Integrating and querying linked datasets through ontological rules

Birkbeck, University of London
Department of Computer Science

Integrating and Querying Linked Data Sets Through Ontological Rules

Mirko Dimartino

Submitted in part fulfilment of the requirements for the degree of
Doctor of Philosophy in Computer Science and Information Systems
September 2019

Abstract

The Web of Linked Open Data has developed from a few datasets in 2007 into a large data space containing billions of RDF triples published and stored in hundreds of independent datasets, so as to form the so called Linked Open Data Cloud. This information cloud, ranging over a wide set of data domains, poses a challenge when it comes to reconciling heterogeneous schemas or vocabularies adopted by data publishers. Motivated by this challenge, in this thesis was address the problem of integrating and querying multiple heterogeneous Linked Data sets through ontological rules. Firstly, we propose a formalisation of the notion of a peer-to-peer Linked Data integration system, where the mappings between peers comprise schema-level mappings and equality constraints between different IRIs; we call this formalism an *RDF Peer System* (RPS). We show that the semantics of the mappings preserve tractability of answering Basic Graph Pattern (BGP) SPARQL queries against the data stored in the RDF sources and the set of constraints given by the RPS mappings. Then, we address the problem of SPARQL query rewriting under RPSs and we show that it is not possible to rewrite an input BGP SPARQL query into a SPARQL 1.0 query under general RPSs, as the RPS peer mappings are not first-order-rewritable rules; this is a major drawback of general RPSs since data materialisation is required to exploit their full semantics.

With the adoption of the more recent standard SPARQL 1.1 and its property paths we are able to extend the expressivity of the target language beyond first-order by including regular expressions in the body of the target SPARQL queries, that is, by expressing conjunctive two-way regular path queries (C2RPQs). Following this idea, in the second part of the thesis we step away from the language of RPSs to conduct a study on C2RPQ-rewritability under a broader ontology language. We define $\mathcal{ELHI}_h^{lin}$ (harmless linear $\mathcal{ELHI}$), an ontology language that generalises both the DL-Lite$_\mathcal{R}$ and linear $\mathcal{ELH}$ description logics. We prove the rewritability of instance queries (queries with a single atom in their body) under $\mathcal{ELHI}_h^{lin}$ knowledge bases with C2RPQs as the target language, presenting a query rewriting algorithm that makes use of non-deterministic finite-state automata. Following from that,

we propose a query rewriting algorithm for answering conjunctive queries under $\mathcal{ELHI}_h^{lin}$ knowledge bases, with C2RPQs as the target language. Since C2RPQs can be straightforwardly expressed in SPARQL 1.1 by means of property paths, we believe that our approach is directly applicable to real-world querying settings.

Lastly, we undertake a complexity analysis for query answering under $\mathcal{ELHI}_h^{lin}$. We analyse the computational cost of query answering in terms of both data complexity (where the ontology and the query are fixed and the data alone is a variable input) and combined complexity (where query, ontology and data all constitute the variable input). We show that answering instance queries under $\mathcal{ELHI}_h^{lin}$ is NLogSpace-complete for data complexity and in PTime for combined complexity; we also show that answering CQs under $\mathcal{ELHI}_h^{lin}$ is NLogSpace-complete for data complexity and NP-complete for combined complexity.

Acknowledgements

Firstly, I would like to express my sincere gratitude to my supervisors. Specifically, I wish to thank: *Professor Alexandra Poulovassilis*, for being continuously supportive and warm, for her patience and for giving me the strength to never give up over the years; I definitely could not have reached the finish line without her support. *Doctor Andrea Calì*, for always stimulating my creative process and inspiring me to think outside of the box. *Professor Peter Wood*, for being a solid and consistent presence over the years; his guidance and ever-positive attitude made me feel safe at all times during my PhD journey. I sincerely could not have imagined having better mentors for my PhD study.

Besides my supervisors, I would like to thank my thesis examiners, *Doctor Andreas Pieris* and *Doctor Emanuel Sallinger*, for their insightful comments.

I wish to thank my wonderful parents, *Alessandro* and *Nina*, for nourishing me and supporting me since day one of my life, and for continuing to do so. My sincere thanks also goes to my four sisters, my friends and my fellow SGI Buddhists, for being my best cheerleaders.

Lastly, I would like to express special thanks to my friend *Ryan Williams* and to my therapist *Doctor Riccardo Ambrosi*, for being key figures in my support system.

Dedication

I dedicate this thesis to my mentor in life, *Doctor Daisaku Ikeda*.

‘The real struggle in life is with ourselves. The true secret of success is the refusal to give up, the refusal to fail; it lies in the struggle to win the battle against one’s own weaknesses.’

Daisaku Ikeda

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation and Objectives

Linked Data[1] is a term used to describe recommended best practices for exposing, sharing, and connecting pieces of data, information, and knowledge on the Semantic Web using the Resource Description Framework[2] (RDF). RDF is a family of World Wide Web Consortium (W3C) specifications[3] originally designed as a metadata model for resources on the World-Wide-Web (WWW, Web). RDF describes *resources* that are connected by means of *properties*; resources are represented by Internationalized Resource Identifiers (IRIs)[4]; when resources are unknown or unavailable they are represented by *blank nodes*. Properties denote relationships between resources and are predefined IRIs published within RDF vocabularies. An *RDF dataset* is a set of *triples* of the form $\langle subject, predicate, object \rangle$, where the *subject* is a resource, the predicate is a property and the object is either a resource or a literal (e.g. a string, a piece of text, a number, etc...). RDF can also be used to

[1] `https://www.w3.org/standards/semanticweb/data`
[2] `https://www.w3.org/RDF/`
[3] `http://www.dblab.ntua.gr/~bikakis/XMLSemanticWebW3CTimeline.pdf`
[4] `https://www.w3.org/International/O-URL-and-ident.html`

describe data schemas; for this purpose, the W3C has published two main vocabularies: RDF-Schema, or RDFS[5], which provides a basic ontology language, and the more expressive OWL[6], which can be used to represent richer and more complex knowledge about entities. In order to query RDF databases, the W3C has proposed the SPARQL[7] query language, which can be used to express queries across diverse data sources, whether the data is stored natively as RDF or viewed as RDF via middleware.

When Linked Data can be freely used and distributed by anyone, it is called Linked Open Data (LOD). The Web of Linked Open Data has developed from a few datasets in 2007 into a large data space containing billions of RDF triples published and stored in hundreds of independent datasets, so as to form the so called *Linked Open Data Cloud*[8]. This information cloud, ranging over a wide set of data domains, poses a challenge when it comes to reconciling heterogeneous schemas or vocabularies adopted by data publishers. According to Linked Open Data best practices [66], data publishers should reuse terms from widely-used vocabularies already present in the cloud, in order to enable the discovery of additional data and to support the integration of data from multiple sources. Following this vision, Linked Data applications should be able to access an open, global data space with an approach similar to how a single database can be queried, in order to obtain more extensive answers as new data sources are published on the Web. However, existing vocabularies often do not provide all the terms needed to completely describe the content of a dataset. Thus, data providers need to define proprietary terms as sets of new IRIs, again published on the cloud. This trend leads to the formation of islands of data describing overlapping domains, rather than generating a single consistent global knowledge base.

[5] `https://www.w3.org/TR/rdf-schema/`
[6] `https://www.w3.org/OWL`
[7] `https://www.w3.org/TR/rdf-sparql-query/`
[8] `https://lod-cloud.net/`

Examples of this trend include the *DBpedia*[9], *YAGO*[10], *WordNet*[11] and *Freebase*[12] cross-domain datasets, all defining proprietary vocabularies. Taking it a step further, we notice several other overlapping datasets, such as *ACM*[13], *IEEE*[14], *DBLP*[15] and *ePrints*[16] in the domain of publications, *PubMed*[17], *GeneID*[18], *Drug Bank*[19] and *Gen Bank*[20] in the life sciences, *GeoNames*[21], *Linked GeoData*[22] and *Geo Linked Data*[23] in the geographic domain, as well as *Last.FM*[24], *MySpace*[25], *BBC Music*[26] and *Music Brainz*[27] in the domain of media. Numerous other examples can be seen in the Web of Data graph[28]. The ever-increasing overlap of similar vocabularies is also observed in a recent survey on the adoption of Linked Data best practices [99], where the authors show the results of a crawl that includes 1014 different datasets in the Linked Data cloud: out of the 638 different vocabularies only 263 (41.22%) are non-proprietary, while 375 vocabularies (58.77%) are proprietary[29]. In this regard, the more datasets that are published, the more crucial is the issue of managing interoperability of such highly heterogeneous vocabularies.

