lift this limitation to support joint querying of the data and ontology; we list them in the **SPARQL** row in Table 4.1. In these works, reasoning is also taken into account, which allows inferring new ontological triples.

Works such as [Kontchakov et al., 2014, Botoeva et al., 2018, Calvanese et al., 2017] rely on query rewriting outlined in the former article to support the OWL 2 QL entailment regime. As explained in Section 3.1.2, this query rewriting assumes a typing of query variables in order to know if they will match classes, properties or instances of the OWL knowledge base. This assumption makes impossible the use of this technique with domain metamodeling-capable graphs.

In [Pinto et al., 2011], domain metamodeling at the global level of integration systems is enabled for an extension of DL-Lite$_R$ which allows reasoning and querying of the terminological part of knowledge bases.

4.1.2 Mapping Language

A second dimension is how the source (or local) schemas are connected to the global (integration) schema, using *mappings* [Doan et al., 2012]. We identify three types of mappings, each corresponding to a column in Table 4.1, introduced in Section 2.2.

Global-As-View

The simplest mappings define each element of the global schema, e.g., each relation (if the global schema is relational), as a view over the local schemas; this is known as Global-As-View, or **GAV** in short. In a GAV system, a query over the global (virtual) schema is easily transformed into a query over the local schemas, by *unfolding* each global schema relation, i.e., replacing it with its definition (recall Section 2.2.2).

Most of the abovementioned OBDA works, following [Poggi et al., 2008], use GAV mappings. This is the case, specifically, for all the implementated OBDA mediators [Calvanese et al., 2011, Sequeda et al., 2014, Calvanese et al., 2017], which rely on GAV mappings.

Local-As-View

In contrast, Local-As-View (**LAV**) mappings define elements of the local schemas as views over the global one. Query answering in this context requires *rewriting the query with the views* describing the local sources (recall Section 2.2.3).

Models	Mappings		
	GAV	**LAV**	**GLAV**
DB	[Garcia-Molina et al., 1997], [Deutsch and Tannen, 2003, Duggan et al., 2015]	[Amann et al., 2002], [Deutsch and Tannen, 2003, Manolescu et al., 2001], [Alotaibi et al., 2019]	[Calvanese et al., 2012]
CQ	[Poggi et al., 2008], [Calvanese et al., 2011], [Pinto et al., 2013], [Rodriguez-Muro et al., 2013], [Lanti et al., 2017]	[Levy et al., 1996], [Amann and Fundulaki, 1999], [Amann et al., 2000], [Goasdoué et al., 2000], [Goasdoué and Rousset, 2004], [Abdallah et al., 2009]	[Calvanese et al., 2009]
SPARQL-data	[Sequeda et al., 2014], [Hovland et al., 2017]	[Smits et al., 2014]	[De Giacomo et al., 2018]
SPARQL	[Pinto et al., 2011], [Kontchakov et al., 2014], [Botoeva et al., 2018], [Calvanese et al., 2017]		our work

Table 4.1: Outline of the positioning of our contributions.

Global-Local-As-View

Global-Local-As-View (**GLAV**) data integration generalizes both GAV and LAV. A GLAV mapping pairs a query q_1 over one or several local schemas to a query q_2 over the global schema, having the same answer variables. The semantics is that for each answer of q_1, the integration system exposes the data comprised in a corresponding answer of q_2 (recall Section 2.2.2).

GLAV maximizes flexibility, or, equivalently, *integration expressive power*: unlike LAV, a GLAV mapping may expose only part of a given source's data, and may combine data from several sources; unlike GAV, a GLAV mapping may include joins or complex expressions over the global schema. A benefit of our using GLAV is the ability to support a form of *incomplete information*, naturally present in RDF through the so-called *blank nodes*, in the virtual RDF graph exposed by the mediator.

Formal OBDA frameworks based on GLAV mappings have been defined, for instance, in [Calvanese et al., 2009], without concretely deployed systems. A technique for simulating GLAV mappings through GAV ones under certain conditions, provided with so-called Skolem functions on answer variables, is suggested in [De Giacomo et al., 2018], however this solution has several drawbacks (see details in Section 4.3.6).

4.1.3 Contributions

As Table 4.1 shows, our work is, to the best of our knowledge, *the first to support the integration of multiple data sources through GLAV mappings, for general BGPQ querying over the data and the ontology*.

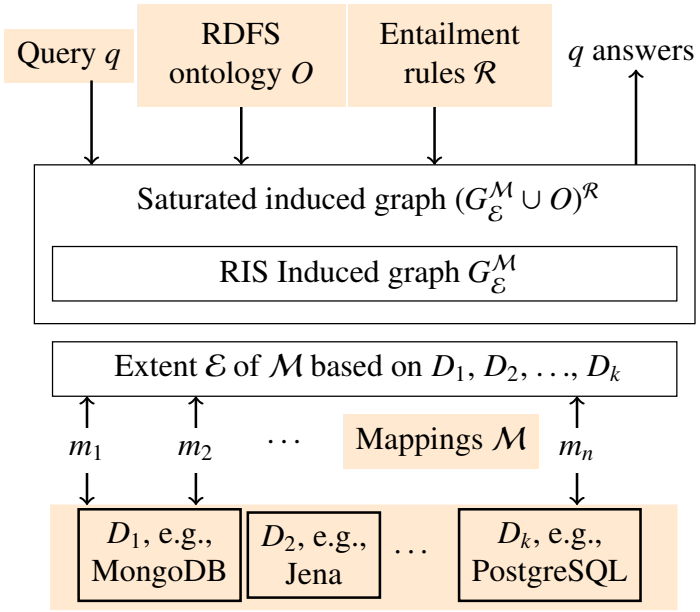


Figure 4.1: Outline of an RDF Integration System.

The main contributions of this chapter are the following:

- We formally define **RDF Integration Systems** (RIS, in short), outlined in Figure 4.1. These are OBDA mediators capable of exposing data from heterogeneous sources through GLAV mappings, under the form of an RDF graph with full metamodeling (see Section 3.1.1). We formalize the problem of **general BGP query**

answering over the RDF induced graph and an **RDFS ontology** exposed in such systems w.r.t. to a **set of RDF entailment rules**. We introduce the RIS formalism and its related query answering problem in Section 4.2.

- We describe **several query answering strategies in RIS** using the RDFS entailment rules following materialization and rewriting-based approaches. The latter are based on transforming *RIS mappings into relational LAV mappings*, and on reducing query answering to rewriting it using LAV mappings. Two rewriting-based strategies are novel, and rely on a form of *mapping saturation*. We show that a smart decomposition of reasoning between offline (precomputation) and query time makes one of these strategies much faster than the others. These query answering strategies are exposed in Section 4.3, and experimented with in Section 4.5.

- We developed **Obi-Wan**, a system implementing the query answering strategies in RIS in Section 4.4.

- We generalize our mapping saturation framework for a subset of existential rules in Section 4.6.

4.2 RDF Integration Systems

Below, we first formalize the notion of *RDF integration system* in Section 4.2.1. Then, we state the associated query answering problem in Section 4.2.2, for which Section 4.3 provides solutions.

4.2.1 RDF Integration System (RIS) Definition

An RDF integration system (RIS in short) relies on a set $\mathcal{S}$ of *heterogeneous source schemas*, each of which may use its own data model and query language. We assume all queries in these languages return *tuples of bindings*, which is the case for SQL, XPath (and any tree pattern language), and BGPQs. A set of instance sets, one for each schema in $\mathcal{S}$, are integrated into an RDF graph, consisting of an (RDFS) ontology and of data triples derived from the sources by means of GLAV-style mappings. Mappings allow (*i*) specifying and (*ii*) transforming the data made available from the sources, as well as (*iii*) organizing it according to the RIS ontology.

In order to specify which data are available from *a source*, a mapping specifies a query on it. The results of this query are tuples of values extracted from the sources; these values may or may not be proper RDF values, i.e., IRI, literal or blank nodes. In order to form RDF values from source query results, we consider a set of value transformation functions; applying a function on a tuple of source values returns a proper RDF value.

Definition 4.1 (RIS mappings)**.** *A* RIS mapping *m on a source schema* $S \in \mathcal{S}$ *is of the form:*

$$m = q_1(\bar{x}) \rightsquigarrow q_2(f_1(\bar{x}), f_2(\bar{x}), \ldots, f_n(\bar{x}))$$

where:

- q_1 *is a query on* S *of arity* k*, called the* body of *m,*

- q_2 *is a BGPQ of arity* n*, called the* head of *m,*

- *for* $1 \leq i \leq n$, f_i *is a function of k inputs whose result is an IRI, a literal or a blank node. We call the* f_i*s the* functions of *m.*

In the following, we will replace the term $f_i(\bar{x})$ by x_i the ith variable in $\bar{x}$, when f_i is the projection on its ith input, i.e., $f_i(\bar{x}) = x_i$.

Consider that a RIS mapping has a body q_1 and a head q_2 having the same arity, whose functions f_i are projections on the ith input. Such a RIS mapping is also a (sound) GLAV mapping as defined in Section 2.2.4, and has the form $q_1(\bar{x}) \rightsquigarrow q_2(\bar{x})$. Hence, RIS mappings extends the GLAV mappings on a RDF global schema by introducing a set of functions allowing to translate the source values into proper RDF values.

Intuitively, a mapping m translates the answer tuples of q_1 into tuples of RDF values using its functions. These new tuples are exposed to the RIS as the result of its head BGP query q_2. This intuition will be formalized shortly below by defining *mapping extensions* and *induced graph*.

Among mappings, we identify *data mappings* which expose only data triples. These mappings are often the only ones considered in OBDA settings [Calvanese et al., 2011, Kontchakov et al., 2014, Sequeda et al., 2014, Lanti et al., 2017]. An exception is [Pinto et al., 2011] where mappings exposing schema constraints are also considered.

Definition 4.2 (Data mapping)**.** *A RIS mapping m is a* data mapping*, if* q_2*, the head of m, has a body containing only triple patterns of the following forms:*

- $(\_, \tau, \mathsf{c})$*, where* c *is an IRI,*
- $(\_, \mathsf{p}, \_)$*, where* p *is an IRI different from RDFS properties and* τ*.*

A triple position denoted using _ *means no constraint on that position.*

We define mapping extensions, inspired by the canonical extensions of views given by the GAV integration part of a GLAV integration, as exposed in Section 2.2.4. These extensions associate to each mapping m a view name denoted V_m. This set of views creates an intermediate layer in the integration system formed by a RIS, populated by the RDF values exposed as tuples.

Definition 4.3 (Mapping Extension)**.** *The* extension of a RIS mapping *m on a source schema S w.r.t. D, an instance of S, denoted* $\mathrm{ext}(m, D)$*, is the set of tuples defined by:*

$$\mathrm{ext}(m, D) = \{V_m(f_1(\bar{v}), f_2(\bar{v}), \ldots, f_n(\bar{v})) \mid \bar{v} \in q_1(D)\},$$

where $q_1(D)$ *is the answer set of m's body on D that m integrates, and* $(f_1, f_2, \ldots, f_n)$ *are the functions of m.*

Example 4.1 (Mappings)**.** *Consider the two mappings below, on the source schemas* S_1*, respectively* S_2*:*

$$m_1 = q_1^{m_1}(x) \rightsquigarrow (f_1(x), :\mathrm{pilotOf}, y), (y, \tau, :\mathrm{StarShip})$$

$$m_2 = q_1^{m_2}(x, y) \rightsquigarrow (g_1(x), :\mathrm{usesWeapon}, g_2(y)), (g_2(y), \tau, :\mathrm{LightSaber}).$$

This mapping notation has been simplified, in particular, the mapping heads are replaced by their bodies, i.e., sets of triple patterns, where the answer variables have been substituted by their respective function terms. Some function terms have also been reduced. In

m_2, the mapping functions $g_1(x, y)$ and $g_2(x, y)$ have been respectively replaced by $g_1(x)$ and $g_2(y)$, meaning that g_1 only takes into account its first input, while g_2 only its second.

Suppose that the body of m_1 returns $\langle p_1^{D_1} \rangle$ as its result, and that the f_1 function maps the value $p_1^{D_1}$ from D_1, an instance of S_1, to the IRI :Luke. *Then, the extension of m_1 is:* $\text{ext}(m_1) = \{V_{m_1}(\text{:Luke})\}$. *Further, suppose that the body of m_2 returns $\langle p_2^{D_2}, a^{D_2} \rangle$, and that g_1 and g_2 map the values $p_2^{D_2}, a^{D_2}$ from D_2, an instance of S_2, to the IRIs* :Rey, *respectively* :a. *Then, the extension of m_2 is:* $\text{ext}(m_2) = \{V_{m_2}(\text{:Rey}, \text{:a})\}$.

We now define the notion of *extent of RIS mapping set*:

Definition 4.4 (Extent of RIS mapping set)**.** *Given a set of RIS mappings $\mathcal{M}$ on schemas in $\mathcal{S}$ and $\mathcal{D}$, a set of instances of these schemas, the* extent *of $\mathcal{M}$ w.r.t. $\mathcal{D}$, denoted $\mathcal{E}_\mathcal{D}$, is the union of the mappings' extensions:*

$$\mathcal{E}_\mathcal{D} = \bigcup_{m \in \mathcal{M}} \text{ext}(m, D_m),$$

where D_m is the instance in $\mathcal{D}$ of the schema on which the mapping m is defined.

We say that $\mathcal{E}$ is an extent of $\mathcal{M}$*, if there exists $\mathcal{D}$, a set of instances of the schemas in $\mathcal{S}$, such that $\mathcal{E}$ is the extent of $\mathcal{M}$ w.r.t. $\mathcal{D}$.*

We remark that if $\mathcal{M}$ is a set of RIS mappings that contains only GLAV mappings as defined in Section 2.2.4 (i.e., without mapping functions), then the extent of $\mathcal{M}$ w.r.t. a set of instances is equal to the canonical extensions w.r.t. the same instances of the views in the GAV integration part of $\langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$, where $\mathcal{G}$ is schema for RDF.

Given an extent $\mathcal{E}$ of a mapping set, we denote by $\text{Val}(\mathcal{E})$ the set of RDF values it contains. In the following, we will often consider that an extent of the mappings is given. This amounts to considering the problem of heterogeneity among data sources solved, since an extent contains RDF values in relational views. Further, following the separation of GLAV mappings into GAV and LAV mappings presented in Section 2.2.4, we will consider the GAV mappings defined by the extent of the RIS mappings, and only have to work with the remaining LAV mappings. In other words: *knowing the extent of mappings, we can consider a RIS mapping m as a LAV mapping of the form $V_m(\bar{y}) \rightsquigarrow q_2(\bar{y})$.* This will allow us to focus on the upper part of the RDF integration system represented in the Figure 4.1, that is: the integration of the mapping extensions into RDF following an RDFS schema, which enables to infer new triples through a set of RDF entailment rules.

We can now define the RIS data triples *induced* by some mappings and an extent thereof. These form *all the data which is exposed (can be queried) through a RIS*.

Definition 4.5 (RIS induced graph)**.** *Given a set $\mathcal{M}$ of RIS mappings and an extent $\mathcal{E}$ of $\mathcal{M}$, the* RIS induced graph *by $\mathcal{M}$ and $\mathcal{E}$ is the RDF graph $G_\mathcal{E}^\mathcal{M}$ defined as follows:*

$$G_\mathcal{E}^\mathcal{M} = \bigcup_{m = q_1(\bar{x}) \rightsquigarrow q_2(\bar{x}) \in \mathcal{M}} \{\text{bgp2rdf}(body(q_2)_{[\bar{x} \leftarrow \bar{t}]}) \mid V_m(\bar{t}) \in \mathcal{E}\}$$

where

- *$body(q_2)_{[\bar{x} \leftarrow \bar{t}]}$ is the BGP body of a mapping's q_2, in which the answer variables $\bar{x}$ are bound to the values in the tuple $V_m(\bar{t})$, part of $\mathcal{E}$;*
- $\text{bgp2rdf}(\cdot)$ *is a function that transforms a BGP into an RDF graph, by replacing each variable with a fresh blank node.*

Observe that, because we use GLAV mappings, the RIS induced graph may include *incomplete information* modeled by fresh blank nodes, as exemplified below; these correspond to the existential variables allowed in GLAV mappings, as discussed at the end of Section 2.2.4.

Example 4.2. *Reusing the mappings from Example 4.1, let* $\mathcal{M} = \{m_1, m_2\}$ *and its extent* $\mathcal{E} = \{V_{m_1}(\text{:Luke}), V_{m_2}(\text{:Rey}, \text{:a})\}$. *The RIS induced graph they lead to is:*

$$G_{\mathcal{E}}^{\mathcal{M}} = \begin{array}{lll} \text{:Luke} & \text{:pilotOf} & \_{:}b_s \\ \_{:}b_s & \tau & \text{:StarShip} \\ \text{:Rey} & \text{:usesWeapon} & \text{:a} \\ \text{:a} & \tau & \text{:LightSaber} \end{array}$$

In particular, the first and second triples contain the blank node $\_{:}b_s$*, introduced by* bgp2rdf *instead of the variable* y *in the head (query* q_2*) of* m_1*. Importantly, only non-answer variables in a mapping head lead to blank nodes introduced this way: by Definition 4.5, answer variables (here* x *for* m_1 *and* x, y *for* m_2*) are replaced with values from* $V_m(\bar{t})$*, thus from* $\text{Val}(\mathcal{E})$.

Remark that the above example contains data triples, which are very similar to the ones in Figure 2.1. The main structure of the graph example has been kept, the blank node $\_{:}b_c$ (subclass of :Vehicle) has been replaced by :StarShip. The other changes are made to better highlight RIS query answering details below. This new graph is depicted in Figure 4.2.

We are now able to define an RDF Integration System by including an RDFS ontology (Definition 2.4) and a set of RDF entailment rules (Definition 2.6):

Definition 4.6 (RDF Integration System)**.** *An* RDF Integration System (RIS) *is a tuple* $S = \langle O, \mathcal{R}, \mathcal{M}, \mathcal{E} \rangle$ *stating that* S *allows to access (query), with the reasoning power given by the set* $\mathcal{R}$ *of RDF entailment rules, the RDF graph comprising the ontology* O*, and the graph induced by the set of mappings* $\mathcal{M}$ *and their extent* $\mathcal{E}$.

Equivalently, we could define a RIS by $S = \langle O, \mathcal{R}, \mathcal{M}, \mathcal{D} \rangle$ using $\mathcal{D}$ a set of instances of the sources instead of $\mathcal{E}$ an extent of $\mathcal{M}$. The RIS S obtained in this way would be equal to the RIS $\langle O, \mathcal{R}, \mathcal{M}, \mathcal{E}_{\mathcal{D}} \rangle$.

4.2.2 Query answering problem

First, we introduce a new notion of query answering in graphs, which returns only the answers that take their values in a given set:

Definition 4.7 (Restricted query evaluation)**.** *Given a subset* V *of* $\mathscr{I} \cup \mathscr{L} \cup \mathscr{B}$ *and a BGP query* q *on an RDF graph* G*, the* evaluation of q in G restricted to V *is:*

$$q_{|V}(G) = \{\varphi(\bar{x}) \mid \varphi \text{ homomorphism from } body(q) \text{ to } G\}$$

where $\varphi(\bar{x})$ *comprises only values from* V.

The problem we consider is answering BGPQs in a RIS. We define certain answers in a RIS setting as follows:

Definition 4.8 (Certain answer set)**.** *The* certain answer set *of a BGPQ* q *on a RIS* $S = \langle O, \mathcal{R}, \mathcal{M}, \mathcal{E} \rangle$ *is:*

$$\text{cert}(q, S) = q_{|\text{Val}(\mathcal{E})}\left((O \cup G_{\mathcal{E}}^{\mathcal{M}})^{\mathcal{R}}\right)$$

The certain answer set cert(q, S) is thus the subset of $q(O \cup G^{\mathcal{M}}_{\mathcal{E}}, \mathcal{R})$ restricted to tuples fully built from source values, i.e., we exclude tuples with blank nodes introduced by the mappings (see Definition 4.5). Note, however, that these blank nodes can be exploited to answer queries, as shown below.

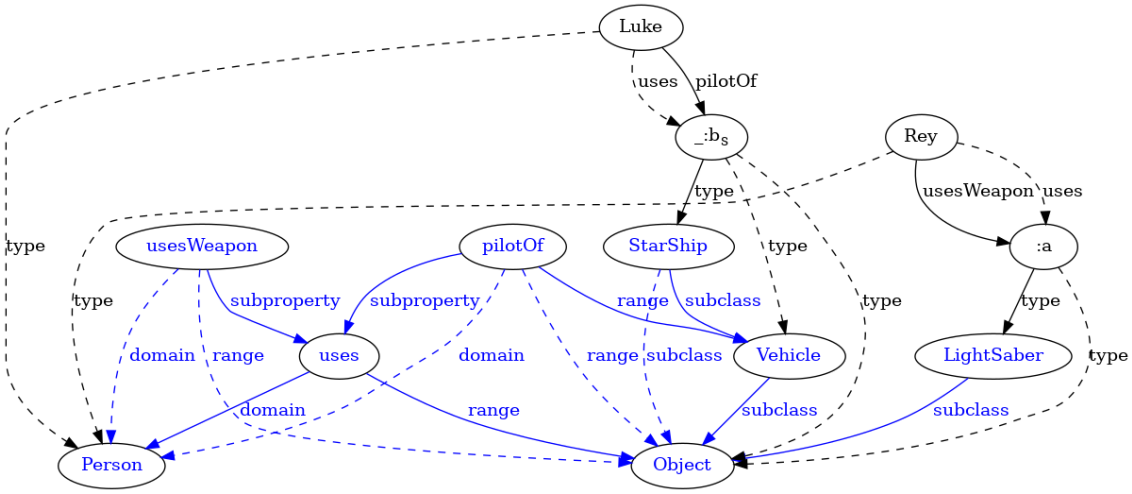


Figure 4.2: Illustration of the saturation of $G^{\mathcal{M}}_{\mathcal{E}}$ example graph and its ontology O_{ex} (shown in blue) w.r.t. $\mathcal{R}_{\text{RDFS}}$.

Example 4.3 (Certain answers)**.** *Consider the RIS S consisting of the ontology O_{ex} in Figure 4.2, the set $\mathcal{R}_{\text{RDFS}}$ of RDFS entailment rules shown in Table 2.2, and the set of mappings $\mathcal{M}$, together with the extent $\mathcal{E}$ from Example 4.2. Let q be the query asking "who uses which vehicle" defined by:*

$$q(x, y) \leftarrow (x, :\text{uses}, y), (y, \tau, :\text{Vehicle}),$$

while the query q' asking "who uses some vehicle" is defined by:

$$q'(x) \leftarrow (x, :\text{uses}, y), (y, \tau, :\text{Vehicle}).$$

The only difference between them is that y is an answer variable in q and not in q'. The certain answer set of q is $\emptyset$, while the certain answer set of q' is $\{\langle :\text{Luke}\rangle\}$. This answer results from the RIS triples (:Luke, :uses, $\_:b_s$), ($\_:b_s$, τ, :Vehicle)*, which are entailed from:*

- *the $G^{\mathcal{M}}_{\mathcal{E}}$ triples* (:Luke, :pilotOf, $\_:b_s$), ($\_:b_s$, τ, :StarShip)*, with the blank node $\_:b_s$ discussed in Example 4.2, and:*
- *either the O triples* (:pilotOf, $\prec_{sp}$, :uses), (:pilotOf, $\hookrightarrow_r$, :Vehicle) *together with the $\mathcal{R}_{\text{RDFS}}$ rules* `rdfs3` *and* `rdfs7` *(recall Table 2.2), or the O triples* (:pilotOf, $\prec_{sp}$, :uses), (:StarShip, $\prec_{sc}$, :Vehicle) *together with the $\mathcal{R}_{\text{RDFS}}$ rules* `rdfs3` *and* `rdfs9`.

The query q has no answer because it requires a value not available from the source, namely the vehicle (starship) that :Luke *uses; the RIS only knows the existence of such value through the blank node $\_:b_s$ begotten by* bgp2rdf *in its induced graph. In contrast, q' allows finding out that* :Luke *uses (as pilot of) some vehicle, even though the mapping m_1 (the only one involving starships, so vehicles) does not expose the vehicle IRI through the RIS.*

The problem we study in the next section is:

Problem 4.1. *Given a RIS S, compute the certain answer set of a BGPQ q on S, i.e.,* cert(q, S).

4.3 Query answering techniques on RDF Integration Systems

Throughout this section, we will consider a RIS $S = \langle O, \mathcal{R}, \mathcal{M}, \mathcal{E} \rangle$ and a BGP query q having the following characteristics:

- q does not contain blank nodes;
- the ontology O is a **First Order (FO) ontology** (Definition 3.4) and **does not contain blank nodes**;
- the RDF entailment rule set is $\mathcal{R}_{\text{RDFS}}$, the **RDFS entailment rule set** (Table 2.2), i.e., $\mathcal{R} = \mathcal{R}_{\text{RDFS}}$;
- the mappings in $\mathcal{M}$ are **data mappings** (Definition 4.2), exposing only data triples.

Since $G_{\mathcal{E}}^{\mathcal{M}}$ contains only data triples and O is a FO ontology, it follows from Property 3.2 that the graph induced with the ontology, $G_{\mathcal{E}}^{\mathcal{M}} \cup O$, is an RDF graph with split reasoning (Definition 3.3). Hence, the results on query reformulation presented in Section 3.2 can be used for this graph. Moreover, non-standard query evaluation can be avoided, because O and q do not contain blank nodes.

This section is devoted to *RIS query answering strategies*. The three tasks involved follow from Definition 4.8: (*i*) computing the RIS induced graph $G_{\mathcal{E}}^{\mathcal{M}}$; (*ii*) reasoning on this graph, with the ontology O, under the RDFS entailment rules $\mathcal{R}$; (*iii*) computing answers to the user query q on the graph resulting from the reasoning.

Five RIS query answering strategies are described below; they differ in *when and how* the above steps are performed. Specifically, we first describe two methods which start by actually computing the RIS induced graph; this is reminiscent of an extract-transform-load (or warehouse) scenario in classical data integration literature, e.g., [Özsu and Valduriez, 2011], followed by an RDF query answering stage (as outlined in Section 2.1.3). These strategies use a *materialization-based query answering approach*. The three next strategies are our most innovative ones: the RIS induced graph $G_{\mathcal{E}}^{\mathcal{M}}$ is not materialized, hence the saturation of $O \cup G_{\mathcal{E}}^{\mathcal{M}}$ cannot be computed to answer queries. Instead, they use the RIS mappings as *LAV mappings*, in order to reduce query answering in a RIS to relational view-based query rewriting (Section 2.2.3). These three strategies differ in how the ontological reasoning is incorporated: *all*, *some* or *none* of the reasoning effort is performed at query time, as outlined in Figure 4.3. These strategies use a *rewriting-based query answering approach*.

4.3.1 Materialization-based query answering strategies: MAT and MAT-CA

We first present two strategies which *materialize* the RIS induced graph, then reduce RIS query answering to *RDF query answering*: either based on graph saturation (MAT), or on query reformulation (MAT-CA, whose acronym reflects reformulation with both the $\mathcal{R}_c$ and

$\mathcal{R}_a$ rules, i.e., $\mathcal{R}_{\text{RDFS}}$ rules from Figure 3.1). Figure 4.1 contains most of the elements used in our discussion.

During materialization, the bodies of the mappings in $\mathcal{M}$ are executed on their respective data sources, then their results are transformed by applying the respective mapping functions to form the content of $\mathcal{E}$, the mappings extent. They are then converted as per Definition 4.5 into the RIS induced graph $G_{\mathcal{E}}^{\mathcal{M}}$, which is stored into an RDF data management system.

Subsequently, when a BGPQ q is asked against the RDF graph made of the RIS induced graph and the ontology O, it can be answered:

- by the saturation-based technique on the graph $G_{\mathcal{E}}^{\mathcal{M}} \cup O$ saturated with the set $\mathcal{R}_{\text{RDFS}}$ of RDFS entailment rules, or
- by the reformulation-based technique, reformulating q using O and $\mathcal{R}$ into a UBGPQ $Q_{c,a}$ (recall Section 3.2.6), then evaluating $Q_{c,a}$ on the RIS data triples only.

The next theorem states the correctness of the two above-mentioned RIS query answering methods:

Theorem 4.1 (MAT and MAT-CA correctness)**.** *Let q be a BGPQ asked on the RIS $S = \langle O, \mathcal{R}_{\text{RDFS}}, \mathcal{M}, \mathcal{E} \rangle$, then*

- *using MAT:* $\text{cert}(q, S) = q_{|\text{Val}(\mathcal{E})}\left((O \cup G_{\mathcal{E}}^{\mathcal{M}})^{\mathcal{R}_{\text{RDFS}}}\right)$;
- *using MAT-CA:* $\text{cert}(q, S) = (Q_{c,a})_{|\text{Val}(\mathcal{E})}\left(G_{\mathcal{E}}^{\mathcal{M}}\right)$

The proofs of the above two statements follow from the definition of certain answers in a RIS (Definition 4.8), and either from that of RDF query answers (Definition 2.16) for the saturation-based approach (MAT correctness) or from Theorem 3.5 for the reformulation-based approach (MAT-CA correctness).

Example 4.4. *Consider the RIS of Example 4.3, whose ontology O_{ex} and $G_{\mathcal{E}}^{\mathcal{M}}$ data triples form exactly the RDF graph (without saturation) in Figure 4.2. In the MAT approach, the saturation of this latter graph is stored in an RDF database (triple store). One can check that the evaluation of the query $q'(x) \leftarrow (x, :\text{uses}, y), (y, \tau, :\text{Vehicle})$ on the saturation returns the certain answer $\langle :\text{Luke} \rangle$, as explained in Example 4.3. Once the RIS induced graph is materialized, the approach MAT-CA is similar to reformulation-based query answering. This technique has been illustrated in Example 3.8.*

These two methods push the extent materialization work before queries are received, thus speeding up query answering. However, the materialized RIS data triples must be maintained when the data sources are updated. In turn, changes to the RIS induced graph require maintaining $(G_{\mathcal{E}}^{\mathcal{M}})^{\mathcal{R}_{\text{RDFS}}}$ for the saturation-based strategy. Hence, approaches based on materializing RIS data triples are generally not adapted to contexts where data sources are frequently updated.

4.3.2 Rewriting-based query answering strategies: REW-CA, REW-C and REW

In this section, we briefly introduce the three rewriting-based query answering approaches, schematized in Figure 4.3.

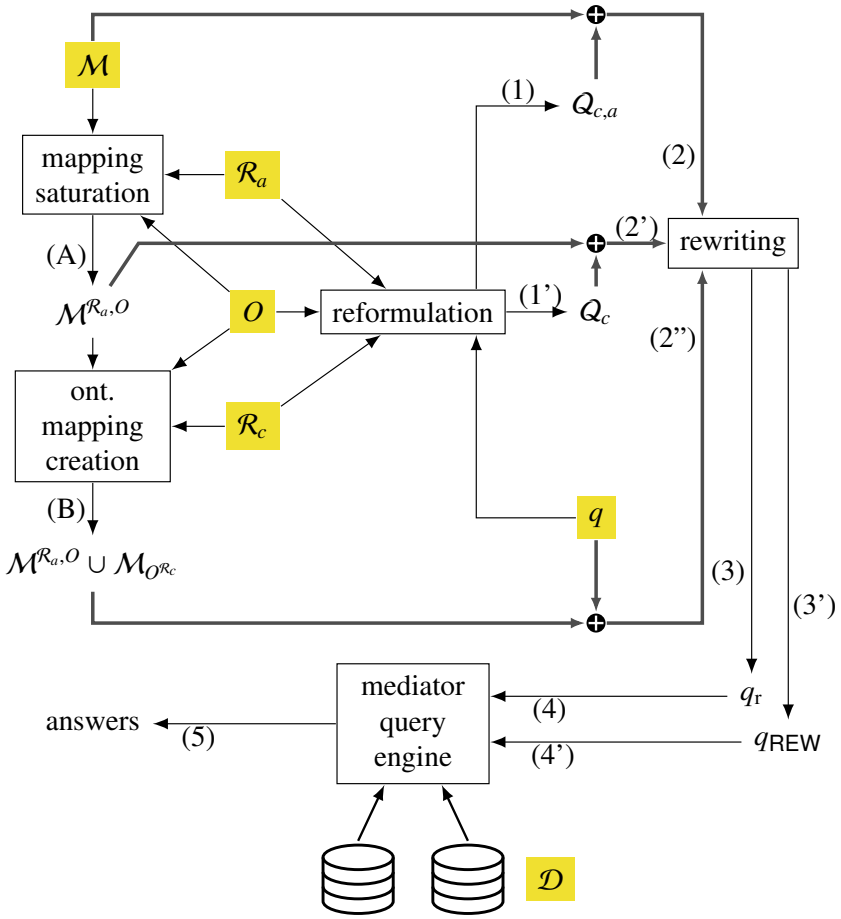


Figure 4.3: Outline of query answering strategies.

All reasoning at query time (REW-CA) The first strategy will be detailed in Section 4.3.3. First, it reduces the RIS query answering problem to *standard query evaluation* in an RDF data management system, by reformulating (step (1) in Figure 4.3) the query q based on the RIS ontology O and entailment rules $\mathcal{R}_{\text{RDFS}} = \mathcal{R}_c \cup \mathcal{R}_a$. The obtained reformulated query $Q_{c,a}$ thus yields the expected certain answers when evaluated on the RIS data triples (recall Section 3.2.6), provided that answers with blank nodes introduced by the *bgp2rdf* function are discarded (recall Section 4.2.2). Since these data triples are not materialized, the RDF query evaluation problem is in turn reduced to *relational view-based query answering*, by rewriting $Q_{c,a}$ using the RIS GLAV mappings $\mathcal{M}$ seen as LAV mappings (step (2)). This produces a relational rewriting q_r over the mappings extension (step (3)), whose evaluation with a mediator query engine provides the desired certain answers (steps (4) and (5)).

Some reasoning at query time (REW-C) The second strategy (detailed in Section 4.3.4) is a main contribution of this chapter. First, it reduces the RIS query answering problem to *saturation-based query answering* by reformulating (step (1')) the query q based on O and $\mathcal{R}_c$ only (not $\mathcal{R}_{\text{RDFS}} = \mathcal{R}_c \cup \mathcal{R}_a$ as above). The obtained reformulation Q_c thus yields the expected certain answer set when evaluated on the RIS data triples *saturated with* $\mathcal{R}_a$ (recall Section 3.2.2), again provided that the answers with blank nodes introduced by the *bgp2rdf* function are discarded (as above). Since these triples are not materialized in a RIS, hence cannot be saturated with $\mathcal{R}_a$, the saturation-based query answering problem is in turn reduced to *relational view-based query answering*, by rewriting Q_c using the RIS GLAV mappings *saturated by* O *and* $\mathcal{R}_a$, seen as LAV mappings. These saturated map-

pings, denoted $\mathcal{M}^{\mathcal{R}_a,O}$, are obtained (step (A)) from the original ones by adding to their head queries (q_2) all the implicit data triples they model w.r.t. O and $\mathcal{R}_a$. Then, the partially reformulated query Q_c is rewritten using $\mathcal{M}^{\mathcal{R}_a,O}$ (step (2')) and the resulting query (step (3)) is evaluated as in the first strategy (steps (4) and (5)). Importantly, mappings are saturated offline, and need to be updated only when some mapping or the ontology changes. This limits both the reasoning effort at query time *and* the complexity of the reformulated query to rewrite, hence the rewriting time needed to obtain a rewriting q_r over the data sources, as our experiments show (Section 4.5).

No reasoning at query time (REW) Finally, the third strategy (detailed in Section 4.3.5) reduces the RIS query answering problem directly to view-based query answering. Here, the mappings are saturated offline as above (step (A)), in order to model all explicit and implicit RIS data triples. Also, these mappings are complemented with a set of RIS mappings, noted $\mathcal{M}_{O^{\mathcal{R}_c}}$ (step (B)), comprising all the explicit and implicit RIS schema triples w.r.t. O and $\mathcal{R}_{\text{RDFS}}$; since only $\mathcal{R}_c$ rules entail new schema triples (Property 3.1), $O^{\mathcal{R}_{\text{RDFS}}}$ is actually equal to $O^{\mathcal{R}_c}$. This second set of mappings is also computed offline, and only needs to be updated when the ontology changes. A query q just needs to be rewritten based on the above mappings $\mathcal{M}^{\mathcal{R}_a,O} \cup \mathcal{M}_{O^{\mathcal{R}_c}}$ seen as LAV mappings (step (2")), in order to obtain, as above, a rewriting q_{REW} over the data sources (step(3'), followed by the evaluation steps (4') and (5)).

4.3.3 Rewriting fully-reformulated queries using LAV mappings: REW-CA

Before going into the technical details of the above strategies, we introduce a set of simple functions.