Over the past years, researchers in the Semantic Web community have attempted to tackle these challenges, proposing several approaches based on semantics-preserving SPARQL rewriting algorithms [42, 80, 81]. These methods allow users to pose

[9] `https://wiki.dbpedia.org/`
[10] `https://github.com/yago-naga/yago3`
[11] `https://wordnet.princeton.edu/`
[12] `https://www.wikidata.org/wiki/Q15241312`
[13] `https://dl.acm.org/`
[14] `https://www.ieee.org/`
[15] `https://dblp.uni-trier.de/`
[16] `https://wiki.eprints.org/w/API:EPrints/Database`
[17] `https://www.ncbi.nlm.nih.gov/pubmed/`
[18] `https://www.ncbi.nlm.nih.gov/gene`
[19] `https://www.drugbank.ca/`
[20] `https://www.ncbi.nlm.nih.gov/genbank/`
[21] `https://www.geonames.org/`
[22] `http://linkedgeodata.org/`
[23] `https://old.datahub.io/dataset/geolinkeddata`
[24] `https://www.last.fm/`
[25] `https://myspace.com/`
[26] `https://www.bbc.co.uk/music/`
[27] `https://musicbrainz.org/`
[28] `http://lod-cloud.net`
[29] The authors consider a vocabulary to be proprietary if it is used only by a single dataset.

SPARQL queries using a preferred vocabulary and a rewriting algorithm provides translations of the query using similar terms from other vocabularies. To rewrite queries, these techniques typically utilise reasoning rules with respect to a set of mappings specified between the RDF sources. The rewritten query is evaluated over the sources and a more complete answer is returned to the user than would be returned by the original query. Query rewriting approaches in the literature are typically based on a two-tier global-to-local schema integration paradigm [74], where queries are expressed over a global schema and are reformulated into the language of the source vocabularies, to be then evaluated over the data stored in the data sources.

In contrast, in the Linked Data cloud each data store is an autonomous resource whose vocabulary should represent part of the global schema, available on the web. Linked Data consumers should be able to pose queries adopting any of the source vocabularies and to access similar sources through query translation, in a transparent way and without relying on a single global schema. We believe that a peer-to-peer query rewriting approach is more suitable in this setting than a global-to-local approach because it provides a decentralised architecture where peers act both as clients and as servers during the query reformulation process [58]. In addition, the global-to-local approaches typically require a comprehensive global schema design before they can be used, thus they are difficult to scale because schema evolution may break backwards compatibility. Scalability is a key property of LOD-oriented data mediation systems, due to the continuous increase of data published on the web.

Implementations or re-adaptations of existing SPARQL query rewriting techniques in a peer-to-peer environment are impractical. For instance, a query rewriting algorithm which is guaranteed to terminate on computing a query translation from one source to another may run indefinitely when adopted in a distributed peer-to-peer

scenario, typically in the presence of *cyclic* mappings (i.e., mappings through which peers are mutually dependent on each other). As well as scalability, termination of the algorithms with arbitrary mappings is a fundamental requirement in the context of LOD, since the mappings between overlapping domains are not under the control of any single authority.

Motivated by this challenge, the first part of this thesis proposes a formalisation of the notion of a peer-to-peer Linked Data integration system that allows us to explore the decidability of the query answering problem when reasoning over a set of peer-to-peer mappings between RDF data sources. We call this formalism an *RDF Peer System* (RPS). The mappings between peers comprise both schema-level mappings and equality constraints which entail the semantics of the OWL `sameAs`[30] property . In Chapter 3 of the thesis we will show that the semantics of the mappings preserve tractability of the *conjunctive SPARQL query answering problem over RPSs*, that is, answering queries expressed in the conjunctive fragment of the SPARQL query language against the data stored in the RDF sources and the set of constraints given by the RPSs mappings.

To specify our query rewriting procedure over RPSs we study several works in the literature that address query rewritability properties for a class of data constraints called *Tuple Generating Dependencies* (TGDs) [29]. For a query q expressed in a language $\mathcal{Q}$, a language $\mathcal{L}$, a set of constraints Σ expressed in a language $\mathcal{L}_\Sigma$, we say that q and Σ are $\mathcal{L}$*-rewritable* if there exists a reformulation of q, q_Σ, expressed in the language $\mathcal{L}$ such that q_Σ evaluated over a data source D yields the same result as q evaluated against Σ and D. If q and Σ are $\mathcal{L}$*-rewritable*, then we say that q and Σ *enjoy* the $\mathcal{L}$*-rewritability property*, or that Σ *is* $\mathcal{L}$*-rewritable with respect to* q. Also, given a query language $\mathcal{Q}$, we say that $\mathcal{Q}$ and $\mathcal{L}_\Sigma$ are $\mathcal{L}$*-rewritable* if, for each $q \in \mathcal{Q}$ and $\Sigma \in \mathcal{L}_\Sigma$, q and Σ are $\mathcal{L}$*-rewritable*.

[30] `http://sameas.org`

By translating the RPS mappings into TGDs we will see that for (general) RPSs it is not possible to design a semantics-preserving SPARQL rewriting algorithm. In fact, we will see that conjunctive SPARQL queries and the set of TGDs corresponding to (general) RPSs do not enjoy the First Order-rewritability property. The following example illustrates this.

Example 1.1.1. Consider the mapping given by the following rule:

$$triple(x, \textbf{is\_younger\_than}, z), triple(z, \textbf{is\_younger\_than}, y) \rightarrow triple(x, \textbf{is\_younger\_than}, y),$$

where **is_younger_than** is the IRI for the relation *is younger than*. This rule states that *if x is younger than z and z is younger then y, then x is younger than y*. This rule is not FO-rewritable since it captures the transitive closure of the relation *is younger than*, and therefore does not allow a rewriting to a finite number of FO queries [32]. For instance, let us consider the following SPARQL query:

```
SELECT ?x
WHERE { ?x is_younger_than Paul }
```

which asks for all the people younger than Paul. It is easy to see that a naive rewriting technique produces an infinite union in the body of the SPARQL query:

```
SELECT ?x
WHERE { { ?x is_younger_than Paul }
UNION
    { ?x is_younger_than ?z . ?z is_younger_than Paul }
UNION
    { ?x is_younger_than ?z . ?z is_younger_than ?y .
```

```
                                    ?y is_younger_than Paul }
 UNION
 ...
```

In Chapter 3, we will identify subsets of RPSs that do enjoy the FO-rewritability property.

With the adoption of the more recent standard SPARQL 1.1 query language, LOD database systems should be able to answer queries that are more expressive than FO queries. Using the *property paths*[31] of SPARQL 1.1 it is possible to define regular expressions on predicates in the body of the query. In fact, any conjunctive regular path query (CRPQ) can be translated to a SPARQL 1.1 query [10]. It may thus be possible to find larger subsets of RPSs for which it is possible to answer queries via query rewriting in SPARQL 1.1.

Example 1.1.2. Let us consider again the query and the TGD in Example 1.1.1. With the use of the property paths feature of SPARQL 1.1, we can have a finite rewriting:

```
SELECT ?x
WHERE  { ?x is_younger_than+ Paul }
```

where *is_younger_than+* is a property path expression. Query evaluation determines all matches of a property path expression and binds the subject or object as appropriate, in this case a path of one or more occurrences of *is_younger_than* such that the final object is Paul.

As in the case of FO-rewritability, we wish to identify (larger) subsets of RPSs that have the CRPQ-rewritability property. The problem of identifying CRPQ-

[31] https://www.w3.org/TR/sparql11-property-paths/

rewritability of ontological constraints has not been previously addressed in the literature. To the best of our knowledge, the closest work is that of [37], which treats rewriting of regular path queries to regular path queries over views (which are in turn regular path queries); they do not consider conjunctions of regular path queries. This gap motivates our exploration of the problem of identifying CRPQ-rewritability of ontological constraints in Chapters 4 and 5. For this purpose, we adopt a logic-based formalism that underpins the OWL Web Ontology Language, known as *Description Logic* (DL), which is a family of knowledge representation formalisms that are able to capture a wide range of ontological constructs [8]. DLs are based on *concepts* (unary predicates representing classes of individuals) and *roles* (binary predicates representing relations between classes).

In our recent work [48], we first consider a DL we call $\mathcal{ELH}^{\ell in}$ also known in the literature as DL-Lite^{+} [91]. This is a language that does not allow a finite rewriting of FO queries into FO queries [55]. We show how to encode CRPQ rewritings under $\mathcal{ELH}^{\ell in}$ by means of a finite-state automaton; intuitively, the automaton is able to encode infinite sequences of rewriting steps. In this thesis, we extend this approach to rewrite conjunctive queries (CQs) into *Conjunctive Two-Way Regular Path Queries*[32] (C2RPQs) under a more expressive language that we call *harmless linear* $\mathcal{ELHI}$, denoted by $\mathcal{ELHI}_h^{\ell in}$, which is a generalisation of $\mathcal{ELH}^{\ell in}$. We show that CQs and $\mathcal{ELHI}_h^{\ell in}$ enjoy C2RPQ-rewritability and therefore that under $\mathcal{ELHI}_h^{\ell in}$ it is possible to answer conjunctive SPARQL queries with a pure rewriting approach that leverages the expressive power of SPARQL 1.1 property paths.

Since the complexity of answering C2RPQs is in the highly tractable class NLogspace with respect to *data complexity* [10] (where the ontology and the query are fixed and the data alone is a variable input) it follows that under C2RPQ-rewritable rules, the query answering problem would also be highly tractable. As well as considering

[32]C2RPQs are CRPQs with the additional capability of expressing query navigation in two directions, forwards and backwards.

data complexity, we assess in Chapter 5 query answering under C2RPQ-rewritable rules in terms of *combined complexity*, where query, ontology and data all constitute the variable input.

1.2 Thesis Contributions

The contributions of the thesis are the following:

- We propose a formalisation of the notion of a peer-to-peer Linked Data integration system, where the mappings between peers comprise schema-level mappings and equality constraints between different IRIs.
- We show that answering conjunctive SPARQL queries on an RDF peer system can be done in polynomial time in terms of data complexity.
- We present a novel query rewriting technique, based on non-deterministic finite-state automata, for *instance queries* (i.e., queries having a single atom in the query body) on $\mathcal{ELHI}_h^{lin}$ knowledge bases, with C2RPQs as the target language.
- Based on the rewriting technique for instance queries, we present a technique for rewriting CQs into C2RPQs under $\mathcal{ELHI}_h^{lin}$. Since C2RPQs can be straightforwardly expressed in SPARQL 1.1, exploiting its property paths, our approach is applicable to real-world RDF knowledge bases that are queryable via SPARQL endpoints.
- We undertake a complexity analysis for query answering under $\mathcal{ELHI}_h^{lin}$. We analyse the computational cost of query answering in terms of both *data complexity* (where the query is fixed, and the complexity is expressed in terms of the size of the database) and *combined complexity* (where the complexity is measured in the size of both the query and the database). We show that answering instance queries under $\mathcal{ELHI}_h^{lin}$ is NLogSpace-complete for data

complexity and in PTIME for combined complexity. We also show that answering CQs under $\mathcal{ELHI}_h^{lin}$ is NLOGSPACE-complete for data complexity and NP-complete for combined complexity.

- We formally prove the correctness of our query rewriting algorithms, and also that they comply with the upper complexity bounds.

1.3 Outline of the Thesis

The thesis is structured as follows: In Chapter 2 we review fundamentals of relational databases, conjunctive queries, tuple-generating dependencies and the chase procedure. We review the description logic $\mathcal{ELH}^{lin}$ and its model-theoretic semantics. In the same chapter we review the current state of the art in frameworks for SPARQL query rewriting, peer-to-peer systems and query rewriting under description logics. In Chapter 3 we propose a formalisation of the notion of a peer-to-peer Linked Data integration system, RPS, and we show that it supports tractability of the conjunctive SPARQL query answering problem. Then, we assess FO-rewritability of RPSs. We show that RPSs are not generally FO-rewritable and we identify a subset that do enjoy this property. Then we propose a rewriting algorithm for the restricted form of RPSs and we devise a rewriting in Datalog for the general case. In the same chapter we propose optimisations for the rewriting algorithms, and we present a case study for the restricted form of RPSs.

In Chapter 4 we define the description logic $\mathcal{ELHI}_h^{lin}$ (harmless linear $\mathcal{ELHI}$) and we prove the rewritability of *instance queries* (queries with a single atom in their body) under $\mathcal{ELHI}_h^{lin}$ knowledge bases with C2RPQs as the target language, presenting a query rewriting algorithm that makes use of non-deterministic finite-state automata. In Chapter 5 we propose a query rewriting algorithm for answering *conjunctive queries* under $\mathcal{ELHI}_h^{lin}$ knowledge bases, with C2RPQs as target language. Then,

we undertake a complexity analysis for query answering under $\mathcal{ELHI}_h^{lin}$. We analyse the computational cost of query answering in terms of both data complexity and combined complexity.

We conclude this thesis in Chapter 6 with a discussion on the results achieved and directions for future work.

1.4 Publications

- *Peer-to-Peer Semantic Integration of Linked Data* [47], covered in Chapter 3;
- *Peer-based Query Rewriting in SPARQL for Semantic Integration of Linked Data* [45], covered in Chapter 3;
- *Implementing Peer-to-Peer Semantic Integration of Linked Data* [46], covered in Chapter 3;
- *Combining Flexible Queries and Knowledge Anchors to facilitate the exploration of Knowledge Graphs* [92], covered in Section 3.4 of Chapter 3;
- *Query Rewriting under Linear EL Knowledge Bases* [48], covered in Chapter 4;
- *Efficient Ontological Query Answering by Rewriting into Graph Queries* [49], covered in Chapter 5.

Chapter 2

Background Theory and Related Work

In this chapter we introduce some background theory and a discussion of the literature related to the work described in this thesis. The background theory is covered in Section 2.1, where we review the principles of relational, RDF and graph databases. A review of the literature relevant to our work follows in Section 2.2, focussing on frameworks for SPARQL query rewriting (Section 2.2.1), peer-to-peer systems (Section 2.2.2), and query rewriting under DLs (Section 2.2.3).

2.1 Background Theory

This section is divided into three subsections introducing the worlds of *relational*, *RDF* and *graph databases*:

- *Relational Databases:* in Section 2.1.1 we introduce the foundamentals of relational databases and conjunctive queries, data integration and tuple-generating

dependencies; this section establishes the foundational work on databases and database management systems.

- *RDF Databases:* in Section 2.1.2 we present the definitions of RDF, SPARQL, RDFS and OWL which form the basis of our formalisation of a peer-to-peer Linked Data integration system.

- *Graph Databases:* in Section 2.1.3 we introduce the DL $\mathcal{ALCHI}$ with its model-theoretic semantics and the definitions of regular languages and Conjunctive Regular Path queries, to inform our work on query rewriting using regular languages under ontologies expressed in DL.

2.1.1 Relational Databases

Relational Databases and Conjunctive Queries

Consider two pairwise disjoint (infinite) sets of symbols Δ_c and Δ_z such that: Δ_c is a set of *constants* (which constitutes the domain of a database), and Δ_z is a set of *labelled nulls* (used as placeholders for unknown values). Different constants represent different values (*unique name assumption*), while different nulls may represent the same value. Throughout this thesis, we denote by $\boldsymbol{X}$ sequences of variables, e.g., $X_1, \ldots, X_k$, where $k \geqslant 0$, and by $[n]$ the set $1, \ldots, n$, for any $n \geqslant 1$.

A *relational schema* $\mathcal{R}$ (or simply a *schema*) is a set of *relational symbols* (or *predicate symbols*), each with its associated arity. A *position* $r[i]$ is identified by a predicate $r \in \mathcal{R}$ and its i-th argument.

A *term* t is a constant, labelled null, or variable. An *atomic formula* (or simply *atom*) has the form $r(t_1, \ldots, t_n)$, where $r \in \mathcal{R}$ has arity n, and $t_1, \ldots, t_n$ are terms. A *conjunction of atoms* has the form $a_1 \wedge a_2 \wedge \cdots \wedge a_n$, where $a_1, a_2 \ldots a_n$ are atoms. Conjunctions of atoms are often identified by the sets of their atoms.

A *substitution* from one set of symbols S_1 to another set of symbols S_2 is a function $h : S_1 \to S_2$. Given an atom $a = r(t_1, \ldots, t_n)$ and a substitution h, $h(a)$ denotes the atom $r(h(t_1), \ldots, h(t_n))$. Given a two sets of atoms A_1, A_2, both over the same schema $\mathcal{R}$, and a substitution h from the set of terms of A_1 to the set of terms of A_2, we say that h is a *homomorphism* from A_1 to A_2 if the following conditions hold:

(i) if $t \in \Delta_c$, then $h(t) = t$;

(ii) if $r(t_1, \ldots, t_n)$ is in A_1, then $h(r(t_1, \ldots, t_n)) = r(h(t_1), \ldots, h(t_n))$ is in A_2.

The notion of homomorphism naturally extends to conjunctions of atoms.

Example 2.1.1. Let us consider the set of atoms $T_1 := \{t(x, a, y), t(z, a, u)\}$ and $T_2 := \{t(v, a, w), r(v, b, w)\}$, where a, b are constants and x, y, z, v, w are variables. An example of homomorphism h from T_1 to T_2 is the function $\{x \to v, y \to w, z \to v, u \to w, a \to a\}$. By applying h to T_1 we have:

$$h(T_1) = \{t(h(x), h(a), h(y)), t(h(z), h(a), h(u))\} = \{t(v, a, w)\},$$

with $\{t(v, a, w)\} \subseteq T_2$.