The *bgp2ca* function transforms a BGP into a conjunction of atoms with ternary predicate T (standing for "triple") as follows:

$$bgp2ca(\{(\mathtt{s}_1, \mathtt{p}_1, \mathtt{o}_1), \ldots, (\mathtt{s}_n, \mathtt{p}_n, \mathtt{o}_n)\}) = T(\mathtt{s}_1, \mathtt{p}_1, \mathtt{o}_1) \wedge \cdots \wedge T(\mathtt{s}_n, \mathtt{p}_n, \mathtt{o}_n).$$

The *bgpq2cq* function transforms a BGPQ $q(\bar{x}) \leftarrow body(q)$ into the CQ

$$q(\bar{x}) \leftarrow bgp2ca(body(q)).$$

Finally, the function *ubgpq2ucq* function transforms a UBGPQ $\bigcup_{i=1}^{n} q_i(\bar{x}_i)$ into a UCQ by applying the above *bgpq2cq* function to each of its q_i BGPQs.

Based on Section 3.2.6, the first step of this strategy, (1) in Figure 4.3, reformulates a query q w.r.t. the RIS ontology O and entailment rules $\mathcal{R}_{\text{RDFS}} = \mathcal{R}_c \cup \mathcal{R}_a$ into a query $Q_{c,a}$. This allows obtaining the certain answers directly from the RIS induced graph, and not from their saturation after they have been augmented with O (recall Definition 4.8). Indeed, the correctness of the reformulation ensures that the certain answers of q on the RIS S correspond precisely to those of $Q_{c,a}$ asked on S when disregarding O and $\mathcal{R}_{\text{RDFS}}$, as formally expressed in the next lemma. Of course, this still does not provide a concrete method to compute the certain answers through standard query evaluation, since the RIS data triples $G_{\mathcal{E}}^{\mathcal{M}}$ are not materialized.

Lemma 4.1. *Let $S = \langle O, \mathcal{R}_{\text{RDFS}}, \mathcal{M}, \mathcal{E}\rangle$ be a RIS, q be a BGPQ and $Q_{c,a}$ its UBGPQ reformulation w.r.t. $O, \mathcal{R}_{\text{RDFS}} = \mathcal{R}_c \cup \mathcal{R}_a$ using Section 3.2.6. Then:*

$$\text{cert}(q, S) = \text{cert}(Q_{c,a}, \langle \emptyset, \emptyset, \mathcal{M}, \mathcal{E}\rangle)$$

Proof.

$$\begin{aligned}\text{cert}(q, S) &= q_{|\text{Val}(\mathcal{E})}\left((O \cup G^{\mathcal{M}}_{\mathcal{E}})^{\mathcal{R}_{\text{RDFS}}}\right)\\ &= (Q_{c,a})_{|\text{Val}(\mathcal{E})}\left((O \cup G^{\mathcal{M}}_{\mathcal{E}}) \setminus O\right) \quad \text{(Theorem 3.5)}\\ &= (Q_{c,a})_{|\text{Val}(\mathcal{E})}\left(G^{\mathcal{M}}_{\mathcal{E}}\right)\\ &= \text{cert}(Q_{c,a}, \langle \emptyset, \emptyset, \mathcal{M}, \mathcal{E}\rangle)\end{aligned}$$

□

Recall that the RIS data triples are defined from the mappings $\mathcal{M}$ as follows: for every RIS mapping $m = q_1(\bar{x}) \rightsquigarrow q_2(f_1(\bar{x}), \ldots, f_n(\bar{x})) \in \mathcal{M}$

- evaluate the mapping body $q_1(\bar{x})$ on the data source D and applying the functions f_i on the results to produce its extension $\text{ext}(m, D) \in \mathcal{E}$ defining the content of V_m, and then
- instantiate the mapping head $q_2(y_1, \ldots, y_n)$ with its extension.

This is also how the instance of a data integration system based on LAV mappings and their extensions is defined in a relational setting (Section 2.2.3). Based on this analogy and on the splitting of GLAV integration system into GAV and LAV integration systems presented in Section 2.2.4, we *recast the RIS query answering problem* of the above Lemma, into a *relational view-based query answering* one. To this aim, we treat our RIS mappings as LAV mappings:

Definition 4.9 (RIS mappings as relational LAV mappings)**.** *Let m be a RIS mapping and q_2 its head having the answer variables $\bar{y}$. The corresponding* relational LAV mapping of *m is the (sound) LAV mapping defines by:*

$$V_m(\bar{y}) \rightarrow bgp2ca(body(q_2)).$$

We denote the set of LAV mappings derived from all the RIS mappings $\mathcal{M}$ by $\text{Views}(\mathcal{M})$. Crucially, the extent $\mathcal{E}$ of the mapping set $\mathcal{M}$ is also an extent for the corresponding set of views $\text{Views}(\mathcal{M})$.

Example 4.5. *The relational LAV mappings corresponding to the mappings m_1, m_2 from Example 4.1 are:*

- $V_{m_1}(x) \rightarrow T(x, :\text{pilotOf}, y), T(y, \tau, :\text{StarShip})$
- $V_{m_2}(x, y) \rightarrow T(x, :\text{usesWeapon}, y), T(y, \tau, :\text{LightSaber})$

Based on the above Lemma 4.1, treating mappings and their extent as relational LAV mappings and their extent, and seeing (U)BGPQs as (U)CQs with the help of the functions introduced in the beginning of this section, we reduce the RIS query answering problem to view-based query answering:

Theorem 4.2 (REW-CA correctness)**.** *Let $S = \langle O, \mathcal{R}_{\text{RDFS}}, \mathcal{M}, \mathcal{E}\rangle$ be a RIS and q be a BGPQ. Let $Q_{c,a}$ be the reformulation of q w.r.t. O and $\mathcal{R}_{\text{RDFS}}$ as defined in Section 3.2.6. Then:*

$$\text{cert}(q, S) = \text{cert}(ubgpq2ucq(Q_{c,a}), \text{Views}(\mathcal{M}), \mathcal{E})$$

where $\text{cert}(ubgpq2ucq(Q_{c,a}), \text{Views}(\mathcal{M}), \mathcal{E})$ *denotes the certain answer set of $ubgpq2ucq(Q_{c,a})$ over* $\text{Views}(\mathcal{M})$ *and $\mathcal{E}$.*

$$
\begin{aligned}
\mathcal{Q}_{c,a} = \; & q(x, :\text{pilotOf}) \leftarrow && T(x, :\text{pilotOf}, z), T(z, \tau, :\text{StarShip}),\\
& && T(x, :\text{uses}, a), T(a, \tau, :\text{LightSaber})\\
& \cup\, q(x, :\text{pilotOf}) \leftarrow && T(x, :\text{pilotOf}, z), T(z, \tau, :\text{StarShip}),\\
& && T(x, :\text{usesWeapon}, a), T(a, \tau, :\text{LightSaber})\\
& \cup\, q(x, :\text{pilotOf}) \leftarrow && T(x, :\text{pilotOf}, z), T(z, \tau, :\text{StarShip}),\\
& && T(x, :\text{pilotOf}, a), T(a, \tau, :\text{LightSaber})\\
& \cup\, q(x, :\text{usesWeapon}) \leftarrow && T(x, :\text{usesWeapon}, z), T(z, \tau, :\text{StarShip}),\\
& && T(x, :\text{uses}, a), T(a, \tau, :\text{LightSaber})\\
& \cup\, q(x, :\text{usesWeapon}) \leftarrow && T(x, :\text{usesWeapon}, z), T(z, \tau, :\text{StarShip}),\\
& && T(x, :\text{usesWeapon}, a), T(a, \tau, :\text{LightSaber})\\
& \cup\, q(x, :\text{usesWeapon}) \leftarrow && T(x, :\text{usesWeapon}, z), T(z, \tau, :\text{StarShip}),\\
& && T(x, :\text{pilotOf}, a), T(a, \tau, :\text{LightSaber})
\end{aligned}
$$

Figure 4.4: Sample reformulation for Example 4.6.

Importantly, this provides an effective solution to RIS query answering problem by using state-of-the-art view-based query rewriting techniques [Halevy, 2001], in particular for step (2) in Figure 4.3.

Example 4.6 (REW-CA query answering)**.** *Consider again the RIS in Example 4.3 and the query:*

$$q(x, y) \leftarrow (x, y, z), (z, \tau, t), (y, \prec_{sp}, :\text{uses}), (t, \prec_{sc}, :\text{Vehicle}), (x, :\text{uses}, a), (a, \tau, :\text{LightSaber})$$

asking "who uses light sabers, and how do they use vehicles". Its UBGPQ reformulation, seen as an UCQ, is shown in Figure 4.4. Its maximally-contained rewriting based on the LAV mappings obtained from the RIS mappings (see Example 4.5) is:

$$q_r(x, :\text{pilotOf}) \leftarrow V_{m_1}(x), V_{m_2}(x, y),$$

obtained from the second CQ in the above union. This becomes clear when the view symbols are replaced according to the LAV mappings :

$$
\begin{aligned}
q(x, :\text{pilotOf}) \leftarrow \; & T(x, :\text{pilotOf}, y_1), T(y_1, \tau, :\text{StarShip}),\\
& T(x, :\text{usesWeapon}, y_2), T(y_2, \tau, :\text{LightSaber}).
\end{aligned}
$$

Note that the other CQs cannot be rewritten given the available LAV mappings.

With the current RIS, this rewriting yields an empty certain answer set to q, i.e., $\text{cert}(q, S) = \emptyset$*, because the extent of the mappings, hence of the LAV mappings, is:* $\mathcal{E} = \{V_{m_1}(:\text{Luke}), V_{m_2}(:\text{Rey}, :\text{a})\}$*. However, if we add* $V_{m_2}(:\text{Luke}, :\text{a})$ *to* $\mathcal{E}$*, then* $\text{cert}(q, S) = \{\langle :\text{Luke}, :\text{pilotOf}\rangle\}$.

4.3.4 Rewriting partially-reformulated queries using saturated LAV mappings: REW-C

In contrast with the REW-CA strategy that performs *all* the reasoning w.r.t. O and $\mathcal{R}_{\text{RDFS}} = \mathcal{R}_c \cup \mathcal{R}_a$ at query time, our second strategy, called REW-C, splits the reasoning work between offline preprocessing and query time.

The first step of this strategy, labeled (1') in Figure 4.3, reformulates a query q as explained in Section 3.2.4, but *solely* w.r.t. $O, \mathcal{R}_c$, producing a UBGPQ denoted $\mathcal{Q}_c$. From the correctness of this reformulation step, and the fact that only $\mathcal{R}_a$ needs to be considered

to answer Q_c with respect to the entire set of rules $\mathcal{R}_{\text{RDFS}}$ (recall Section 3.2.2), the certain answer set of q asked on the RIS S is exactly the certain answer set of Q_c asked on S when disregarding $\mathcal{R}_c$. Formally:

Lemma 4.2. *Let $S = \langle O, \mathcal{R}_{\text{RDFS}}, \mathcal{M}, \mathcal{E}\rangle$ be a RIS, q be a BGPQ and Q_c its reformulation w.r.t. $O, \mathcal{R}_c$. Then:*

$$\text{cert}(q, S) = \text{cert}(Q_c, \langle O, \mathcal{R}_a, \mathcal{M}, \mathcal{E}\rangle)$$

Proof.

$$\begin{aligned}
\text{cert}(q, S) &= q_{|\text{Val}(\mathcal{E})}\left((O \cup G^{\mathcal{M}}_{\mathcal{E}})^{\mathcal{R}_{\text{RDFS}}}\right) \\
&= q_{|\text{Val}(\mathcal{E})}\left(\left((O \cup G^{\mathcal{M}}_{\mathcal{E}})^{\mathcal{R}_a}\right)^{\mathcal{R}_c}\right) \quad \text{(Theorem 3.2)} \\
&= (Q_c)_{|\text{Val}(\mathcal{E})}\left((O \cup G^{\mathcal{M}}_{\mathcal{E}})^{\mathcal{R}_a}\right) \quad \text{(Equation 3.20)} \\
&= \text{cert}(Q_c, \langle O, \mathcal{R}_a, \mathcal{M}, \mathcal{E}\rangle)
\end{aligned}$$

□

In other words, the desired answer set can be obtained by evaluating Q_c on the RIS data triples $G^{\mathcal{M}}_{\mathcal{E}}$ saturated by $\mathcal{R}_a$. Again, since the RIS data triples are not materialized, this does not provide a concrete solution. To account for the impact of the ontology O and the entailment rules $\mathcal{R}_a$ on these "virtual" data triples, we rely on *BGPQ saturation* [El Hassad et al., 2017]: given a BGPQ q, O and $\mathcal{R}_a$, the *saturation $q^{\mathcal{R}_a,O}$ of q w.r.t. $\mathcal{R}_a$ and O is q augmented with* all *the triples q implicitly asks for, given the ontology O and the rules $\mathcal{R}_a$*. To compute $q^{\mathcal{R}_a,O}$ we (*i*) saturate body$(q) \cup O$ using $\mathcal{R}_a$, then (*ii*) add to the body of q all triples thus inferred. BGPQ saturation is exemplified below:

Example 4.7 (BGPQ saturation)**.** *Consider the ontology O_{ex} in Figure 4.2 and the query $q(x) \leftarrow (x, :\text{pilotOf}, y), (y, \tau, :\text{StarShip})$ asking for the star ship pilots. Its saturation w.r.t. $\mathcal{R}_a, O$ is (inferred triples are in blue):*

$$q^{\mathcal{R}_a,O}(x) \leftarrow \textit{body}(q), (x, :\text{uses}, y), (x, \tau, :\text{Person}), (y, \tau, :\text{Vehicle}), (y, \tau, :\text{Object}).$$

We use BGPQ saturation to saturate the RIS mapping heads w.r.t. $\mathcal{R}_a, O$, so that the saturated mappings together with $\mathcal{E}$ model the saturated RIS data triples w.r.t. $\mathcal{R}_a, O$.

Definition 4.10 (Mappings saturation)**.** *The saturation of a set $\mathcal{M}$ of RIS mappings w.r.t. the entailment rules $\mathcal{R}_a$ and ontology O is:*

$$\mathcal{M}^{\mathcal{R}_a,O} = \bigcup_{m \in \mathcal{M}} \{q_1(\bar{x}) \rightsquigarrow q_2^{\mathcal{R}_a,O}(\bar{x}) \mid m = q_1(\bar{x}) \rightsquigarrow q_2(\bar{x})\}$$

We saturate mappings offline, and just need to update them when O or the mapping heads change.

Example 4.8 (Saturated mappings)**.** *Recall the RIS of Example 4.3. The mapping heads in $\mathcal{M}^{\mathcal{R}_a,O}$ are shown below (inferred triples are in blue):*

$$\begin{aligned}
m_1: \quad & q_2^{\mathcal{R}_a,O}(x) \leftarrow (x, :\text{pilotOf}, y), (y, \tau, :\text{StarShip}) \\
& \qquad\qquad (x, :\text{uses}, y), (y, \tau, :\text{Vehicle}) \\
& \qquad\qquad (x, \tau{:}\text{Person}), (y, \tau, :\text{Object}) \\
m_2: \quad & q_2^{\mathcal{R}_a,O}(x, y) \leftarrow (x, :\text{usesWeapon}, y), (y, \tau, :\text{LightSaber}) \\
& \qquad\qquad (x, :\text{uses}, y), (y, \tau, :\text{Object}) \\
& \qquad\qquad (x, \tau, :\text{Person})
\end{aligned}$$

From the above Lemma and the use of saturated RIS mappings instead of the original ones, we show:

Lemma 4.3. *Let $S = \langle O, \mathcal{R}_{\text{RDFS}}, \mathcal{M}, \mathcal{E}\rangle$ be a RIS, q be a BGPQ and Q_c its reformulation w.r.t. $O, \mathcal{R}_c$ as defined in Section 3.2.2. Then:*

$$\text{cert}(q, S) = \text{cert}(Q_c, \langle \emptyset, \emptyset, \mathcal{M}^{\mathcal{R}_a, O}, \mathcal{E}\rangle)$$

Proof.

$$\begin{aligned}
\text{cert}(q, S) &= \text{cert}(Q_c, \langle O, \mathcal{R}_a, \mathcal{M}, \mathcal{E}\rangle) \quad \text{(Lemma 4.2)}\\
&= (Q_c)_{|\text{Val}(\mathcal{E})}\left((O \cup G_{\mathcal{E}}^{\mathcal{M}})^{\mathcal{R}_a}\right)\\
&= (Q_c)_{|\text{Val}(\mathcal{E})}\left(O^{\mathcal{R}_a} \cup G_{\mathcal{E}}^{\mathcal{M}^{\mathcal{R}_a, O}}\right) \quad \text{(Theorem 4.5)}\\
&= (Q_c)_{|\text{Val}(\mathcal{E})}\left(O \cup G_{\mathcal{E}}^{\mathcal{M}^{\mathcal{R}_a, O}}\right) \quad (O \text{ is an FO ontology and Property 3.2})\\
&= (Q_c)_{|\text{Val}(\mathcal{E})}\left(G_{\mathcal{E}}^{\mathcal{M}^{\mathcal{R}_a, O}}\right) \quad \text{(Equation 3.21)}\\
&= \text{cert}(Q_c, \langle \emptyset, \emptyset, \mathcal{M}^{\mathcal{R}_a, O}, \mathcal{E}\rangle)
\end{aligned}$$

□

This result allows solving the RIS query answering problem by relational view-based query rewriting (step (2') in Figure 4.3):

Theorem 4.3 (REW-C correctness)**.** *Let $S = \langle O, \mathcal{R}_{\text{RDFS}}, \mathcal{M}, \mathcal{E}\rangle$ be a RIS, q be a BGPQ and Q_c its reformulation w.r.t. $O, \mathcal{R}_c$.Then:*

$$\text{cert}(q, S) = \text{cert}(ubgpq2ucq(Q_c), \text{Views}(\mathcal{M}^{\mathcal{R}_a, O}), \mathcal{E})$$

A proof of this result is given in Section 4.6. We illustrate REW-C query answering on an example:

Example 4.9 (REW-CA)**.** *Consider again the RIS in Example 4.3 and the query q of Example 4.6. Its reformulation Q_c w.r.t. $O, \mathcal{R}_c$, seen as a UCQ, is:*

$$\begin{aligned}
q(x, :\text{pilotOf}) \leftarrow \quad & T(x, :\text{pilotOf}, z), T(z, \tau, :\text{StarShip}),\\
& T(x, :\text{uses}, a), T(a, \tau, :\text{LightSaber})\\
\cup\, q(x, :\text{usesWeapon}) \leftarrow & T(x, :\text{usesWeapon}, z), T(z, \tau, :\text{StarShip}),\\
& T(x, :\text{uses}, a), T(a, \tau, :\text{LightSaber})
\end{aligned}$$

This reformulation is therefore rewritten using the LAV mappings as:

$$q_r(x, :\text{pilotOf}) \leftarrow V_{m_1}(x), V_{m_2}(x, y).$$

This rewriting is due to the first CQ in the above; the second one has no rewriting based on the available LAV mappings. Note that this rewriting is equivalent to the one obtained in Example 4.6, hence it yields the same answers.

4.3.5 Rewriting queries using saturated mappings and ontology LAV mappings: REW

This strategy does not reason at query time at all. Instead, it rewrites a query q based on the saturated RIS mappings $\mathcal{M}^{\mathcal{R}_a,O}$ as above, *and* on a specific set of ontology mappings we build to model the *saturated RIS ontology as a data source*:

Definition 4.11 (Ontology mappings and their extent)**.** *The* ontology mappings *for a RDFS ontology O w.r.t. a set of RDF entailment rules $\mathcal{R}$, denoted $\mathcal{M}_{O^\mathcal{R}}$, is defined as:*

$$\mathcal{M}_{O^\mathcal{R}} = \bigcup_{p \in \{\prec_{sc}, \prec_{sp}, \hookrightarrow_d, \hookrightarrow_r\}} \{m_p \mid m_p = q_1(\mathsf{s}, \mathsf{o}) \rightsquigarrow (\mathsf{s}, p, \mathsf{o})\}$$

The extent *of $\mathcal{M}_{O^\mathcal{R}}$, denoted $\mathcal{E}_{O^\mathcal{R}}$, is defined by:*

$$\mathcal{E}_{O^\mathcal{R}} = \bigcup_{p \in \{\prec_{sc}, \prec_{sp}, \hookrightarrow_d, \hookrightarrow_r\}} \{V_{m_p}(\mathsf{s}, \mathsf{o}) \mid (\mathsf{s}, p, \mathsf{o}) \in O^\mathcal{R}\}$$

where V_{m_p} comes from the relational LAV mapping corresponding to m_p (Definition 4.9).

We compute $\mathcal{M}_{O^{\mathcal{R}_c}}$, the ontology mappings, together with their extent $\mathcal{E}_{O^{\mathcal{R}_c}}$ offline, and only need to update them when the ontology changes. The extent $\mathcal{E}_{O^{\mathcal{R}_c}}$, which is treated like an extra data source introduced for the needs of this query answering strategy, stores all the explicit and implicit RIS ontology triples (recall from Section 2.1.3 that only $\mathcal{R}_c$ lead to such triples). Importantly, this leads to the observation that a query triple that refers to the ontology (schema) can be evaluated on the ontology mapping extensions alone. Formally:

Lemma 4.4. *Let $S = \langle O, \mathcal{R}_{\text{RDFS}}, \mathcal{M}, \mathcal{E}\rangle$ be a RIS and q be a BGPQ. Then:*

$$\text{cert}(q, S) = \text{cert}(q, \langle \emptyset, \mathcal{R}_a, \mathcal{M}_{O^{\mathcal{R}_c}} \cup \mathcal{M}, \mathcal{E}_{O^{\mathcal{R}_c}} \cup \mathcal{E}\rangle).$$

Proof.

$$\begin{aligned}
\text{cert}(q, S) &= q_{|\text{Val}(\mathcal{E})}\left((O \cup G^{\mathcal{M}}_{\mathcal{E}})^{\mathcal{R}_{\text{RDFS}}}\right) \\
&= q_{|\text{Val}(\mathcal{E})}\left(\left((O \cup G^{\mathcal{M}}_{\mathcal{E}})^{\mathcal{R}_c}\right)^{\mathcal{R}_a}\right) \quad \text{(Theorem 3.2)} \\
&= q_{|\text{Val}(\mathcal{E})}\left((O^{\mathcal{R}_c} \cup G^{\mathcal{M}}_{\mathcal{E}})^{\mathcal{R}_a}\right) \quad \text{(Property 3.1)} \\
&= q_{|\text{Val}(\mathcal{E})}\left((G^{\mathcal{M}_{O^{\mathcal{R}_c}}}_{\mathcal{E}_{O^{\mathcal{R}_c}}} \cup G^{\mathcal{M}}_{\mathcal{E}})^{\mathcal{R}_a}\right) \quad \text{(Property 4.8)} \\
&= \text{cert}(q, \langle \emptyset, \mathcal{R}_a, \mathcal{M}_{O^{\mathcal{R}_c}} \cup \mathcal{M}, \mathcal{E}_{O^{\mathcal{R}_c}} \cup \mathcal{E}\rangle)
\end{aligned}$$

□

This lemma effectively "pushes" $\mathcal{R}_c$ reasoning in the set of mappings (to which we add $\mathcal{M}_{O^{\mathcal{R}_c}}$) and the extent (to which we add $\mathcal{E}_{O^{\mathcal{R}_c}}$). Clearly, from the definition of RIS certain answers, $\mathcal{M}_{O^{\mathcal{R}_c}}$ and $\mathcal{E}_{O^{\mathcal{R}_c}}$ provides the explicit and implicit RIS schema triples, while O, $\mathcal{R}_a$, $\mathcal{M}$ and $\mathcal{E}$ provides the explicit and implicit RIS data triples. Next, we rely (as we did for REW-C) on mappings saturation with $O, \mathcal{R}_a$ to also push $\mathcal{R}_a$ reasoning in the mappings, leading to:

Lemma 4.5. *Let $S = \langle O, \mathcal{R}_{\text{RDFS}}, \mathcal{M}, \mathcal{E}\rangle$ be a RIS and q be a BGPQ. Then:*

$$\text{cert}(q, S) = \text{cert}(q, \langle \emptyset, \emptyset, \mathcal{M}_{O^{\mathcal{R}_c}} \cup \mathcal{M}^{\mathcal{R}_a,O}, \mathcal{E}_{O^{\mathcal{R}_c}} \cup \mathcal{E}\rangle)$$

$$
\begin{aligned}
q(x, \text{:pilotOf}) \leftarrow\ & V_{m_1}(x), V_{m_{\prec_{sp}}}(\text{:pilotOf}, \text{:uses}),\\
& V_{m_{\prec_{sc}}}(\text{:StarShip}, \text{:Vehicle}), V_{m_2}(x, a)\\
\cup\ q(x, \text{:pilotOf}) \leftarrow\ & V_{m_1}(x), V_{m_{\prec_{sp}}}(\text{:pilotOf}, \text{:uses}),\\
& V_{m_{\prec_{sc}}}(\text{:Vehicle}, \text{:Vehicle}), V_{m_2}(x, a)\\
\cup\ q(x, \text{:pilotOf}) \leftarrow\ & V_{m_1}(x), V_{m_{\prec_{sp}}}(\text{:pilotOf}, \text{:uses}),\\
& V_{m_{\prec_{sc}}}(\text{:Object}, \text{:Vehicle}), V_{m_2}(x, a)\\
\cup\ q(x, \text{:uses}) \leftarrow\ & V_{m_1}(x), V_{m_{\prec_{sp}}}(\text{:uses}, \text{:uses}),\\
& V_{m_{\prec_{sc}}}(\text{:StarShip}, \text{:Vehicle}), V_{m_2}(x, a)\\
\cup\ q(x, \text{:uses}) \leftarrow\ & V_{m_1}(x), V_{m_{\prec_{sp}}}(\text{:uses}, \text{:uses}),\\
& V_{m_{\prec_{sc}}}(\text{:Vehicle}, \text{:Vehicle}), V_{m_2}(x, a)\\
\cup\ q(x, \text{:uses}) \leftarrow\ & V_{m_1}(x), V_{m_{\prec_{sp}}}(\text{:uses}, \text{:uses}),\\
& V_{m_{\prec_{sc}}}(\text{:Object}, \text{:Vehicle}), V_{m_2}(x, a)\\
\cup\ q(x, \text{:usesWeapon}) \leftarrow\ & V_{m_2}(x, z), V_{m_{\prec_{sp}}}(\text{:usesWeapon}, \text{:uses}),\\
& V_{m_{\prec_{sc}}}(\text{:LightSaber}, \text{:Vehicle}), V_{m_2}(x, a)\\
\cup\ q(x, \text{:usesWeapon}) \leftarrow\ & V_{m_2}(x, z), V_{m_{\prec_{sp}}}(\text{:usesWeapon}, \text{:uses}),\\
& V_{m_{\prec_{sc}}}(\text{:Object}, \text{:Vehicle}), V_{m_2}(x, a)\\
\cup\ q(x, \text{:uses}) \leftarrow\ & V_{m_2}(x, z), V_{m_{\prec_{sp}}}(\text{:uses}, \text{:uses}),\\
& V_{m_{\prec_{sc}}}(\text{:LightSaber}, \text{:Vehicle}), V_{m_2}(x, a)\\
\cup\ q(x, \text{:uses}) \leftarrow\ & V_{m_2}(x, z), V_{m_{\prec_{sp}}}(\text{:uses}, \text{:uses}),\\
& V_{m_{\prec_{sc}}}(\text{:Object}, \text{:Vehicle}), V_{m_2}(x, a)\\
\cup \bigcup_{\mathtt{r} \in \{\prec_{sc}, \prec_{sp}, \hookleftarrow_d, \hookrightarrow_r\}} q(x, \mathtt{r}) \leftarrow\ & V_{m_{\mathtt{r}}}(x, z), V_{m_2}(v, z),\\
& V_{m_{\prec_{sp}}}(\mathtt{r}, \text{:uses}),\\
& V_{m_{\prec_{sc}}}(\text{:LightSaber}, \text{:Vehicle}),\\
& V_{m_2}(x, a)\\
\cup\ q(x, \mathtt{r}) \leftarrow\ & V_{m_{\mathtt{r}}}(x, z), V_{m_2(v,z)},\\
& V_{m_{\prec_{sp}}}(\mathtt{r}, \text{:uses}),\\
& V_{m_{\prec_{sc}}}(\text{:Object}, \text{:Vehicle}),\\
& V_{m_2}(x, a)\\
\cup q(x, \tau) \leftarrow\ & V_{m_2}(x_1, x), V_{m_2}(x_2, \text{:LightSaber}), V_{m_{\prec_{sp}}}(\tau, \text{:uses}),\\
& V_{m_{\prec_{sc}}}(\text{:LightSaber}, \text{:Vehicle}), V_{m_2}(x, a)\\
\cup q(x, \tau) \leftarrow\ & V_{m_2}(x_1, x), V_{m_2}(x_2, \text{:Object}), V_{m_{\prec_{sp}}}(\tau, \text{:uses}),\\
& V_{m_{\prec_{sc}}}(\text{:Object}, \text{:Vehicle}), V_{m_2}(x, a)
\end{aligned}
$$

Figure 4.5: Sample rewriting for Example 4.10.

This lemma is a direct consequence of Theorem 4.5. This allows to reduce RIS query answering to relational view-based query rewriting (step (2") in Figure 4.3):

Theorem 4.4 (REW correctness)**.** *Let $S = \langle O, \mathcal{R}_{\text{RDFS}}, \mathcal{M}, \mathcal{E} \rangle$ be a RIS and q be a BGPQ. Then:*

$$\text{cert}(q, S) = \text{cert}(bgpq2cq(q), \text{Views}(\mathcal{M}_{O^{\mathcal{R}_c}} \cup \mathcal{M}^{\mathcal{R}_a, O}), \mathcal{E}_{O^{\mathcal{R}_c}} \cup \mathcal{E})$$

Example 4.10 (REW)**.** *Consider again the RIS in Example 4.3 and the query q of Example 4.6 seen as a CQ:*

$$
\begin{aligned}
q(x, y) \leftarrow\ & T(x, y, z), T(z, \tau, t), T(y, \prec_{sp}, \text{:uses}), T(t, \prec_{sc}, \text{:Vehicle}),\\
& T(x, \text{:uses}, a), T(a, \tau, \text{:LightSaber})
\end{aligned}
$$

Its maximally-contained rewriting q_{REW} based on the LAV mappings obtained from the RIS saturated *mappings and* ontology *mappings appears in Figure 4.5. It shows a rewriting containing 22 CQs, which is not minimized, in order to ease the understanding of the rewriting process. Nevertheless, we remark that the 6th CQ is contained in the 10th one and the CQs from 7th to 10th contain a redundant atom using the view symbol V_{m_2}. This rewriting is much larger than the ones of the two preceding techniques: this is due to the ontology mappings. If we assume that $\mathcal{E}$ also contains V_{m_2}(:Luke, :a), as we did in Example 4.6, we obtain again* $\text{cert}(q, S) = \{\langle$:Luke, :pilotOf$\rangle\}$*, which results from the evaluation of the first CQ in the UCQ rewriting; the other CQs yield empty results because some required $\prec_{sc}$ or $\prec_{sp}$ constraints are not found in the extensions of RIS ontology mappings.*

How do our strategies compare? Since they are all correct, they lead to the same RIS certain answer set, however they do not necessarily compute the same view-based rewritings. Indeed, REW considers the additional set $\mathcal{M}_{O^{\mathcal{R}_c}}$ of ontology mappings. Hence, for queries over the ontology, i.e., featuring in a property position $\prec_{sc}$, $\prec_{sp}$, $\hookleftarrow_d$, $\hookrightarrow_r$, or a variable, a REW rewriting is larger than a REW-CA or REW-C rewriting and, to be answered, requires the additional ontology source modeled by $\mathcal{E}_{O^{\mathcal{R}_c}}$. In contrast, REW-CA and REW-C yield logically equivalent rewritings; we minimize them both to avoid possible redundancies, thus they become identical (up to variable renaming). Hence, REW-CA and REW-C do not differ in how these rewritings are evaluated. Instead, they differ in how the rewritings are *computed*, or, equivalently, on *the distribution of the reasoning effort on the data and mappings, across various query answering stages*. As our experiments show, given the computational complexity of view-based query rewriting [Pottinger and Halevy, 2001], this difference has a significant impact on their performance.

4.3.6 Remarks on related techniques

We discuss here how GLAV mappings could be simulated by GAV mappings and the drawbacks of this solution. We also relate our mapping saturation technique to a similar technique developed in the context of OBDA with description logics.

Simulation of GLAV mappings by GAV mappings

To some extent, GLAV mappings may be simulated by GAV mappings provided with so-called Skolem functions on answer variables, as suggested for instance in [De Giacomo et al., 2018]. The name "Skolem function" comes from the Skolemization operation in first-order logic, which replaces all existentially quantified variables in a formula (in a specific form) with functional terms, using a fresh set of (Skolem) function symbols. By analogy, existential variables in the head of a mapping may be replaced by functions on the answer variables of the head.

To illustrate, consider the GLAV mapping from Example 4.1 (we omit here the function f_1 on x to focus on the functions that will be introduced):

$$m_1 = q_1(x) \rightsquigarrow (x, \text{:pilotOf}, y), (y, \tau, \text{:StarShip})$$

The existential variable y could be replaced by a Skolem function $f(x)$, which would yield two GAV mappings, namely m_1^1 and m_1^2, defined by:

$$m_1^1 = q_1(x) \rightsquigarrow (x, \text{:pilotOf}, f(x))$$

$$m_1^2 = q_1(x) \rightsquigarrow (f(x), \tau, \text{:StarShip}).$$

In a materialization scenario, note that Skolem functions would have to produce syntactically correct RDF values. Still in a materialization scenario, query answering would require some post-processing to prevent the values built by the Skolem functions to be accepted as answers.

In a query rewriting scenario, Skolem values would also have to be dealt with in a special way, which in particular prevents to use off-the-shelf view-based query rewriting algorithms. Hence, value invention would be simulated here at the price of technically more complex mappings and processing. Second, the break-up of GLAV mappings into several GAV mappings would lead to higher conceptual complexity since intrinsically connected triples, as those associated with $(x, \text{:pilotOf}, y)$ and $(y, \tau, \text{:StarShip})$ in the example, could not be exposed together by a single mapping. Last but not least, it leads to less efficient query rewriting algorithms. In particular, using such GAV mappings to rewrite a query according to LAV mappings has been proposed by the Inverse-rules algorithm [Abiteboul et al., 2011], however this technique produces highly redundant rewritings and is considerably slower than the Minicon algorithm, which directly works on LAV mappings [Pottinger and Halevy, 2001].

Mapping saturation

As already mentioned, our mapping saturation (Definition 4.15) is inspired by a query saturation technique introduced in [El Hassad et al., 2017] to compute least general generalizations of BGPQs under RDFS background knowledge.

However, it appears that it can be seen as a generalization to GLAV mappings of the $\mathcal{T}$-mapping technique introduced in [Rodriguez-Muro et al., 2013] (and further developed in [Sequeda et al., 2014]) to optimize query rewriting in a classical OBDA context with description logics. The $\mathcal{T}$-mapping technique consists of completing the original set of GAV mappings with new ones, encapsulating information inferred from the DL ontology. For instance, given a GAV mapping:

$$m = q_1(x) \rightsquigarrow C(x)$$

with C a class, and a DL constraint specifying that C is a subclass of D, a new mapping:

$$m' = q_1(x) \rightsquigarrow D(x)$$

is created by composing m and the DL constraint. On this example, we would saturate the head of m into $q_2^{\mathcal{R},O}(x) \leftarrow C(x) \wedge D(x)$, which is semantically equivalent to adding the mapping m'. However, when mappings are GLAV and not GAV, one cannot simply add new mappings. For instance, consider the GLAV mapping:

$$m_1 = q_1(x) \rightsquigarrow (x, \text{:pilotOf}, y), (y, \tau, \text{:StarShip})$$

Given the entailment rule `rdfs9` and the ontological triple $(\text{:StarShip}, \prec_{sc}, \text{:Vehicle})$, the saturation of m_1 adds the triple $(y, \tau, \text{:Vehicle})$ to its head. Creating instead a new mapping of the form $m_1' = q_1(\bar{x}) \rightsquigarrow (y, \tau, \text{:Vehicle})$ would of course be unsatisfactory as y in m_1' should correspond to the same object as y in m_1.

4.3.7 Landscape of query answering strategies

We now propose a structured overview of the query answering strategies enabled by the previously exposed RIS transformations and query reformulations to handle the reasoning w.r.t. $\mathcal{R}_{\text{RDFS}}$ In this section, we present respectively nine and four strategies, following the materialization and, respectively, the rewriting-based query answering approaches; among these are the strategies detailed in the previous subsections. We explore these strategies by introducing a general, extensible framework, using small transformations of a *query answering problem on RIS*.

Definition 4.12 (QA problem)**.** *A* Query Answering (QA) problem *is pair* (q, S) *of a BGPQ* q *and a RIS* $S = \langle O, \mathcal{R}, \mathcal{M}, \mathcal{E} \rangle$.