A *relational instance* I for a schema $\mathcal{R}$ is a (possibly infinite) set of atoms of the form $r(t_1, t_2, \ldots, t_n)$, where $r \in \mathcal{R}$ has arity n and $t_1, t_2, \ldots, t_n \in (\Delta_c \cup \Delta_z)$. A *database* is a finite relational instance. A *conjunctive query* (CQ) q of arity n over a schema $\mathcal{R}$ is a formula of the form $ans_q(\boldsymbol{X}) \leftarrow \phi(\boldsymbol{X}, \boldsymbol{Y})$, where $\phi(\boldsymbol{X}, \boldsymbol{Y})$ is a conjunction of atoms over $\mathcal{R}$, and ans_q is an n-ary predicate not in $\mathcal{R}$. $\phi(\boldsymbol{X}, \boldsymbol{Y})$ is called the body of q, denoted as $body(q)$, and $ans_q(\boldsymbol{X})$ is the head of q, denoted as $head(q)$. We use the symbol **CQ** to denote the class of all CQs. The *answer* to a CQ q of arity n over an instance I, denoted as $q(I)$, is the set of all atoms $ans_q(\boldsymbol{t})$ where $\boldsymbol{t} \in (\Delta_c)^n$ and for which there exists a homomorphism $h : \boldsymbol{X} \cup \boldsymbol{Y} \to \Delta_c \cup \Delta_z$ such that $h(\phi(\boldsymbol{X}, \boldsymbol{Y})) \subseteq I$ and $h(\boldsymbol{X}) = \boldsymbol{t}$.

As we will see in Section 2.1.2, RDF databases can contain *blank nodes* [1], which represent resources for which a URI or literal is not given. Thus, to have a more direct mapping between the RDF and the relational worlds, we include null values in the definition of relational databases.

A *boolean conjunctive query* (BCQ) is a CQ of arity zero. A BCQ q has either the empty set or the empty atom $ans_q(\langle\rangle)$ as a possible answer; in the latter case it is said to have a positive answer. Formally, a BCQ q has a *positive* answer over I, denoted as $I \models q$, if and only if $ans_q(\langle\rangle) \in q(I)$.

A *union* of CQs (UCQ) Q of arity n is a set of CQs, where each $q \in Q$ has the same arity n and uses the same predicate symbol in the head. The *answer* to Q over an instance I, denoted as $Q(I)$, is the set of atoms $\{\ ans_Q(\boldsymbol{t}) \mid$ there exists $q \in Q$ such that $ans_q(\boldsymbol{t}) \in q(I)\ \}$.

Example 2.1.2. Consider the database

$$I \coloneqq \{t(a,b,c), t(a,a,b), r(a,b,c)\}$$

and the CQs

$$q_1 \coloneqq ans_{q_1}(Y,Z) \leftarrow t(a,Y,Z) \wedge r(X,Y,Z)$$
$$q_2 \coloneqq ans_{q_2} \leftarrow t(X,X,Y)$$
$$q_3 \coloneqq ans_{q_3} \leftarrow r(X,X,Y)$$

where a, b, c are constants and X, Y, Z are variables. We note that q_2 and q_3 are BCQs since they are of arity zero. We have that $q_1(I) = \{ans_{q_1}(b,c)\}$ with the homomorphism $\{X \to a, Y \to b, Z \to c, a \to a\}$, and $q_2(I) = \{ans_{q_2}(\langle\rangle)\}$ with the homomorphism $\{X \to a, Y \to b\}$. q_3 does not have a positive answer over I (i.e., $q_3(I) = \varnothing$) since there does not exist any homomorphism h such that $h(\{r(X,X,Y)\}) \subseteq I$.

[1] `https://www.w3.org/TR/rdf11-mt/#blank-nodes`

Tuple-Generating Dependencies

Answering queries under data or schema constraints is a recurring problem in databases and has been employed in schema integration, information integration, and service discovery [39]. The problem of answering queries under constraints is related to the problem of query containment [41] and, in fact, the two problems are mutually reducible [27].

Here we consider the class of constraints called *tuple-generating dependencies* (TGDs), which is a generalisation of inclusion dependencies [2]. As we shall see in Chapter 3, the mappings defined in our peer-to-peer data integration system can be interpreted as TGDs that satisfy certain properties.

TGDs are first-order constraints that express an implication from one conjunction of atoms to another, and extend the well known Datalog language [2] by allowing existential quantifiers in rule heads. This feature is also known as value invention [79, 22]. We now give definitions of the syntax and semantics of TGDs.

A TGD σ over a schema $\mathcal{R}$ is a first-order formula $\forall \boldsymbol{X} \forall \boldsymbol{Y} \phi(\boldsymbol{X}, \boldsymbol{Y}) \rightarrow \exists \boldsymbol{Z} \psi(\boldsymbol{X}, \boldsymbol{Z})$, where $\phi(\boldsymbol{X}, \boldsymbol{Y})$ and $\psi(\boldsymbol{X}, \boldsymbol{Z})$ are conjunctions of atoms over $\mathcal{R}$, called the *body* and the *head* of σ, denoted as $body(\sigma)$ and $head(\sigma)$, respectively. To avoid verbosity, we will omit here the universal quantifiers in TGDs. A TGD σ is *satisfied* by an instance I of $\mathcal{R}$ if and only if, whenever there exists a homomorphism h such that $h(\phi(\boldsymbol{X}, \boldsymbol{Y})) \subseteq I$, there exists an extension h' of h (i.e., $h' \supseteq h$) such that $h'(\phi(\boldsymbol{X}, \boldsymbol{Z})) \subseteq I$. The satisfaction relation naturally extends to a set of TGDs. Formally, an instance I of $\mathcal{R}$ *satisfies* a set of TGDs Σ if, for each $\sigma \in \Sigma$, σ is satisfied by I.

Example 2.1.3. Consider again the database $I = \{t(a, b, c), t(a, a, b), r(a, b, c)\}$

defined in Example 2.1.2, and the following set of TGDs $\Sigma \coloneqq \{\sigma_1, \sigma_2\}$, with

$$\sigma_1 \coloneqq t(X, Y, Z) \rightarrow r(X, Y, Z)$$
$$\sigma_2 \coloneqq r(X, Y, Z) \rightarrow \exists W\ t(Y, Y, W).$$

σ_1 is satisfied if, for each atom in I of the form $t(X, Y, Z)$, there is an atom of the form $r(X, Y, Z)$. σ_2 is satisfied if, for each atom in I of the form $r(X, Y, Z)$, there is an atom of the form $t(Y, Y, W)$, with W being any constant. We observe that σ_1 is satisfied in I whereas σ_2 is not.

We now define the notion of *query answering* under TGDs. Given a database D for $\mathcal{R}$, a set Σ of TGDs over $\mathcal{R}$, and an instance I of $\mathcal{R}$, we say that $I \models D \cup \Sigma$ if $I \supseteq D$ and I satisfies Σ; I is said to be a *model* of D with respect to Σ. The *set of models* of D with respect to Σ, denoted as $mods(D, \Sigma)$, is the set of all instances I such that $I \models D \cup \Sigma$. The *answer* to a CQ q with respect to D and Σ, denoted as $\texttt{ans}(q, D, \Sigma)$, is the set $\{\boldsymbol{t} \mid \boldsymbol{t} \in q(I) \text{ for all } I \in mods(D, \Sigma)\}$. The *answer* to a BCQ q with respect to D and Σ is *positive*, denoted as $D \cup \Sigma \models q$, if and only if $\texttt{ans}(q, D, \Sigma) \neq \varnothing$.

Query answering under general TGDs is undecidable [12], even when the schema and the set of TGDs are fixed [27]. The two problems of answering CQs and BCQs under TGDs are LOGSPACE-equivalent [41]: we can enumerate the polynomially many tuples of constants which are possible answers to a CQ q, and then, instead of answering the given query q, we answer the polynomially many BCQs obtained by replacing the variables in the body of q with the appropriate constants; a certain tuple $\boldsymbol{t}$ of constants is in the answer of q if and only if the answer to the BCQ obtained from $\boldsymbol{t}$ is true.

Data Integration

Data integration is the problem of combining data stored in different sources and providing the user with a unified view of this data [74]. The design of data integration systems is of great importance in real world applications. A clear example of this is the Web of Linked Open Data, which has developed from a few datasets in 2007 into a large data space containing billions of RDF triples published and stored in hundreds of independent datasets. This huge information cloud, ranging over a large number of application domains, poses a great challenge when it comes to reconciling the heterogeneous schemas, vocabularies and entity names adopted by data publishers.

Data integration systems are often modelled by an architecture based on a global schema and a set of data sources. The sources contain the actual data, while the global schema provides a reconciled, integrated, and virtual view of the underlying sources. Data can also be "materialised" in the modelling language of the global schema using the set of mappings between the sources and the global schema. An example of data materialisation can be found later in this section, where we illustrate the *chase* procedure (see Section 2.1.1).