Solving (q, S) means finding the certain answers of q in S. A QA strategy is a fixed method to solve a class of QA problems. A QA strategy can start by *QA transformations* which turn the original QA problem into a equivalent, slightly different one.

Definition 4.13 (QA transformation)**.** *A* Query Answering (QA) transformation *is defined by two applications* f *and* g*. It takes as input a QA problem* (q, S) *and returns the new QA problem* $(f(q, S), g(S))$.

We say that a QA transformation defined by f *and* g *is* safe for a class of QA problems*, if for any QA problem* (q, S) *of that class, the certain answers of* q *in* S *are equal to the certain answers of* $f(q, S)$ *in* $g(S)$.

Above, we have investigated QA strategies for the class of QA problems defined by the hypothesis presented at the beginning of Section 4.3. These QA strategies are implicitly based on safe QA transformations, allowing to distribute the reasoning w.r.t. $\mathcal{R}_{\text{RDFS}}$ in the query or inside the RIS components. We can group these QA transformations in three ordered categories:

1. **query reformulation** allows to handle the reasoning w.r.t. $\mathcal{R}_a$ (respectively, $\mathcal{R}_c$ and $\mathcal{R}_{\text{RDFS}}$) as introduced in Section 3.2. The QA transformation is defined as follows:
   - $f(q, S)$ returns Q_a (respectively, Q_c and $Q_{c,a}$), the reformulation of q w.r.t. the ontology of S;
   - $g(S)$ returns $\langle O, \mathcal{R} \backslash \mathcal{R}_a, \mathcal{M}, \mathcal{E} \rangle$ (respectively, $\langle O, \mathcal{R} \backslash \mathcal{R}_c, \mathcal{M}, \mathcal{E} \rangle$ and $\langle \emptyset, \emptyset, \mathcal{M}, \mathcal{E} \rangle$)
2. **mappings transformation** allows to handle the reasoning w.r.t. $\mathcal{R}_a$ by mappings saturation (Definition 4.10) (respectively, w.r.t. $\mathcal{R}_c$ by adding of ontology mappings and their extent (Definition 4.11)). The QA transformation is defined as follows:
   - $f(q, S) = q$;
   - $g(S)$ returns $\langle O, \mathcal{R} \setminus \mathcal{R}_a, \mathcal{M}^{\mathcal{R}_a, O}, \mathcal{E} \rangle$ (respectively, $\langle \emptyset, \mathcal{R} \setminus \mathcal{R}_c, \mathcal{M} \cup \mathcal{M}_{O^{\mathcal{R}_c}}, \mathcal{E} \cup \mathcal{E}_{O^{\mathcal{R}_c}} \rangle$).
3. **materialization saturation** allows, only for materialization-based approaches, to handle the reasoning w.r.t. $\mathcal{R}$ among $\mathcal{R}_a$, $\mathcal{R}_c$ and $\mathcal{R}_{\text{RDFS}}$. The QA transformation is defined as follows:
   - $f(q, S) = q$;

- $g(S)$ returns $\langle \emptyset, \emptyset, \{m_t\}, \left(G^{\mathcal{M}}_{\mathcal{E}} \cup O\right)^{\mathcal{R}} \rangle$, where the fourth input of the RIS, instead of the mappings extent, is an RDF graph considered as a datasource (as outlined below Definition 4.6) and m_t is the mapping defined by $q_t(s,p,o) \rightsquigarrow q_t(s,p,o)$, with q_t the BGPQ defined by $q_t(s,p,o) \leftarrow (s,p,o)$.

To fully handle the RDFS reasoning, a QA strategy needs to apply on a QA problems: either two QA transformations (one for $\mathcal{R}_a$ and another for $\mathcal{R}_c$) or only one for $\mathcal{R}_{\text{RDFS}}$, in the order of the categories to which they belong. The order is required, to avoid that a first QA transformation removes an input required by a subsequent one. For example, the QA strategy REW-C starts by chaining the QA transformation w.r.t. $\mathcal{R}_c$, in the query reformulation category, with the one w.r.t. $\mathcal{R}_a$, in the mapping transformation category. After application of those QA transformations, the resulting QA problem (q', S') is such that the rule set of S is empty, as well as its ontology. A QA strategy takes this QA problem and reduces it to:

- following the **materialization-based approach**, the (restricted) evaluation problem of q' in the RIS induced graph of S',
- following the **rewriting-based approach**, the view-based query answering problem of q' using the mappings of S' as LAV mappings and their extent.

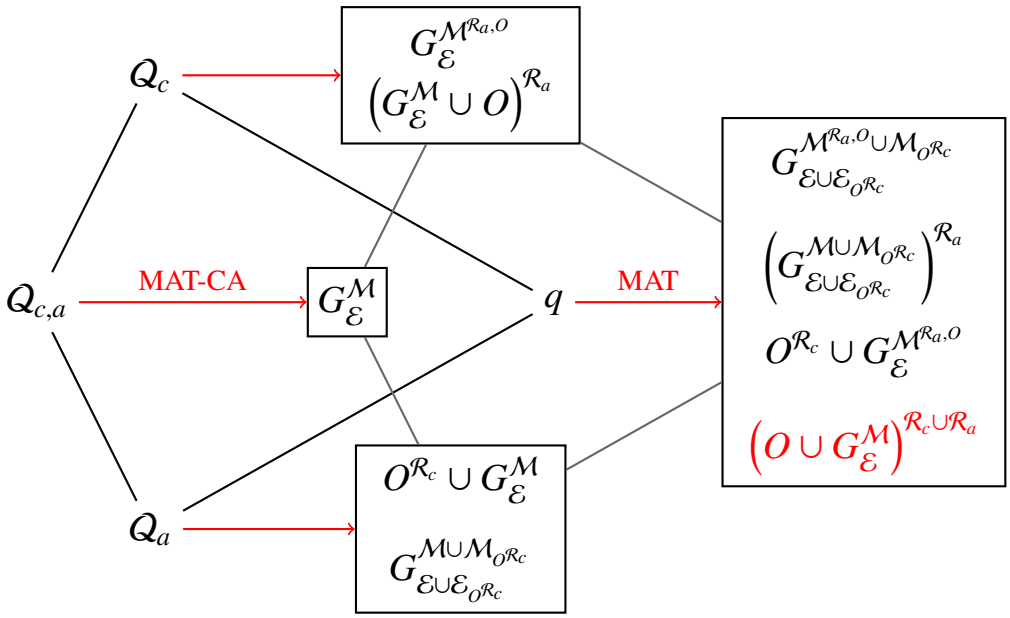


Figure 4.6: Possible RIS query answering strategies based on graph materialization.

Figure 4.6 and Figure 4.7 show the thirteen reduced QA problems obtained using the QA strategies based on the QA transformations outlined above, through a materialization- or a rewriting-based approach. In each figure, the pair formed by the query at the beginning of an red arrow and by one of the elements in the pointed box represents the QA problem resulting from the QA transformations applied by a QA strategy (induced graphs are displayed instead of RIS for materialization-based approaches).

In Figure 4.6 and Figure 4.7, two queries are connected, if there is a query reformulation w.r.t. $\mathcal{R}_a$ or $\mathcal{R}_c$ allowing to reformulate one into the other. For example, in Figure 4.6,

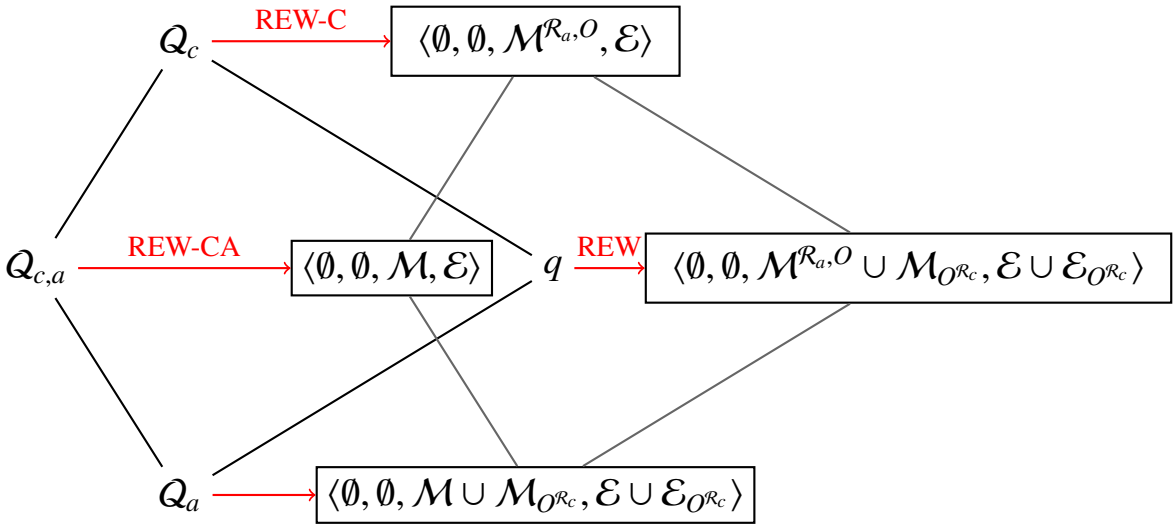


Figure 4.7: Possible RIS query answering strategies based on query rewriting.

one can find the materialization-based query answering strategy MAT (Section 4.3.1) represented by $q{\rightarrow}(O \cup G^{\mathcal{M}}_{\mathcal{E}})^{\mathcal{R}_a \cup \mathcal{R}_c}$. We can see that the strategy MAT is based on the QA transformation w.r.t. $\mathcal{R}_{\text{RDFS}}$ in the materialization category.

The query answering strategies detailed in Section 4.3.1-4.3.5 are presented by their labels, above each corresponding arrow.

In Figure 4.7, the unlabelled strategy proposed to rewrite the reformulated query Q_a w.r.t. the mappings $\mathcal{M}^{\mathcal{R}_a,O} \cup \mathcal{M}_{O^{\mathcal{R}_c}}$. It is easy to see that this strategy has the same drawback as REW (Section 4.3.5), since Q_a can contain triples matching the ontology and we rewrite it using a mapping set containing the ontology mappings.

The decomposition (classification) of query answering strategies presented in this section (by means of QA transformations) has guided our design for the strategies presented in Sections 4.3.1-4.3.5, which we have actually implemented and experimented with within OBI-WAN. The next section provides more information about the implementation.

4.4 A Platform for RDF Integration Systems: OBI-WAN

We developed a system, called OBI-WAN, implementing the five RIS query answering strategies detailed above. OBI-WAN is developed in Java 1.8 (18K lines of code), and leverages existing software to perform the different query answering steps.

For **materialization-based query answering** strategies, including MAT and MAT-CA, we use OntoSQL[1], a Java platform providing efficient RDF storage, saturation w.r.t. $\mathcal{R}_{\text{RDFS}}$, and query evaluation on top of an RDBMS [Bursztyn et al., 2015, Goasdoué et al., 2013], relying on Postgres v9.6. To save space, OntoSQL encodes IRIs and literals into integers, and a dictionary table which allows going from one to the other. It can store the resources following the three storage layout T, CP and TCP introduced in Section 3.3; for each layout, all tables are completely indexed. OntoSQL also contains a implementation of the query reformulation algorithms w.r.t. $\mathcal{R}_c$, $\mathcal{R}_a$ and $\mathcal{R}_{\text{RDFS}}$ introduced in Section 3.2; this implementation is used both for materialization and rewriting-based approaches.

[1] `https://ontosql.inria.fr`

We rely on the Graal engine [Baget et al., 2015] for **view-based query rewriting**. Graal is a Java toolkit dedicated to query answering algorithms in knowledge bases with existential rules (a.k.a. tuple-generating dependencies). Since the LAV mapping $V_m(\bar{x}) \rightarrow bgp2ca(\text{body}(q_2))$ corresponding to a GLAV mapping m (recall Def. 4.9) can be seen as a specific existential rule, the query reformulation algorithm of Graal can be used to rewrite the UCQ translation of a BGPQ with respect to a set of RIS mappings. We specialized Graal code to exploit obvious specificities of RIS mappings, such as the fact that they correspond to *source-to-target TGDs*.

Finally, to **execute queries against heterogeneous data sources**, we use Tatooine [Bonaque et al., 2016, Alotaibi et al., 2019], a Java-based mediator (or polystore) system handling JSON, relational, key-value and RDF data (based on MongoDB, Postgres, Redis and Jena TDB, respectively), capable (unlike other polystores, e.g., [Duggan et al., 2015]) of evaluating joins within the mediator engine. We implemented new optimizations inside Tatooine, allowing it to push queries in underlying data sources.

4.4.1 Query answering in Obi-Wan

The inputs given to Obi-Wan are:

- a list of **BGP queries**,
- a **Query Answering (QA) strategy** characterized by a list of QA transformations and a QA approach (Section 4.3.7);
- a **RIS specification**, which is a JSON document; it can also be a R2RML file in the particular case when the mappings are GAV and the sources are relational. The RIS specification contains the data sources description, the RDFS ontology specified as a file containing ontology triples, and the mappings description. Each such description specifies: its data sources, its body query, and its functions which transform each answer of the mapping body into tuples of IRIs, literals or blank nodes.

  Concretely, we implemented these functions with the help of **string patterns**, that assign a common prefix to all the values built from the bindings of a given variable. For instance, the string pattern <`https://starwars.com/{id}`> is used to create IRIs out of bindings of the variable *id*. Thus, from the binding `luke`, we obtain the IRI <`https://starwars.com/luke`>.

  Optionally, a mapping can specify a *primary key constraint* known to hold on its extent.

At **preprocessing time**, Obi-Wan loads the RIS and applies on it the functions g defined by the QA transformations (Definition 4.13) from the input QA strategy. The output RIS is called *effective RIS*. In case of a materialization-based approach, a RIS materialization is performed as follows. First, each mapping is triggered on its data source, and the resulting triples are stored in a file (fresh blank nodes introduced by existential variables in mapping head use a common prefix, e.g., _:*exVar*). Second, triples from this file are loaded into OntoSQL, which saturates the resulting graph, if it is specified by a QA transformation.

At **query time**, Obi-Wan processes the input BGP queries in order. It applies the function f of the QA transformations on the current BGP query and on the appropriate

RIS, and returns a (reformulated) query in the form of (U)BGPQ. Depending on the QA approach, OBI-WAN computes the certain answers:

- in a **materialization-based approach**, by passing the (reformulated) query to OntoSQL, which evaluates it on the materialization of induced graph of the effective RIS, and returns the certain answers, from which the answers containing the blank nodes prefixed by $\_:exVar$ have been removed,
- in a **rewriting-based approach**, by rewriting the (reformulated) query using Graal (as described above). This returns the *query rewriting*, a UCQ on the views V_m of the mappings of the effective RIS, which is unfolded using the q_1 parts of the mappings (see Section 2.2.2), to form a *mediated plan* executed by Tatooine, which returns the certain answers.

In a rewriting-based approach, the query rewriting and the mediated plan are optimized before being, respectively, unfolded and executed. These optimizations are described in the next section.

4.4.2 Query rewriting and mediated plan optimizations

We have devised and implemented some optimizations for pruning the CQs in the query rewriting and simplifying (thus, improving the performance of) the mediated plan.

Query rewriting optimizations

The query rewriting is a UCQ on the views V_m. Before its unfolding, some optimizations are applied on it, in the following order:

1. **Chasing with primary key constraints** As mentioned above, the RIS specification can optionally define a primary key constraint on each view V_m. When present, such constraints allow to infer equalities between the variables of each CQ in the query rewriting. They make the following optimizations more important.
2. **Elimination of unsatisfiable joins** In a RIS, the columns content of each view V_m (described by the extension of the mapping m, see Definition 4.4) is the output of $f_1, \ldots f_n$, the functions of m. The CQs of the query rewriting lead to empty results on the data sources, if they contain a join of two views on columns, whose associated functions have disjoint ranges. This disjointness can be easily inferred thanks to the fact that our functions are implemented using string patterns (Section 4.4.1). Such CQs are removed from the query rewriting union.
3. **Union minimisation** Any CQs in the query rewriting, which are contained within another such CQ, is removed, since its results can be obtained without it.
4. **CQs minimisation** Each CQ in the query rewriting is minimized, by eliminating the redundant view atoms it contains.

Mediated plan optimizations

Before sending to Tatooine a mediated plan on the heterogeneous data sources, OBI-WAN unfolds the optimized query rewriting, especially the views V_m, using the functions and the body of each mapping m. We illustrate this on the mediated plan displayed at the top

of the Figure 4.8. This plans results from a query rewriting in a heterogeneous RIS (more specifically, S_3 in Section 4.5), defined using MongoDB and Postgres data sources.

Before any optimization, mediated plans have a general shape that directly mirrors, at the physical algebra level, a union of conjunctive queries. From top to bottom, such a plan comprises: a *distinct operator* removing duplicate answers, followed by a *union operator* beneath which lies one plan for each CQ in the rewriting query. The root of each such plan is a *projection operator* keeping only the answers from the tuples returned by the *join operators* connected in a tree (usually, a left-deep one), to join sub-plans returning mapping extensions. These sub-plans are at the leaf level in the mediated plan. Each sub-plan consists *operator evaluating a mapping body (q_1) on its data source*, whose outputs are input to another operator that *applies the mapping functions* to return the mapping extensions. A *selection operator* may be inserted between one of these sub-plans and its parent (a join operator), when a query rewriting contains a constant.

In the example, at the top of the Figure 4.8, the leaves of the plan, from left to right, are two SQL queries whose evaluation is delegated to Postgres, followed by two queries evaluated in MongoDB. Over these operators, functions translate the answers into tuples of serialized IRIs, literals and blank nodes using string patterns. In the example, a selection operator is the left child of the left-most join operator, selecting the tuples in which the sixth column is equal to the literal "4".

These mediated plans are not efficiently evaluated by Tatooine, for the following reasons.

- The plans may extract large volumes of data from the sources, only to eliminate them later through selections, projections, duplicate eliminations etc. This suggests to push (sub)queries in the data sources, in order to eliminate such useless values early on. Query pushing within data sources is a well-known heuristic [Özsu and Valduriez, 2011].

- Since the function call operators are parents of a query evaluation operator, each answer of a mapping body leads to a function invocation. Or, in some cases, many of these function outputs are discarded by others operators higher up in the plan (notably selections, projections and distinct operators).

- The operators of the mediated plan are often performed more efficiently by data source query engines than by Tatooine, especially because the former can often use indexes on the data.

Therefore, **we implemented a logical plan optimizer for Tatooine**, which allows to:

- move and merge the mapping function invocation operators, in one operator at the root of the mediated plan, when possible, by reversing the order between these operators and their parents,

- push, to the extent possible, the others operators into the data source query operators, by pushing operators whose children are query evaluation operators on the same data sources. In some cases, this also requires re-ordering joins in the mediated plan.

In our example, at the bottom in Figure 4.8 we show the previous mediated plan example after applying the optimizations. Note that the optimized plan contains only one join between two query evaluation operators on MongoDB and on Postgres. The others

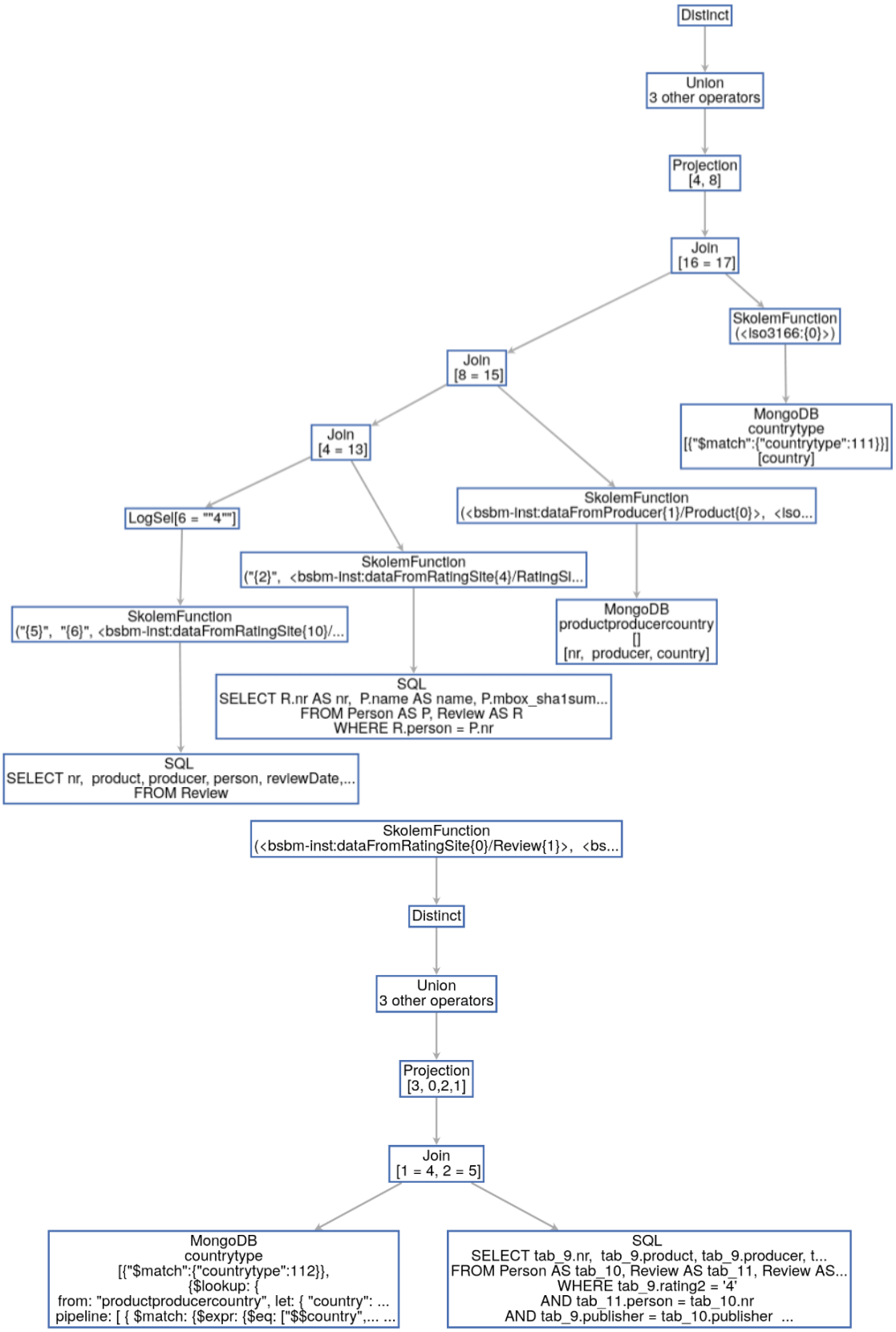


Figure 4.8: A mediated plan (top), the same plan after optimizations (bottom).

joins have been pushed in these remaining query evaluation operators. Also, the mapping function operator is now at the root of the mediated plan, and the selection has been pushed in the SQL query (visible on the first line of the WHERE statement). These **optimizations considerably reduce the evaluation time of mediated plans** in Tatooine[2], allowing to perform experiments, where query answering times of materialization and rewriting-based approaches can be compared.

4.5 Experimental evaluation

We now describe our experiments with RIS query answering using the strategies outlined in Section 4.3 and implemented by Obi-Wan.

Hardware We used servers with 2,7 GHz Intel Core i7 processors and 160 GB of RAM, running CentOs Linux 7.5.

4.5.1 Experimental scenarios

RDF Integration Systems

Our first goal was to study scalability of RIS query answering, in particular in the *relational* setting studied in many prior works. To achieve this, we used the BSBM benchmark relational data generator[3] to build databases consisting of 10 relations named producer, product, offer, review etc. Using two different benchmark scale factors, we obtained a data source DS_1 of 154.054 tuples across the relations, respectively, DS_2 of 7.843.660 tuples; both are stored in Postgres.

We used two RDFS ontologies O_1 respectively O_2, containing, first, subclass hierarchies of 151 (resp. 2011) product types, which come with DS_1, respectively, DS_2. To O_1 and O_2, we add a natural RDFS ontology for BSBM[4] composed of 26 classes and 36 properties, used in 40 subclass, 32 subproperty, 42 domain and 16 range statements.

Relational-sources RIS

We devised two sets $\mathcal{M}_1, \mathcal{M}_2$ of 307, respectively, 3863 mappings, which expose the relational data from DS_1, respectively, DS_2 as RDF graphs. The relatively high number of mappings is because: (*i*) each product type (of which there are many, and their number scales up with the BSBM data size) appears in the head of a mapping, enabling fine-grained and high-coverage exposure of the data in the integration graph; (*ii*) we also generated more complex GLAV mappings, partially exposing the results of join queries over the BSBM data; interestingly, these mappings expose incomplete knowledge, in the style of Example 4.2.

The mapping sets lead to the **RIS graphs of** $2.0 \cdot 10^6$, respectively, $108 \cdot 10^6$ triples. Their **saturated** versions comprise respectively $3.4 \cdot 10^6$ and $185 \cdot 10^6$ triples. Our first two RIS are thus: $S_1 = \langle O_1, \mathcal{R}_{\text{RDFS}}, \mathcal{M}_1, \mathcal{E}_1 \rangle$ and $S_2 = \langle O_2, \mathcal{R}_{\text{RDFS}}, \mathcal{M}_2, \mathcal{E}_2 \rangle$, where $\mathcal{E}_i$ for i in $\{1, 2\}$ are the extents resulting from DS_i and $\mathcal{M}_i$.

[2]See more details at: https://pages.saclay.inria.fr/maxime.buron/posts/tatooine-cq-plan-optimization/index.html

[3]https://downloads.sourceforge.net/project/bsbmtools/bsbmtools/bsbmtools-0.2

[4]https://pages.saclay.inria.fr/maxime.buron/projects/het2onto-benchmark/bsbm/onto-core/bsbm-onto.png

Heterogeneous-sources RIS

To build a heterogeneous RIS, we converted a third (33%) of DS_1, DS_2 into JSON documents, and stored them into MongoDB, leading to the JSON data sources denoted $DS_{j,1}, DS_{j,2}$; the relational sources $DS_{r,1}, DS_{r_2}$ store the remaining (relational) data. Conceptually, for i in $\{1, 2\}$, the extension based on $DS_{r,i}$ and extension based on $DS_{j,i}$ form a partition of $\mathcal{E}_i$.

We devised a set of **JSON-to-RDF mappings** to expose $DS_{j,1}$ and $DS_{j,2}$ into RDF, and denote $\mathcal{M}_3$ the set of mappings exposing $DS_{r,1}$ and $DS_{j,1}$, together, as an RDF graph; similarly, the mappings $\mathcal{M}_4$ expose $DS_{r,2}$ and $DS_{j,2}$ as RDF. Our last two RIS are thus: $S_3 = \langle O_1, \mathcal{R}_{\text{RDFS}}, \mathcal{M}_3, \mathcal{E}_3 \rangle$ and $S_4 = \langle O_2, \mathcal{R}_{\text{RDFS}}, \mathcal{M}_4, \mathcal{E}_4 \rangle$, where $\mathcal{E}_3$ is the extent of $\mathcal{M}_3$ based on $DS_{r,1}$ and $DS_{j,1}$, while $\mathcal{E}_4$ is the extent of $\mathcal{M}_4$ based on $DS_{r,2}$ and $DS_{j,2}$. *The RIS induced graph and ontology triples of S_1 and S_3 are identical; thus, the difference between these two RIS is only due to the heterogeneity of their underlying data sources*. More precisely, $\mathcal{M}_1$ and $\mathcal{M}_3$ contains the same number of mappings and mappings in $\mathcal{M}_3$ on the relation source $DS_{r,1}$ are also in $\mathcal{M}_1$ and the others mappings in $\mathcal{M}_3$ are build from mappings in $\mathcal{M}_1$, in which the mapping body, a relational query on DS_1, is replaced by a query on $DS_{j,1}$ having the same answers. Thus the mappings extent $\mathcal{E}_1$ and $\mathcal{E}_3$ contains the set of tuples stored in different views. The same holds for S_2 and S_4.

Queries

We devised a set of 28 BGP queries having from 1 to 11 triple patterns (5.5 on average), of varied selectivity (they return between 2 and $330 \cdot 10^3$ results in S_1 and S_3 and between 2 and $4.4 \cdot 10^6$ results in S_2 and S_4); 6 among them query the data *and* the ontology, a capability which most others systems lack. Table 4.2 reports three query properties impacting query answering performance: the number of induced triples (N_{TRI}), the number of BGPQ reformulations on the ontology ($|Q_{c,a}|$, ranging from 1 to 1225; this strongly determines the performance of answering such large union queries, recall Example 4.6), and its number of answers (N_{ANS}) on the two RIS groups (S_1, S_3 and S_2, S_4). To further study the impact of the ontology on query evaluation complexity, we created *query families* denoted Q_X, Q_{Xa} etc. by replacing the classes and properties appearing in Q_X with their super classes or super properties in the ontology. In such a family, *Q_X is the most selective, and queries are sorted in the increasing order of their number of reformulations*.

Our ontologies, mappings, queries, and experimental details are available online[5]. The queries also appear in Appendix A.3.1.

4.5.2 Query answering performance

REW inefficiency

We have conducted experiments using our six queries on ontological triples showing, as in Example 4.10 and Figure 4.5, an explosion of the size of the rewriting (number of CQs), compared to the rewriting produced by the two other approaches.

On queries (also) over the ontology, as explained in Section 4.3.5, we noted that the size of the rewriting produced by REW is larger (by a multiplicative factor of 29 to 74 in S_1 and S_3, and of 33 to 969 in S_2 and S_4) than the rewritings of the two other strategies,

[5]Experiment web site: `https://gitlab.inria.fr/mburon/org/blob/master/projects/het2onto-benchmark/bsbm/`

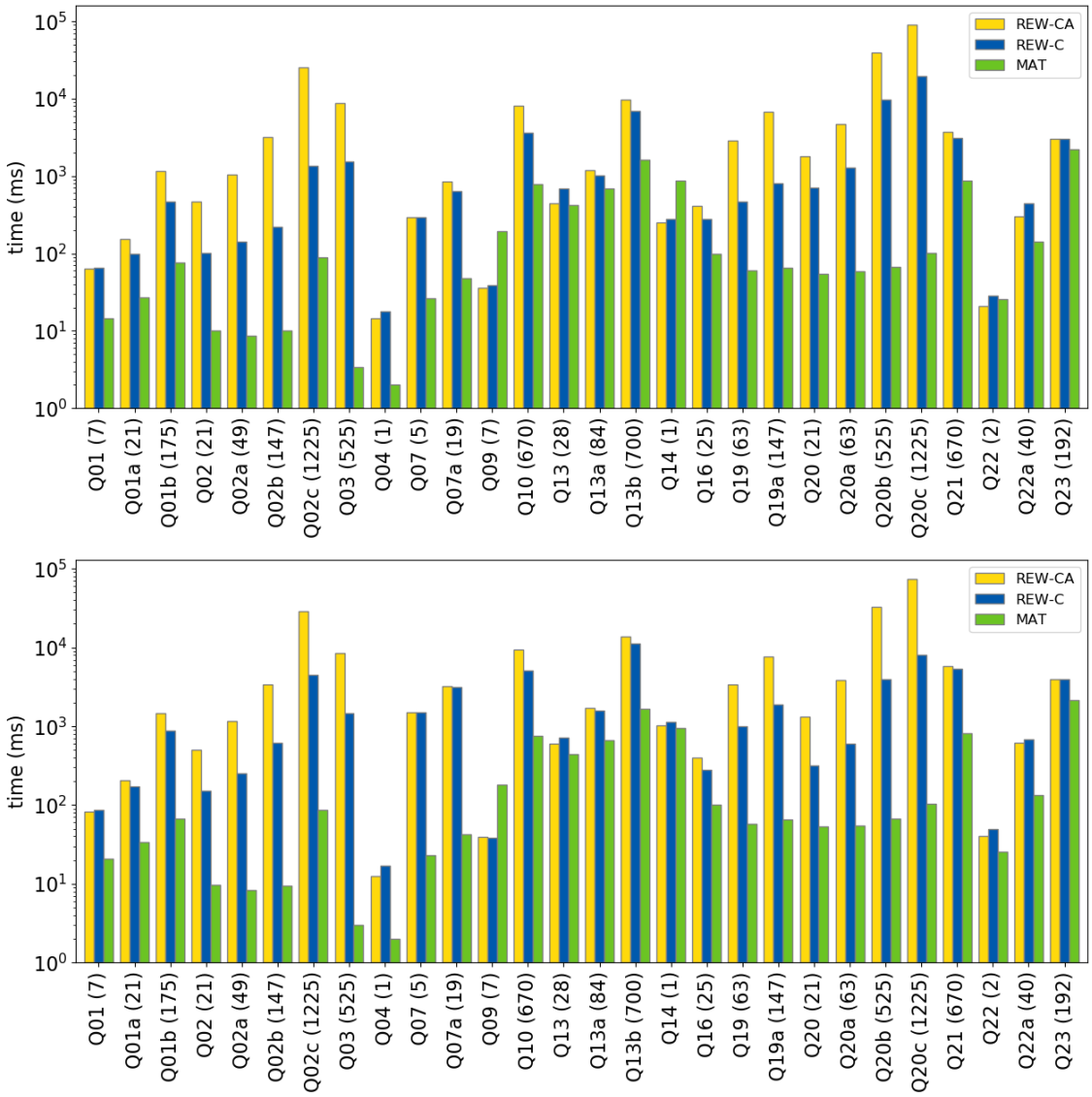


Figure 4.9: Query answering times on the smaller RIS S_1 (top, relational sources) and S_3 (bottom, heterogeneous sources).

which led to an explosion of the time spent minimizing the rewriting, and made REW overall unfeasible; the details of these tests can be found in Appendix A.3.2.

On queries that do not carry over the ontology, REW produces the same rewritings as the other methods. Thus, we do not report further REW performance below.

Query answering time comparison

Figure 4.9 depicts the query answering times, on the smaller RIS, of REW-CA, REW-C and MAT. *The size of (number of BGPQs in) the reformulation of each query w.r.t. $\mathcal{R}$, $|Q_{c,a}|$ appears in parentheses after the query name, in the labels along the x axis*. Given that S_1, S_3 have the same RIS data triples, the MAT strategy coincides among these two RIS. Figure 4.10 shows the corresponding times for the largest RIS S_2 and S_4; the same observations apply. Note the logarithmic time axes.

A first observation is that *our query set is quite diverse*; their answering times range from a few to more than 10^5 ms.

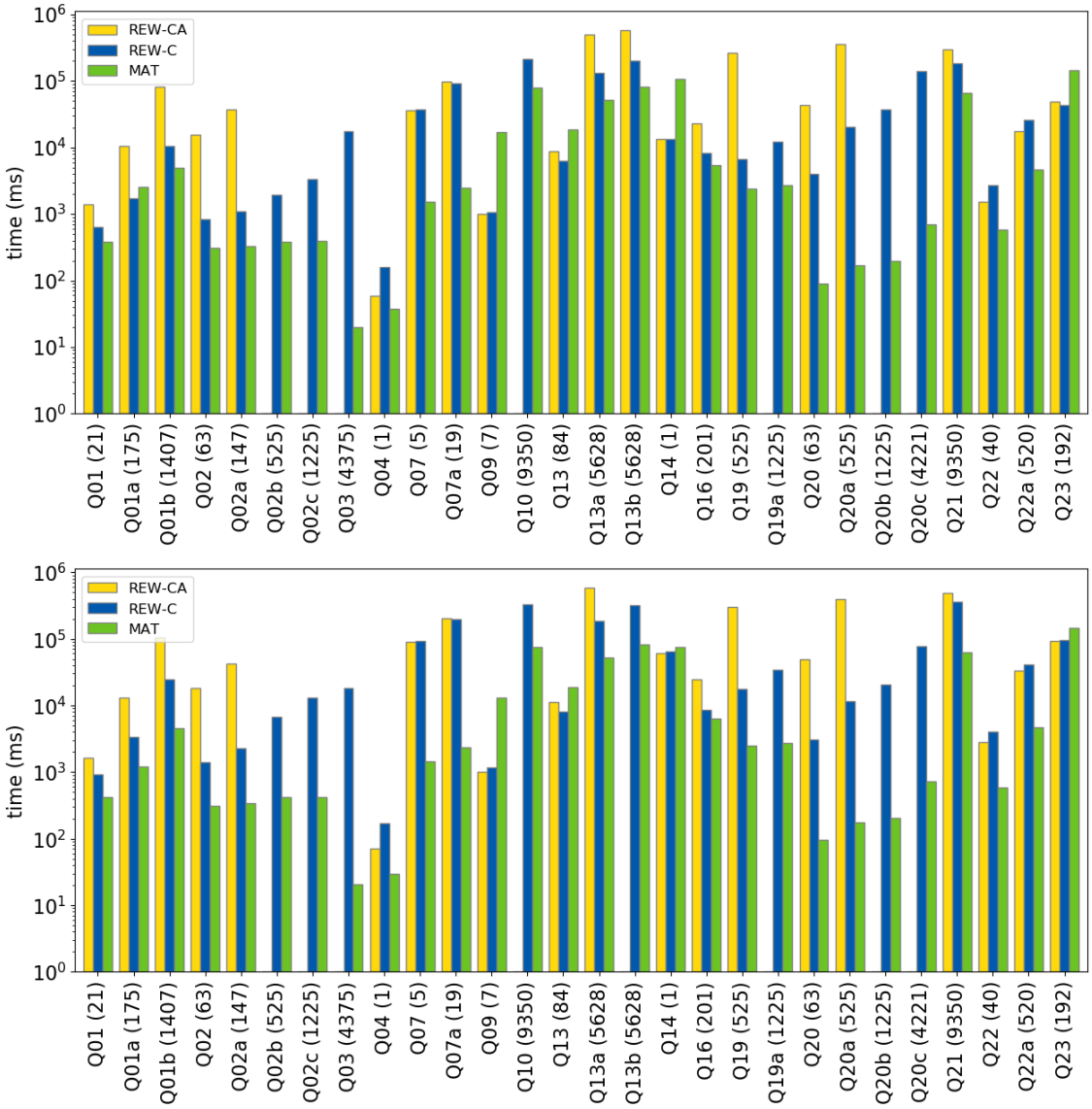


Figure 4.10: Query answering times on the larger RIS S_2 (top, relational sources) and S_4 (bottom, heterogeneous sources).