Here we introduce the formal definition of a data integration system taken from [74]. A *data integration system* is a triple $\langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ where:

- $\mathcal{G}$ is the global schema, expressed in a language $\mathcal{L}_\mathcal{G}$ over an alphabet $\mathcal{A}_\mathcal{G}$. The alphabet comprises a symbol for each element of $\mathcal{G}$ (i.e., relation if $\mathcal{G}$ is relational, class if G is object-oriented, etc.).
- $\mathcal{S}$ is the source schema, expressed in a language $\mathcal{L}_\mathcal{S}$ over an alphabet $\mathcal{A}_\mathcal{S}$. The alphabet $\mathcal{A}_\mathcal{S}$ includes a symbol for each element of the sources.
- $\mathcal{M}$ is the mapping between $\mathcal{G}$ and $\mathcal{S}$, constituted by a set of assertions of the

forms $q_{\mathcal{S}} \rightsquigarrow q_{\mathcal{G}}$, $q_{\mathcal{G}} \rightsquigarrow q_{\mathcal{S}}$ where $q_{\mathcal{S}}$ and $q_{\mathcal{G}}$ are two queries of the same arity, respectively over the source schema $\mathcal{S}$, and over the global schema $\mathcal{G}$. Queries $q_{\mathcal{S}}$ are expressed in a query language $\mathcal{L}_{\mathcal{M},\mathcal{S}}$ over the alphabet AS , and queries $q_{\mathcal{G}}$ are expressed in a query language $\mathcal{L}_{\mathcal{M},\mathcal{G}}$ over the alphabet $\mathcal{A}_{\mathcal{G}}$. Intuitively, an assertion $q_{\mathcal{S}} \rightsquigarrow q_{\mathcal{G}}$ specifies that the concept represented by the query $q_{\mathcal{S}}$ over the sources corresponds to the concept in the global schema represented by the query $q_{\mathcal{G}}$ (similarly for an assertion of type $q_{\mathcal{G}} \rightsquigarrow q_{\mathcal{S}}$).

Intuitively, the source schema describes the structure of the sources, where the real data are, while the global schema provides a reconciled, integrated, and virtual view of the underlying sources. The assertions in the mapping establish the connection between the elements of the global schema and those of the source schema. When users pose queries over the data integration system, they pose their queries on $\mathcal{G}$, and these are reformulated to queries on $\mathcal{S}$ using the mappings $\mathcal{M}$.

Two common ways to model this correspondence exist: *Global as View* or GAV and *Local as View* or LAV [74, 25].

In the GAV approach, the mapping $\mathcal{M}$ associates to each element g in $\mathcal{G}$ a query $q_{\mathcal{S}}$ over $\mathcal{S}$. In other words, the query language $\mathcal{L}_{\mathcal{M},\mathcal{G}}$ allows only expressions constituted by one symbol of the alphabet $\mathcal{A}_{\mathcal{G}}$. Therefore, a GAV mapping is a set of assertions, one for each element g of $\mathcal{G}$, of the form $g \rightsquigarrow q_{\mathcal{S}}$. In the case where $\mathcal{L}_{\mathcal{M},\mathcal{G}}$ is a set of TGDs, then a GAV assertion is of the form:

$$\forall \boldsymbol{X} g(\boldsymbol{X}) \leftarrow \exists \boldsymbol{Z} \psi_{A_{\mathcal{S}}}(\boldsymbol{X}, \boldsymbol{Z}),$$

where $\psi_{\mathcal{A}_{\mathcal{S}}}(\boldsymbol{X}, \boldsymbol{Z})$ is conjunctions of atoms over $\mathcal{A}_{\mathcal{S}}$.

From the modeling point of view, the GAV approach is based on the idea that the content of each element g of the global schema should be characterized in terms of a view $q_{\mathcal{S}}$ over the sources. In some sense, the mapping explicitly tells the system how

to retrieve the data when one wants to evaluate the various elements of the global schema.

For example, consider if one of the sources served a weather website. The designer would add an element corresponding to the concept 'weather' in the global schema. Then the bulk of effort concentrates on specifying mappings that will transform a global query on 'weather' into a query over the weather website. This effort can become complex if some other source also relates to 'weather', because the designer may need to understand how to combine the results from the two sources into the global schema.

In the LAV approach, the mapping $\mathcal{M}$ associates to each element s of the source schema $\mathcal{S}$ a query $q_{\mathcal{G}}$ over $\mathcal{G}$. In other words, the query language $\mathcal{L}_{\mathcal{M},\mathcal{S}}$ allows only expressions constituted by one symbol of the alphabet $\mathcal{A}_{\mathcal{S}}$. Therefore, a LAV mapping is a set of assertions, one for each element s of $\mathcal{S}$, of the form $s \rightsquigarrow q_{\mathcal{G}}$. In the case where $\mathcal{L}_{\mathcal{M},\mathcal{S}}$ is a set of TGDs, then a LAV assertion is of the form:

$$\forall \boldsymbol{X} s(\boldsymbol{X}) \leftarrow \exists \boldsymbol{Z} \psi_{A_{\mathcal{G}}}(\boldsymbol{X}, \boldsymbol{Z}),$$

where $\psi_{\mathcal{A}_{\mathcal{G}}}(\boldsymbol{X}, \boldsymbol{Z})$ is conjunctions of atoms over $\mathcal{A}_{\mathcal{G}}$.

From the modeling point of view, the LAV approach is based on the idea that the content of each source s should be characterized in terms of a view $q_{\mathcal{G}}$ over the global schema.

Consider again if one of the sources serves a weather website. The designer would add a corresponding element for 'weather' to the global schema if none existed already. Then, the designer would specify a LAV mapping to transform a local query on the weather website to a global query on 'weather'. This requires less effort on the part of the designer than with the GAV approach. However, global query processing with LAV mappings requires the more complex process of answering queries using

views [75].

The TGD Chase. The *chase procedure* (or simply *chase*) is a fundamental algorithmic tool introduced for generating the implications of a set of dependencies [78], and later for checking query containment [68]. The chase provides a process for repairing a database with respect to a set of dependencies so that the resulting database satisfies the dependencies. We shall use the term "chase" interchangeably for both the procedure and its result. The chase works on an instance through the *TGD chase rule*.

TGD Chase Rule: Consider a database D for a schema $\mathcal{R}$, and a TGD $\sigma : \phi(\boldsymbol{X}, \boldsymbol{Y}) \rightarrow \exists \boldsymbol{Z} \psi(\boldsymbol{X}, \boldsymbol{Z})$ over $\mathcal{R}$. If σ is *applicable* to D, i.e., there exists a homomorphism h such that $h(\phi(\boldsymbol{X}, \boldsymbol{Y})) \subseteq D$, then:

- define $h' \supseteq h$ such that $h'(Z_i) = z_i$, for each $Z_i \in \boldsymbol{Z}$, where $z_i \in \Delta_z$ is a "fresh" labelled null not introduced before in the chase procedure, and
- add to D the set of atoms in $h'(\psi(\boldsymbol{X}, \boldsymbol{Z}))$, if there does not exist $h'' \supseteq h$ such that $h''(\psi(\boldsymbol{X}, \boldsymbol{Z}))$ is already in D.

Given a database D and a set of TGDs Σ, the chase algorithm for D and Σ consists of a repeated application of the TGD chase rule until a fixpoint is reached. Its result is a (possibly infinite) chase for D and Σ, denoted as $chase(D, \Sigma)$ [28]. The (possibly infinite) chase for D and Σ is a *universal model* of D with respect to Σ, i.e., for each instance $I \in mods(D, \Sigma)$, there exists a homomorphism from $chase(D, \Sigma)$ to I [44]. From this it can be shown that $D \cup \Sigma \models q$ if and only if $chase(D, \Sigma) \models q$, for every boolean conjunctive query q [28].

The TGD chase rule given above is known as *restricted* [44], since it checks whether the TGD under consideration is already satisfied, that is, it adds atoms to the given

instance only if necessary. The version of the TGD chase rule that applies looser criteria to the applicability of TGDs, with the aim of adding atoms to the given instance even if not necessary, is called *oblivious* [44]. In this thesis, we adopt the restricted version, since the termination argument given in Theorem 3.1 relies on the fact that the restricted chase is used.

Example 2.1.4. Consider the database I and the set of TDGs Σ in Example 2.1.3. We have seen that I does not satisfy the TGD σ_2. We can apply the chase procedure to extend I with respect to the TDGs in Σ so that the resulting database does satisfy Σ. As we said above, σ_1 is satisfied in I so it does not generate any modification to the database during the chase procedure. By applying the TGD Chase Rule to the TGD σ_2 we find that σ_2 is applicable to I with the homomorphism $h := \{X \rightarrow a, Y \rightarrow b, Z \rightarrow c\}$, with $h(body(\sigma_2)) = \{r(a, b, c)\} \subseteq I$. As there does not exist any homomorphism $h'' \supseteq h$ such that $h''(head(\sigma_2)) \subseteq I$, we define a homomorphism $h' \supseteq h$ such that $h' = \{X \rightarrow a, Y \rightarrow b, Z \rightarrow c, W \rightarrow \xi\}$, where $\xi \in \Delta_z$ is a "fresh" labelled null. Then, we add the set of atoms $h'(head(\sigma_2)) = \{t(b, b, \xi)\}$ to I. At this point, I is $\{t(a, b, c), t(a, a, b), r(a, b, c), t(b, b, \xi)\}$ and satisfies Σ. The chase procedure therefore terminates as a fixpoint has been reached.

Query rewriting under TGDs. CQ answering over TGDs can be achieved, in certain cases, by a technique called *query rewriting*. In query rewriting, starting from a given query q and a database I, a new query q' is computed according to a set of TGDs Σ, such that the answers to q over I and Σ are obtained by evaluating q' over I only; it is said that q is *rewritten* into q' and that q' is the *perfect rewriting of q with respect to Σ* [54]. The language of q', called the *target language*, can be more expressive than that of q.

Example 2.1.5 (Query rewriting)**.** Let us consider the TGDs

$$\sigma_1 \coloneqq c(X) \rightarrow a(X)$$
$$\sigma_2 \coloneqq a(X) \rightarrow \exists Y\ s(X,Y), c(Y),$$

the database $D \coloneqq \{a(A_c)\}$ and the query $q \coloneqq ans_q(X) \leftarrow s(X,Y)$. Following the chase procedure, we obtain a universal model $J = \{a(A_c), s(A_c,Z_0), c(Z_0)\}$, where Z_0 is a labelled null. Then we have that the answer to q over the database I and the TGDs $\{\sigma_1,\sigma_2\}$ is $\{ans_q(A_c)\}$, since $s(A_c,Z_0) \in J$. Now, let us rewrite q into the query $q' \coloneqq ans_{q'}(X) \leftarrow a(X) \cup s(X,Y)$; intuitively, q' captures the fact that, to search for terms that appear in an atom of the form $s(t_1,t_2)$ at position $s[1]$, we need also to consider atoms of the form $a(t)$, because the TGDs might infer the former from the latter atom. The evaluation of q' on I returns the correct answer, $\{ans_{q'}(A_c)\}$, without the need to materialise the universal solution J. Note that, in this case, the target language is more expressive than that of the original query, since we obtain a union of CQs as the perfect rewriting of the original CQ.

Several works have addressed rewriting of CQ under TGDs. [30, 31] introduced sets of TDGs, namely *sticky* sets, that enjoy the property of being *FO-rewritable*, i.e., for every query that needs to be evaluated under such dependencies it is possible to compute a first-order query, and thus a SPARQL 1.0 query, as a perfect rewriting.

Stickiness is a sufficient syntactic condition that ensures the so-called sticky property of the chase, which is as follows. For every instance D, assume that during the chase of D under a set Σ of TGDs, we apply a TGD σ that has a variable V appearing more than once in its body; assume also that V maps (via a homomorphism) onto the constant z, and that by virtue of this application the atom a is generated by the chase step. In this case, for each atom b in the body of σ, we say that a is derived from b. Then, we have that z appears in a, and in all atoms resulting from

some chase derivation sequence starting from a, "sticking" to them (hence the name "sticky sets of TGDs").

The formal definition of sticky sets of TGDs, given in [31], is an efficient testable condition involving variable marking.

Definition 2.1 ([30, 31])**.** Consider a set Σ of TGDs over a relational alphabet $\mathcal{R}$. A *position* $r[i]$ in $\mathcal{R}$ is identified by the predicate $r \in \mathcal{R}$ and its i-th argument (or attribute). We mark the variables that occur in the body of the TGDs of Σ according to the following procedure. First, for each TGD $\sigma \in \Sigma$ and for each variable V in $body(\sigma)$, if there exists an atom a in $head(\sigma)$ such that V does not appear in a, then we mark each occurrence of V in $body(\sigma)$. Now, we apply exhaustively (i.e., until a fixpoint is reached) the following step: for each TGD $\sigma \in \Sigma$, if a marked variable in $body(\sigma)$ appears at a position π, then for every TGD $\sigma' \in \Sigma$ (including the case $\sigma' = \sigma$), we mark each occurrence of the variables in $body(\sigma')$ that appear in $head(\sigma')$ at the same position π. We say that Σ is *sticky* if and only if there is no TGD $\sigma \in \Sigma$ such that a marked variable occurs in $body(\sigma)$ more than once.

Two decidable classes of TGDs that do not enjoy the FO-rewritability property are *Weakly-acyclic* and *Weakly-guarded* sets of TGDs [51, 27]. We introduce the definitions of weakly-acyclic and weakly-guarded TGDs, which are drawn from [51] and [27], respectively.

Definition 2.2 (Weakly acyclic set of TGDs)**.** Let Σ be a set of TGDs over a relational alphabet $\mathcal{R}$. Construct a directed graph, called the *dependency graph*, as follows: (1) there is a node for every position $r[i]$ in $\mathcal{R}$; (2) add edges as follows: for every TGD $\phi(\boldsymbol{X}, \boldsymbol{Z}) \rightarrow \exists \boldsymbol{Y}\ \psi(\boldsymbol{X}, \boldsymbol{Y})$ in Σ and for every X in $\boldsymbol{X}$:

- For every occurrence of X in ϕ in position $r[i]$:
  - (a) for every occurrence of X in ψ in position $s[j]$, add an edge $r[i] \rightarrow s[j]$ (if it does not already exist);

(b) in addition, for every existentially quantified variable Y and for every occurrence of Y in ψ in position $t[k]$, add a special edge $r[i] \xrightarrow{*} t[k]$ (if it does not already exist).

Note that there may be two edges in the same direction between two nodes, if exactly one of the two edges is special. Then Σ is weakly acyclic if the dependency graph has no cycle going through a special edge.

Definition 2.3 (Weakly-guarded sets of TGDs)**.** To define weakly guarded sets of TGDs, we first give the notion of an *affected position* in a predicate of a relational schema, given a set of TGDs Σ. Intuitively, a position π is affected in a set of TGDs Σ if there exists a database D such that a labelled null appears in some atom of $chase(D, \Sigma)$ at position π. Given a relational schema R and a set of TGDs Σ over R, a position π of a predicate p of R is affected with respect to Σ if either:

- (base case) for some $\sigma \in \Sigma$, an existentially quantified variable appears in π in $head(\sigma)$, or
- (inductive case) for some $\sigma \in \Sigma$, the variable appearing at position π in $head(\sigma)$ also appears in $body(\sigma)$, and only at affected positions.

Consider a set of TGDs Σ on a schema R. A TGD $\sigma \in \Sigma$ is said to be *weakly guarded* with respect to Σ if there is an atom in $body(\sigma)$, called a weak guard, that contains all the universally quantified variables of σ that appear in affected positions with respect to Σ and do not also appear in non-affected positions with respect to Σ. The set Σ is said to be a weakly guarded set of TGDs if each TGD $\sigma \in \Sigma$ is weakly guarded with respect to Σ.

2.1.2 RDF Databases

RDF and SPARQL Syntax

To formalise our peer-to-peer RDF integration system in Chapter 3, we first need to define the notion of *basic graph pattern* (BGP) SPARQL queries[2] over RDF databases. The following formalisations of RDF databases and BGP SPARQL queryes are drawn from [56].

Assume there are pairwise disjoint infinite sets I, B, and L (of IRIs [50], Blank nodes, and Literals, respectively). A triple $(s, p, o) \in (I \cup B) \times I \times (I \cup B \cup L)$ is called an *RDF triple*. In this triple, s is the *subject*, p the *predicate*, and o the *object*. An *RDF database* (also called an *RDF graph*) is a set of RDF triples. Given an RDF database D, we denote by $dom(D)$ the set of *identified resources*, that is, the elements in $(I \cup L)$ that occur in D.

Here, RDF databases are sets of triples, which by definition do not contain duplicate entries. However, RDF database management systems may implement a variation of this model where RDF sources are *bags* (or multisets) of triples, with duplicates allowed [4]. Systems that are based on bags rather than sets are said to implement *bag semantics*, rather than *set semantics*. We adopt set semantics to harmonise with the relational model, where relations are sets of tuples.

Assume also the existence of an infinite set of variables V disjoint from $(I \cup B \cup L)$. A *basic graph pattern* is defined recursively as follows:

1. A tuple from $(I \cup L \cup V) \times (I \cup V) \times (I \cup L \cup V)$ is a *basic graph pattern*. Specifically, it is a *triple pattern*.

2. If P_1 and P_2 are basic graph patterns, then the expression $(P_1$ AND $P_2)$ is a *basic graph pattern*.

[2] `https://www.w3.org/TR/rdf-sparql-query/#BasicGraphPatterns`

We denote by $var(P)$ the set of variables $V_P \subseteq V$ that appear in the basic graph pattern P.

A *basic graph pattern (BGP) SPARQL query* q of arity n is of the form

$$\boldsymbol{X} \leftarrow P_q$$

where P_q is a basic graph pattern, and $\boldsymbol{X} = x_1, \ldots, x_n \in var(P_q)$ denote the *free variables* of q. All the elements in $var(P_q)$ that are not free variables are the *existentially quantified variables* of q.

SPARQL Semantics

We now discuss the semantics of BGP SPARQL queries. The following formalisation is drawn from [89, 21] which defines the evaluation of a basic graph patterns over an RDF database.

A *mapping* μ from V to $(I \cup B \cup L)$ is a partial function $\mu : V \rightarrow (I \cup B \cup L)$. The domain of μ, denoted by $dom(\mu)$, is the subset of V on which μ is defined. Given a mapping μ and a variable $v \in dom(\mu)$, $\mu(v)$ denotes the value in $(I \cup B \cup L)$ obtained by applying μ to v. For a triple pattern t, $\mu(t)$ denotes the triple obtained by replacing all variables v in t according with $\mu(v)$. Two mappings μ_1 and μ_2 are *compatible* if for all $x \in dom(\mu_1) \cap dom(\mu_2)$, it is the case that $\mu_1(x) = \mu_2(x)$.