Strategy performance analysis

As expected, query answering in MAT *is the fastest in most cases*, since it has no reasoning work to do at query answering time. However, it required, for S_1, S_3, $1.2 \cdot 10^5$ ms to build the materialization and $1.49 \cdot 10^5$ ms more to saturate it, whereas for S_2, S_4, these times are 14h46 ($5.31 \cdot 10^7$ ms), respectively, 1h28 ($5.28 \cdot 10^6$ ms). Not only these are *orders of magnitude more than all query answering times*; recall also that materializing $G^{\mathcal{M}}_{\mathcal{E}}$ requires maintaining it when the underlying data changes, and its saturation $(G^{\mathcal{M}}_{\mathcal{E}} \cup O)^{\mathcal{R}_{\text{RDFS}}}$ needs a second level of maintenance. Thus, MAT is not practical when data sources change.

We were surprised to see REW-C and REW-CA somehow faster than MAT for queries Q_{09} and Q_{14}. Answering these queries through MAT within OntoSQL leads to producing many results that involve *mapping-generated blank nodes*, tuples which should not appear in our certain answers, as per Definition 4.8. We remove such tuples in post-processing mode, which leads to a performance overhead for MAT. REW-C and REW-CA, in contrast,

are answered by evaluating rewritings, and do not have to apply such a result pruning. It remains to be seen if this pruning could be pushed in an RDFDB; note that not *all* answers including blank nodes should be pruned, only those whose blank nodes are due to mappings.

In each scenario, we observe that REW-C is faster or takes as long as REW-CA. Since the two approaches produce the same rewritings, the difference is due to steps before the step (3) in Figure 4.3. It turns out it is due to the rewriting time, which in turn strongly depends on the size of the reformulation it receives as input. In REW-C, the reformulations w.r.t. $\mathcal{R}_c$ are of size 1 (no union, just one BGP) for queries on data triples only, and never exceed 64 in S_1 and S_3 and 200 in S_2 and S_4, whereas, in REW-CA the reformulation sizes are much larger. REW-C is most often faster than REW-CA, by up to two orders of magnitude e.g., for Q_{02a}, Q_{19} and Q_{20a} on S_2, the latter two on S_4 etc. One order of magnitude speed-up is noticeable even on the smaller RIS S_1, S_3 (Figure 4.9) for Q_{02a}. As a consequence, REW-C completes successfully in all scenarios we study, whereas REW-CA fails to complete for many queries with timeout set to 10min (missing yellow bars in Figure 4.10), in close correlation with the increased number of reformulations.

Scaling in the data size

As stated in Section 4.5.1, there is a *scale factor of about 50* between S_1, S_3 on one hand, and S_2, S_4 on the other. Figures 4.9 and 4.10 show that the query answering times generally grow by less than 50, when moving from S_1 to S_2, and from S_3 to S_4. This is mostly due to the good scalability of PostgreSQL (in the all-relational RIS), Tatooine (itself building on PostgreSQL and MongoDB, in the heterogeneous RIS), and OntoSQL (for MAT). As discussed above, computation steps we implemented outside these systems are strongly impacted by the *mappings, ontology and query*; intelligently distributing the reasoning effort, as REW-C does, avoids the heavy performance penalties that from which REW-CA and REW sometimes suffer.

Impact of heterogeneity

REW-CA and REW-C incur a (modest) overhead when combining data from PostgreSQL and MongoDB (heterogeneous RIS) w.r.t. the relational-sources RIS. Part of this is due to the cost of marshalling data across system boundaries; the rest is due to imperfect optimization within Tatooine. Overall, the comparison demonstrates that RIS query answering is feasible and quite efficient even on heterogeneous data sources.

Experiment conclusion

In a setting where the data, ontology and mappings do not change, MAT is an efficient and robust query answering technique, at a rather high cost to materialize and saturate the RIS instance. In contrast, in a dynamic setting, REW-C *smartly combines partial reformulation and view-based query rewriting to efficiently compute query answers*. The changes it requires when the ontology and mappings change (basically re-saturating mapping heads) are light and likely to be very fast. Thus, we conclude that REW-C is the best query answering strategy for dynamic RIS.

4.6 Extending the framework to more general rules

In this section, we present theoretical results on mapping saturation, in a more general framework than the one of Section 4.3. Notably, we extend the previous framework by supporting RDF entailment rules that go beyond RDFS entailment rules (Table 2.2) and allow for existential variables in their head.

We first recall the notion of *saturation of a BGPQ w.r.t. an RDFS ontology* by a set of RDF entailment rules, defined in [El Hassad et al., 2017]. We will rely on it to define a more general notion of mapping saturation. The saturation of a BGPQ contains in its body all the triples entailed from the BGPQ body and a given ontology, but not those entailed by the ontology alone as illustrated by Figure 4.11:

Definition 4.14 (BGPQ saturation [El Hassad et al., 2017])**.** *Let $\mathcal{R}$ be a set RDF entailment rules, O an RDFS ontology, and q a BGPQ. The* saturation *of q w.r.t. O, denoted by $q^{\mathcal{R},O}$, is the BGPQ with the same answer variables as q and whose body, denoted by $body(q^{\mathcal{R},O})$, is the maximal subset of $(body(q) \cup O)^{\mathcal{R}}$ such that for any of its subsets S: if $O \models_{\mathcal{R}} S$ holds, then $body(q) \models_{\mathcal{R}} S$ holds.*

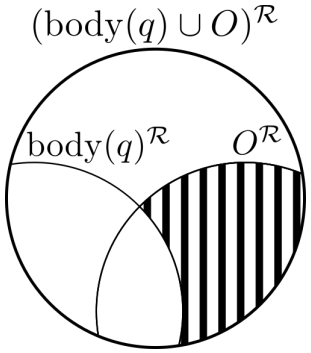


Figure 4.11: The circle represents $(\mathrm{body}(q) \cup O)^{\mathcal{R}}$, the hatched area is removed from $q^{\mathcal{R},O}$, because it is consequence of O only, hence not relevant to q.

Similarly to RDF graphs, the saturation of a query w.r.t. an ontology and (a subset of) $\mathcal{R}_{RDFS}$ is finite. An example of BGPQ saturation has been given in Example 4.7. We now introduce the notion of a single *RIS mapping saturation*:

Definition 4.15 (Mapping saturation)**.** *Let O be an RDFS ontology, $\mathcal{R}$ be a set of RDF entailment rules and $m = q_1(\bar{x}) \rightsquigarrow q_2(f_1(\bar{x}), \ldots, f_n(\bar{x}))$ be a RIS mapping, the* saturation of *m w.r.t. $\mathcal{R}$ and O is the RIS mapping, denoted by $m^{\mathcal{R},O}$, defined by:*

$$m^{\mathcal{R},O} = q_1(\bar{x}) \rightsquigarrow q_2^{\mathcal{R},O}(f_1(\bar{x}), \ldots, f_n(\bar{x}))$$

Given a set $\mathcal{M}$ of RIS mappings, a set $\mathcal{R}$ of RDF entailment rules and an RDFS ontology O, the *saturation of* $\mathcal{M}$ w.r.t. $\mathcal{R}$ and O, denoted by $\mathcal{M}^{\mathcal{R},O}$, is the set of the saturations of the mappings in $\mathcal{M}$ w.r.t. $\mathcal{R}$ and O. It corresponds to a generalization of Definition 4.10.

4.6.1 Restricted RIS

In this section, we introduce restricted rules, which are specific RDF entailment rules (hence their name) that generalize RDFS entailment rules. Together with an FO ontology (Definition 3.4), data mappings (Definition 4.2) and ontology mappings (Definition 4.11), they lead to *restricted RDF Integration Systems*.

Restricted rules

We consider RDF entailment rules that comply with some restrictions. This yields two kinds of entailment rules, namely *ontological rules* and *instance rules*, which respectively allow to infer knowledge about the ontology and about individuals.

The goal of the restriction that leads to instance rules is to allow using RDFS triples when inferring facts about individuals, which is one of the interests of RDF, while ensuring the completeness of saturation based on mapping heads. Essentially, the restrictions ensure that, given any set of extensions for the mappings, the graph obtained by the saturated mappings is equal to the saturation of the graph obtained by the initial mappings, as stated in Theorem 4.5.

Definition 4.16 (Restricted rules)**.** *We call* restricted rule *an RDF entailment rule r which is either an* ontological rule *or an* instance rule *as defined below:*

1. *(**Ontological rule**) body*(r) *and head*(r) *contain solely RDFS triples such that* $\text{Var}(head(\text{r})) \subseteq \text{Var}(body(\text{r}))$
2. *(**Instance rule**)* $body(r) = \{t_r\} \cup \text{body}_O(r)$*, where*
   - *a)* $\text{body}_O(r)$ *is a (possibly empty) set of RDFS triples*
   - *b)* t_r *is of one of the following forms:*
     - *i.* $(x, \mathtt{p}, y)$ *where* $x, y \in \mathscr{V} \setminus \text{Var}(\text{body}_\text{O}(\text{r}))$*,* $x \neq y$ *and* $\mathtt{p} \in (\mathscr{I} \setminus \{\prec_{sc}, \prec_{sp}, \hookleftarrow_d, \hookrightarrow_r, \tau\}) \cup \text{Var}(\text{body}_\text{O}(\text{r}))$*,*
     - *ii.* (x, τ, z) *where* $x \in \mathscr{V} \setminus \text{Var}(\text{body}_\text{O}(\text{r}))$*,* $z \in \mathscr{I} \cup \text{Bl}(\text{body}_O(r))$

   and $head(r)$ *contains solely* $(\mathtt{s}, \mathtt{p}, \mathtt{o})$ *triples such that:*
   - *c)* $\mathtt{p} \in \mathscr{I} \setminus \{\prec_{sc}, \prec_{sp}, \hookleftarrow_d, \hookrightarrow_r\}$ *or* $\mathtt{p} \in \text{Var}(\text{body}_\text{O}(\text{r}))$*,*
   - *d) if* $\mathtt{p} = \tau$*, then* $\mathtt{o} \in \mathscr{I}$ *or* $\mathtt{o} \in \text{Var}(\text{body}_\text{O}(\text{r}))$*.*

We first point out below that the standard RDFS entailment rules from Table 2.2 are specific restricted rules.

Example 4.11. *Consider Table 2.2. Rules* `rdfs5`*,* `rdfs11`*,* `ext1`*,* `ext2`*,* `ext2`*,* `ext3` *and* `ext4`*, i.e., the rules in* $\mathcal{R}_c$*, are ontological rules. Indeed, their body and head are composed of RDFS statements, e.g.,* `rdfs5`*:* $(p_1, \prec_{sp}, p_2), (p_2, \prec_{sp}, p_3) \rightarrow (p_1, \prec_{sp}, p_3)$*. The other RDFS rules, i.e., in* $\mathcal{R}_a$*, are instance rules, whose body is composed of an RDFS triple and a triple of the form* t_r*, and the head has a single triple. In Rule* `rdfs2` *defined as* $(p, \hookleftarrow_d, c), (s, p, o) \rightarrow (s, \tau, o)$*,* t_r *fulfills Restriction 2(b)i, while the head fulfills Restriction 2d. The same holds for Rule* `rdfs3`*. In Rule* `rdfs7`*:* $(p_1, \prec_{sp}, p_2), (s, p_1, o) \rightarrow (s, p_2, o)$*,* t_r *complies with 2(b)i and the head with 2c. Finally, in Rule* `rdfs9` $= (c_1, \prec_{sc}, c_2), (s, \tau, c_1) \rightarrow (s, \tau, c_2)$*,* t_r *complies with 2(b)ii and the head with 2d.*

The syntax of restricted rules also allows for user-specific rules **beyond standard RDFS entailment rules**, as for instance the rule $r_{\text{ex}} = (s, \tau, :\text{Jedi}) \rightarrow (s, :\text{usesWeapon}, w)$ from the running example, stating that if someone is a Jedi, then he/she uses some weapon.

Since entailment rules will be applied to saturate the mapping heads, the termination of saturation is a crucial requirement. Obviously, this requirement is fulfilled by ontological rules. Indeed, rule heads do not introduce new blank nodes (i.e., for any rule r,

Var(head(r)) $\subseteq$ Var(body(r))). However, termination is not ensured for instance rules: e.g., a rule of the form $(x, \mathtt{p}, y) \rightarrow (y, \mathtt{p}, z)$ (intuitively, for all x and y, if x is related to y by $\mathtt{p}$, there exists z such that y is related to z by $\mathtt{p}$) leads to infinite saturation, as each rule application produces a new individual, which leads to a new rule application. However, we prefer not to further restrict instance rules to enforce termination, because of the variety of candidate syntactic restrictions. Indeed, RDF entailment rules can be logically translated into specific tuple-generating dependencies or existential rules, as mentioned in Section 2.1.3 ; many acyclicity conditions for sets of such rules have been defined in the literature (see, e.g., [Thomazo, 2013, Grau et al., 2014, Rocher, 2016] for syntheses) and can be imported in our setting. Hence, in the following, we will silently assume that the considered set of restricted rules ensures the termination of saturation, as is the case, for instance, of the set of rules $\mathcal{R}_{\text{RDFS}} \cup \{r_{\text{ex}}\}$.

We first point out that ontological rules can only be applied on triples of an FO ontology, and that any restricted rule that can be applied on an FO ontology is an ontological rule (next Property 4.1); second, the saturation of an FO ontology by restricted rules (hence, necessarily ontological rules) remains an FO ontology (next Property 4.2); third, given a graph whose set of RDFS triples is an FO ontology, all RDFS triples that can be brought by application of restricted rules come from ontological rules (next Property 4.3). Finally, restricted rules behave as expected when they are applied to any RDF graph G composed of an FO ontology and data triples: the ontological rules compute exactly the saturation of the ontological part of the graph (next Property 4.4), while the instance rules add triples about the individuals (possibly using ontological triples as well, be they initially present in the graph or inferred by the ontological rules). In the following, given an RDF graph G, we will denote its RDFS ontology by RDFS(G).

Property 4.1. *Let r be a restricted rule and O be an FO ontology, if $O \models^{\varphi} body(r)$, then r is an ontological rule, i.e., it fulfills Restriction 1 of Definition 4.16.*

Property 4.2. *Let O be an FO ontology and $\mathcal{R}$ be a set of restricted rules, then $O^{\mathcal{R}}$ is also an FO ontology.*

Property 4.3. *Let r be a restricted rule and G be an RDF graph whose set of RDFS triples is an FO ontology. If the direct entailment of G by $\{r\}$ (denoted $C_{G,\{r\}}$ in Section 2.1.3) contains an RDFS triple, then r is an ontological rule.*

Property 4.4. *Let O be an FO ontology, $\mathcal{R}$ be a set of restricted rules and G be an RDF graph such that* RDFS(G) = O*, it holds that:*

$$\text{RDFS}(G^{\mathcal{R}}) = O^{\mathcal{R}}$$

Figure 4.12 schematizes the properties of restricted rules applied on graphs with an FO ontology, which are similar to the ones of RDFS entailment rules on graphs with split reasoning (recall Figure 3.2). The above properties of ontological rules are summarized by the loop on the FO ontology. The body of an instance rule r is composed of t_r a triple and $\text{body}_O(r)$ a set of RDFS triples. When r is applied on a graph G, the triple t_r (resp. $\text{body}_O(r)$) is necessarily mapped to a data triple (resp. to schema triples) of G, furthermore the triples produced by this application are necessarily data triples.

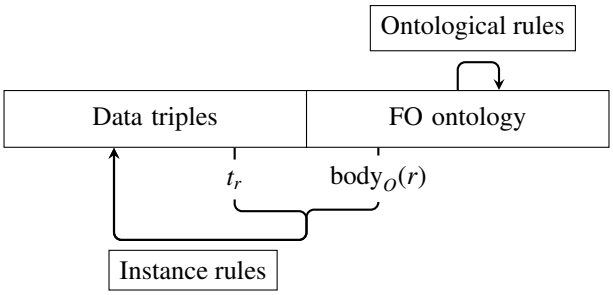


Figure 4.12: Restricted rule entailments.

Saturation of data mapping heads

We now investigate the properties of data mapping (Definition 4.2) saturation with restricted rules. The triples in the head of a data mapping yield data triples, when they are instantiated. The next example shows why we cannot consider RIS mappings that would extend data mappings by allowing for head triples with a variable at class position.

Example 4.12. *Consider the RIS mapping m defined by $q_1(x, y) \rightsquigarrow (x, \tau, y)$ and its extension $\text{ext}(m, D) = \{V_m(s, C_1)\}$ on a data source D, as well as the RDF standard entailment rule* `rdfs9` *defined by $(s, \prec_{sc}, o), (s_1, \tau, s) \rightarrow (s_1, \tau, o)$ and the FO ontology $O = \{(C_1, \prec_{sc}, C_2)\}$. Given $\mathcal{M} = \{m\}$, $\mathcal{E} = \text{ext}(m, D)$ and $\mathcal{R} = \{\texttt{rdfs9}\}$, the saturation of the RIS induced graph is:*

$$\left(G_{\mathcal{E}}^{\mathcal{M}} \cup O\right)^{\mathcal{R}} = \{(s, \tau, C_1), (s, \tau, C_2), (C_1, \prec_{sc}, C_2)\}.$$

However, there is no homomorphism from body(`rdfs9`*) to $(x, \tau, y), (C_1, \prec_{sc}, C_2)$, hence the BGPQ saturation $(head(m))^{\mathcal{R},O}$ is equal to head(m). It follows that the triple (s, τ, C_2) is missing in $(G_{\mathcal{E}}^{\mathcal{M}^{\mathcal{R},O}} \cup O)$. For this reason, we do not consider such mapping m, and stick to data mappings.*

The next property partially explains why no information is lost when we locally saturate the heads of mappings instead of saturating the graph induced by them. See Figure 4.13: for an instance rule r, if body(r) is mapped by a homomorphism φ to $\{v(t)\} \cup O$, where O is an FO ontology and $v(t)$ is a triple from the head of a data mapping instantiated by a homomorphism v, then (1) there is a homomorphism φ' that applies r to $\{t\} \cup O$, and (2) the composition $\varphi' \circ v$ is exactly φ.

Property 4.5. *Let O be an FO ontology, t a triple in the head of a data mapping, and v a homomorphism from t (i.e., a substitution $\text{Var(t)} \rightarrow \mathscr{B} \cup \mathscr{I} \cup \mathscr{L}$). For any restricted rule r (necessarily an instance rule), if $\{v(t)\} \cup O \models^{\varphi} body(r)$ then there exists a homomorphism φ' such that $\{t\} \cup O \models^{\varphi'} body(r)$ and $\varphi(body(r)) = \varphi'(v(body(r)))$.*

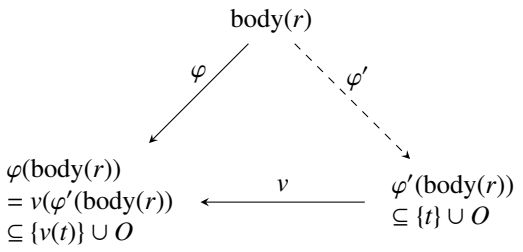


Figure 4.13: Illustration of Property 4.5.

The next property expresses that, when a restricted rule is applied to the head of a data mapping, the added triples keep the property of being a data mapping.

Property 4.6. *Let O be an FO ontology, t a triple of the head of a data mapping $m = q_1(\bar{x}) \rightsquigarrow q_2(f_1(\bar{x}), \ldots, f_n(\bar{x}))$ and r an instance rule such that $\{t\} \cup O \models^{\varphi'} body(r)$. Then the mapping $m' = q_1(\bar{x}) \rightsquigarrow q'_2(f_1(\bar{x}), \ldots, f_n(\bar{x}))$ with $q'_2(y_1, \ldots, y_n) = body(q_2) \cup \varphi'(head(r))^{safe}$ is a data mapping.*

The two previous properties hold in particular for RDFS entailment rules and can be checked on the previous illustration of saturated mappings in Example 4.8. Below, we illustrate these properties using a restricted rule outside $\mathcal{R}_{\text{RDFS}}$.

Example 4.13. *Consider a data mapping m with head $q_2(y) \leftarrow (y, \tau, :\text{Jedi})$ generating the triple $(:\text{Rey}, \tau, :\text{Jedi})$ by the homomorphism $v = \{y \mapsto :\text{Rey}\}$ applied on the only triple t in q_2's body. The restricted rule $r_{ex} = (x, \tau, :\text{Jedi}) \rightarrow (x, :\text{usesWeapon}, z)$, introduced above, can be applied on the generated triple $v(t)$ using the homomorphism φ from $body(r_{ex})$ to $v(t)$, defined by $\varphi = \{x \mapsto :\text{Rey}\}$. Since r_{ex} is an instance rule without RDFS triple pattern, we do not need to specify an ontology. As stated by Property 4.5, there exists a homomorphism from $body(r)$ to $\{t\}$, which is $\varphi' = \{x \mapsto y\}$. We check that $v(\varphi'(body(r))) = \varphi(body(r))$, which ensures that applying r_{ex} on the head of m with φ' and then instantiating its saturated head with v returns the same triples as applying r_{ex} with φ on the head of m with v. Moreover, Property 4.6 ensures that the saturated mapping $m^{\{r_{ex}\},\emptyset}$, whose head is $q_2^{\{r_{ex}\},\emptyset}(x) \leftarrow (x, \tau, :\text{Jedi}), (x, :\text{usesWeapon}, z)$, is, indeed, a data mapping like m.*

The following example is relative to the definition of restricted rules and illustrates the importance of the condition $x \neq y$ in the triple t_r.

Example 4.14. *Assume the rule $r = (x, p, x) \rightarrow (x, q, x)$ would be allowed, and let the mapping $m = q_1(x, y) \rightsquigarrow (x, p, y)$ and its extension $\text{ext}(m, D) = \{V_m(a, a)\}$ w.r.t. a data source D. Let $\mathcal{M} = \{m\}$, $\mathcal{E} = \text{ext}(m)$ and $\mathcal{R} = \{\texttt{rdfs9}\}$. Then the saturation of the induced RDF graph is:*

$$\left(G_{\mathcal{E}}^{\mathcal{M}}\right)^{\mathcal{R}} = \{(a, p, a), (a, q, a)\}.$$

However, there is no homomorphism from $body(r)$ to (x, p, y), hence the saturation of the mapping m with $\mathcal{R}$ is exactly m. Therefore, the triple (a, q, a) is missing in the saturated mapping graph $G_{\mathcal{E}}^{\mathcal{M}^{\mathcal{R},\emptyset}}$. It is the reason why the condition $x \neq y$ is enforced in 2(b)i, hence r is not a restricted rule.

Restricted RDF Integration System

We have now investigated suitable restrictions for each component of a RIS, which yield a restricted RIS.

Definition 4.17 (Restricted RIS)**.** *We say that an RDF Integration System $S = \langle O, \mathcal{R}, \mathcal{M}, \mathcal{E} \rangle$ is a* restricted RDF Integration System*, if O is an FO ontology, $\mathcal{R}$ a set of restricted rules and $\mathcal{M}$ a set of data mappings.*

As defined in Section 4.3.5, the ontology mappings associated with an FO ontology O allow one to integrate the ontological statements of O into an RDF graph, as follows:

Property 4.7. *Given an FO ontology O and a set $\mathcal{R}$ of restricted rules, it holds that:*

$$G_{\mathcal{E}_{O^{\mathcal{R}}}}^{\mathcal{M}_{O^{\mathcal{R}}}} = O^{\mathcal{R}}$$

Futhermore, no other RDFS triples are created by data mappings and applications of instance rules, hence:

Property 4.8. *For any restricted RIS $S = \langle O, \mathcal{R}, \mathcal{M}, \mathcal{E} \rangle$, the RDFS ontology of $(G^{\mathcal{M} \cup \mathcal{M}_{O^\mathcal{R}}}_{\mathcal{E} \cup \mathcal{E}_{O^\mathcal{R}}})^\mathcal{R}$ is exactly $O^\mathcal{R}$.*

4.6.2 Correctness of the Method

We now present the **main arguments that prove the correctness** of the query answering strategies presented in this chapter, and refer the reader to Appendix A.4 for detailed proofs. Note that these proofs consider the restricted entailment rules introduced in the preceding section, hence a generalization of RDFS entailment rules.

Assume first that we adopt a classical materialization approach: starting from the extent $\mathcal{E} \cup \mathcal{E}_{O^\mathcal{R}}$, we trigger the mappings $\mathcal{M} \cup \mathcal{M}_{O^\mathcal{R}}$, then saturate the obtained graph with the entailment rules $\mathcal{R}$, and finally get the graph $\left(G^{\mathcal{M} \cup \mathcal{M}_{O^\mathcal{R}}}_{\mathcal{E} \cup \mathcal{E}_{O^\mathcal{R}}}\right)^\mathcal{R}$, on which we can ask BGP queries and obtain a complete certain answer set.

Now, instead of saturating the graph produced by the mappings, we proceed as follows: (1) we saturate the mappings (actually their head) with the entailment rules, then (2) we trigger the mappings. We thus obtain the graph $G^{\mathcal{M}^{\mathcal{R},O}}_{\mathcal{E}} \cup G^{\mathcal{M}_{O^\mathcal{R}}}_{\mathcal{E}_{O^\mathcal{R}}}$. The next theorem states that the graphs obtained by the two ways of doing are equal (up to bijective renaming of blank nodes introduced the existential variables in mapping heads).

Theorem 4.5. *Given an FO ontology O, a set $\mathcal{R}$ of restricted rules and a set $\mathcal{M}$ of data mappings, it holds that:*

$$\left(G^{\mathcal{M}}_{\mathcal{E}} \cup O\right)^\mathcal{R} = G^{\mathcal{M}^{\mathcal{R},O} \cup \mathcal{M}_{O^\mathcal{R}}}_{\mathcal{E} \cup \mathcal{E}_{O^\mathcal{R}}}$$

Of course, this equality does not hold for general RIS. In Section 4.6.1, we have illustrated by examples the role of the main restrictions we enforce and highlighted some key properties ensured by these restrictions. An important characteristic of restricted RIS is the distinction between ontology and data mappings, and similarly between ontological and instance entailment rules. This allows one to consider two induced graphs, whose union yields $G^{\mathcal{M}^{\mathcal{R},O} \cup \mathcal{M}_{O^\mathcal{R}}}_{\mathcal{E} \cup \mathcal{E}_{O^\mathcal{R}}}$:

- $G^{\mathcal{M}_{O^\mathcal{R}}}_{\mathcal{E}_{O^\mathcal{R}}}$, which is equal to the saturated ontology $O^\mathcal{R}$ (Property 4.7).
- $G^{\mathcal{M}^{\mathcal{R},O}}_{\mathcal{E}}$, which materializes exactly the data triples of the saturated graph $\left(G^{\mathcal{M}}_{\mathcal{E}} \cup O\right)^\mathcal{R}$. This equality also relies on the form of the restricted instance rules, which ensures that every application of an instance rule involved in the saturation of the graph produced the mappings can be similarly performed on a mapping head (Property 4.5); in particular, no application of an instance rule requires data triples coming from two different mappings.

4.7 Summary

In this chapter, we tackled the issue of *querying heterogeneous data sources* using *RDF as the integration formalism* and *RDFS to represent and reason with ontological knowl-*

edge. Our contributions are of different nature: theoretical, algorithmic, software and experimental.

We formalized our framework in the form of *RDF Integration Systems* (RIS). In the spirit of Ontology-Based Data Access architectures, the data sources in a RIS are connected to the conceptual (or ontological) level by means of mappings. Compared to the state of the art, our originality is to process *GLAV mappings*, which provide the framework with a greater data integration power than GAV mappings, as well as *general BGP queries*, which allow one to query schema and data triples together.

As in the OBDA approach, we specially investigated mediator-style query answering strategies, in which the global RDF graph remains virtual, which leads to consider query rewriting techniques. In particular, we proposed two novel rewriting-based query answering techniques, based on a form of *mapping saturation*. Besides, we were also interested in delineating the space of possible strategies, which can be viewed as different ways of distributing reasoning on the query or inside the RIS components. Hence, we proposed a *classification of strategies* grouped in two approaches, respectively based on graph materialization and on query rewriting, and further refined according to the distribution of reasoning using query reformulation, mappings transformation and saturation of the graph materialization. This led to the identification of thirteen strategies.

This way of analyzing RIS query answering strategies guided the implementation of our *RIS query answering system* Obi-Wan. Indeed, although Obi-Wan implements a subset of selected strategies, it is designed to be easily extensible to test new strategies. We also implemented optimization techniques, on the one hand to simplify the query produced by query rewriting, and on the other hand to provide Tatooine, the underlying polystore system, with an optimized translation of mediated query plans into logical query plans, which in particular takes advantage of the data source query engine capabilities.

We carried out several experiments, using extensions of the well-known BSBM benchmark, first to assess and compare the scalability of RIS query answering strategies, and second to estimate the overhead due to heterogeneity. These experiments demonstrated the feasibility of RIS query answering. We also concluded that REW-C, a strategy that combines partial query reformulation and view-based query rewriting, was the best choice for dynamic RIS.

Finally, we extended the RIS framework with RDF entailment rules that go beyond the standard RDFS rules (so-called *restricted entailment rules*, in contrast with general RDF entailment rules). These rules may have existential variables in their head, hence are able to infer the existence of unknown individuals. In particular, we proved the correctness of the studied query answering strategies in this extended framework. Better understanding the expressive power of restricted entailment rules, in particular with respect to lightweight description logics used in OBDA systems, is ongoing work. More generally, the interaction between (GLAV) mappings and RDF entailment rules remains to be studied in more depth, from both theoretical and algorithmic viewpoints.

RIS		Q01	Q01a	Q01b	Q02	Q02a	Q02b	Q02c	Q03	Q04	Q07	Q07a	Q09	Q10	Q13
all	N_{TRI}	5	5	5	6	6	6	6	5	2	3	3	1	3	4
S_1, S_3	$\lvert Q_{c,a}\rvert$	7	21	175	21	49	147	1225	525	1	5	19	7	670	28
S_1, S_3	N_{ANS}	1272	4376	22738	16	56	174	1342	19	91	2	3	5617	9	13190
S_2, S_4	$\lvert Q_{c,a}\rvert$	21	175	1407	63	147	525	1225	4375	1	5	19	7	9350	84
S_2, S_4	N_{ANS}	15514	111793	863729	124	598	1058	1570	5	4487	2	3	299902	10	167760

RIS		Q13a	Q13b	Q14	Q16	Q19	Q19a	Q20	Q20a	Q20b	Q20c	Q21	Q22	Q22a	Q23
all	N_{TRI}	4	4	3	4	9	9	11	11	11	11	3	4	4	7
S_1, S_3	$\lvert Q_{c,a}\rvert$	84	700	1	25	63	147	21	63	525	1225	670	2	40	192
S_1, S_3	N_{ANS}	43157	330142	56200	8114	2015	3515	0	236	2312	7564	1085	28	434	25803
S_2, S_4	$\lvert Q_{c,a}\rvert$	5628	5628	1	201	525	1225	63	525	1225	4221	9350	40	520	192
S_2, S_4	N_{ANS}	4416946	10049829	2998948	249004	39826	60834	904	7818	10486	51988	37176	1528	18588	1329887

Table 4.2: Characteristics of the queries used in our experiments.

CHAPTER

5 Conclusion and perspectives

Conclusion

In this thesis, we considered the issue of efficiently querying heterogeneous data sources using RDF as the integration formalism and RDF entailment rules to reason on RDF graphs. To take advantage of RDF inherent metamodeling capabilities, we relied on general basic graph pattern queries. We made several kinds of contributions: theoretical, algorithmic, software and experimental.

In Chapter 3, we first focused on *querying RDF graphs provided with built-in RDFS entailment rules*, which allow one to reason with basic ontological knowledge, namely subclass, subproperty, domain and range. We carefully considered the structure of RDFS entailment rules, which led to partition them into the constraint and assertion rule sets and study how both parts interact. This allowed us to define a desirable property of RDF graphs, notably, *split reasoning*, which is ensured by *FO-restriction*, a syntactic restriction, and still enables interesting forms of metamodeling. We then designed a *query reformulation* algorithm for RDF graphs with split reasoning. The experiments we carried out show the effectiveness of our algorithm. We finally investigated the impact of *storage layout* on the efficiency of query answering with general BGPQs. We introduced the novel workload-unaware *TCP layout*, which combines the data structures of two classical layouts, namely T and CP, and experimentally assessed its interest, on a relational and a native RDF database. Besides, we introduced *summary-based pruning*, an optimization technique of independent interest, and we found that it generally reduces query answering cost.

Then, in Chapter 4, we studied the more general problem of *querying RDF integration of heterogeneous data sources* with respect to built-in RDFS entailment rules. We formalized the notion of *RDF Integration Systems* (RIS) in the spirit of Ontology-Based Data Access architectures. The novelties of RIS, compared to the state of the art, are, first, the *GLAV mappings* linking the sources content with the concepts defined by an RDFS ontology, which are more expressive than GAV mappings and, second, the support of *general BGP queries*, which allow one to query schema and data triples together. As in the OBDA approach, we specially investigated query answering strategies based on mediation, in which the query on the global RDF graph is rewritten into a query on the sources. We presented a form of *mapping saturation*, which allows one to handle a part

of the reasoning as a preprocessing step. We also presented a classification of the different query answering approaches, which has inspired the implementation of our *RIS query answering system* Obi-Wan, designed to be easily extendable with new query answering strategies. Using Obi-Wan, we carried out several experiments, first to assess and compare the scalability of RIS query answering strategies, and second to estimate the overhead due to heterogeneity. These experiments demonstrated the feasibility of RIS query answering and that mappings saturation improves the performances of the query rewriting step. Finally, we proved the correctness of mappings saturation for query answering in RIS, for a subset of RDF entailment rules, containing the built-in RDFS entailment rules, and in which rules may have existential variables in their head.

Perspectives

This work can be pursued in a number of ways. We list below some ideas to extend the expressivity of our framework and further improve the efficiency of query answering in RDF integration systems and databases.

Ontological expressivity beyond RDFS

Throughout the thesis, we focused on RDFS for simplicity and because it is the basis of most of the ontology languages used in semantic web applications. We would like to extend the properties we introduced about the reasoning behaviour of RDFS ontologies and together with RDFS entailment rules (see Section 3.2.1) to more general *decidable subsets of OWL-Full*, by still enabling the metamodeling capabilities of RDF, as investigated in [Motik, 2005, Giacomo et al., 2011].

Cost-driven translation on TCP layout

We showed that TCP is a robust storage layout for general BGP query answering, on which we proposed a single query translation. In reformulation-based query answering, further optimizations could be obtained, by investigating a space of reformulated query translations using a *cost-based algorithm for selecting the translation with the lowest cost* from this space, as proposed in [Bursztyn et al., 2015] for the CP layout.

Interactions between mappings and rules

Better understanding the *expressive power of restricted entailment rules*, in particular with respect to lightweight description logics used in OBDA systems, is ongoing work. More generally, the *interaction between (GLAV) mappings and RDF entailment rules* remains to be studied in more depth, from both the theoretical and algorithmic viewpoints.

Heterogeneous data sources mediation

Supporting the integration of *web services as sources*, which can be seen as views with binding patterns [Rajaraman et al., 1995] could greatly expand the scope of integration systems, like Obi-Wan. This requires further work on query rewriting algorithms [Romero et al., 2020] as well as on mediated plan optimizations [Florescu et al., 1999, Benedikt et al., 2015]. In particular, the efficient evaluation of mediated plans requires

an optimizer capable of searching for an optimal plan among a space of equivalent plans (generated, in particular, by *ordering the join operators*) based on evaluation cost estimations [Lanti et al., 2017].

Bibliography

[OWL, a] OWL 2 Web Ontology Language Document Overview (Second Edition). https://www.w3.org/TR/owl2-overview/.

[OWL, b] OWL 2 Web Ontology Language Mapping to RDF Graphs (Second Edition). https://www.w3.org/TR/2012/REC-owl2-mapping-to-rdf-20121211/#Translation_of_Annotations.