Let Ω_1 and Ω_2 be sets of mappings. Then the *join* of Ω_1 and Ω_2 is defined as follows:

$$\Omega_1 \bowtie \Omega_2 = \{\mu_1 \cup \mu_2 \mid \mu_1 \in \Omega_1, \mu_2 \in \Omega_2 \text{ and } \mu_1, \mu_2 \text{ are compatible mappings}\}$$

The semantics of basic graph patterns are defined through a function $[\![\cdot]\!]_D$ over an RDF database, which takes a basic graph pattern as input and returns a set of mappings defined as follows [21, 89].

Definition 2.4. (Evaluation of a basic graph pattern). The evaluation of a basic graph pattern P over an RDF database D, denoted by $[\![P]\!]_D$, is defined recursively as follows:

1. If P is a triple pattern t, then $[\![P]\!]_D = \{\mu \mid dom(\mu) = var(t) \text{ and } \mu(t) \in D\}$
2. If P is of the form (P_1 AND P_2), then $[\![P]\!]_D = [\![P_1]\!]_D \bowtie [\![P_2]\!]_D$.

Given a BGP query q of arity n of the form $\boldsymbol{X} \leftarrow P_q$, where $\boldsymbol{X} = x_1, \ldots x_n$, we denote by q^D the set of n-tuples returned by the evaluation q over D, where:

$$q^D := \{(\ \mu(x_1), \ldots, \mu(x_n)) \mid \mu \in [\![P_q]\!]_D\ \}.$$

RDFS and OWL

RDF Schema (*Resource Description Framework Schema*, variously abbreviated as *RDFS*, *RDF(S)*, *RDF-S*, or *RDF/S*) is a set of classes and properties expressed in RDF, providing basic elements for the definition of ontologies (also called RDF vocabularies) with the aim of adding semantics to RDF resources [3, 103]. Ontologies are a formal way to describe taxonomies and classification networks, essentially defining the structure of knowledge for various domains.

The data described by an ontology is interpreted as a set of "individuals" and a set of "property assertions" which relate these individuals to each other. An ontology consists of a set of axioms which place constraints on sets of individuals (called "classes") and the types of relationships permitted between them. These axioms provide semantics by allowing systems to infer additional information based on the data explicitly provided[3]. Ontologies can import other ontologies, adding information from the imported ontology to the current ontology.

[3] `https://www.w3.org/TR/owl-guide/`

Here we illustrate some of the classes and properties of the RDF Schema 1.1 specification of the W3C[4]:

Classes. `rdfs:Resource` is the class of everything. All entities described by RDF are resources. `rdfs:Class` declares a resource to be a class of other resources. An example of an `rdfs:Class` is `foaf:Person` in the *Friend of a Friend*[5] (FOAF) vocabulary. An instance of `foaf:Person` is a resource that is linked to the class `foaf:Person` using the `rdf:type property`, such as in the following RDF representation of the natural-language sentence 'John is a Person':

```
ex:John rdf:type foaf:Person
```

Here, `rdf:type` is a property used to state that a resource is an instance of a class. A commonly used abbreviation for this property is "`a`". `rdf:`, `foaf:` and `ex:` are namespace prefixes.

The definition of `rdfs:Class` is recursive: `rdfs:Class` is also the class of classes, and so it is an instance of itself:

```
rdfs:Class rdf:type rdfs:Class
```

Properties. Properties are instances of the class `rdf:Property` and describe a relationship between subject resources and object resources. When used as such, a property is regarded as being a predicate.

The property `rdfs:domain` of an `rdf:Property`, P, identifies the class of the subject of any triple whose predicate is P. The property `rdfs:range` of an `rdf:Property`, P, identifies the class or datatype of the object of any triple whose predicate is P. For example, the following declarations are used to express that the property `ex:employer` relates a subject of type `foaf:Person` to an object of type `foaf:Organization`:

[4] `https://www.w3.org/TR/rdf-schema/`
[5] `http://xmlns.com/foaf/spec/`

```
ex:employer rdfs:domain foaf:Person

ex:employer rdfs:range foaf:Organization
```

Given the previous two declarations, the following triple requires that `ex:John` is necessarily of type `foaf:Person`, and `ex:CompanyX` is necessarily of type `foaf:Organization`:

```
ex:John ex:employer ex:CompanyX
```

`rdfs:subClassOf` allows the declaration of hierarchies of classes. For example, the following declares that 'Every Person is an Agent':

```
foaf:Person rdfs:subClassOf foaf:Agent
```

`rdfs:subPropertyOf` is an instance of `rdf:Property` that is used to state that all resources related by one property are also related by another. `rdfs:label` is an instance of `rdf:Property` that may be used to provide a human-readable version of a resource's name. `rdfs:comment` is an instance of `rdf:Property` that may be used to provide a human-readable description of a resource.

The *Web Ontology Language* (OWL) is a family of knowledge representation languages for defining ontologies[6] that provide more expressiveness than RDF/S. OWL has a richer vocabulary which enables a more precise description of properties and classes. For instance, an OWL ontology describing families might include axioms stating that a "hasMother" property is only present between two individuals when "hasParent" is also present, and that individuals of the class "HasTypeOBlood" are never related via "hasParent" to members of the class "HasTypeABBlood". Thus, if it is stated that the individual Harriet is related via "hasMother" to the individual Sue, and that Harriet is a member of "HasTypeOBlood", then it can be inferred that Sue is not a member of "HasTypeABBlood".

[6] `https://www.w3.org/OWL/`

2.1.3 Graph Databases

Description Logics

We now introduce Description Logics (DLs) [7], a family of knowledge representation languages that further generalise OWL. The name *description logics* is motivated by the fact that the important notions of the domain are described by concept *descriptions*, i.e., expressions that are built from atomic concepts (unary predicates) and atomic roles (binary predicates) using the concept and role constructors provided by the particular DL. Here we focus on the DL $\mathcal{ALCHI}$ [7], from which the logics that we address in this thesis are derived.

Syntax The alphabet of $\mathcal{ALCHI}$ contains three pairwise disjoint and countably infinite sets of *concept names* A, *role names* R, and *individual names* I. The alphabet of $\mathcal{ALCHI}$ also contains a set of *roles* P, such that each $P \in \mathsf{P}$ is either a role name in R or its *inverse*, denoted by R^-. A *complex concept* C is constructed from concept names, roles and two special primitive concepts $\top$ ('top'), and $\bot$ ('bottom'), using the following grammar:

$$C ::= A \mid \top \mid \bot \mid \neg C \mid C_1 \sqcap C_2 \mid C_1 \sqcup C_2 \mid \exists P.C \mid \forall P.C,$$

where $A \in \mathsf{A}$ and $P \in \mathsf{P}$. Note that a DL vocabulary can be seen as a restricted FO vocabulary containing only unary predicates (concept names), binary predicates (role names), and constants (individual names). The set of complex concepts is denoted by C. An example of a complex concept is given by $\exists R.A \sqcap B$ which denotes all the elements of type B that are also connected to elements of type A via the role R.

An $\mathcal{ALCHI}$ *terminological box* (or TBox) $\mathcal{T}$ is a finite set of *concept* and *role*

inclusion axioms (or simply *inclusions*) of the form

$$C_1 \sqsubseteq C_2 \quad \text{and} \quad P_1 \sqsubseteq P_2,$$

where $C_1, C_2 \in \mathsf{C}$ and $P_1, P_2 \in \mathsf{P}$. An $\mathcal{ALCHI}$ *assertion box* (or ABox) $\mathcal{A}$ is a finite set of *concept* and *role assertions* of the form

$$A(a) \quad \text{and} \quad R(a, b),$$

where $A \in \mathsf{A}$, $R \in \mathsf{R}$ and $a, b \in \mathsf{I}$. Given an ABox $\mathcal{A}$, we denote by $\mathsf{ind}(\mathcal{A})$ the set of individual names that occur in $\mathcal{A}$. Taken together, $\mathcal{T}$ and $\mathcal{A}$ comprise a *knowledge base* (or KB) $\mathcal{K} = (\mathcal{T}, \mathcal{A})$. In a KB, terminological boxes specify general properties of concepts and roles, and constrain the way all objects in the domain can participate in the different concepts and roles; on the other hand, assertion boxes are facts about specific objects in the domain, that is, they assert that an individual participates in some concept, or that some role holds between a pair of individuals.