[RFC, a] RFC 3986 - Uniform Resource Identifier (URI): Generic Syntax. https://tools.ietf.org/html/rfc3986.

[RFC, b] RFC 3987 - Internationalized Resource Identifiers (IRIs). https://tools.ietf.org/html/rfc3987.

[Res, 2004] (2004). Resource Description Framework (RDF): Concepts and Abstract Syntax. https://www.w3.org/TR/rdf-concepts/.

[SPA, 2013] (2013). SPARQL 1.1 Query Language. https://www.w3.org/TR/sparql11-query/.

[RDF, 2014a] (2014a). RDF 1.1 Concepts and Abstract Syntax. https://www.w3.org/TR/rdf11-concepts/.

[RDF, 2014b] (2014b). RDF 1.1 Semantics. https://www.w3.org/TR/rdf11-mt/#rdfs-entailment.

[RDF, 2014c] (2014c). RDF Schema 1.1. https://www.w3.org/TR/2014/REC-rdf-schema-20140225/.

[Abadi et al., 2007] Abadi, D. J., Marcus, A., Madden, S. R., and Hollenbach, K. (2007). Scalable Semantic Web Data Management Using Vertical Partitioning. *PVLDB*.

[Abdallah et al., 2009] Abdallah, N., Goasdoué, F., and Rousset, M. (2009). DL-LITER in the light of propositional logic for decentralized data management. In *IJCAI*.

[Abiteboul and Duschka, 1998] Abiteboul, S. and Duschka, O. M. (1998). Complexity of answering queries using materialized views. ACM Press.

[Abiteboul et al., 1995] Abiteboul, S., Hull, R., and Vianu, V. (1995). *Foundations of Databases*. Addison-Wesley.

[Abiteboul et al., 2011] Abiteboul, S., Manolescu, I., Rigaux, P., Rousset, M.-C., and Senellart, P. (2011). *Web Data Management*. Cambridge University Press, USA.

[Adjiman et al., 2007] Adjiman, P., Goasdoué, F., and Rousset, M.-C. (2007). SomeRDFS in the semantic web. *JODS*, 8.

[Alotaibi et al., 2019] Alotaibi, R., Bursztyn, D., Deutsch, A., Manolescu, I., and Zampetakis, S. (2019). Towards Scalable Hybrid Stores: Constraint-Based Rewriting to the Rescue. In *SIGMOD*.

[Amann et al., 2002] Amann, B., Beeri, C., Fundulaki, I., and Scholl, M. (2002). Querying XML Sources Using an Ontology-Based Mediator. In *CoopIS*. Springer Berlin Heidelberg.

[Amann and Fundulaki, 1999] Amann, B. and Fundulaki, I. (1999). Integrating ontologies and thesauri to build RDF schemas. In *ECDL*.

[Amann et al., 2000] Amann, B., Fundulaki, I., and Scholl, M. (2000). Integrating ontologies and thesauri for RDF schema creation and metadata querying. *Int. J. on Digital Libraries*, 3(3).

[Arenas et al., 2009] Arenas, M., Gutierrez, C., and Pérez, J. (2009). Foundations of RDF Databases. In *Reasoning Web. Semantic Technologies for Information Systems*, Lecture Notes in Computer Science, pages 158–204. Springer, Berlin, Heidelberg.

[Artale et al., 2009] Artale, A., Calvanese, D., Kontchakov, R., and Zakharyaschev, M. (2009). The DL-Lite family and relations. *Journal of artificial intelligence research*, 36(1):1–69.

[Atre et al., 2010] Atre, M., Chaoji, V., Zaki, M. J., and Hendler, J. A. (2010). Matrix "bit" loaded: a scalable lightweight join query processor for RDF data. In *WWW*.

[Baader et al., 2003] Baader, F., Calvanese, D., McGuinness, D. L., Nardi, D., and Patel-Schneider, P. F., editors (2003). *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press.

[Baader et al., 2007] Baader, F., Calvanese, D., McGuinness, D. L., Nardi, D., and Peter F. Patel-Schneider (2007). The Description Logic Handbook: Theory, Implementation and Applications. /core/books/description-logic-handbook/F050683766E57EE9BB07BC01BB7A7069.

[Baget et al., 2015] Baget, J.-F., Leclère, M., Mugnier, M., Rocher, S., and Sipieter, C. (2015). Graal: A toolkit for query answering with existential rules. In *RuleML*.

[Baget et al., 2011] Baget, J.-F., Leclère, M., Mugnier, M.-L., and Salvat, E. (2011). On rules with existential variables: Walking the decidability line. *Artificial Intelligence*, 175(9):1620–1654.

[Benedikt et al., 2015] Benedikt, M., Leblay, J., and Tsamoura, E. (2015). Querying with access patterns and integrity constraints. *Proc. VLDB Endow.*, 8(6):690–701.

[Bischof et al., 2014] Bischof, S., Krötzsch, M., Polleres, A., and Rudolph, S. (2014). Schema-Agnostic Query Rewriting in SPARQL 1.1. In Mika, P., Tudorache, T., Bernstein, A., Welty, C., Knoblock, C., Vrandečić, D., Groth, P., Noy, N., Janowicz, K., and Goble, C., editors, *The Semantic Web – ISWC 2014*, volume 8796, pages 584–600. Springer International Publishing, Cham.

[Bishop et al., 2011] Bishop, B., Kiryakov, A., Ognyanoff, D., Peikov, I., Tashev, Z., and Velkov, R. (2011). OWLIM: A family of scalable semantic repositories. *Semantic Web*, 2(1).

[Bonaque et al., 2016] Bonaque, R., Cao, T. D., Cautis, B., Goasdoué, F., Letelier, J., Manolescu, I., Mendoza, O., Ribeiro, S., Tannier, X., and Thomazo, M. (2016). Mixed-instance querying: a lightweight integration architecture for data journalism. In *VLDB*.

[Bornea et al., 2013] Bornea, M. A., Dolby, J., Kementsietsidis, A., Srinivas, K., Dantressangle, P., Udrea, O., and Bhattacharjee, B. (2013). Building an efficient RDF store over a relational database. In *SIGMOD*.

[Botoeva et al., 2018] Botoeva, E., Calvanese, D., Cogrel, B., Corman, J., and Xiao, G. (2018). A generalized framework for ontology-based data access. In *AI*IA*.

[Broekstra and Kampman, 2003] Broekstra, J. and Kampman, A. (2003). Inferencing and truth maintenance in RDF schema. In *PSSS1 Workshop*.

[Broekstra et al., 2002] Broekstra, J., Kampman, A., and van Harmelen, F. (2002). Sesame: A generic architecture for storing and querying RDF and RDF schema. In *ISWC*.

[Buron et al., 2020a] Buron, M., Goasdoué, F., Manolescu, I., Merabti, T., and Mugnier, M. (2020a). Revisiting RDF storage layouts for efficient query answering. Report, INRIA.

[Buron et al., 2019] Buron, M., Goasdoué, F., Manolescu, I., and Mugnier, M. (2019). Reformulation-based query answering for RDF graphs with RDFS ontologies. In Hitzler, P., Fernández, M., Janowicz, K., Zaveri, A., Gray, A. J. G., López, V., Haller, A., and Hammar, K., editors, *The Semantic Web - 16th International Conference, ESWC 2019, Portorož, Slovenia, June 2-6, 2019, Proceedings*, volume 11503 of *Lecture Notes in Computer Science*, pages 19–35. Springer.

[Buron et al., 2020b] Buron, M., Goasdoué, F., Manolescu, I., and Mugnier, M. (2020b). 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 23nd International Conference on Extending Database Technology, EDBT 2020, Copenhagen, Denmark, March 30 - April 02, 2020*, pages 299–310. OpenProceedings.org.

[Buron et al., 2018] Buron, M., Goasdoué, F., Mugnier, M.-L., and Manolescu, I. (2018). Rewriting-Based Query Answering for Semantic Data Integration Systems (Informal publication). In *BDA: Gestion de Données - Principes, Technologies et Applications*.

[Buron et al., 2020c] Buron, M., Goasdoué, F., Manolescu, I., and Mugnier, M.-L. (2020c). Obi-Wan: Ontology-based RDF integration of heterogeneous data (demonstration). In *Proceedings of VLDB*. Accepted also for informal presentation at BDA 2020.

[Bursztyn et al., 2015] Bursztyn, D., Goasdoué, F., and Manolescu, I. (2015). Optimizing reformulation-based query answering in RDF. In *EDBT*.

[Bursztyn et al., 2016] Bursztyn, D., Goasdoué, F., and Manolescu, I. (2016). Teaching an RDBMS about ontological constraints. *PVLDB*, 9(12).

[Calì et al., 2009] Calì, A., Gottlob, G., and Lukasiewicz, T. (2009). Datalog extensions for tractable query answering over ontologies. In *Semantic Web Information Management - A Model-Based Perspective*, pages 249–279.

[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).

[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., 2009] Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R., and Ruzzi, M. (2009). Using owl in data integration. In De Virgilio, R., Giunchiglia, F., and Tanca, L., editors, *Semantic Web Information Management – A Model-Based Perspective*. Springer.

[Calvanese et al., 2012] Calvanese, D., De Giacomo, G., Lenzerini, M., and Vardi, M. Y. (2012). Query processing under GLAV mappings for relational and graph databases. *PVLDB*, 6(2).

[Calvanese et al., 2007] Calvanese, D., Giacomo, G. D., Lembo, D., Lenzerini, M., and Rosati, R. (2007). Tractable reasoning and efficient query answering in description logics: The DL-Lite family. *Journal of Automated Reasoning (JAR)*, 39(3).

[Cebiric et al., 2018] Cebiric, S., Goasdoué, F., Kondylakis, H., Kotzinos, D., Manolescu, I., Troullinou, G., and Zneika, M. (2018). Summarizing Semantic Graphs: A Survey. *VLDB Journal*.

[Chebotko et al., 2009] Chebotko, A., Lu, S., and Fotouhi, F. (2009). Semantics preserving SPARQL-to-SQL translation. *Data Knowl. Eng.*, 68(10).

[Christophides et al., 1997] Christophides, V., Doerr, M., and Fundulaki, I. (1997). A semantic network approach to semi-structured documents repositories. In *ECDL*.

[De Giacomo et al., 2018] De Giacomo, G., Lembo, D., Lenzerini, M., Poggi, A., and Rosati, R. (2018). Using Ontologies for Semantic Data Integration. In *A Comprehensive Guide Through the Italian Database Research Over the Last 25 Years*, volume 31. Springer, Cham.

[Deutsch and Tannen, 2003] Deutsch, A. and Tannen, V. (2003). MARS: A System for Publishing XML from Mixed and Redundant Storage. In *VLDB*.

[Doan et al., 2012] Doan, A., Halevy, A., and Ives, Z. G. (2012). *Principles of Data Integration*. Morgan Kaufmann, Waltham, MA.

[Duggan et al., 2015] Duggan, J., Elmore, A. J., Stonebraker, M., Balazinska, M., Howe, B., Kepner, J., Madden, S., Maier, D., Mattson, T., and Zdonik, S. B. (2015). The BigDAWG polystore system. *SIGMOD*, 44(2).

[El Hassad et al., 2017] El Hassad, S., Goasdoué, F., and Jaudoin, H. (2017). Learning commonalities in SPARQL. In *ISWC*.

[Florescu et al., 1999] Florescu, D., Levy, A. Y., Manolescu, I., and Suciu, D. (1999). Query optimization in the presence of limited access patterns. In Delis, A., Faloutsos, C., and Ghandeharizadeh, S., editors, *SIGMOD 1999, Proceedings ACM SIGMOD International Conference on Management of Data, June 1-3, 1999, Philadelphia, Pennsylvania, USA*, pages 311–322. ACM Press.

[Friedman et al., 1999] Friedman, M., Levy, A. Y., and Millstein, T. D. (1999). Navigational plans for data integration. *AAAI/IAAI*, 1999:67–73.

[Garcia-Molina et al., 1997] Garcia-Molina, H., Papakonstantinou, Y., Quass, D., Rajaraman, A., Sagiv, Y., Ullman, J. D., Vassalos, V., and Widom, J. (1997). The TSIMMIS approach to mediation: Data models and languages. *J. Intell. Inf. Syst.*, 8(2).

[Giacomo et al., 2011] Giacomo, G. D., Lenzerini, M., and Rosati, R. (2011). Higher-order description logics for domain metamodeling. In Burgard, W. and Roth, D., editors, *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2011, San Francisco, California, USA, August 7-11, 2011*. AAAI Press.

[Goasdoué et al., 2020] Goasdoué, F., Guzewicz, P., and Manolescu, I. (2020). RDF Graph Summarization for First-sight Structure Discovery. *The VLDB Journal*.

[Goasdoué et al., 2011] Goasdoué, F., Karanasos, K., Leblay, J., and Manolescu, I. (2011). View selection in semantic web databases. *PVLDB*, 5(2).

[Goasdoué et al., 2000] Goasdoué, F., Lattès, V., and Rousset, M. (2000). The use of CARIN language and algorithms for information integration: The PICSEL system. *Int. J. Cooperative Inf. Syst.*, 9(4).

[Goasdoué et al., 2012] Goasdoué, F., Manolescu, I., and Roatis, A. (2012). BGP Query Answering against Dynamic RDF Databases. Report, INRIA.

[Goasdoué et al., 2013] Goasdoué, F., Manolescu, I., and Roatis, A. (2013). Efficient query answering against dynamic RDF databases. In *EDBT*.

[Goasdoué and Rousset, 2004] Goasdoué, F. and Rousset, M. (2004). Answering queries using views: A KRDB perspective for the semantic web. *ACM Trans. Internet Techn.*, 4(3).

[Grau et al., 2014] Grau, B. C., Horrocks, I., Krötzsch, M., Kupke, C., Magka, D., Motik, B., and Wang, Z. (2014). Acyclicity notions for existential rules and their application to query answering in ontologies. *CoRR*, abs/1406.4110.

[Guo et al., 2005] Guo, Y., Pan, Z., and Heflin, J. (2005). LUBM: A benchmark for OWL knowledge base systems. *J. Web Sem.*, 3(2-3).

[Halevy, 2000] Halevy, A. Y. (2000). Theory of answering queries using views. *ACM SIGMOD Record*, 29(4):40–47.

[Halevy, 2001] Halevy, A. Y. (2001). Answering queries using views: A survey. *The VLDB Journal*, 10(4).

[Hovland et al., 2017] Hovland, D., Kontchakov, R., Skjæveland, M. G., Waaler, A., and Zakharyaschev, M. (2017). Ontology-based data access to Slegge. In *ISWC*.

[Jarke, 2003] Jarke, M. (2003). *Fundamentals of data warehouses, 2nd Edition*. Springer.

[Kaoudi et al., 2008] Kaoudi, Z., Miliaraki, I., and Koubarakis, M. (2008). RDFS reasoning and query answering on DHTs. In *ISWC*.

[König et al., 2013] König, M., Leclère, M., Mugnier, M.-L., and Thomazo, M. (2013). Sound, Complete and Minimal UCQ-Rewriting for Existential Rules. *arXiv:1311.3198 [cs]*.

[Kontchakov et al., 2014] Kontchakov, R., Rezk, M., Rodriguez-Muro, M., Xiao, G., and Zakharyaschev, M. (2014). Answering SPARQL queries over databases under OWL 2 QL entailment regime. In *ISWC*.

[Lanti et al., 2017] Lanti, D., Xiao, G., and Calvanese, D. (2017). Cost-driven ontology-based data access. In *ISWC*.

[Lassila and Swick, 1999] Lassila, O. and Swick, R. R. (1999). Resource Description Framework (RDF) Model and Syntax Specification. https://www.w3.org/TR/1999/REC-rdf-syntax-19990222/.

[Lenzerini, 2002] Lenzerini, M. (2002). Data integration: A theoretical perspective. In *Proceedings of the Twenty-First ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 233–246. ACM.

[Levy et al., 1996] Levy, A. Y., Rajaraman, A., and Ordille, J. J. (1996). Querying heterogeneous information sources using source descriptions. In *VLDB*.

[Levy et al., 1995] Levy, A. Y., Srivastava, D., and Kirk, T. (1995). Data model and query evaluation in global information systems. *J. Intell. Inf. Syst.*, 5(2).

[Lutz et al., 2013] Lutz, C., Seylan, I., Toman, D., and Wolter, F. (2013). The combined approach to OBDA: taming role hierarchies using filters. In *ISWC*.

[Manolescu et al., 2001] Manolescu, I., Florescu, D., and Kossmann, D. (2001). Answering XML queries on heterogeneous data sources. In *VLDB*.

[Motik, 2005] Motik, B. (2005). On the Properties of Metamodeling in OWL. In Gil, Y., Motta, E., Benjamins, V. R., and Musen, M. A., editors, *The Semantic Web – ISWC 2005*, Lecture Notes in Computer Science, pages 548–562, Berlin, Heidelberg. Springer.

[Mugnier and Thomazo, 2014] Mugnier, M.-L. and Thomazo, M. (2014). An Introduction to Ontology-Based Query Answering with Existential Rules. In Koubarakis, M., Stamou, G., Stoilos, G., Horrocks, I., Kolaitis, P., Lausen, G., and Weikum, G., editors, *Reasoning Web. Reasoning on the Web in the Big Data Era*, volume 8714, pages 245–278. Springer International Publishing, Cham.

[Nenov et al., 2015] Nenov, Y., Piro, R., Motik, B., Horrocks, I., Wu, Z., and Banerjee, J. (2015). RDFox: A Highly-Scalable RDF Store. In Arenas, M., Corcho, O., Simperl, E., Strohmaier, M., d'Aquin, M., Srinivas, K., Groth, P., Dumontier, M., Heflin, J., Thirunarayan, K., and Staab, S., editors, *The Semantic Web - ISWC 2015*, Lecture Notes in Computer Science, pages 3–20, Cham. Springer International Publishing.

[Neumann and Weikum, 2010] Neumann, T. and Weikum, G. (2010). The RDF-3X engine for scalable management of RDF data. *VLDB J.*

[Özsu and Valduriez, 2011] Özsu, M. T. and Valduriez, P. (2011). *Distributed and Parallel Database Systems (3rd. ed.)*. Springer.

[Pham et al., 2015] Pham, M., Passing, L., Erling, O., and Boncz, P. A. (2015). Deriving an emergent relational schema from RDF data. In *WWW*.

[Pinto et al., 2011] Pinto, F. D., Giacomo, G. D., Lenzerini, M., and Rosati, R. (2011). Mapping data to higher-order description logic knowledge bases. In Rosati, R., Rudolph, S., and Zakharyaschev, M., editors, *Proceedings of the 24th International Workshop on Description Logics (DL 2011), Barcelona, Spain, July 13-16, 2011*, volume 745 of *CEUR Workshop Proceedings*. CEUR-WS.org.

[Pinto et al., 2013] Pinto, F. D., Lembo, D., Lenzerini, M., Mancini, R., Poggi, A., Rosati, R., Ruzzi, M., and Savo, D. F. (2013). Optimizing query rewriting in ontology-based data access. In *EDBT*.

[Poggi et al., 2008] Poggi, A., Lembo, D., Calvanese, D., De Giacomo, G., Lenzerini, M., and Rosati, R. (2008). Linking data to ontologies. *J. Data Semantics*, 10.

[Pottinger and Halevy, 2001] Pottinger, R. and Halevy, A. Y. (2001). Minicon: A scalable algorithm for answering queries using views. *VLDB J.*, 10.

[Prud'hommeaux and Seaborne, 2004] Prud'hommeaux, E. and Seaborne, A. (2004). SPARQL Query Language for RDF. https://www.w3.org/TR/2004/WD-rdf-sparql-query-20041012/.

[Rajaraman et al., 1995] Rajaraman, A., Sagiv, Y., and Ullman, J. D. (1995). Answering queries using templates with binding patterns. In Yannakakis, M., editor, *Proceedings of the Fourteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 22-25, 1995, San Jose, California, USA*, pages 105–112. ACM Press.

[Rocher, 2016] Rocher, S. (2016). *Querying Existential Rule Knowledge Bases: Decidability and Complexity*. Theses, Université de Montpellier.

[Rodriguez-Muro et al., 2013] Rodriguez-Muro, M., Kontchakov, R., and Zakharyaschev, M. (2013). Ontology-based data access: Ontop of databases. In *ISWC*.

[Romero et al., 2020] Romero, J., Preda, N., Amarilli, A., and Suchanek, F. M. (2020). Equivalent rewritings on path views with binding patterns. In Harth, A., Kirrane, S., Ngomo, A.-C. N., Paulheim, H., Rula, A., Gentile, A. L., Haase, P., and Cochez, M., editors, *The Semantic Web - 17th International Conference, ESWC 2020, Heraklion, Crete, Greece, May 31-June 4, 2020, Proceedings*, volume 12123 of *Lecture Notes in Computer Science*, pages 446–462. Springer.

[Sequeda et al., 2014] Sequeda, J. F., Arenas, M., and Miranker, D. P. (2014). OBDA: query rewriting or materialization? in practice, both! In *ISWC*.

[Sidirourgos et al., 2008] Sidirourgos, L., Goncalves, R., Kersten, M., Nes, N., and Manegold, S. (2008). Column-store support for RDF data management: not all swans are white. *PVLDB*, 1(2).

[Smits et al., 2014] Smits, G., Pivert, O., Jaudoin, H., and Paulus, F. (2014). AGGREGO SEARCH: interactive keyword query construction. In *EDBT*.

[Thomazo, 2013] Thomazo, M. (2013). *Conjunctive Query Answering Under Existential Rules - Decidability, Complexity, and Algorithms*. PhD thesis, Montpellier 2 University, France.

[Udrea et al., 2007] Udrea, O., Pugliese, A., and Subrahmanian, V. S. (2007). GRIN: A graph based RDF index. In *AAAI*.

[Urbani et al., 2014] Urbani, J., Piro, R., van Harmelen, F., and Bal, H. E. (2014). Hybrid reasoning on OWL RL. *Semantic Web*, 5(6).

[Urbani et al., 2011] Urbani, J., van Harmelen, F., Schlobach, S., and Bal, H. (2011). QueryPIE: Backward reasoning for OWL Horst over very large knowledge bases. In *ISWC*.

[Weiss et al., 2008] Weiss, C., Karras, P., and Bernstein, A. (2008). Hexastore: Sextuple indexing for Semantic Web data management. *PVLDB*.

[Wiederhold, 1992] Wiederhold, G. (1992). Mediators in the architecture of future information systems. *IEEE Computer*, 25(3).

[Wilkinson et al., 2003] Wilkinson, K., Sayers, C., Kuno, H., and Reynolds, D. (2003). Efficient RDF storage and retrieval in Jena2. In *SWDB*.

Appendix

A Appendix

A.1 Appendix of Section 3.2

A.1.1 Proofs

Proof of Proposition 3.3

Proof. We provide an upper bound the number of reformulations explored during the reformulation of a BGPQ by analyzing the producer-consumer dependencies among rules w.r.t. the form of the query. Given a query form Q and an ontology O, we denote by $\#\text{explored}(Q, O)$ the number of explored reformulations during the execution of $\text{Reformulate}_c(q, O)$ for a BGPQ q of form Q.

First, we notice that for the most general query form $Q(\bar{x}) \leftarrow t_1, t_2, \ldots, t_n$ (where the t_i are triples), it holds that:

$$\#\text{explored}(Q, O) \leq \prod_{1 \leq i \leq n} \#\text{explored}(Q_i, O)$$

where Q_i has the form $Q_i(x_i) \leftarrow t_i$, with x_i the list of variables in t_i. We now analyze the reformulations obtained for the different forms of queries composed of a single triple.

Let us consider the query form $Q_0(\bar{x}) \leftarrow (s, v, o)$, where s and o are values or variables and v is a variable. The rules in (3.3) are the only rules that can consume Q_0. They produce queries of the form $Q_1(\bar{x_1}) \leftarrow (s, \hookleftarrow_d, o)$ or $Q_2(\bar{x_2}) \leftarrow (s, \prec_{sc}, o)$ (and there are two similar cases with properties $\hookrightarrow_r$ and $\prec_{sp}$). Since no rule feeds rule (3.3), it holds that:

$$\#\text{explored}(Q_0, O) \leq 2(\#\text{explored}(Q_1, O) + \#\text{explored}(Q_2, O))$$

Queries of the form Q_2 only feed rules from (3.14) to (3.18). These rules always produce queries of the form $Q_2(\bar{x_2}) \leftarrow (s, \prec_{sc}, o)$, where s and o are either values from $\text{Val}(O)$ or variables. Moreover, there are at most 2 variables in $\text{Var}(\text{Q}_2)$, which can only be instantiated by values from $\text{Val}(O)$ or variables.

In the end,

$$\#\text{explored}(Q_2, O) \leq (\#\text{Val}(O) + 1)^2(\#\text{Val}(O) + 1)^2 = (\#\text{Val}(O) + 1)^4.$$

Concerning a query of the form Q_1, either rule (3.5) can be applied, then produced queries have the form $Q_3(\bar{x_3}) \leftarrow (v_1, \prec_{sp}, \mathtt{p}), (\mathtt{c}, \prec_{sc}, v_2)$, or a rule from (3.6) to (3.13) can be applied. In the latter case, we observe that all further produced queries will have the form $Q_4(\bar{x_4}) \leftarrow (s, \mathtt{p}, o)$ with $\mathtt{p} \in \{\hookleftarrow_d, \prec_{sc}, \prec_{sp}\}$, s and o belonging to $\mathrm{Val}(O) \cup \mathrm{Var}(\mathrm{Q}_4)$, and there is at most one variable among s and o.

So, we obtain by counting the number of possible values by position that

$$\#\mathrm{explored}(Q_3, O) \leq ((\#\mathrm{Val}(O) + 1)^2 \#\mathrm{Val}(O))^2$$

and

$$\#\mathrm{explored}(Q_4, O) \leq 3(\#\mathrm{Val}(O) + 1)^4.$$

In the end,

$$\begin{aligned} \#\mathrm{explored}(Q_1, O) &\leq (\#\mathrm{Val}(O) + 1)^4 \#\mathrm{Val}(O)^2 + 3(\#\mathrm{Val}(O) + 1)^4 \\ &\leq (\#\mathrm{Val}(O) + 1)^6 + 3(\#\mathrm{Val}(O) + 1)^4 \end{aligned}$$

It follows that the maximal number of explored reformulations for an input query with a single triple is $O(\#\mathrm{Val}(O)^6)$, hence in $O(\#\mathrm{Val}(O)^{6|q|})$ for a BGPQ q. Note that $|q|$ is a rough upper bound, since one should only consider the number of triples in q with an RDFS property or a variable in property position.

□

Proof of Lemma 3.1

The only entailment rule in $\mathcal{R}_c$ that allows one to infer a new triple with property $\prec_{sc}$ (respectively $\prec_{sp}$) is the rule `rdfs11` (resp. `rdfs5`). Since this rule states the transitivity of the property $\prec_{sc}$ (resp. $\prec_{sp}$), it holds that $(\mathtt{c}, \prec_{sc}, \mathtt{c}') \in G^{\mathcal{R}_c}$ iff G contains a non-empty $\prec_{sc}$-chain from $\mathtt{c}$ to $\mathtt{c}'$ (resp. $(\mathtt{p}, \prec_{sp}, \mathtt{p}') \in G^{\mathcal{R}_c}$ iff G contains a non-empty empty $\prec_{sp}$-chain from $\mathtt{p}$ to $\mathtt{p}'$).

Assume now that $(\mathtt{p}', \hookleftarrow_d, \mathtt{c}') \in G^{\mathcal{R}_c}$. The only entailment rules in $\mathcal{R}_c$ that entail a triple with property $\hookleftarrow_d$ are `ext1` and `ext3`. The body of these rules contain a triple with property $\hookleftarrow_d$, so there exists an entailment chain (of triples with $\hookleftarrow_d$ property) of length $l \geq 0$ starting from G and using only rules `ext1` and `ext3`. We prove by induction on l that G contains a triple $(\mathtt{p}, \hookleftarrow_d, \mathtt{c})$, a (possibly empty) $\prec_{sp}$-chain from $\mathtt{p}'$ to $\mathtt{p}$ and a (possibly empty) $\prec_{sc}$-chain from $\mathtt{c}$ to $\mathtt{c}'$.

- If $l = 0$, then $(\mathtt{p}', \hookleftarrow_d, \mathtt{c}') \in G$ and there are an empty $\prec_{sp}$-chain from $\mathtt{p}'$ to $\mathtt{p}'$ and an empty $\prec_{sc}$-chain from from $\mathtt{c}'$ to $\mathtt{c}'$.
- Otherwise ($l > 0$), the last rule applied in the chain is:
  - either `ext1`, so $G^{\mathcal{R}_c}$ contains a triple $(\mathtt{p}', \hookleftarrow_d, \mathtt{c}_1)$, which results from an entailment chain of length $l-1$ starting from G and using only rules `ext1` and `ext3`, and a triple $(\mathtt{c}_1, \prec_{sc}, \mathtt{c}')$. By induction hypothesis, we know that G contains a triple $(\mathtt{p}, \hookleftarrow_d, \mathtt{c})$, a (possibly empty) $\prec_{sp}$-chain from $\mathtt{p}'$ to $\mathtt{p}$ and a (possibly empty) $\prec_{sc}$-chain from $\mathtt{c}$ to $\mathtt{c}_1$. Moreover, by using the first point of the lemma (proved above), $(\mathtt{c}_1, \prec_{sc}, \mathtt{c}') \in G^{\mathcal{R}_c}$ implies that G contains a non-empty $\prec_{sc}$-chain from $\mathtt{c}_1$ to $\mathtt{c}'$. So, concatenating the two $\prec_{sc}$-chains, we obtain a $\prec_{sc}$-chain from $\mathtt{c}$ to $\mathtt{c}'$. Hence, G contains a triple $(\mathtt{p}, \hookleftarrow_d, \mathtt{c})$, a (possibly empty) $\prec_{sp}$-chain from $\mathtt{p}'$ to $\mathtt{p}$ and a $\prec_{sc}$-chain from $\mathtt{c}$ to $\mathtt{c}'$.

– or `ext3`, and the proof is similar to that for `ext1`, replacing $\prec_{sc}$-chains by $\prec_{sp}$-chains.

We have proven that $(\mathtt{p}', \hookrightarrow_d, \mathtt{c}') \in G^{\mathcal{R}_c}$ implies that G contains a triple $(\mathtt{p}, \hookrightarrow_d, \mathtt{c})$, a (possibly empty) $\prec_{sp}$-chain from $\mathtt{p}'$ to $\mathtt{p}$ and a (possibly empty) $\prec_{sc}$-chain from $\mathtt{c}$ to $\mathtt{c}'$. The converse implication is straightforward: from the two first points of the lemma, we obtain $(\mathtt{c}, \prec_{sc}, \mathtt{c}') \in G^{\mathcal{R}_c}$ and $(\mathtt{p}', \prec_{sp}, \mathtt{p}) \in G^{\mathcal{R}_c}$, then by one application of each entailment rule `ext1` and `ext3`, we obtain $(\mathtt{p}', \hookrightarrow_d, \mathtt{c}') \in G^{\mathcal{R}_c}$.

Proof of Theorem 3.3

For the sake of readability, we assume in the following that G does not contain blank nodes. So, we do not need non-standard query evaluation. This assumption can be done without loss of generality. Indeed, we may define a one-to-one mapping f from the blank nodes of G to fresh IRIs, apply f to G before any processing, and apply the inverse mapping f^{-1} to the answer tuples obtained considering $f(G)$ to get answers considering G.

With the above assumption, to prove statement 3.20, it remains to prove that $q(G, \mathcal{R}_c) = Q_c(G)$ holds. We first prove $Q_c(G) \subseteq q(G, \mathcal{R}_c)$ (soundness) then $q(G, \mathcal{R}_c) \subseteq Q_c(G)$ (completeness).

(soundness) We want to prove that for all $q'_{\sigma'}$ reformulation of q in Q_c, for all tuple t answer to $q'_{\sigma'}$ in G, there is G' obtained from G by application of some entailment rules to G such that t is an answer to q in G'. In other words, we want to prove that $q'_{\sigma'}(G, \emptyset) \subseteq q(G, \mathcal{R}_c)$. Since $q'_{\sigma'}(G, \emptyset) \subseteq q'_{\sigma'}(G, \mathcal{R}_c)$, it is sufficient to prove that $q'_{\sigma'}(G, \mathcal{R}_c) \subseteq q(G, \mathcal{R}_c)$.

The proof is done by induction on the length l of a sequence of reformulation rules leading to $q'_{\sigma'}$, starting from O and q.

Base step For $l = 0$, we have $q'_{\sigma'} = q$, so $q'_{\sigma'}(G, \mathcal{R}_c) \subseteq q(G, \mathcal{R}_c)$.

Inductive step For $l < \alpha$, suppose that $q'_{\sigma'}(G, \mathcal{R}_c) \subseteq q(G, \mathcal{R}_c)$ holds. Now at $l = \alpha$, $q'_{\sigma'}$ has been produced from $q''_{\sigma''}$ by the application of a reformulation rule (i) and $q''_{\sigma''}$ is a reformulation of q. So that sequence being of length $< \alpha$, we get $q''_{\sigma''}(G) \subseteq q(G, \mathcal{R}_c)$ by induction hypothesis. We will show that $q'_{\sigma'}(G, \mathcal{R}_c) \subseteq q''_{\sigma''}(G, \mathcal{R}_c)$.

There are basically two cases:

- the reformulation rule (i) instantiates a variable of $q''_{\sigma''}$ to generate $q'_{\sigma'}$ i.e., rule (i) is one of the following (3.3), (3.6)-(3.9), (3.14)-(3.16). In this case, $q'_{\sigma'}$ is contained in $q''_{\sigma''}$, so $q'_{\sigma'}(G, \mathcal{R}_c) \subseteq q''_{\sigma''}(G, \mathcal{R}_c)$.

- the reformulation rule (i) has the form $\frac{t_1 \in q_\sigma, t_2 \in O}{q_\sigma[t_1/t_3]}$ that replaces a triple in $q''_{\sigma''}$ by another one (or two for the rule (3.5)). Observe here that $\sigma' = \sigma''$ holds. If $\varphi(\bar{x}_{\sigma'}) \in q'_{\sigma'}(G, \mathcal{R}_c)$, then $\varphi(t_{3\sigma'}) \in G^{\mathcal{R}_c}$. Furthermore, the reformulation rules ensure that $\varphi(t_{3\sigma'}), t_2 \models_{\mathcal{R}_c} \varphi(t_{1\sigma''})$. As a result, $\varphi(t_{1\sigma''}) \in G^{\mathcal{R}_c}$, and φ is a total assignment of the variables of $q''_{\sigma''}$ such that $\varphi(\bar{x}_{\sigma'}) = \varphi(\bar{x}_{\sigma''}) \in q''_{\sigma''}(G, \mathcal{R}_c)$.

In each case, we get $q'_{\sigma'}(G, \mathcal{R}_c) \subseteq q''_{\sigma''}(G, \mathcal{R}_c)$ which concludes the proof of $q'_{\sigma'}(G, \emptyset) \subseteq q(G, \mathcal{R}_c)$, so $Q_c(G) \subseteq q(G, \mathcal{R}_c)$.

(completeness) We now show that $q(G, \mathcal{R}_c) \subseteq Q_c(G)$ with $Q_c(G) = \bigcup_{q'_{\sigma'} \in \text{Reformulate}_c(q,O)} q'_{\sigma'}(G, \emptyset)$, i.e., for each answer tuple $a \in q(G, \mathcal{R}_c)$, there exists $q'_{\sigma'} \in Q_c$ a reformulation of q using O such that $a \in q'_{\sigma'}(G, \emptyset)$.

In the following, we will consider that Q_c contains queries in which all the instantiated RDFS triples that belong to the ontology are kept; in other words, the triples removed by applications of rule (3.4) are restored in the resulting queries. This has no impact on the completeness of the algorithm, since the reformulations output in both versions have the same answers in G.

Let the query q be defined by $q(\bar{x}) \leftarrow t_1, t_2, \ldots, t_n$ with t_i being the body triples of q. An answer from $q(G, \mathcal{R}_c)$ has the form $\varphi(\bar{x})$, where φ is a homomorphism from body(q) to $G^{\mathcal{R}_c}$. If for all triples t_i from the body of q, $\varphi(t_i)$ is not an RDFS triple, then $\varphi(\text{body}(q)) \in G$ (because data triples are not entailed by $\mathcal{R}_c$), so a valid reformulation of q is q itself, since $q(G, \mathcal{R}_c) = q(G, \emptyset)$. Otherwise, there exists a triple t_i from the body of q such that $\varphi(t_i) \in O^{\mathcal{R}_c}$ and we will show that there exists $q'_{\sigma'}$ a reformulation of q where only t_i has been replaced by a BGP P such that $P \subseteq O$ and $\varphi(\bar{x}) = \varphi(\bar{x}_{\sigma'})$.

First case, $\varphi(t_i) = (\mathsf{c}, \prec_{sc}, \mathsf{c}') \in G^{\mathcal{R}_c}$; according to Lemma 3.1, there is $C = ((\mathsf{c}_i, \prec_{sc}, \mathsf{c}_{i+1}))_{1 \leq i < c}$ a $\prec_{sc}$-chain in G such that $\mathsf{c} = \mathsf{c}_1$ and $\mathsf{c}' = \mathsf{c}_c$. The triple t_i can have one of the following forms:

- $(\mathsf{c}, \prec_{sc}, \mathsf{c}')$, then we consider $q'_{\sigma'}$ obtained from q by applying rule (3.17) for each triple of C; finally $(\mathsf{c}, \prec_{sc}, \mathsf{c}')$ is replaced by $(\mathsf{c}_{c-1}, \prec_{sc}, \mathsf{c}') \in O$. Since $\sigma' = \emptyset$, $\varphi(\bar{x}) = \varphi(\bar{x}_{\sigma'})$.

- $(\mathsf{c}, \prec_{sc}, v')$, then we consider $q'_{\sigma'}$ obtained from q by applying rule (3.17) for each triple of C then (3.16); finally $(\mathsf{c}, \prec_{sc}, v')$ is replaced by $(\mathsf{c}_{c-1}, \prec_{sc}, \mathsf{c}') \in O$. Since $\sigma' = \{v' \mapsto \mathsf{c}'\}$, $\varphi(\bar{x}) = \varphi(\bar{x}_{\sigma'})$.

- (c, v_p, v), then we consider $q'_{\sigma'}$ obtained from q by applying rule (3.3) then (3.17) for each triple of C then (3.16); finally (c, v_p, v) is replaced by $(\mathsf{c}_{c-1}, \prec_{sc}, \mathsf{c}') \in O$. Since $\sigma' = \{v_p \mapsto \prec_{sc}, v \mapsto \mathsf{c}'\}$, $\varphi(\bar{x}) = \varphi(\bar{x}_{\sigma'})$.

- $(v, \prec_{sc}, \mathsf{c}')$, then we consider $q'_{\sigma'}$ obtained from q by applying rule (3.18) for each triple of C in inverse order then (3.15); finally $(v, \prec_{sc}, \mathsf{c}')$ is replaced by $(\mathsf{c}, \prec_{sc}, \mathsf{c}_2) \in O$. Since $\sigma' = \{v \mapsto \mathsf{c}\}$, $\varphi(\bar{x}) = \varphi(\bar{x}_{\sigma'})$.

- (v, v_p, c'), then we consider $q'_{\sigma'}$ obtained from q by applying rule (3.3) then (3.18) for each triple of C in inverse order then (3.15); finally (v, v_p, c') is replaced by $(\mathsf{c}, \prec_{sc}, \mathsf{c}_2) \in O$. Since $\sigma' = \{v_p \mapsto \prec_{sc}, v \mapsto \mathsf{c}\}$, $\varphi(\bar{x}) = \varphi(\bar{x}_{\sigma'})$.

- $(v, \prec_{sc}, v')$, then we consider $q'_{\sigma'}$ obtained from q by applying rule (3.14) then (3.17) for each triple of C then (3.16); finally $(v, \prec_{sc}, v')$ is replaced by $(\mathsf{c}_{c-1}, \prec_{sc}, \mathsf{c}') \in O$. Since $\sigma' = \{v' \mapsto \mathsf{c}', v \mapsto \mathsf{c}\}$, $\varphi(\bar{x}) = \varphi(\bar{x}_{\sigma'})$.

- (v, v_p, v'), then we consider $q'_{\sigma'}$ obtained from q by applying rule (3.3) then (3.14) then (3.17) for each triple of C then (3.16) ; finally (v, v_p, v') is replaced by $(\mathsf{c}_{c-1}, \prec_{sc}, \mathsf{c}') \in O$. Since $\sigma' = \{v_p \mapsto \prec_{sc}, v' \mapsto \mathsf{c}', v \mapsto \mathsf{c}\}$, $\varphi(\bar{x}) = \varphi(\bar{x}_{\sigma'})$.

Second case, $\varphi(t_i) = (\mathsf{p}, \hookleftarrow_d, \mathsf{c}) \in G^{\mathcal{R}_c}$; according to Lemma 3.1, there are three cases, depending on whether a chain is empty or not. We describe the case where none of the chains is empty, hence assuming that there exists $P = ((\mathsf{p}_i, \prec_{sp}, \mathsf{p}_{i+1}))_{1 \leq i \leq p}$ a $\prec_{sp}$-chain in G from p to p' and $(\mathsf{p}', \hookleftarrow_d, \mathsf{c}') \in G$ and there exists $C = ((\mathsf{c}_i, \prec_{sc}, \mathsf{c}_{i+1}))_{1 \leq i \leq c}$ a $\prec_{sc}$-chain in G from c' to c. The other cases are handled similarly using also rules (3.6) and (3.7). The triple t_i can have the following forms:

- $(\mathsf{p}, \hookleftarrow_d, \mathsf{c})$, then we consider $q'_{\sigma'}$ obtained from q by applying rule (3.12) for each triple in C in inverse order then (3.13) for each triple in P ; finally $(\mathsf{p}, \hookleftarrow_d, \mathsf{c})$ is replaced by $(\mathsf{p}', \hookleftarrow_d, \mathsf{c}') \in O$. Since $\sigma' = \emptyset$, $\varphi(\bar{x}) = \varphi(\bar{x}_{\sigma'})$.

- $(\mathsf{p}, \hookleftarrow_d, v')$, then we consider $q'_{\sigma'}$ obtained from q by applying rule (3.13) for each triple in P then (3.11) then (3.17) for each triple in C then (3.16) ; finally $(\mathsf{p}, \hookleftarrow_d, \mathsf{c})$ is replaced by $(\mathsf{c}_{c-1}, \prec_{sc}, \mathsf{c}) \in O$. Since $\sigma' = \{v' \mapsto \mathsf{c}\}$, $\varphi(\bar{x}) = \varphi(\bar{x}_{\sigma'})$.

- (p, v_p, v'), then we consider $q'_{\sigma'}$ obtained from q by applying rule (3.3) then (3.13) for each triple in P then (3.11) then (3.17) for each triple in C then (3.16) ; finally (p, v_p, v') is replaced by $(\mathsf{c}_{c-1}, \prec_{sc}, \mathsf{c}) \in O$. Since $\sigma' = \{v_p \mapsto \hookleftarrow_d, v' \mapsto \mathsf{c}\}$, $\varphi(\bar{x}) = \varphi(\bar{x}_{\sigma'})$.

- $(v, \hookleftarrow_d, \mathsf{c})$, then we consider $q'_{\sigma'}$ obtained from q by applying rule (3.12) for each triple in C in inverse order then (3.10) then (3.18) for each triple in P in inverse order then (3.15) ; finally $(v, \hookleftarrow_d, \mathsf{c})$ is replaced by $(\mathsf{p}, \prec_{sp}, \mathsf{p}_2) \in O$. Since $\sigma' = \{v \mapsto \mathsf{p}\}$, $\varphi(\bar{x}) = \varphi(\bar{x}_{\sigma'})$.

- (v, v_p, c), then we consider $q'_{\sigma'}$ obtained from q by applying rule (3.3) then (3.12) for each triple in C in inverse order then (3.10) then (3.18) for each triple in P in inverse order then (3.15) ; finally (v, v_p, c) is replaced by $(\mathsf{p}, \prec_{sp}, \mathsf{p}_2) \in O$. Since $\sigma' = \{v_p \mapsto \hookleftarrow_d, v \mapsto \mathsf{p}\}$, $\varphi(\bar{x}) = \varphi(\bar{x}_{\sigma'})$.

- $(v, \hookleftarrow_d, v')$, then we consider $q'_{\sigma'}$ is obtained from q by applying rule (3.5) then (3.18) for each triple in P inverse order then (3.15) then on the other triple, (3.17) for each triple in C then (3.16) ; finally $(v, \hookleftarrow_d, v')$ is replaced by $(\mathsf{p}, \prec_{sp}, \mathsf{p}_2), (\mathsf{c}_{c-1}, \prec_{sp}, \mathsf{c}) \in O$. Since $\sigma' = \{v \mapsto \mathsf{p}, v' \mapsto \mathsf{c}\}$, $\varphi(\bar{x}) = \varphi(\bar{x}_{\sigma'})$.

- (v, v_p, v'), then we consider $q'_{\sigma'}$ obtained from q by applying rule (3.3) then (3.5) then (3.18) for each triple in P in inverse order then (3.15) then on the other triple, (3.17) for each triple in C then (3.16) ; finally (v, v_p, v') is replaced by $(\mathsf{p}, \prec_{sp}, \mathsf{p}_2), (\mathsf{c}_{c-1}, \prec_{sp}, \mathsf{c}) \in O$. Since $\sigma' = \{v \mapsto \mathsf{p}, v' \mapsto \mathsf{c}\}$, $\varphi(\bar{x}) = \varphi(\bar{x}_{\sigma'})$.

Hence, for each triple t_i in q such that $\varphi(t_i) \in O^{\mathcal{R}_c}$, there is $q'_{\sigma'}$, a reformulation of q, where only t_i has been replaced by a BGP P such that $P \subseteq O$ and $\varphi(\bar{x}) = \varphi(\bar{x}_{\sigma'})$. It follows that there is $q''_{\sigma''}$, a reformulation of q, in which all body triples of q mapped by φ to $O^{\mathcal{R}_c}$ have been replaced by triples that belong to O, such that $\varphi(\bar{x}) = \varphi(\bar{x}_{\sigma''})$. Since the other triples of q are necessarily mapped by φ to G (actually, $G \setminus O$), we conclude that $\varphi(\bar{x}) = \varphi(\bar{x}_{\sigma''})$ is an answer to $q''_{\sigma''}$ in G. This concludes the proof of statement (3.20), which is the only part of Theorem 3.3 needed in the proof of Theorem 3.5. Statement (3.21) actually follows from the next lemma (Lemma A.1).

Proof of Theorem 3.5

Lemma A.1. *For all RDF graph G, it holds that:*

$$G^{\mathcal{R}_a \cup \mathcal{R}_c} = \left(G^{\mathcal{R}_a}\right)^{\mathcal{R}_c}$$

Proof. For one direction: $\left(G^{\mathcal{R}_a}\right)^{\mathcal{R}_c} \subseteq G^{\mathcal{R}_a \cup \mathcal{R}_c}$. The proof is trivial.

For the converse direction $G^{\mathcal{R}_a \cup \mathcal{R}_c} \subseteq \left(G^{\mathcal{R}_a}\right)^{\mathcal{R}_c}$. We take a triple $t \in G^{\mathcal{R}_a \cup \mathcal{R}_c}$, and differentiate two cases:

- either t is not an RDFS triple, then by applying Theorem 1 of [Goasdoué et al., 2013], $t \in G^{\mathcal{R}_a}$. In other words, assertion rules suffice to entail all RDF data triples.
- or t is an RDFS triple. Since the RDFS ontology O of G does not contain an RDFS property as subject or object, the entailment rule `rdfs7` does not entail RDFS triples. So, $t \in O$ or t has been produced by a rule in $\mathcal{R}_c$. Moreover, all rules in $\mathcal{R}_c$ have a body that contains only RDFS triples, so $t \in O$ or t has been entailed from O using rules in $\mathcal{R}_c$, i.e., $t \in O^{\mathcal{R}_c}$. We also know that $O^{\mathcal{R}_c} \subseteq \left(G^{\mathcal{R}_a}\right)^{\mathcal{R}_c}$, so $t \in \left(G^{\mathcal{R}_a}\right)^{\mathcal{R}_c}$.

  In both cases, we have proven that $t \in \left(G^{\mathcal{R}_a}\right)^{\mathcal{R}_c}$.

□

of the theorem.

$$
\begin{aligned}
q(G, \mathcal{R}_a \cup \mathcal{R}_c) &= q(G^{\mathcal{R}_a \cup \mathcal{R}_c}) \\
&= q(\left(G^{\mathcal{R}_a}\right)^{\mathcal{R}_c}) \quad \text{by Lemma A.1} \\
&= q(G^{\mathcal{R}_a}, \mathcal{R}_c) \quad \text{by definition of query answering} \\
&= \overbrace{Q_c(G^{\mathcal{R}_a})} \quad \text{by Theorem 3.3} \\
&= \overbrace{Q_c(G, \mathcal{R}_a)} \quad \text{by definition of query answering} \\
&= \overbrace{Q_{c,a}(G)} \quad \text{by Theorem 6 of [Goasdoué et al., 2013] since } Q_c \text{ is without RDFS triples}
\end{aligned}
$$

□

A.1.2 Experiments Appendix

This section describes a set of experiments related to RDF query reformulation under RDFS ontologies.

Experiments with our software: OntoSQL

The experiments are presented in Section 3.2.7.

The benchmark lubm-exists is available online as well as a dataset generator: `http://www.informatik.uni-bremen.de/~clu/combined/`

We have used a restricted version to RDFS triples of the LUBM exists ontology initially in OWL leading to 209 triples.

The list of executed queries is the following:

```
Q01<$X, $Y> :-
    triple($X, <rdf:type>, <univ:Employee>),
    triple($X, <univ:worksFor>, <http://www.Department0.University0.edu>),
    triple($P, <rdfs:subPropertyOf>, <univ:degreeFrom>),
    triple($X, $P, $Y);

Q02<$X, $Y> :-
    triple($X, <rdf:type>, <univ:Employee>),
    triple($X, <univ:worksFor>, <http://www.Department0.University0.edu>),
```

```
    triple($P, <rdfs:domain>, <univ:Employee>),
    triple($X, $P, $Y);

Q03<$X, $Y> :-
    triple($X, <rdf:type>, <univ:Employee>),
    triple($X, <univ:worksFor>, <http://www.Department0.University0.edu>),
    triple($P, <rdfs:range>, <univ:Course>),
    triple($X, $P, $Y);

Q04<$X, $Y, $U, $V, $W> :-
    triple($C, <rdfs:subClassOf>, <univ:Employee>),
    triple($X, <rdf:type>, $C),
    triple($X, <univ:worksFor>, <http://www.Department0.University0.edu>),
    triple($X, <univ:degreeFrom>, $Y),
    triple($X, <univ:name>, $U),
    triple($X, <univ:emailAddress>, $V),
    triple($X, <univ:telephone>, $W);

Q05<$X, $Y> :-
    triple($C, <rdfs:subClassOf>, <univ:Employee>),
    triple($X, <rdf:type>, $C),
    triple($X, <univ:worksFor>, <http://www.Department0.University0.edu>),
    triple($X, <univ:doctoralDegreeFrom>, $Y);

Q06<$X, $Y, $Z> :-
    triple($Y, <rdfs:subClassOf>, <univ:Professor>),
    triple($X, <rdf:type>, $Y),
    triple($X, <univ:doctoralDegreeFrom>, $U),
    triple($X, <univ:memberOf>, $Z);

Q07<$X, $Y, $Z> :-
    triple($C, <rdfs:subClassOf>, <univ:Student>),
    triple($X, <rdf:type>, $C),
    triple($X, <rdf:type>, $C),
    triple($X, <univ:advisor>, $Y),
    triple($Y, <univ:teacherOf>, $Z),
    triple($X, <univ:takesCourse>, $Z);

Q08<$X, $W, $Y, $Z> :-
    triple($W, <rdfs:subClassOf>, <univ:Person>),
    triple($X, <rdf:type>, $W),
    triple($X, <univ:advisor>, $Y),
    triple($Y, <univ:teacherOf>, $Z),
    triple($X, <univ:takesCourse>, $Z);

Q09<$X, $Y, $Z> :-
    triple($C, <rdfs:subClassOf>, <univ:Faculty>),
    triple($X, <rdf:type>, $C),
    triple($X, <univ:degreeFrom>, $Y),
```

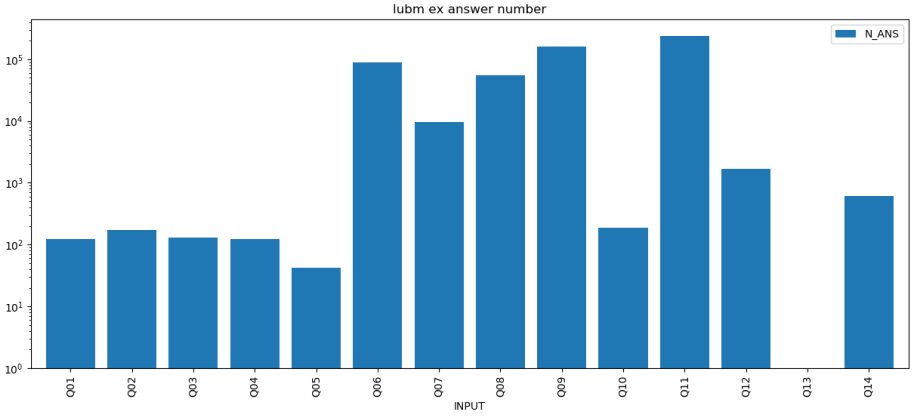


Figure A.1: Number of answers per query.

```
    triple($X, <univ:memberOf>, $Z);

Q10<$X> :-
    triple($P, <rdfs:subPropertyOf>, <univ:degreeFrom>),
    triple($X, $P, $Y),
    triple($X, <univ:memberOf>, <http://www.Department0.University0.edu>);

Q11<$X> :-
    triple($X, $P, $Y),
    triple($P, <rdfs:subPropertyOf>, <univ:degreeFrom>),
    triple($X, <univ:memberOf>, $Y);

Q12<$X, $Y, $P> :-
    triple($P, <rdfs:subPropertyOf>, <univ:worksFor>),
    triple($X, $P, <http://www.Department0.University0.edu>),
    triple($Y, $P, <http://www.Department0.University0.edu>);

Q13<$X, $Y, $P> :-
    triple($P, <rdfs:range>, <univ:Publication>),
    triple($X, $P, $Z),
    triple($Y, $P, $Z); Q14<$CX, $CY> :-
    triple($CX, <rdfs:subClassOf>, <univ:Professor>),
    triple($CY, <rdfs:subClassOf>, <univ:Course>),
    triple($X, <rdf:type>, $CX),
    triple($Y, <rdf:type>, $CY),
    triple($X, <univ:teacherOf>, $Y);
```

We plot the number of answers per query in Figure A.1.

Reformulation

We plot the number of conjunctive queries in the union output by the global reformulation algorithm, which output is denoted $Q_{c,a}$ in Figure A.2.

Saturation

The initialization time comprises 2 steps:

- creating a file containing the data triples and the saturation of the ontology. (**T_DUMP**),

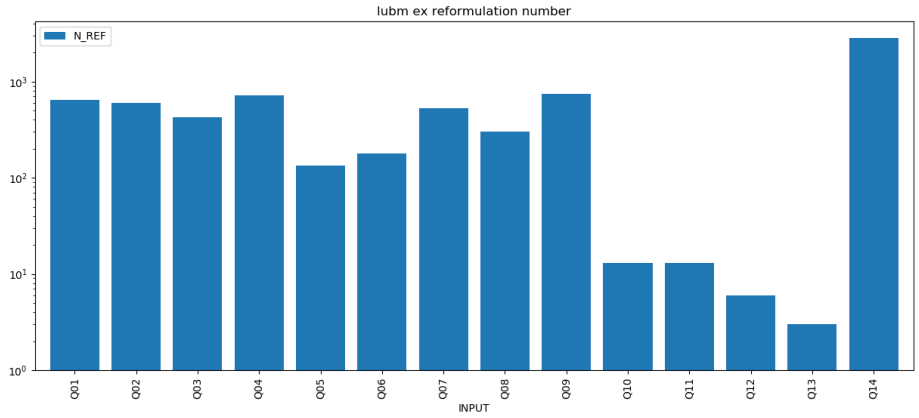


Figure A.2: Reformulation size in number of BGPQs in their union.

- loading the whole graph into OntoSQL and saturating the data triples (**T_LOAD**).

The times in milliseconds are:

T_DUMP	T_LOAD	TOTAL
18137	1271592	1289729

Experiments with other platforms

We tested Stardog and AllegroGraph (last publicly available versions as of Nov 30, 2018) with respect to the completeness of their answers.

Stardog

We used a simple dataset retrieved by the following query:

```
// Query a
// no reasoning

stardog query Gex "select distinct * where { ?s ?p ?o }"
```

s	p	o
urn:posterCP	rdfs:subClassOf	urn:ConfP
_:b0	rdfs:subClassOf	urn:ConfP
urn:inproc	rdfs:domain	urn:ConfP
urn:inproc	rdfs:range	urn:Conference
urn:a	urn:inproc	urn:edbt
urn:a	rdf:type	urn:posterCP
urn:b	urn:inproc	_:bc
urn:b	rdf:type	_:b0

Then we ask more queries:

```
// Query b
// -r means reasoning

stardog query -r Gex "select distinct * where { ?s ?p ?o }"
```

s	p	o	Commentary
urn:a	urn:inproc	urn:edbt	OK (explicit triple)
urn:b	urn:inproc	_:bc	OK (explicit triple)
urn:a	rdf:type	urn:ConfP	OK (implicit triple: a inproc edbt, inproc domain ConfP)
urn:b	rdf:type	urn:ConfP	OK (implicit triple: b inproc _:bc, inproc domain ConfP)
urn:b	rdf:type	_:b0	OK (explicit triple)
urn:edbt	rdf:type	urn:Conference	OK (implicit triple: a inproc edbt, inproc range Conference)
_:bc	rdf:type	urn:Conference	OK (implicit triple: b inproc _:bc, inproc range Conference)
urn:a	rdf:type	urn:posterCP	OK (explicit triple)
urn:a	rdf:type	owl:Thing	
urn:b	rdf:type	owl:Thing	

Note that all the ontology triples have disappeared . At the same time, we have four new triples typing all non-schema resources as owl:Thing (which is a bit surprising given that RDFS, not OWL, reasoning was intended).

Then we ask:

```
// Query c
stardog query -r Gex "select distinct * where {
    ?c rdfs:subClassOf <urn:ConfP> .
    ?a rdf:type ?c .
    ?a <urn:inproc> ?conf .
}"
```

c	a	conf
urn:posterCP	urn:a	urn:edbt
urn:ConfP	urn:a	urn:edbt
:b0	urn:b	:bc
urn:ConfP	urn:b	_:bc

This result is correct. We introduce a second dataset, a slightly different RDF graph:

s	p	o
<urn:posterCP>	rdfs:subClassOf	<urn:ConfP>
_:b0	rdfs:subClassOf	<urn:ConfP>
<urn:inproc>	rdfs:domain	<urn:ConfP>
<urn:inproc>	rdfs:range	<urn:Conference>
<urn:a>	rdf:type	<urn:posterCP>
<urn:b>	rdf:type	_:b0
<urn:a>	<urn:inproc>	<urn:edbt>
<urn:b>	<urn:inproc>	<_:bc>
<urn:ConfP>	rdfs:subClassOf	<urn:Paper>

Then we ask:

```
// Query d

stardog query -r Gex "select distinct * where {
    ?s rdfs:subClassOf <urn:Paper>
}"
```

s
urn:Paper
urn:ConfP
urn:posterCP
owl:Nothing
_:b0

This result is correct. We extend the same query to also go over the data:

```
// Query e

stardog query -r Gex "select distinct * where {
    ?c rdfs:subClassOf <urn:Paper> .
    ?a rdf:type ?c .
}"
```

c	a
urn:Paper	urn:a
urn:Paper	urn:b
urn:ConfP	urn:a
urn:ConfP	urn:b
urn:posterCP	urn:a
_:b0	urn:b

The result is correct. Then we extend the query with one extra triple:

```
// Query f

stardog query -r Gex "select distinct * where {
    ?c ?p <urn:Paper> .
    ?a rdf:type ?c .
}"
```

The result is empty, whereas it should have been the same as above. It appears that Stardog is unable to bind property variables on schema triples. If the property is specified, reasoning is performed; otherwise, it only matches property variables on the data, which is incomplete. A similar result is obtained with domain querying:

```
// Query g

stardog query -r Gex "select distinct * where {
    ?c rdfs:domain <urn:Paper>
}"
```

c
owl:bottomDataProperty
owl:bottomObjectProperty
urn:inproc

The result is OK (again, despite the surprising apparition of OWL when we wanted RDFS). However, a slight change:

```
// Query h

stardog query -r Gex "select distinct * where {
    ?c ?p <urn:Paper>
}"
```

c	p
urn:a	rdf:type
urn:b	rdf:type

loses the urn:inproc expected result! Thus, Stardog is incomplete for the RDFS (subclass, subproperty, domain, range) inference setting.

Allegrograph

We loaded it the second dataset above and asked some of preceding queries:

```
// Query d

select distinct * where {
    ?s rdfs:subClassOf <urn:Paper>
}
```

s
urn: ConfP
urn:posterCP
_bDF3893B2x1

Compared to Stardog, it misses urn:Paper.

```
// Query e

select distinct * where {
    ?c rdfs:subClassOf <urn:Paper> .
    ?a rdf:type ?c .
}
```

a	c
urn:b	_:bDF3893B2x1
urn:a	urn:posterCP
urn:b	urn:ConfP
urn:a	urn:ConfP

Compared to Stardog, we are missing that a and b are papers (subclass is not considered reflexive). However, we see that AllegroGraph does reason on the schema and the data.

```
// Query f

select distinct * where {
    ?c ?p <urn:Paper> .
```

```
    ?a rdf:type ?c .
}
```

a	c	p
urn:a	urn:ConfP	rdfs:subClassOf
urn:b	urn:ConfP	rdfs:subClassOf
urn:a	urn:posterCP	rdfs:subClassOf
urn:b	_:bDF3893B2x1	rdfs:subClassOf

AllegroGraph behaves better on this query than Stardog. However:

```
// Query g

select distinct * where {
    ?c rdfs:domain <urn:Paper>
}
```

yields no results, whereas we should have obtained urn:inProc. This shows that AllegroGraph is incomplete when it comes to reasoning with domain and range constraints.

A.2 Experiments details of Section 3.3.6

After detailing the queries and DBLP ontology used in experiments from Section 3.3.6, we conducted tests on Virtuoso to check its reformulation capabilities in Section A.2.2. You can find the code and the instructions to reproduce the experiments online[1].

A.2.1 Queries and DBLP ontology

LUBM

BGP queries on the LUBM dataset:

```
# star query
# one class answer variable
Q01<$X, $Y, $Z> :-
  triple($X, <rdf:type>, $Y),
  triple($X, <lubm:doctoralDegreeFrom>, $U),
  triple($X, <lubm:memberOf>, $Z);

# square query
# one class answer variable
Q02<$X, $W, $Y, $Z> :-
  triple($X, <rdf:type>, $W),
  triple($X, <lubm:advisor>, $Y),
  triple($Y, <lubm:teacherOf>, $Z),
  triple($X, <lubm:takesCourse>, $Z);
```

[1] https://gitlab.inria.fr/mburon/graph-layout-experiments/-/blob/8eacd5267af39b327e3bcb0b7da1a82360ebb452/README.org

```
# triangle query
# one class answer variable
Q03<$Z, $W> :-
  triple($X, <rdf:type>, <lubm:Student>),
  triple($Y, <rdf:type>, <lubm:GraduateStudent>),
  triple($Z, <rdf:type>, $W),
  triple($X, <lubm:advisor>, $Z),
  triple($Y, <lubm:advisor>, $Z);

# star query
# one property variable
# one schema triple
Q04<$X, $Y> :-
    triple($X, <rdf:type>, <lubm:Employee>),
    triple($X, $P, $Y),
    triple($P, <rdfs:range>, <lubm:University>);

# 5-path query
# two class answer variables
# two schema triples
Q05<$P, $PC, $O, $OC> :-
   triple($O, <rdf:type>, $OC),
   triple($OC, <rdfs:subClassOf>, <lubm:Organization>),
   triple($P, <lubm:memberOf>, $O),
   triple($P, <rdf:type>, $PC),
   triple($PC, <rdfs:subClassOf>, <lubm:Professor>);

# 4-path query
# one class answer variable
Q06<$Z,$S1> :-
  triple($Z, <rdf:type>, $S1),
  triple($Y, <rdf:type>, <lubm:GraduateStudent>),
  triple($V, <rdf:type>, <lubm:GraduateCourse>),
  triple($Z, <lubm:teacherOf>, $V),
  triple($Y, <lubm:takesCourse>, $V);

# 3-path query
# one class answer variable
Q07<$X, $Y, $Z, $PC> :-
   triple($X, <rdf:type>, $PC),
   triple($Y, <rdf:type>, <lubm:Department>),
   triple($X, <lubm:memberOf>, $Y),
   triple($Y, <lubm:subOrganizationOf>, <http://www.University0.edu>),
   triple($X, <lubm:emailAddress>, $Z);

# triangle query
# one property answer variable
# one triple schema
Q08<$X, $P, $O> :-
```

```
   triple($X, <lubm:memberOf>, $O),
   triple($O, <lubm:subOrganizationOf>, $SO),
   triple($X, $P, $SO),
   triple($P, <rdfs:subPropertyOf>, <lubm:degreeFrom>);

# complex query
# one property variable
# one class variable
# two schema triples
Q09<$X, $OX, $EX, $Y, $OY, $TY> :-
   triple($X, <lubm:memberOf>, $OX),
   triple($X, <lubm:emailAddress>, $EX),
   triple($OX, <lubm:subOrganizationOf>, $SO),
   triple($X, $P, $SO),
   triple($SO, <rdf:type>, $SOC),
   triple($P, <rdfs:range>, $SOC),
   triple($SOC, <rdfs:subClassOf>, <lubm:Organization>),
   triple($X, <lubm:advisor>, $Y),
   triple($Y, <lubm:memberOf>, $OY),
   triple($OY, <lubm:subOrganizationOf>, $SO),
   triple($Y, <lubm:telephone>, $TY);

# atomic query
# one property variable
Q10<$X> :-
        triple($X, $P, $Y);

Q11<$X> :-
        triple($P, <rdfs:subPropertyOf>, <lubm:degreeFrom>),
        triple($X, $P, $Y),
        triple($X, <lubm:memberOf>, <http://www.Department0.University0.edu>);

Q12<$X, $Y, $P> :-
        triple($X, $P, <http://www.Department0.University0.edu>),
        triple($Y, $P, <http://www.Department0.University0.edu>);

Q13<$X, $P, $P1> :-
        triple($X, $P, <http://www.Department0.University0.edu>),
        triple($X, $P1, <http://www.Department0.University0.edu>);

Q14<$X, $Y, $P> :-
         triple($X, $P, $Z),
         triple($X, <lubm:advisor>, $Y),
         triple($Y, $P, $Z);
```

DBLP

We use the DBLP ontology in Figure A.3 for our experiments. BGP queries on the DBLP dataset:

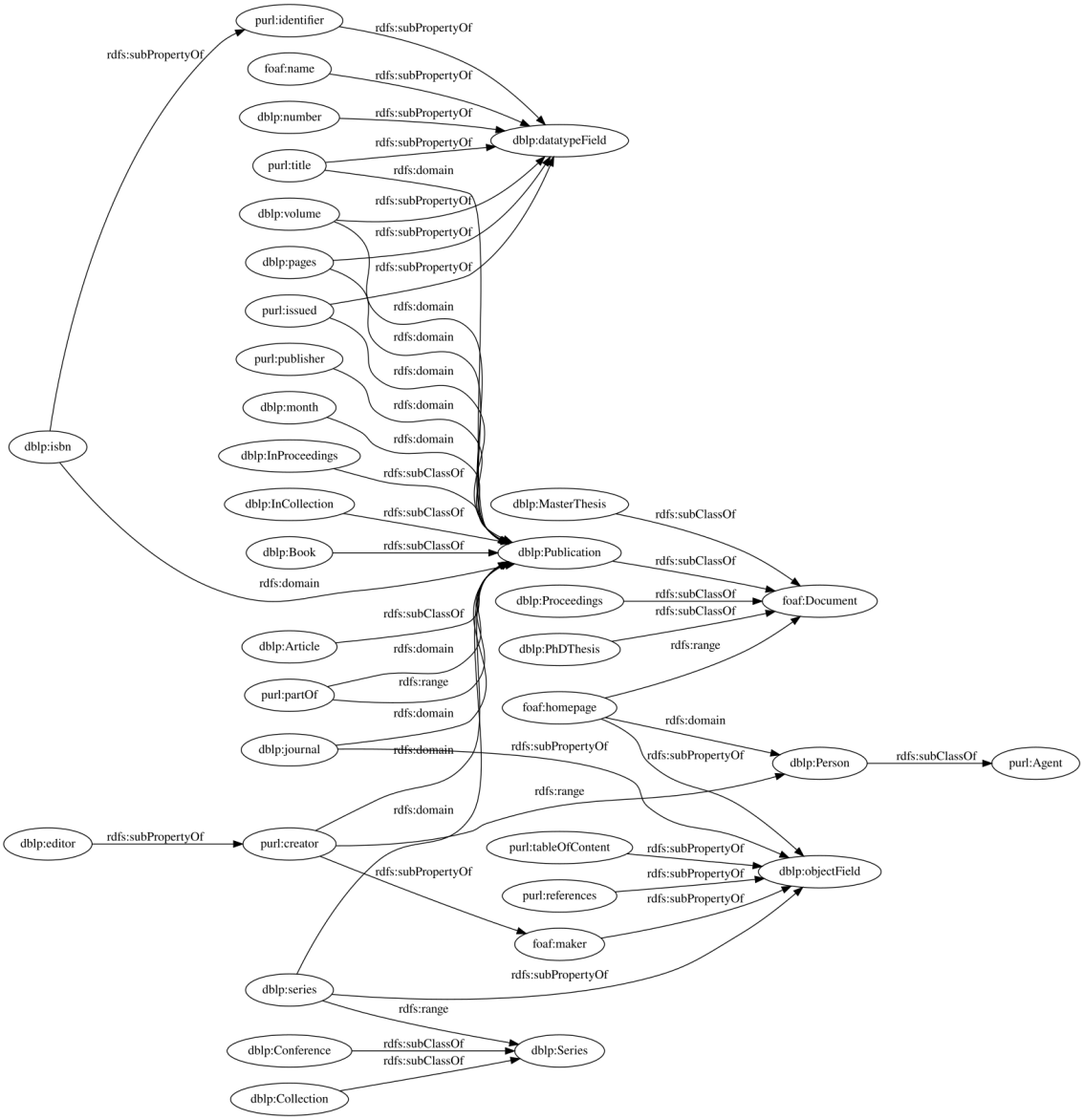


Figure A.3: DBLP ontology.

```
# square query
# no constant
Q01<$X, $Z, $V> :-
triple($X, <rdf:type>, $Z),
        triple($X, <dblp:datatypeField>, $V),
        triple($X, <purl:references>, $Y),
        triple($X, <dblp:series>, $S),
        triple($Y, <dblp:series>, $S);

# star query
# one constant
# one class answer variable
Q02<$X, $Z> :-
triple($X, <rdf:type>, $Z),
        triple($X, <dblp:objectField>, <http://dblp.l3s.de/.../www>);
```

```
# star query
# no constant
# one class answer variable
Q03<$X, $Y, $Z> :-
triple($X, <rdf:type>, $Z),
        triple($X, <dblp:datatypeField>, $Y);

# star query
# one constant
# one class answer variable
Q04<$X, $Y, $Z> :-
        triple($X, <rdf:type>, $Z),
        triple($X, <dblp:datatypeField>, $Y),
        triple($X, <purl-elt:publisher>, "Springer");

# bistar query
# one constant
# one prop answer variable
Q05<$X, $D, $H1, $U, $V, $H2, $T> :-
        triple($X, <purl:issued>, $D),
        triple($X, <http://xmlns.com/foaf/0.1/homepage>, $H1),
        triple($X, <dblp:objectField>, <http://dblp.l3s.de/.../www>),
        triple($X, $U, $V),
        triple($V, <http://xmlns.com/foaf/0.1/homepage>, $H2),
        triple($V, <purl-elt:title>, $T);

# bistar query
# one constant
# one prop answer variable
# one class answer variable
# one schema triple
Q06<$X, $U, $V> :-
        triple($X, <dblp:objectField>, <http://.../article156.html>),
        triple($X, $U, $V),
        triple($V, <rdf:type>, $Z),
        triple($Z, <rdfs:subClassOf>, <purl-elt:Agent>);

# strange shaped query
# two constants
# one prop answer variable
# one class answer variable
Q07<$X, $Y, $U, $V> :-
        triple($X, $U, $V),
        triple($Y, $U, $V),
        triple($V, <rdf:type>, $Z),
        triple($Z, <rdfs:subClassOf>, <purl-elt:Agent>),
        triple($X, <dblp:month>, "July"),
        triple($Y, <dblp:month>, "November");
```

```
# star query
# no constant
# one class answer variable
# one schema triple
Q08<$X, $C, $Z, $R, $P, $D, $H> :-
triple($X, <rdf:type>, $C),
        triple($C, <rdfs:subClassOf>, <dblp:Publication>),
triple($X, <dblp:objectField>, $Z),
triple($X, <purl-elt:title>, $R),
triple($X, <purl-elt:publisher>, $P),
        triple($X, <purl:issued>, $D),
        triple($X, <http://xmlns.com/foaf/0.1/homepage>, $H);

# bistar query
# no constant
# one class answer variable
# one schema triple
Q09<$X, $C, $F, $R, $P, $D, $H> :-
triple($X, <rdf:type>, $C),
        triple($C, <rdfs:subClassOf>, <dblp:Publication>),
triple($X, <dblp:objectField>, $F),
triple($X, <purl-elt:title>, $R),
triple($X, <purl-elt:publisher>, $P),
        triple($X, <purl:issued>, $D),
        triple($X, $Y, $Z),
        triple($Z, <rdf:type>, <purl-elt:Agent>),
        triple($Z, <http://xmlns.com/foaf/0.1/homepage>, $H);

# bistar query
# no constant
# one class answer variable
# one schema triple
# one property answer variable
Q10<$X, $C, $F, $R, $P, $D, $Y, $H> :-
triple($X, <rdf:type>, $C),
        triple($C, <rdfs:subClassOf>, <dblp:Publication>),
triple($X, <dblp:objectField>, $F),
triple($X, <purl-elt:title>, $R),
triple($X, <purl-elt:publisher>, $P),
        triple($X, <purl:issued>, $D),
        triple($X, $Y, $Z),
        triple($Z, <rdf:type>, <purl-elt:Agent>),
        triple($Z, <http://xmlns.com/foaf/0.1/homepage>, $H);

# it is a path query of homepage-1.maker-1.references.series
# none of these properties is selective
# it is a path query of homepage-1.maker-1.references.objectField
# none of these properties is selective
```

```
Q11<$W, $T1, $O, $V> :-
        triple($A, <http://xmlns.com/foaf/0.1/homepage>, $W),
        triple($Y, <http://xmlns.com/foaf/0.1/maker>, $A),
        triple($Y, <purl-elt:title>, $T1),
        triple($Y, <purl:references>, $Z),
       triple($Y, <dblp:objectField>, $O),
        triple($O, <dblp:datatypeField>, $V);

# it is a path query of homepage-1.maker-1.references.series
# none of these properties is selective
# with one unspecified class triple
Q12<$W, $T1, $T2, $C2, $S> :-
        triple($A, <http://xmlns.com/foaf/0.1/homepage>, $W),
        triple($Y, <http://xmlns.com/foaf/0.1/maker>, $A),
        triple($Y, <purl-elt:title>, $T1),
        triple($Y, <purl:references>, $Z),
        triple($Z, <purl-elt:title>, $T2),
        triple($Z, <rdf:type>, $C2),
        triple($Z, <dblp:series>, $S);

# it is a path query of homepage-1.maker-1.references.subject
# none of these properties is selective, except subject (~10%)
# with two unspecified class triples
Q13<$W, $T1, $C1, $T2, $C2, $D> :-
        triple($A, <http://xmlns.com/foaf/0.1/homepage>, $W),
        triple($Y, <http://xmlns.com/foaf/0.1/maker>, $A),
        triple($Z, <rdf:type>, $C1),
        triple($Y, <purl-elt:title>, $T1),
        triple($Y, <purl:references>, $Z),
        triple($Z, <purl-elt:title>, $T2),
        triple($Z, <rdf:type>, $C2),
        triple($Z, <purl-elt:subject>, $D);

# it is a path query of homepage-1.maker-1.$P.subject
# none of these properties is selective, except subject (~10%)
# with one generic triple
Q14<$W, $T1, $T2, $D> :-
        triple($A, <http://xmlns.com/foaf/0.1/homepage>, $W),
        triple($Y, <http://xmlns.com/foaf/0.1/maker>, $A),
        triple($Y, <purl-elt:title>, $T1),
        triple($Y, $P, $Z),
        triple($Z, <purl-elt:title>, $T2),
        triple($Z, <purl-elt:subject>, $D);

# it is a long path query of maker-1.references.references.series
# none of these properties is selective, except subject (~10%)
# with one generic triple
Q15<$V, $T, $U> :-
        triple($A, <dblp:datatypeField>, $V),
```

```
triple($X, <http://xmlns.com/foaf/0.1/maker>, $A),
triple($X, <purl:references>, $Y),
triple($Y, <purl-elt:title>, $T),
triple($Y, <purl:references>, $Z),
triple($Z, <purl:issued>, "1982"),
triple($Z, <dblp:series>, $S),
triple($S, <dblp:datatypeField>, $U);
```

A.2.2 Reasoning in Virtuoso

In Virtuoso, reasoning is done at query time by default. Virtuoso does not fully support reasoning with RDFS entailment rules (see Table 2 of the paper, recalled below). Therefore in our experiments, we provided Virtuoso with the reformulated queries in SPARQL. A main limitation of reasoning in Virtuoso is its partial support for subclass and subproperty constraints, i.e., only w.r.t. the rules **rdfs7** and **rdfs9**, as checked in Section called "SubClass and SubProperty Reasoning on Data triples". Moreover, it does not support the reasoning w.r.t. domain and range constraints (rules **rdfs2** and **rdfs3**), as explained in the official documentation. In Section "Transitivity of SubClassOf and SubPropertyOf", we check that there is no way to support the transitivity of the subClassOf and subPropertyOf properties (defined by **rdfs5** and **rdfs11**) by using the transitivity embedded in Virtuoso. Finally in Section A.2.2, we describe our attempt to encode RDFS reasoning using Virtuoso 8's new SPIN rules, however as we point out, that leads to some inconsistencies in query answers.

Rule	
rdfs2	$(\mathsf{p}, \hookleftarrow_d, \mathsf{o}), (\mathsf{s}_1, \mathsf{p}, \mathsf{o}_1) \rightarrow (\mathsf{s}_1, \tau, \mathsf{o})$
rdfs3	$(\mathsf{p}, \hookrightarrow_r, \mathsf{o}), (\mathsf{s}_1, \mathsf{p}, \mathsf{o}_1) \rightarrow (\mathsf{o}_1, \tau, \mathsf{o})$
rdfs5	$(\mathsf{p}_1, \prec_{sp}, \mathsf{p}_2), (\mathsf{p}_2, \prec_{sp}, \mathsf{p}_3) \rightarrow (\mathsf{p}_1, \prec_{sp}, \mathsf{p}_3)$
rdfs7	$(\mathsf{p}_1, \prec_{sp}, \mathsf{p}_2), (\mathsf{s}, \mathsf{p}_1, \mathsf{o}) \rightarrow (\mathsf{s}, \mathsf{p}_2, \mathsf{o})$
rdfs9	$(\mathsf{s}, \prec_{sc}, \mathsf{o}), (\mathsf{s}_1, \tau, \mathsf{s}) \rightarrow (\mathsf{s}_1, \tau, \mathsf{o})$
rdfs11	$(\mathsf{s}, \prec_{sc}, \mathsf{o}), (\mathsf{o}, \prec_{sc}, \mathsf{o}_1) \rightarrow (\mathsf{s}, \prec_{sc}, \mathsf{o}_1)$
ext1	$(\mathsf{p}, \hookleftarrow_d, \mathsf{o}), (\mathsf{o}, \prec_{sc}, \mathsf{o}_1) \rightarrow (\mathsf{p}, \hookleftarrow_d, \mathsf{o}_1)$
ext2	$(\mathsf{p}, \hookrightarrow_r, \mathsf{o}), (\mathsf{o}, \prec_{sc}, \mathsf{o}_1) \rightarrow (\mathsf{p}, \hookrightarrow_r, \mathsf{o}_1)$
ext3	$(\mathsf{p}, \prec_{sp}, \mathsf{p}_1), (\mathsf{p}_1, \hookleftarrow_d, \mathsf{o}) \rightarrow (\mathsf{p}, \hookleftarrow_d, \mathsf{o})$
ext4	$(\mathsf{p}, \prec_{sp}, \mathsf{p}_1), (\mathsf{p}_1, \hookrightarrow_r, \mathsf{o}) \rightarrow (\mathsf{p}, \hookrightarrow_r, \mathsf{o})$

RDFS Reasoning without Custom Inference Rules

In the official documentation, we can see that Virtuoso reasoning w.r.t. RDFS rules is limited to `rdfs7` and `rdfs9`. Reasoning w.r.t. domain and range is not supported.

See also:

- `http://docs.openlinksw.com/virtuoso/rdfsparqlruleenableinfr/`
- `http://vos.openlinksw.com/owiki/wiki/VOS/VirtSPARQLReasoningTutorial`

SubClass and SubProperty Reasoning on Data triples

Here, we test reasoning according to subClassOf and subPropertyOf using the following graph. It works as expected, except for an error at Point 4.

```
<file:///qa//Erik> <file:///qa//phoneNumber> "245470000" .
<file:///qa//Erik> <file:///qa//livesIn> <file:///qa//London> .
<file:///qa//Alice> <file:///qa//livesIn> <file:///qa//Paris> .
<file:///qa//Alice> <rdf:type> <file:///qa//Person> .
<file:///qa//Bob> <file:///qa//livesIn> <file:///qa//Berlin> .
<file:///qa//Bob> <rdf:type> <file:///qa//Person> .
<file:///qa//Dany> <file:///qa//livesIn> <file:///qa//Madrid> .
<file:///qa//Dany> <rdf:type> <file:///qa//Person> .


<file:///qa//Thing> <rdfs:subClassOf> <file:///qa//Thing> .
<file:///qa//Person> <rdfs:subClassOf> <file:///qa//Thing> .
<file:///qa//Place> <rdfs:subClassOf> <file:///qa//Thing> .
<file:///qa//City> <rdfs:subClassOf> <file:///qa//Place> .
<file:///qa//livesIn> <rdfs:range> <file:///qa//City> .
<file:///qa//contact> <rdfs:domain> <file:///qa//Person> .
<file:///qa//phoneNumber> <rdfs:subPropertyOf> <file:///qa//contact> .


<file:///qa//contact> <rdf:type> <rdf:Property> .
<file:///qa//Place> <rdf:type> <rdfs:Class> .
```

1. Clear and load the graph:

   ```
   isql-vt 1111 dba dba exec="SPARQL
   CLEAR GRAPH <urn:qa-test-graph>;"

   # move the graph file to vsp directory of virtuoso

   cp test-graph.nt loading/virtuoso-opensource/vsp/

   isql-vt 1111 dba dba exec="SPARQL
   LOAD <file:/test-graph.nt>
   INTO <urn:qa-test-graph>;"
   ```

2. Load the ontology

   ```
   isql-vt 1111 dba dba exec="sparql
   clear graph <urn:qa-test-onto>;"

   cp test-schema.nt loading/virtuoso-opensource/vsp/

   isql-vt 1111 dba dba exec="SPARQL
   LOAD <file:/test-schema.nt> INTO
   <urn:qa-test-onto>;"
   ```

3. Build the rule set from the schema

   ```
   isql-vt 1111 dba dba exec="
   ```

```
DELETE FROM sys_rdf_schema WHERE RS_NAME='urn:owl:inference:rules:qa-test';"

isql-vt 1111 dba dba exec="
rdfs_rule_set('urn:owl:inference:rules:qa-test', 'urn:qa-test-onto');"
```

4. Find the type of each subject using reasoning:

```
isql-vt 1111 dba dba exec="SPARQL
DEFINE input:inference 'urn:owl:inference:rules:qa-test'
SELECT *
FROM <urn:qa-test-graph>
WHERE { ?s a ?o}
ORDER BY ?s ?o;"
```

s	o
file:///qa//Bob	file:///qa//Thing
file:///qa//Alice	file:///qa//Person
file:///qa//Alice	file:///qa//Thing
file:///qa//Dany	file:///qa//Thing
2file:///qa//Place	rdfs:Class
file:///qa//contact	rdfs:Property

The answers set is not correct because Bob, Alice and Dany are of type Person and Person is a subclass of Thing, but only Alice is of type Person in the answers. We notice that Erik doesn't have a type, even if the domain of phoneNumber is Person, so domain is not used (the same for range, with London for example).

5. Find all the subjects of type Thing (correct result):

```
isql-vt 1111 dba dba exec="SPARQL
DEFINE input:inference 'urn:owl:inference:rules:qa-test'
SELECT *
FROM <urn:qa-test-graph>
WHERE { ?s a <file:///qa//Thing>}
ORDER BY ?s ;"
```

s
file:///qa//Bob
file:///qa//Alice
file:///qa//Dany

6. Query all the subjects and objects of the property contact, populated by its sub-property phoneNumber (correct result) :

```
isql-vt 1111 dba dba exec="SPARQL
DEFINE input:inference 'urn:owl:inference:rules:qa-test'
SELECT *
FROM <urn:qa-test-graph>
WHERE { ?s <file:///qa//contact> ?o}
ORDER BY ?s, ?o ;"
```

s	o
file:///qa//Erik	245470000

Transitivity of SubClassOf and SubPropertyOf

We show that there is no way to use the transitivity capabilities of Virtuoso to support the transitivity of the properties subClassOf and subPropertyOf.

1. Query all the subclass relations, no reasoning appears to have taken place:

```
isql-vt 1111 dba dba exec="SPARQL
DEFINE input:inference 'urn:owl:inference:rules:qa-test'
SELECT *
FROM <urn:qa-test-graph>
WHERE { ?s <rdfs:subClassOf> ?o }
ORDER BY ?s ?o ;"
```

s	o
file:///qa//City	file:///qa//Place
file:///qa//Person	file:///qa//Thing
file:///qa//Place	file:///qa//Thing
file:///qa//Thing	file:///qa//Thing

2. We try again by adding first, a triple that declares `rdfs:subClassOf` as a transitive property. It returns a error, because **the transitivity of a property is not supported in a query with a variable as subject**.

```
isql-vt 1111 dba dba exec="SPARQL
INSERT
  {
    GRAPH <urn:qa-test-onto> {
      <rdfs:subClassOf> a <owl:TransitiveProperty> .
    }
  };"

isql-vt 1111 dba dba exec="
rdfs_rule_set('urn:owl:inference:rules:qa-test', 'urn:qa-test-onto');"

isql-vt 1111 dba dba exec="SPARQL
DEFINE input:inference 'urn:owl:inference:rules:qa-test'
SELECT *
FROM <urn:qa-test-graph>
WHERE { ?s <rdfs:subClassOf> ?o }
ORDER BY ?s ?o ;"

Connected to OpenLink Virtuoso
Driver: 06.01.3127 OpenLink Virtuoso ODBC Driver
OpenLink Interactive SQL (Virtuoso), version 0.9849b.
Type HELP; for help and EXIT; to exit.
```

```
*** Error 37000: [Virtuoso Driver][Virtuoso Server]TR...:
transitive start not given
at line 0 of Top-Level:
SPARQL DEFINE input:inference 'urn:owl:inference:rules:qa-test'
 SELECT * FROM <urn:qa-test-graph> WHERE
 { ?s <rdfs:subClassOf> ?o } ORDER BY ?s ?o
```

3. We now ask for the superclasses of City (should be Place and Thing). This raises an out-of-memory error.

```
isql-vt 1111 dba dba exec="SPARQL
DEFINE input:inference 'urn:owl:inference:rules:qa-test'
SELECT DISTINCT *
FROM <urn:qa-test-graph>
WHERE { <file:///qa//City> <rdfs:subClassOf> ?o OPTION (TRANSITIVE)}
ORDER BY ?o ;"


Connected to OpenLink Virtuoso
Driver: 06.01.3127 OpenLink Virtuoso ODBC Driver
OpenLink Interactive SQL (Virtuoso), version 0.9849b.
Type HELP; for help and EXIT; to exit.

*** Error 42000: [Virtuoso Driver][Virtuoso Server]TN...:
Exceeded 100000000 bytes in transitive temp memory.

at line 0 of Top-Level:
SPARQL DEFINE input:inference 'urn:owl:inference:rules:qa-test'
SELECT * FROM <urn:qa-test-graph>
WHERE { <file:///qa//City> <rdfs:subClassOf> ?o } ORDER BY ?o
```

4. In fact, there are specific settings for transitivity, so we have to specify "distinct" near the transitive triples in the query (as illustrated below). However, the suject of a triple has to be defined, which is not always the case in our queries.

```
isql-vt 1111 dba dba exec="SPARQL
DEFINE input:inference 'urn:owl:inference:rules:qa-test'
SELECT DISTINCT *
FROM <urn:qa-test-graph>
WHERE { <file:///qa//City> <rdfs:subClassOf> ?o
OPTION (TRANSITIVE, t_distinct)}
ORDER BY ?o ;"
```

trans-subj-s_{30}	o
file:///qa//City	file:///qa//Place
file:///qa//City	file:///qa//Thing

Custom Inference Rules (SPIN)

Custom inference rules are available only in Virtuoso 8.0, which is the commercial version.

These rules are built using the SPIN (for SPARQL Inference Notation) language[2][3][4]. We define a SQL script[5] in order to test reasoning in Virtuoso. This script has three input variables.

- `$GRAPH` the RDF graph,
- `$RULES` the SPIN rules,
- `$QUERY` is the query to be answered on the graph.

Simple Case We will consider the following RDF graph:

```
<file:///qa//Erik> <file:///qa//livesIn> <file:///qa//London> .
<file:///qa//Erik> a <file:///qa//#Thing> .
```

The following SPIN rule says that the domain of livesIn is Person.

```
CONSTRUCT { ?this a <file:///qa//Person> }
WHERE {
{ ?this <file:///qa//livesIn> ?y . }
}
```

We query for entities of type Person and type Thing.

```
DEFINE input:macro-lib <urn:rules>
WITH <urn:graph>
SELECT ?s
WHERE { ?s a <file:///qa//#Thing> .
        ?s a <file:///qa//Person> . };
```

s
file:///qa//Erik

The SPIN rule has been defined on the class Thing, and it seems mandatory to specify the type of Thing in the query to activate the reasoning process ... So, **the answer Erik is missing of the following query**.

```
DEFINE input:macro-lib <urn:rules>
WITH <urn:graph>
SELECT *
WHERE { ?s a <file:///qa//Person> .};
```

[2] https://medium.com/virtuoso-blog/magic-sets-and-custom-inference-rules-in-virtuoso-8-x-db783f8d98d2

[3] https://medium.com/virtuoso-blog/virtuoso-8-0-creating-a-custom-inference-rules-using-spin-vocabulary-d7a060f859ef

[4] https://stackoverflow.com/questions/51513968/how-virtuoso-do-if-then-rules-inference

[5] https://gitlab.inria.fr/mburon/graph-layout-experiments/-/blob/8eacd5267af39b327e3bcb0b7da1a82360ebb452/README.org

s

Neither does the following query provide all expected answers:

```
DEFINE input:macro-lib <urn:rules>
WITH <urn:graph>
SELECT ?s ?c
WHERE { ?s a <file:///qa//#Thing> .
        ?s a ?c . };
```

s	c
file:///qa//Erik	file:///qa//#Thing

Trying to Reason using Domain Specification

We define Person as the domain of livesIn.

```
<file:///qa//Erik> <file:///qa//livesIn> <file:///qa//London> .
<file:///qa//Erik> a <file:///qa//#Thing> .
<file:///qa//livesIn> <rdfs:domain> <file:///qa//Person> .
```

We define the domain reasoning as defined by the rule **rdfs2**:

```
CONSTRUCT { ?this a ?c }
WHERE {
{ ?this ?p ?y .
  ?p  <rdfs:domain> ?c . }
}
```

We query for entities of type Person and of type Thing.

```
DEFINE input:macro-lib <urn:rules>
WITH <urn:graph>
SELECT ?s
WHERE { ?s a <file:///qa//#Thing> .
        ?s a <file:///qa//Person> . };
```

s
file:///qa//Erik

The SPIN rule has been defined on the class Thing, and it seems mandatory to specify the type of Thing in the query to activate the reasoning process ... This is illustrated by the following query, for which the answer Erik is missing.

```
DEFINE input:macro-lib <urn:rules>
WITH <urn:graph>
SELECT *
WHERE { ?s a <file:///qa//Person> .};
```

—

In this case, it is possible to generalize the query as:

```
DEFINE input:macro-lib <urn:rules>
WITH <urn:graph>
SELECT ?s ?c
WHERE { ?s a <file:///qa//#Thing> .
        ?s a ?c . };
```

s	c
file:///qa//Erik	file:///qa//Person
file:///qa//Erik	file:///qa//#Thing

Neither can we get the desired result by asking the following query (Erik should be of type Person):

```
DEFINE input:macro-lib <urn:rules>
WITH <urn:graph>
SELECT ?s ?p ?o
WHERE { ?s ?p ?o .};
```

s	p	o
file:///qa//Erik	rdf:type	file:///qa//#Thing
file:///qa//livesIn	rdfs:domain	file:///qa//Person
file:///qa//Erik	file:///qa//livesIn	file:///qa//London

Neither does the following query provide all expected answers (Erik should be known of type Person):

```
DEFINE input:macro-lib <urn:rules>
WITH <urn:graph>
SELECT ?s ?p ?o
WHERE { ?s a <file:///qa//#Thing> .
        ?s ?p ?o . };
```

s	p	o
file:///qa//Erik	rdf:type	file:///qa//#Thing
file:///qa//Erik	file:///qa//livesIn	file:///qa//London

A.3 Appendix of the query answering strategies in RIS experiments

This appendix is related to Section 4.5.

A.3.1 Experiments Queries

```
Q01<$label, $featureLabel, $value> :-
              triple($product, <rdfs:label>, $label),
              triple($product, <rdf:type>, <bsbm-int:ProductType110>),
              triple($product, <bsbm:productFeature>, $productFeature),
              triple($productFeature, <rdfs:label>, $featureLabel),
```

```
              triple($product, <bsbm:productPropertyNumeric>, $value);

Q01a<$label, $featureLabel, $value> :-
              triple($product, <rdfs:label>, $label),
              triple($product, <rdf:type>, <bsbm-int:ProductType8>),
              triple($product, <bsbm:productFeature>, $productFeature),
              triple($productFeature, <rdfs:label>, $featureLabel),
              triple($product, <bsbm:productPropertyNumeric>, $value);

Q01b<$label, $featureLabel, $value> :-
              triple($product, <rdfs:label>, $label),
              triple($product, <rdf:type>, <bsbm-int:ProductType4>),
              triple($product, <bsbm:productFeature>, $productFeature),
              triple($productFeature, <rdfs:label>, $featureLabel),
              triple($product, <bsbm:productPropertyNumeric>, $value);

Q02<$label, $value, $country> :-
              triple($product, <rdfs:label>, $label),
              triple($product, <rdf:type>, <bsbm-int:ProductType110>),
              triple($product, <bsbm:productPropertyNumeric>, $value),
              triple($product, <bsbm:producer>, $producer),
              triple($producer,<bsbm:country>, $country),
              triple($country, <rdf:type>, <bsbm:CountryType12>);

Q02a<$label, $value, $country> :-
              triple($product, <rdfs:label>, $label),
              triple($product, <rdf:type>, <bsbm-int:ProductType110>),
              triple($product, <bsbm:productPropertyNumeric>, $value),
              triple($product, <bsbm:producer>, $producer),
              triple($producer,<bsbm:country>, $country),
              triple($country, <rdf:type>, <bsbm:CountryType1>);

Q02b<$label, $value, $country> :-
              triple($product, <rdfs:label>, $label),
              triple($product, <rdf:type>, <bsbm-int:ProductType8>),
              triple($product, <bsbm:productPropertyNumeric>, $value),
              triple($product, <bsbm:producer>, $producer),
              triple($producer,<bsbm:country>, $country),
              triple($country, <rdf:type>, <bsbm:CountryType1>);

Q02c<$label, $value, $country> :-
              triple($product, <rdfs:label>, $label),
              triple($product, <rdf:type>, <bsbm-int:ProductType2>),
              triple($product, <bsbm:productPropertyNumeric>, $value),
              triple($product, <bsbm:producer>, $producer),
              triple($producer,<bsbm:country>, $country),
              triple($country, <rdf:type>, <bsbm:CountryType1>);

Q03<$product1, $product2> :-
```

```
            triple($product1, <rdf:type>, <bsbm-int:ProductType12>),
            triple($product1, <bsbm:productPropertyNumeric>, "774"),
            triple($product1, <bsbm:producer>, $producer),
            triple($product2, <rdf:type>, <bsbm-int:ProductType2>),
            triple($product2, <bsbm:producer>, $producer);

Q04<$vendor, $vendorLabel, $vendorHomepage> :-
              triple($vendor, <rdfs:label>, $vendorLabel),
              triple($vendor, <foaf:homepage>, $vendorHomepage);

Q07<$country> :-
             triple($org, <rdf:type>, <foaf:Organization>),
             triple($org, <bsbm:country>, $country),
             triple($country, <rdf:type>, <bsbm:CountryType222>);

Q07a<$country> :-
             triple($org, <rdf:type>, <foaf:Organization>),
             triple($org, <bsbm:country>, $country),
             triple($country, <rdf:type>, <bsbm:CountryType22>);

Q09<$product, $value> :-
               triple($product, <bsbm:productPropertyNumeric>, $value);

Q10<$country, $agentType> :-
             triple($agent, <rdf:type>, $agentType),
             triple($agent, <bsbm:country>, $country),
             triple($country, <rdf:type>, <bsbm:CountryType222>);

Q13<$offer, $prop, $value> :-
               triple($offer, <bsbm:product>, $product),
               triple($product, <rdf:type>, <bsbm-int:ProductType110>),
               triple($product, $prop, $value),
               triple($prop, <rdfs:domain>, <bsbm:Product>);

Q13a<$offer, $prop, $value> :-
               triple($offer, <bsbm:product>, $product),
               triple($product, <rdf:type>, <bsbm-int:ProductType8>),
               triple($product, $prop, $value),
               triple($prop, <rdfs:domain>, <bsbm:Product>);

Q13b<$offer, $prop, $value> :-
               triple($offer, <bsbm:product>, $product),
               triple($product, <rdf:type>, <bsbm-int:ProductType2>),
               triple($product, $prop, $value),
               triple($prop, <rdfs:domain>, <bsbm:Product>);

Q14<$product, $p, $offer, $vendor> :-
               triple($product, <bsbm:producer>, $p),
               triple($offer, <bsbm:product>, $product),
```

```
              triple($offer, <bsbm:vendor>, $vendor);


Q16<$review, $product, $title, $text> :-
             triple($review, <bsbm:reviewFor>, $product),
             triple($product, <rdf:type>, <bsbm-int:ProductType2>),
             triple($review, <http://purl.org/dc/elements/1.1/title>, $title),
             triple($review, <http://purl.org/stuff/rev#text>, $text);

Q19<$offer, $offerURL, $price, $deliveryDays, $value> :-
             triple($offer, <bsbm:offerWebpage>, $offerURL),
             triple($offer, <bsbm:price>, $price),
             triple($offer, <bsbm:deliveryDays>, $deliveryDays),
             triple($offer, <bsbm:product>, $product),
             triple($product, <rdf:type>, <bsbm-int:ProductType8>),
             triple($product, <bsbm:productPropertyNumeric>, $value),
             triple($product, <bsbm:producer>, $producer),
             triple($producer,<bsbm:country>, $country),
             triple($country, <rdf:type>, <bsbm:CountryType12>);

Q19a<$offer, $offerURL, $price, $deliveryDays, $value> :-
             triple($offer, <bsbm:offerWebpage>, $offerURL),
             triple($offer, <bsbm:price>, $price),
             triple($offer, <bsbm:deliveryDays>, $deliveryDays),
             triple($offer, <bsbm:product>, $product),
             triple($product, <rdf:type>, <bsbm-int:ProductType8>),
             triple($product, <bsbm:productPropertyNumeric>, $value),
             triple($product, <bsbm:producer>, $producer),
             triple($producer,<bsbm:country>, $country),
             triple($country, <rdf:type>, <bsbm:CountryType2>);

Q20<$offer, $offerURL, $price, $deliveryDays, $value> :-
             triple($offer, <bsbm:offerWebpage>, $offerURL),
             triple($offer, <bsbm:price>, $price),
             triple($offer, <bsbm:deliveryDays>, $deliveryDays),
             triple($offer, <bsbm:product>, $product),
             triple($product, <bsbm:productFeature>, $productFeature),
             triple($productFeature, <rdfs:label>, "biographies"),
             triple($product, <rdf:type>, <bsbm-int:ProductType110>),
             triple($product, <bsbm:productPropertyNumeric>, $value),
             triple($product, <bsbm:producer>, $producer),
             triple($producer,<bsbm:country>, $country),
             triple($country, <rdf:type>, <bsbm:CountryType12>);

Q20a<$offer, $offerURL, $price, $deliveryDays, $value> :-
             triple($offer, <bsbm:offerWebpage>, $offerURL),
             triple($offer, <bsbm:price>, $price),
             triple($offer, <bsbm:deliveryDays>, $deliveryDays),
             triple($offer, <bsbm:product>, $product),
```

```
            triple($product, <bsbm:productFeature>, $productFeature),
            triple($productFeature, <rdfs:label>, "biographies"),
            triple($product, <rdf:type>, <bsbm-int:ProductType8>),
            triple($product, <bsbm:productPropertyNumeric>, $value),
            triple($product, <bsbm:producer>, $producer),
            triple($producer,<bsbm:country>, $country),
            triple($country, <rdf:type>, <bsbm:CountryType12>);

Q20b<$offer, $offerURL, $price, $deliveryDays, $value> :-
            triple($offer, <bsbm:offerWebpage>, $offerURL),
            triple($offer, <bsbm:price>, $price),
            triple($offer, <bsbm:deliveryDays>, $deliveryDays),
            triple($offer, <bsbm:product>, $product),
            triple($product, <bsbm:productFeature>, $productFeature),
            triple($productFeature, <rdfs:label>, "biographies"),
            triple($product, <rdf:type>, <bsbm-int:ProductType2>),
            triple($product, <bsbm:productPropertyNumeric>, $value),
            triple($product, <bsbm:producer>, $producer),
            triple($producer,<bsbm:country>, $country),
            triple($country, <rdf:type>, <bsbm:CountryType12>);

Q20c<$offer, $offerURL, $price, $deliveryDays, $value> :-
            triple($offer, <bsbm:offerWebpage>, $offerURL),
            triple($offer, <bsbm:price>, $price),
            triple($offer, <bsbm:deliveryDays>, $deliveryDays),
            triple($offer, <bsbm:product>, $product),
            triple($product, <bsbm:productFeature>, $productFeature),
            triple($productFeature, <rdfs:label>, "biographies"),
            triple($product, <rdf:type>, <bsbm-int:ProductType2>),
            triple($product, <bsbm:productPropertyNumeric>, $value),
            triple($product, <bsbm:producer>, $producer),
            triple($producer,<bsbm:country>, $country),
            triple($country, <rdf:type>, <bsbm:CountryType1>);

Q21<$product, $type> :-
            triple($product, <bsbm:productFeature>, $productFeature),
            triple($productFeature, <rdfs:label>, "biographies"),
            triple($product, <rdf:type>, $type);

Q22<$product, $type> :-
            triple($product, <bsbm:productFeature>, $productFeature),
            triple($productFeature, <rdfs:label>, "biographies"),
            triple($product, <rdf:type>, $type),
            triple($type, <rdfs:subClassOf>, <bsbm-int:ProductType12>);

Q22a<$product, $type> :-
            triple($product, <bsbm:productFeature>, $productFeature),
            triple($productFeature, <rdfs:label>, "biographies"),
            triple($product, <rdf:type>, $type),
```

```
            triple($type, <rdfs:subClassOf>, <bsbm-int:ProductType2>);

Q23<$x, $class, $superProp, $value> :-
                triple($x, $prop, $value),
                triple($prop, <rdfs:subPropertyOf>, $superProp),
                triple($x, $p, $agent),
                triple($p, <rdfs:range>, <foaf:Agent>),
                triple($p, <rdfs:domain>, $class),
                triple($agent, <bsbm:country>, $country),
                triple($country, <rdf:type>, <bsbm:CountryType12>);
```

A.3.2 Experiments on REW

Our experiments on the strategy REW shows that it is slower that REW-C, because (i) REW has the same behaviour as REW-C at query time if the query does not query the ontology part of the graph and (ii) we have experimentally shown below that on 6 queries (querying the ontology), in the REW strategy, the size of the rewriting explodes in comparison to the REW-C strategy. We can see the size (number of elements in the union) of the rewriting in the column N_REW of the following tables and the corresponding rewriting time in ms in the T_REW column.

This explosion of the size of the rewriting leads to an explosion of the time spent for optimizing the rewriting. These optimization steps are called CLASH, COVER and CORE: they respectively remove CQs with empty answer sets according the functions in the mappings, minimize the output union by removing redundant CQs and compute the core of each rewriting CQ (removing redundancy inside CQs). The column prefixed by N_ contains the size of the rewriting after the optimization and the column prefixed by T_ contains the corresponding time. The last column T_OP contains the time spent for the optimization of the execution plan in Tatooine; some ongoing work may modify this time, but not the previous ones.

You can see the number of reformulations and respectively the reformulation processing time in the column N_REF, resp. T_REF. The number of triples in the query is written in the column N_TRI.

Experiments on a small scenario

In Table A.1 and Table A.2, we observe that the size of the rewriting is larger with a multiplicative factor going from 29 to 74 in the small scenario. The first table concerns REW and the second REW-C.

INPUT	N_{TRI}	N_{REF}	$\mathbf{N_{REW}}$	N_{CLASH}	N_{COVER}	T_{REF}	T_{REW}	T_{CLASH}	T_{COVER}	T_{CORE}	T_{OP}
Q13	4	nan	**1005**	576	304	nan	106	1	119	247	175
Q13a	4	nan	**2010**	1152	606	nan	123	2	371	487	441
Q13b	4	nan	**16080**	9216	4400	nan	435	24	17531	4982	11909
Q22	4	nan	**929**	539	385	nan	70	0	76	54	126
Q22a	4	nan	**929**	539	385	nan	69	1	77	55	123
Q23	7	nan	**5922**	4474	856	nan	555	9	978	268	707

Table A.1: Statistics of REW on S_1 and S_3.

INPUT	N_{TRI}	N_{REF}	$\mathbf{N_{REW}}$	N_{CLASH}	N_{COVER}	T_{REF}	T_{REW}	T_{CLASH}	T_{COVER}	T_{CORE}	T_{OP}
Q13	4	16	**25**	nan	21	1	184	1	14	44	19
Q13a	4	16	**50**	nan	42	1	185	0	31	84	41
Q13b	4	16	**400**	nan	336	1	223	0	268	1166	354
Q22	4	2	**2**	nan	2	1	10	0	0	0	1
Q22a	4	24	**32**	nan	32	0	73	0	1	4	7
Q23	7	64	**80**	56.0	24	738	1650	0	10	39	16

Table A.2: Statistics of REW-C on S_1 and S_3.

Experiment on large scenario

In Table A.3 and Table A.4, we observe that the size of the rewriting is larger with a multiplicative factor going from 33 to 969 in the large scenario. The first table concerns REW and the second REW-C.

INPUT	N_{TRI}	N_{REF}	$\mathbf{N_{REW}}$	N_{CLASH}	N_{COVER}	T_{REF}	T_{REW}	T_{CLASH}	T_{COVER}	T_{CORE}	T_{OP}
Q13	4	nan	**25886**	12992	10018	nan	1660	44	98580	6960	72717
Q13a	4	nan				nan	**≥10min**				
Q13b	4	nan				nan	**≥10min**				
Q22	4	nan	**12865**	6459	6401	nan	1000	18	19320	859	19142
Q22a	4	nan	**12865**	6459	6401	nan	995	18	19646	855	19138
Q23	7	nan	**77538**	56954	12888	nan	7426	130	353692	2477	163584

Table A.3: Statistics of REW on S_2 and S_4.

INPUT	N_{TRI}	N_{REF}	$\mathbf{N_{REW}}$	N_{CLASH}	N_{COVER}	T_{REF}	T_{REW}	T_{CLASH}	T_{COVER}	T_{CORE}	T_{OP}
Q13	4	16	**50**	nan	42	4	3198	1	29	89	30
Q13a	4	16	**3200**	nan	2688	3	3504	9	5396	4922	6366
Q13b	4	16	**3200**	nan	2688	3	3177	3	5385	5366	5343
Q22	4	24	**32**	nan	32	1	1878	0	0	4	5
Q22a	4	200	**384**	nan	384	3	12417	3	100	47	110
Q23	7	64	**80**	56.0	24	642	21428	1	7	32	24

Table A.4: Statistics of REW-C on S_2 and S_4.

A.4 Proofs about restricted rules

This section contains the proofs of the results in Section 4.6.

In the following, without loss of generality, we will assume that the mapping functions are always projections on one of their inputs, such that the mappings have the following form: $q_1(\bar{x}) \rightsquigarrow q_2(\bar{x})$.

Proof of Property 4.1

Proof. If we assume that r is an instance rule, then $\text{body}(r) = \{t_r\} \cup \text{body}_O(r)$, and the triple t_r has to be of the form $(x, \mathtt{p}, y)$ with $\mathtt{p} \in \text{Var}(\text{body}_O(r))$, otherwise $\varphi(t_r)$ could not be an RDFS triple (which would contradict the fact that $O \models^{\varphi} \text{body}(r)$). Then $\varphi(\mathtt{p})$ occurs as a subject or an object of a triple in O, because $\text{body}_O(r)$ does not have any variable as a property. However, $\varphi(\mathtt{p})$ cannot be an RDFS IRI, because O is an FO ontology. This

contradicts the fact that $\varphi(\mathtt{p})$ is an RDFS IRI since $\varphi(t_r) \in O$. We conclude that r is necessarily an ontological rule. □

Proof of Property 4.2

Proof. From Property 4.1, only ontological rules can be applied on an FO-ontology. We check that the produced RDFS triples comply with the conditions of an FO-ontology. □

Proof of Property 4.3

Proof. From the definition of the restricted rules, we know that if the direct entailment of G by $\{r\}$ contains an RDFS triple, then r is either an ontological rule, or an instance rule with head containing a triple $t = (\mathtt{s}, \mathtt{p}, \mathtt{o})$, case 2c. The latter case is not possible, because necessarily $\mathtt{p} \in \mathrm{Var}(\mathrm{body}_{\mathrm{O}}(\mathrm{r}))$, i.e., $\mathtt{p}$ also occurs as the subject or the object of an RDFS triple t_b in the body of r. The application of the rule r maps t_b to an RDFS triple of G. Since the set of RDFS triples of G is an FO ontology, $\mathtt{p}$ cannot be mapped to an RDFS IRI, which is absurd. Therefore, r is an ontological rule. □

Proof of Property 4.4

Proof. First, we prove that $O^{\mathcal{R}} \subseteq \mathrm{RDFS}(G^{\mathcal{R}})$. Using Property 4.2, we know that $O^{\mathcal{R}}$ is an FO ontology, so at least $O^{\mathcal{R}}$ is a set of RDFS triples. Since $O \subseteq G$, we have $O^{\mathcal{R}} \subseteq \mathrm{RDFS}(G^{\mathcal{R}})$.

Second, we prove that $\mathrm{RDFS}(G^{\mathcal{R}}) \subseteq O^{\mathcal{R}}$. Let t be a triple in $\mathrm{RDFS}(G^{\mathcal{R}})$, so either t is an RDFS triple of G (and then $t \in O^{\mathcal{R}}$) or t is a RDFS triple in entailment of $\mathcal{R}$ on G. In the latter case, we prove that $t \in O^{\mathcal{R}}$ by induction on $(G_i^{\mathcal{R}})_{i\in\mathbb{N}}$ the saturation sequence of $G^{\mathcal{R}}$.

By Property 4.3, if a restricted rules $r \in \mathcal{R}$ applied on G directly derives at least an RDFS triples, r is an ontological rule. And since, the body of an ontological rule is composed of RDFS triples, a such rule r can be apply only on RDFS triples. Moreover, $\mathrm{RDFS}(G) = O$, so each RDFS triple t in $C_{G,r}$ the direct entailment of r on G is actually in $C_{O,r}$. Finally, we can remark that $\mathrm{RDFS}(G_1^{\mathcal{R}}) \subseteq O^{\mathcal{R}}$.

For the initialization step of the induction, if t a RDFS triple directly entails by G, the preceding remark proves that $t \in O^{\mathcal{R}}$. And the induction is assured by Property 4.2 which shows that $O^{\mathcal{R}}$ is an FO ontology, so $\mathrm{RDFS}(G_1^{\mathcal{R}})$ is an FO ontology. We can start the preceding reasoning again replacing G by $G_1^{\mathcal{R}}$ and O by $\mathrm{RDFS}(G_1^{\mathcal{R}})$. □

Proof of Property 4.5

Proof. We define $\mathrm{body}(r)$ as $t_r \wedge \mathrm{body}_O(r)$. Since $\mathrm{body}_O(r)$ is a set of RDFS triples and t (thus also $v(t)$) is not an RDFS triple, then $\varphi(\mathrm{body}_O(r)) \subseteq O$. We now consider the two possible forms of t_r.

Case *(i)*: t_r has the form of $(x, \mathtt{p}, y)$, Restriction 2(b)i. As in the proof of Property 4.1, we know that $\varphi(\mathtt{p})$ is not an RDFS IRI and with the same reasoning, we can show that $\varphi(\mathtt{p}) \neq \tau$. So, the triple $\varphi((x, \mathtt{p}, y))$ is equal to $v(t)$ and the data mapping triple $t = (x', \mathtt{p}', y')$ with $\mathtt{p}' \in \mathscr{I} \setminus \{\tau\}$. Since we know that x and y are not in $\mathrm{Var}(\mathrm{body}_{\mathrm{O}}(\mathrm{r}))$, we know that $\varphi_{|\mathrm{Var}(\mathrm{body}_{\mathrm{O}}(\mathrm{r}))}(t_r) = (x, \mathtt{p}', y)$. So if we choose $\varphi' = \varphi_{|\mathrm{Var}(\mathrm{body}_{\mathrm{O}}(\mathrm{r}))} \cup \{x \mapsto x', y \mapsto y'\}$, which is indeed a homomorphism because $x \neq y$, then we have $\{t\} \cup O \models^{\varphi'} \mathrm{body}(r)$ and

$\varphi(\text{body}(r)) = v(\varphi'(\text{body}(r)))$.
Case *(ii)*: t_r has the form (x, τ, z), with $x \notin \text{Var}(\text{body}_\text{O}(\text{r}))$ and $z \in \mathscr{I} \cup \text{Var}(\text{body}_\text{O}(\text{r}))$, Restriction 2(b)ii. Since τ is not an RDFS property, we know that $\varphi(t_r) = v(t)$. So $v(t)$ is equal to (a, τ, C) with $a \in \mathscr{B} \cup \mathscr{I}$ and $C \in \mathscr{I}$, and there exists $y \in \mathscr{B} \cup \mathscr{I}$ such that $t = (y, \tau, C)$. We have $\varphi_{|\text{Var}(\text{body}_\text{O}(\text{r}))}(t_r) = (x, \tau, C)$, so if $y \in \mathscr{B}$, then $\varphi' = \varphi_{|\text{Var}(\text{body}_\text{O}(\text{r}))} \cup \{x \mapsto y\}$ satisfies the wanted property. Otherwise, $y \in \mathscr{I}$ and $\varphi' = \varphi_{|\text{Var}(\text{body}_\text{O}(\text{r}))}$ satisfies the wanted property as well, because v is then the identity.

□

Proof of Property 4.6

Proof. Let $\text{body}(r) = \{t_r\} \cup \text{body}_O(r)$. Since $\text{body}_O(r)$ is a set of RDFS triples and t is not an RDFS triple, we have $\varphi'(\text{body}_O(r)) \subseteq O$.

The result is just a consequence of the form of head(r). Let $u = (\mathsf{s}, \mathsf{p}, \mathsf{o})$ be a triple in head(r), we check that in each case $\varphi'(u)^{\text{safe}}$ can be a triple of the head of an data mapping:

- if $\mathsf{p} = \tau$ then $\mathsf{o} \in \mathscr{I} \cup \text{Var}(\text{body}_\text{O}(\text{r}))$. So $\varphi'(\mathsf{o})$ is always an IRI, since $\varphi'(\text{body}_O(r)) \subseteq O$ and O is an FO ontology;
- if $\mathsf{p} \in \mathscr{I} \setminus \{\prec_{sc}, \prec_{sp}, \hookleftarrow_d, \hookrightarrow_r\}$, nothing more is required;
- if $\mathsf{p} \in \text{Var}(\text{body}_\text{O}(\text{r}))$, then $\varphi'(\mathsf{p}) \in \mathscr{I} \setminus \{\prec_{sc}, \prec_{sp}, \hookleftarrow_d, \hookrightarrow_r\}$ and it is OK. Again since $\varphi'(\text{body}_O(r)) \subseteq O$ and O is an FO ontology.

□

Proof of Property 4.7

Proof. By Property 4.2, $O^{\mathcal{R}}$ is an FO ontology, hence contains only RDFS triples. The proof then directly follows from the equalities:

$$
\begin{aligned}
O^{\mathcal{R}} &= \{(\mathsf{s}, \prec_{sc}, \mathsf{o}) \mid (\mathsf{s}, \prec_{sc}, \mathsf{o}) \in O^{\mathcal{R}}\} \\
&\quad \cup \{(\mathsf{s}, \prec_{sp}, \mathsf{o}) \mid (\mathsf{s}, \prec_{sp}, \mathsf{o}) \in O^{\mathcal{R}}\} \\
&\quad \cup \{(\mathsf{s}, \hookleftarrow_d, \mathsf{o}) \mid (\mathsf{s}, \hookleftarrow_d, \mathsf{o}) \in O^{\mathcal{R}}\} \\
&\quad \cup \{(\mathsf{s}, \hookrightarrow_r, \mathsf{o}) \mid (\mathsf{s}, \hookrightarrow_r, \mathsf{o}) \in O^{\mathcal{R}}\} \\
&= \{(s, \prec_{sc}, o)^{safe} \mid m_{\text{subClassOf}} = q_{\text{subClassOf}}(\mathsf{s}, \mathsf{o}) \rightsquigarrow \\
&\quad (\mathsf{s}, \prec_{sc}, \mathsf{o}), (s, o) \in \text{ext}(m_{\text{subClassOf}}) = q_{\text{subClassOf}}(O, \mathcal{R})\} \\
&\quad \cup \{(s, \prec_{sp}, o)^{safe} \mid m_{\text{subPropertyOf}} = q_{\text{subPropertyOf}}(\mathsf{s}, \mathsf{o}) \rightsquigarrow \\
&\quad (\mathsf{s}, \prec_{sp}, \mathsf{o}), (s, o) \in \text{ext}(m_{\text{subPropertyOf}}) = q_{\text{subPropertyOf}}(O, \mathcal{R})\} \\
&\quad \cup \{(s, \hookleftarrow_d, o)^{safe} \mid m_{\text{domain}} = q_{\text{domain}}(\mathsf{s}, \mathsf{o}) \rightsquigarrow \\
&\quad (\mathsf{s}, \hookleftarrow_d, \mathsf{o}), (s, o) \in \text{ext}(m_{\text{domain}}) = q_{\text{domain}}(O, \mathcal{R})\} \\
&\quad \cup \{(s, \hookrightarrow_r, o)^{safe} \mid m_{\text{range}} = q_{\text{range}}(\mathsf{s}, \mathsf{o}) \rightsquigarrow \\
&\quad (\mathsf{s}, \hookrightarrow_r, \mathsf{o}), (s, o) \in \text{ext}(m_{\text{range}}) = q_{\text{range}}(O, \mathcal{R})\} \\
&= G^{\mathcal{M}_{O^{\mathcal{R}}}}_{\mathcal{E}_{O^{\mathcal{R}}}}
\end{aligned}
$$

□

Corollary A.1

Corollary A.1. *The set of RDFS triples of* $G^{\mathcal{M}\cup\mathcal{M}_{O^\mathcal{R}}}_{\mathcal{E}\cup\mathcal{E}_{O^\mathcal{R}}}$ *is exactly* $O^\mathcal{R}$.

Proof. Since no data mapping of $\mathcal{M}$ has RDFS triples in its head, the ontology of $G^{\mathcal{M}\cup\mathcal{M}_{O^\mathcal{R}}}_{\mathcal{E}\cup\mathcal{E}_{O^\mathcal{R}}}$ is included in $G^{\mathcal{M}_{O^\mathcal{R}}}_{\mathcal{E}_{O^\mathcal{R}}}$. Moreover, $G^{\mathcal{M}_{O^\mathcal{R}}}_{\mathcal{E}_{O^\mathcal{R}}}$ contains only RDFS triples, hence the wanted equality holds. □

Proof of Property 4.8

Proof. By Corollary A.1, the set of RDFS triples of $G^{\mathcal{M}\cup\mathcal{M}_{O^\mathcal{R}}}_{\mathcal{E}\cup\mathcal{E}_{O^\mathcal{R}}}$ is $O^\mathcal{R}$. By Property 4.2, $O^\mathcal{R}$ is an FO ontology, so by Property 4.4, the set of RDFS triples of $(G^{\mathcal{M}\cup\mathcal{M}_{O^\mathcal{R}}}_{\mathcal{E}\cup\mathcal{E}_{O^\mathcal{R}}})^\mathcal{R}$ is $O^\mathcal{R}$.
□

Proof of Theorem 4.5

To prove the theorem, we will rely on the next definition and some auxilliary lemmas.

Definition A.1. *We define the following sequence of mappings:*

$$(\mathcal{M})^{\mathcal{R},O}_i = \left\{q_1 \rightsquigarrow (q_2)^{\mathcal{R},O}_i \mid q_1 \rightsquigarrow q_2 \in \mathcal{M}\right\}$$

where $body((q)^{\mathcal{R},O}_i) = max\{S \subseteq (body(q) \cup O)^\mathcal{R}_i \mid \forall T \subseteq S, O \models_\mathcal{R} T \Rightarrow body(q) \models_\mathcal{R} T\}$.

Intuitively, $(q)^{\mathcal{R},O}_i$ is the saturation of (body$(q)\cup O$) at rank i from which triples entailed solely by O are removed, as in Def. 4.14.

Lemma A.2. *Let be q the head of a data mapping, O an FO ontology and* $\mathcal{R}$ *a set of restricted rules, we have:*

$$\forall i \in \mathbb{N},\ body((q)^{\mathcal{R},O}_i) = (body(q) \cup O)^\mathcal{R}_i \setminus \text{RDFS}((body(q) \cup O)^\mathcal{R}_i).$$

Proof. First, let i be a positive integer, we prove that body$((q)^{\mathcal{R},O}_i) \subseteq$ (body$(q) \cup O)^\mathcal{R}_i \setminus$ RDFS((body$(q) \cup O)^\mathcal{R}_i$). Let t be a triple in body$((q)^{\mathcal{R},O}_i)$, so $t \in$ (body$(q) \cup O)^\mathcal{R}_i$. Hence by definition of $(q)^{\mathcal{R},O}_i$, we have either $O \not\models_\mathcal{R} t$ or body$(q) \models_\mathcal{R} t$. We will tackle the both cases separately. We recall that since q is the head of a data mapping, body(q) only contains none RDFS triples. Moreover using Property 4.6, we know that $q^\mathcal{R}$ body only contains none RDFS triples. So if body$(q) \models_\mathcal{R} t$, then $t \in$ (body$(q) \cup O)^\mathcal{R}_i \setminus$ RDFS((body$(q) \cup O)^\mathcal{R}_i$). Inspiring by the proof of Property 4.4, we can prove the following property: for all $u \in$ (body$(q) \cup O)^\mathcal{R}_i$, if $u \in$ RDFS((body$(q) \cup O)^\mathcal{R}_i$) then $O \models_\mathcal{R} u$. Using the contraposition of this property, we know that if $O \not\models_\mathcal{R} t$ then $t \in$ (body$(q) \cup O)^\mathcal{R}_i \setminus$ RDFS((body$(q) \cup O)^\mathcal{R}_i$). So in the both case, $t \in$ (body$(q) \cup O)^\mathcal{R}_i \setminus$ RDFS((body$(q) \cup O)^\mathcal{R}_i$).

Secondly, let i be a positive integer, we prove that (body$(q) \cup O)^\mathcal{R}_i \setminus$ RDFS((body$(q) \cup O)^\mathcal{R}_i$) $\subseteq$ body$((q)^{\mathcal{R},O}_i)$. Let $t \in$ (body$(q) \cup O)^\mathcal{R}_i \setminus$ RDFS((body$(q) \cup O)^\mathcal{R}_i$) and $S \subset$ (body$(q) \cup O)^\mathcal{R}_i$, which satisfies the property $P(S) = \forall T \subseteq S,\ O \models_\mathcal{R} T \Rightarrow$ body$(q) \models_\mathcal{R} T$. We will prove that if $t \notin S$, S is not maximal for the property P, i.e., we will prove $P(S \cup \{t\})$. Let $T \subseteq S \cup \{t\}$, if $t \in T$ then $O \not\models_\mathcal{R} T$, because t is not an RDFS triples and Property 4.2, otherwise $T \subseteq S$. In both cases, $O \models_\mathcal{R} T \Rightarrow$ body$(q) \models_\mathcal{R} T$ holds, so $P(S \cup \{t\})$ also. Finally, $t \in$ body$((q)^{\mathcal{R},O}_i)$. □

We notice that even if this sequence of mappings is not increasing, it induces a **increasing** sequence of RDF graphs $\left(G_{\mathcal{E}}^{(\mathcal{M})_i^{\mathcal{R},O}}\right)_{i\in\mathbb{N}}$.

Lemma A.3. *For any $i \in \mathbb{N}$:*

- $(G_{\mathcal{E}\cup\mathcal{E}_{O^\mathcal{R}}}^{\mathcal{M}\cup\mathcal{M}_{O^\mathcal{R}}})_i^\mathcal{R} \subseteq G_{\mathcal{E}\cup\mathcal{E}_O}^{(\mathcal{M})_i^{\mathcal{R},O}\cup\mathcal{M}_{O^\mathcal{R}}}$
- $(\mathcal{M})_i^{\mathcal{R},O}$ *only contains data mappings*

Proof. We start by proving that the set of RDFS triples of $\left(G_{\mathcal{E}\cup\mathcal{E}_{O^\mathcal{R}}}^{\mathcal{M}\cup\mathcal{M}_{O^\mathcal{R}}}\right)_i^\mathcal{R}$ is a subset of $G_{\mathcal{E}\cup\mathcal{E}_O}^{(\mathcal{M})_i^{\mathcal{R},O}\cup\mathcal{M}_{O^\mathcal{R}}}$, for each $i \in \mathbb{N}$. Using preceding results, we have the following equations for $i \in \mathbb{N}$:

$$\begin{aligned} O^\mathcal{R} &= \text{RDFS}(G_{\mathcal{E}\cup\mathcal{E}_{O^\mathcal{R}}}^{\mathcal{M}\cup\mathcal{M}_{O^\mathcal{R}}}) \quad \text{(Corollary A.1)} \\ &\subseteq \text{RDFS}(\left(G_{\mathcal{E}\cup\mathcal{E}_{O^\mathcal{R}}}^{\mathcal{M}\cup\mathcal{M}_{O^\mathcal{R}}}\right)_i^\mathcal{R}) \\ &\subseteq \text{RDFS}(\left(G_{\mathcal{E}\cup\mathcal{E}_{O^\mathcal{R}}}^{\mathcal{M}\cup\mathcal{M}_{O^\mathcal{R}}}\right)^\mathcal{R}) \\ &= O^\mathcal{R} \quad \text{(Property 4.8)} \end{aligned}$$

We deduce that:

$$\forall i \in \mathbb{N},\ \text{RDFS}(\left(G_{\mathcal{E}\cup\mathcal{E}_{O^\mathcal{R}}}^{\mathcal{M}\cup\mathcal{M}_{O^\mathcal{R}}}\right)_i^\mathcal{R}) = O^\mathcal{R}$$

Moreover, using Property 4.7, we know that:

$$\forall i \in \mathbb{N},\ O^\mathcal{R} \subseteq G_{\mathcal{E}\cup\mathcal{E}_{O^\mathcal{R}}}^{\mathcal{M}\cup\mathcal{M}_{O^\mathcal{R}}} \subseteq G_{\mathcal{E}\cup\mathcal{E}_O}^{(\mathcal{M})_i^{\mathcal{R},O}\cup\mathcal{M}_{O^\mathcal{R}}}.$$

So finally, we prove that:

$$\forall i \in \mathbb{N},\ \text{RDFS}(\left(G_{\mathcal{E}\cup\mathcal{E}_{O^\mathcal{R}}}^{\mathcal{M}\cup\mathcal{M}_{O^\mathcal{R}}}\right)_i^\mathcal{R}) \subseteq G_{\mathcal{E}\cup\mathcal{E}_O}^{(\mathcal{M})_i^{\mathcal{R},O}\cup\mathcal{M}_{O^\mathcal{R}}}.$$

After that, we just have to prove for $i \in \mathbb{N}$ the following statement $P(i)$:

- each non-RDFS triple of $\left(G_{\mathcal{E}\cup\mathcal{E}_{O^\mathcal{R}}}^{\mathcal{M}\cup\mathcal{M}_{O^\mathcal{R}}}\right)_i^\mathcal{R}$ is in $G_{\mathcal{E}\cup\mathcal{E}_O}^{(\mathcal{M})_i^{\mathcal{R},O}\cup\mathcal{M}_{O^\mathcal{R}}}$,
- Let q be an head of a data mappings of $\mathcal{M}$, $\text{NR}_{q,i} = (\text{body}(q)\cup O)_i^\mathcal{R} \backslash \text{RDFS}((\text{body}(q)\cup O)_i^\mathcal{R})$ contains only correct triple for data mapping head. So $(\mathcal{M})_i^{\mathcal{R},O}$ only contains data mappings.

Here, we have to explain why in the second point of this list, the first sentences implies the second. It comes from the fact that if q is an head of a mapping in $(\mathcal{M})_i^{\mathcal{R},O}$, then $\text{body}(q) \subset (\text{body}(q) \cup O)_i^\mathcal{R} \setminus \text{RDFS}((\text{body}(q) \cup O)_i^\mathcal{R})$, according to Lemma A.2.

We will prove $P(i)$ by induction. In the base case, we show that the statement holds for $i = 0$:

- $\left(G_{\mathcal{E}\cup\mathcal{E}_{O^\mathcal{R}}}^{\mathcal{M}\cup\mathcal{M}_{O^\mathcal{R}}}\right)_0^\mathcal{R} = G_{\mathcal{E}\cup\mathcal{E}_{O^\mathcal{R}}}^{\mathcal{M}\cup\mathcal{M}_{O^\mathcal{R}}} = G_{\mathcal{E}\cup\mathcal{E}_O}^{(\mathcal{M})_0^{\mathcal{R},O}\cup\mathcal{M}_{O^\mathcal{R}}}$,

- For q a head of mapping in $\mathcal{M}$, $(\text{body}(q)\cup O)_0^{\mathcal{R}} \setminus \text{RDFS}((\text{body}(q)\cup O)_0^{\mathcal{R}}) = (\text{body}(q)\cup O)\setminus\text{RDFS}(\text{body}(q)\cup O) = (\text{body}(q)\cup O)\setminus O = \text{body}(q)$. So like previously explained we have: $(\mathcal{M})_0^{\mathcal{R},O} = \mathcal{M}$ only contains data mappings.

In the inductive step, we assume that $P(i)$ holds for $i \in \mathbb{N}$, we will prove that $P(i+1)$ also holds. If t' is a non-RDFS triple of $\left(G_{\mathcal{E}\cup\mathcal{E}_{O^{\mathcal{R}}}}^{\mathcal{M}\cup\mathcal{M}_{O^{\mathcal{R}}}}\right)_{i+1}^{\mathcal{R}}$, then there are two cases:

- $t' \in \left(G_{\mathcal{E}\cup\mathcal{E}_{O^{\mathcal{R}}}}^{\mathcal{M}\cup\mathcal{M}_{O^{\mathcal{R}}}}\right)_{i}^{\mathcal{R}}$ so by hypothesis $t' \in G_{\mathcal{E}\cup\mathcal{E}_O}^{(\mathcal{M})_i^{\mathcal{R},O}\cup\mathcal{M}_{O^{\mathcal{R}}}}$,
- or there exists a restricted rule $r \in \mathcal{R}$ such that $\left(G_{\mathcal{E}\cup\mathcal{E}_{O^{\mathcal{R}}}}^{\mathcal{M}\cup\mathcal{M}_{O^{\mathcal{R}}}}\right)_{i}^{\mathcal{R}} \models^{\varphi} \text{body}(r)$ and $t' \in \varphi(\text{head}(r))^{\text{safe}}$.

Since $t' \in \varphi(\text{head}(r))^{\text{safe}}$ is a non-RDFS triple, r is an instance rule. So there exists an $t \in \left(G_{\mathcal{E}\cup\mathcal{E}_{O^{\mathcal{R}}}}^{\mathcal{M}\cup\mathcal{M}_{O^{\mathcal{R}}}}\right)_{i}^{\mathcal{R}}$ such that $\{t\} \cup O^{\mathcal{R}} \models^{\varphi} \text{body}(r)$. By the inductive hypothesis, t is a triple of $G_{\mathcal{E}\cup\mathcal{E}_O}^{(\mathcal{M})_i^{\mathcal{R},O}\cup\mathcal{M}_{O^{\mathcal{R}}}}$. Hence there exists a mapping $m \in \mathcal{M}$ with $m = q_1 \rightsquigarrow q_2$, and a triple $t_m \in \text{body}((q_2)_i^{\mathcal{R},O})$ (defined in Definition A.1) and a tuple $e \in ext(m)$ such that $v_e(t_m) = t$, where v_e is the homomorphism induced by the replacement of answer variables of $(q_2)_i^{\mathcal{R},O}$ by the tuple e. Since $(q_2)_i^{\mathcal{R},O}$ is the head of a mapping of $(\mathcal{M})_i^{\mathcal{R},O}$, we know by induction hypothesis this mapping is actually a data mapping. So according to Property 4.5, there exists a homomorphism φ' such that $t_m \cup O^{\mathcal{R}} \models^{\varphi'} \text{body}(r)$ and $\varphi(\text{body}(r)) = v_e(\varphi'(\text{body}(r)))$. Hence, $\varphi(\text{head}(r))^{\text{safe}} = v_e(\varphi'(\text{head}(r))^{\text{safe}})$. We show that the mapping $q_1 \rightsquigarrow (q_2)_{i+1}^{\mathcal{R},O} \in (\mathcal{M})_{i+1}^{\mathcal{R},O}$ is such that $\varphi'(\text{head}(r)))^{\text{safe}} \subseteq \text{body}((q_2)_{i+1}^{\mathcal{R},O})$. Indeed, it is a consequence of Lemma A.2, because we know that $\varphi'(\text{head}(r)))^{\text{safe}} \subseteq (\text{body}(q_2) \cup O)_{i+1}^{\mathcal{R}}$ and $\varphi'(\text{head}(r)))^{\text{safe}}$ contains only no RDFS triple. Finally, we have proved:

$$\begin{aligned} t' &\in \varphi(\text{head}(r))^{\text{safe}} \\ &= v_e(\varphi'(\text{head}(r))^{\text{safe}}) \\ &\subseteq v_e(\text{body}((q_2)_{i+1}^{\mathcal{R},O})) \\ &\subseteq G_{\mathcal{E}\cup\mathcal{E}_O}^{(\mathcal{M})_{i+1}^{\mathcal{R},O}\cup\mathcal{M}_{O^{\mathcal{R}}}}. \end{aligned}$$

We also have to prove that for q an head of a mapping in $\mathcal{M}$, $\text{NR}_{q,i+1} = (\text{body}(q) \cup O)_{i+1}^{\mathcal{R}} \setminus \text{RDFS}((\text{body}(q) \cup O)_{i+1}^{\mathcal{R}})$ contains only valid triples for data mapping head. By induction hypothesis, we know that $\mathcal{M}, \text{NR}_{q,i}$ verify the willing property. Let t' a triple of $\text{NR}_{q,i+1}$, so there exists $r \in \mathcal{R}$ such that t' is one directly entailed triple by r on $(\text{body}(q) \cup O)_i^{\mathcal{R}}$. If the restricted rule r is an ontological rule, then t' is an RDFS triple. This case is absurd, because t' will be in $\text{RDFS}((\text{body}(q) \cup O)_{i+1}^{\mathcal{R}})$ so $t' \notin \text{NR}_{q,i+1}$. So r is an instance rule and there exists an $t \in \text{NR}_{q,i}$ such that $\{t\} \cup O^{\mathcal{R}} \models \text{body}(r)$. Using Property 4.6, we know that t is a valid triples for data mapping head. Finally we can deduce from it that $(\mathcal{M})_{i+1}^{\mathcal{R},O}$ is a set of data mappings. □

We are now able to prove the Theorem 4.5.

Proof of Theorem 4.5. Instead of proving $\left(G_{\mathcal{E}}^{\mathcal{M}} \cup O\right)^{\mathcal{R}} = G_{\mathcal{E}\cup\mathcal{E}_{O^{\mathcal{R}}}}^{\mathcal{M}^{\mathcal{R},O}\cup\mathcal{M}_{O^{\mathcal{R}}}}$, we will prove that $\left(G_{\mathcal{E}\cup\mathcal{E}_{O^{\mathcal{R}}}}^{\mathcal{M}\cup\mathcal{M}_{O^{\mathcal{R}}}}\right)^{\mathcal{R}} = G_{\mathcal{E}\cup\mathcal{E}_{O^{\mathcal{R}}}}^{\mathcal{M}^{\mathcal{R},O}\cup\mathcal{M}_{O^{\mathcal{R}}}}$, which is equivalent, since:

$$\left(G_{\mathcal{E}\cup\mathcal{E}_{O^{\mathcal{R}}}}^{\mathcal{M}\cup\mathcal{M}_{O^{\mathcal{R}}}}\right)^{\mathcal{R}} = \left(G_{\mathcal{E}}^{\mathcal{M}} \cup O^{\mathcal{R}}\right)^{\mathcal{R}} = \left(G_{\mathcal{E}}^{\mathcal{M}} \cup O\right)^{\mathcal{R}}.$$

First, we prove the inclusion $G_{\mathcal{E}\cup\mathcal{E}_{O^{\mathcal{R}}}}^{\mathcal{M}^{\mathcal{R},O}\cup\mathcal{M}_{O^{\mathcal{R}}}} \subseteq \left(G_{\mathcal{E}\cup\mathcal{E}_{O^{\mathcal{R}}}}^{\mathcal{M}\cup\mathcal{M}_{O^{\mathcal{R}}}}\right)^{\mathcal{R}}$. Let $(\mathsf{s},\mathsf{p},\mathsf{o})$ be a triple from $G_{\mathcal{E}\cup\mathcal{E}_{O^{\mathcal{R}}}}^{\mathcal{M}^{\mathcal{R},O}\cup\mathcal{M}_{O^{\mathcal{R}}}}$. Then, there exists a mapping $m \in \mathcal{M}\cup\mathcal{M}_{O^{\mathcal{R}}}$ such as $m = q_1(\bar{x}) \rightsquigarrow q_2(\bar{x})$ and there exists $\bar{t} \in \text{ext}(m)$ with $(\mathsf{s},\mathsf{p},\mathsf{o}) \in (\text{body}(q_2^{\mathcal{R},O})(\bar{t}))^{\text{safe}}$. We also know that:

- $(\text{body}(q_2)(\bar{t}))^{\text{safe}} \subseteq G_{\mathcal{E}\cup\mathcal{E}_{O^{\mathcal{R}}}}^{\mathcal{M}\cup\mathcal{M}_{O^{\mathcal{R}}}}$
- $O \subseteq G_{\mathcal{E}\cup\mathcal{E}_{O^{\mathcal{R}}}}^{\mathcal{M}\cup\mathcal{M}_{O^{\mathcal{R}}}}$, because of Property 4.7

We know that the saturation operation is monotonous, i.e., if G, G' are RDF graphs such as $G \subseteq G'$, then $G^{\mathcal{R}} \subseteq G'^{\mathcal{R}}$. So if we put everything together, we have (considering inclusion by bijective renaming of blank nodes):

$$\begin{aligned}(\mathsf{s},\mathsf{p},\mathsf{o}) &\in (\text{body}(q_2^{\mathcal{R},O})(\bar{t}))^{\text{safe}}\\ &\subseteq ((\text{body}(q_2)(\bar{t}))^{\text{safe}} \cup O)^{\mathcal{R}}\\ &\subseteq \left(G_{\mathcal{E}\cup\mathcal{E}_{O^{\mathcal{R}}}}^{\mathcal{M}\cup\mathcal{M}_{O^{\mathcal{R}}}}\right)^{\mathcal{R}}\end{aligned}$$

Finally, we have $(\mathsf{s},\mathsf{p},\mathsf{o}) \in \left(G_{\mathcal{E}\cup\mathcal{E}_{O^{\mathcal{R}}}}^{\mathcal{M}\cup\mathcal{M}_{O^{\mathcal{R}}}}\right)^{\mathcal{R}}$.

Secondly, we prove that $\left(G_{\mathcal{E}\cup\mathcal{E}_{O^{\mathcal{R}}}}^{\mathcal{M}\cup\mathcal{M}_{O^{\mathcal{R}}}}\right)^{\mathcal{R}} \subseteq G_{\mathcal{E}\cup\mathcal{E}_{O^{\mathcal{R}}}}^{\mathcal{M}^{\mathcal{R},O}\cup\mathcal{M}_{O^{\mathcal{R}}}}$.

Let t be a triple in $\left(G_{\mathcal{E}\cup\mathcal{E}_{O^{\mathcal{R}}}}^{\mathcal{M}\cup\mathcal{M}_{O^{\mathcal{R}}}}\right)^{\mathcal{R}}$, by definition of the saturation of an RDF graph (Definition 2.8), there exists $i \in \mathbb{N}$ such that :

$$\begin{aligned}t &\in \left(G_{\mathcal{E}\cup\mathcal{E}_{O^{\mathcal{R}}}}^{\mathcal{M}\cup\mathcal{M}_{O^{\mathcal{R}}}}\right)_i^{\mathcal{R}}\\ &\subseteq G_{\mathcal{E}\cup\mathcal{E}_O}^{(\mathcal{M})_i^{\mathcal{R},O}\cup\mathcal{M}_{O^{\mathcal{R}}}} \quad \text{(thanks to Theorem 4.5)}\\ &\subseteq G_{\mathcal{E}\cup\mathcal{E}_{O^{\mathcal{R}}}}^{\mathcal{M}^{\mathcal{R},O}\cup\mathcal{M}_{O^{\mathcal{R}}}}.\end{aligned}$$

□

ECOLE DOCTORALE

Titre : Raisonnement efficace sur des grands graphes hétérogènes

Mots clés : Bases de connaissances, Bases de données, Web sémantique, Intégration de données

Résumé : Le Web sémantique propose des représentations de connaissances, qui permettent d'intégrer facilement des données hétérogènes issues de plusieurs sources en une base de connaissances unifiée. Dans cette thèse, nous étudions des techniques d'interrogation de telles bases de connaissances.
La première partie est dédiée à des techniques de réponse à des requêtes sur une base de connaissances représentée par un graphe RDF sous des contraintes ontologiques. Les connaissances implicites produites par le raisonnement, à partir des règles de déduction RDFS, doivent être prises en compte pour répondre correctement à de telles requêtes. Pour commencer, nous présentons un algorithme de reformulation de requêtes dites Basic Graph Pattern (BGP), qui exploite une partition des règles de déduction en des règles sur les assertions et sur les contraintes. Puis nous introduisons une nouvelle disposition du stockage des graphes RDF, qui combine deux dispositions connues. Pour ces deux contributions, des expérimentations permettent de valider nos résultats théoriques et algorithmiques.
Dans la deuxième partie, nous considérons le problème d'interrogation, par des requêtes BGP, de sources de données hétérogènes intégrées en un graphe RDF. Nous introduisons un cadre d'intégration de données sous des contraintes ontologiques RDFS, utilisant une spécification d'intégration basée sur des mappings Global-Local-As-View, rarement considérée jusqu'ici dans la littérature. Nous présentons plusieurs stratégies de réponse à des requêtes, qui, soit matérialisent les données en un graphe RDF, soit laissent ce graphe virtuel. Ces stratégies diffèrent sur quand et comment le raisonnement RDFS est supporté. Nous avons implémenté ces stratégies dans une plate-forme et mené des expérimentations qui démontrent l'intérêt particulier d'une des stratégies basée sur la saturation des mappings. Finalement, nous montrons que cette dernière technique peut être étendue au delà des règles de déduction RDFS au raisonnement défini par un sous-ensemble des règles existentielles.

Title : Efficient reasoning on large-scale heterogeneous data

Keywords : Knowledge bases, Databases, Semantic Web, Data integration

Abstract : The Semantic Web offers knowledge representations, which allow to integrate heterogeneous data from several sources into a unified knowledge base. In this thesis, we investigate techniques for querying such knowledge bases.
The first part is devoted to query answering techniques on a knowledge base, represented by an RDF graph subject to ontological constraints. Implicit information entailed by the reasoning, enabled by the set of RDFS entailment rules, has to be taken into account to correctly answer such queries. First, we present a sound and complete query reformulation algorithm for Basic Graph Pattern queries, which exploits a partition of RDFS entailment rules into assertion and constraint rules. Second, we introduce a novel RDF storage layout, which combines two well-known layouts. For both contributions, our experiments assess our theoretical and algorithmic results.
The second part considers the issue of querying heterogeneous data sources integrated into an RDF graph, using BGP queries. Following the Ontology-Based Data Access paradigm, we introduce a framework of data integration under an RDFS ontology, using the Global-Local-As-View mappings, rarely considered in the literature. We present several query answering strategies, which may materialize the integrated RDF graph or leave it virtual, and differ on how and when RDFS reasoning is handled. We implement these strategies in a platform, in order to conduct experiments, which demonstrate the particular interest of one of the strategies based on mapping saturation. Finally, we show that mapping saturation can be extended to reasoning defined by a subset of existential rules.

Institut Polytechnique de Paris
91120 Palaiseau, France