Semantics We adopt the semantics of DL defined in terms of interpretations [7]. An *interpretation* $\mathcal{I}$ is a pair $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ that consists of a non-empty *domain of interpretation* $\Delta^{\mathcal{I}}$ and an *interpretation function* $\cdot^{\mathcal{I}}$. The function $\cdot^{\mathcal{I}}$ assigns an element $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$ to each individual name a, a subset $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ to each concept name A, and a binary relation $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ to each role name R. We adopt the *unique name assumption* (UNA), whereby distinct individuals are assumed to be interpreted by distinct domain elements. The interpretation function $\cdot^{\mathcal{I}}$ is extended inductively for

complex concepts by taking:

$$\begin{aligned}
(R^-)^{\mathcal{I}} &= \{(v,u) \mid (u,v) \in R^{\mathcal{I}}\},\\
\top^{\mathcal{I}} &= \Delta^{\mathcal{I}},\\
\bot^{\mathcal{I}} &= \varnothing,\\
(\neg C)^{\mathcal{I}} &= \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}},\\
(C_1 \sqcap C_2)^{\mathcal{I}} &= C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}},\\
(C_1 \sqcup C_2)^{\mathcal{I}} &= C_1^{\mathcal{I}} \cup C_2^{\mathcal{I}},\\
(\exists P.C)^{\mathcal{I}} &= \{u \mid \text{there is a } v \in C^{\mathcal{I}} \text{ such that } (u,v) \in P^{\mathcal{I}}\},\\
(\forall P.C)^{\mathcal{I}} &= \{u \mid \text{for all } v \text{ with } (u,v) \in P^{\mathcal{I}}, v \in C^{\mathcal{I}}\}.
\end{aligned}$$

We now define the satisfaction relation $\models$ for inclusions and assertions:

$$\begin{aligned}
\mathcal{I} \models C_1 \sqsubseteq C_2 &\quad \text{if and only if} \quad C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}\ ,\\
\mathcal{I} \models P_1 \sqsubseteq P_2 &\quad \text{if and only if} \quad P_1^{\mathcal{I}} \subseteq P_2^{\mathcal{I}}\ ,\\
\mathcal{I} \models C(a) &\quad \text{if and only if} \quad a^{\mathcal{I}} \in C^{\mathcal{I}}\ ,\\
\mathcal{I} \models P(a,b) &\quad \text{if and only if} \quad (a^{\mathcal{I}}, b^{\mathcal{I}}) \in P^{\mathcal{I}}\ .
\end{aligned}$$

We say that an interpretation $\mathcal{I}$ is a *model* of a knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, written $\mathcal{I} \models \mathcal{K}$, if it satisfies all concept and role inclusions of $\mathcal{T}$ and all concept and role assertions of $\mathcal{A}$.

Example 2.1.6. We give here an example of an $\mathcal{ALCHI}$ knowledge base. We consider the domain of drinks and menus offered by bars. The vocabulary we use to model this domain contains concept names for drink items, such as PinaColada or CoconutMilk, and for more general types of drinks, such as SugarFree options (SugarFree) or hot drinks (HotDrink). We also use concept names for entities such as Bar, Menu and Drink. The role name hasIngredient is used to relate drinks and their ingredients. The role contains is a generalisation (or *superrole*) of hasIngredient that can also relate drinks with components (such as sugar or alcohol) that would

not typically be considered as ingredients. The role name hasCocktail relates menus to the drinks they contain as cocktails, and we may also have specialised versions of this role such as hasBeer and hasWine. We can also use role names to state that a bar offers some menu, or that it serves a drink. For entities representing specific menus, dishes and bars, we use italic, lower-case letters. With this vocabulary in place, we can write ABox assertions such as::

$\mathsf{offers}(r, m)$	$\mathsf{hasCocktail}(m, d_1)$	$\mathsf{PinaColada}(p_2)$
$\mathsf{hasBeer}(m, p_1)$	$\mathsf{Bellini}(d_1)$	$\mathsf{serves}(r, d_2)$
$\mathsf{Beer}(p_1)$	$\mathsf{serves}(r, p_2)$	$\mathsf{GinTonic}(d_2)$

which intuitively express that some bar (r) offers a menu (m) containing beer and Bellini cocktail, and the bar r also serves Pina Colada and Gin Tonic. Below we give some examples of TBox concept and role inclusion axioms that express general knowledge about this domain. Here $C \equiv D$ is shorthand for the pair of axioms $C \sqsubseteq D$ and $D \sqsubseteq C$.

$$\exists \mathsf{hasDrink}.\top \sqsubseteq \mathsf{Menu} \tag{2.1}$$

$$\exists \mathsf{hasDrink}^{-}.\top \sqsubseteq \mathsf{Drink} \tag{2.2}$$

$$\mathsf{hasCocktail} \sqsubseteq \mathsf{hasDrink} \tag{2.3}$$

$$\mathsf{hasBeer} \sqsubseteq \mathsf{hasDrink} \tag{2.4}$$

$$\mathsf{Menu} \sqsubseteq \exists \mathsf{hasBeer}.\top \tag{2.5}$$

$$\mathsf{PinaColada} \sqsubseteq \mathsf{Cocktail} \sqcap \exists \mathsf{hasIngredient}.\mathsf{PineappleJuice} \tag{2.6}$$

$$\mathsf{GinTonic} \sqsubseteq \exists \mathsf{hasIngredient}.\mathsf{Gin} \tag{2.7}$$

$$\mathsf{PineappleJuice} \sqcup \mathsf{Cachaça} \sqsubseteq \exists \mathsf{contains}.\mathsf{Sugar} \tag{2.8}$$

$$\mathsf{SugarFree} \equiv \forall \mathsf{contains}.\neg \mathsf{Sugar} \tag{2.9}$$

$$\mathsf{hasIngredient} \sqsubseteq \mathsf{contains} \tag{2.10}$$

The concept inclusions (2.1) and (2.2) state respectively that the domain of hasDrink consists of menus, and its range consists of drinks. The role inclusions (2.3) and (2.4) express that hasCocktail and hasBeer are specialisations (or *subroles*) of the role hasDrink. Concept inclusion (2.5) stipulates that a menu has at least one beer. Axiom (2.6) states that Pina Colada is a kind of cocktail that has an ingredient pineapple juice. Axiom (2.7) says that Gin Tonic has gin as an ingredient. Axiom (2.8) states that pineapple juice and Cachaça all contain sugar, and axiom (2.9) defines sugar-free as the class of entities not containing sugar. Finally, the role inclusion (2.10) expresses that hasIngredient is a subrole of contains.

Regular Languages and Conjunctive Regular Path Queries

A *non-deterministic finite state automaton* (NFA) over a set of symbols Σ is a tuple $\alpha = (Q, \Sigma, \delta, q_0, F)$, where Q is a finite set of *states*, $\delta \subseteq Q \times \Sigma \times Q$ is the *transition relation*, $q_0 \in Q$ is the *initial state*, and $F \subseteq Q$ is the set of *final states*. $L(\alpha)$ denotes the language defined by an NFA α, and Σ^* denotes the set of all strings over symbols in Σ, including the *empty string* ϵ. A language that is recognised by a NFA is a *regular language* [14].

To define the queries below, it is assumed that there exists a countably infinite set of *variables* V, *individual names* I, *concept names* A and *role names* R. The alphabet also contains a set of *roles* P, such that each $P \in \mathsf{P}$ is either a role name in R or its *inverse*, denoted by R^-. A *term* t is an individual name in I or a variable in V. An *atom* is of the form $\alpha(t, t')$, where t, t' are terms, and α is an NFA or regular expression defining a regular language over $\mathsf{P} \cup \mathsf{A}$. A string $s \in (\mathsf{P} \cup \mathsf{A})^*$ is a *path*.

A *conjunctive two-way regular path query* (C2RPQ) $\boldsymbol{q}$ of arity n has the form $q(\vec{x}) \leftarrow \exists \vec{y}\ \gamma(\vec{x}, \vec{y})$, where $\vec{x} = x_1, \ldots, x_n$ and $\vec{y} = y_1, \ldots, y_m$ are tuples of variables, and $\gamma(\vec{x}, \vec{y})$ is a conjunction of atoms with variables from $\vec{x}$ and $\vec{y}$ [38]. Atom $q(\vec{x})$ is the

head of $\boldsymbol{q}$, denoted by $head(\boldsymbol{q})$, and $\gamma(\vec{x}, \vec{y})$ is the *body* of $\boldsymbol{q}$, denoted by $body(\boldsymbol{q})$. The variables in $\vec{x}$ are the *answer variables* of $\boldsymbol{q}$, while those in $\vec{y}$ are the *existentially quantified variables* of $\boldsymbol{q}$. A *conjunctive (one-way) regular path query* (CRPQ) is obtained by allowing only symbols from $\mathsf{R} \cup \mathsf{A}$ (i.e., disallowing role inverses) in atoms. A *Boolean C(2)RPQ* is a C(2)RPQ with no answer variables. A *two-way regular path query* (2RPQ) is a C2RPQ with a single atom in its body. A *regular path query* (RPQ) is a CRPQ with a single atom in its body. A *two-way path query* (2PQ) is a 2RPQ $head(\boldsymbol{q}) \leftarrow \alpha(x, y)$ such that $\alpha \in (\mathsf{P} \cup \mathsf{A})^*$. A *path query* (PQ) is an RPQ $head(\boldsymbol{q}) \leftarrow \alpha(x, y)$ such that $\alpha \in (\mathsf{R} \cup \mathsf{A})^*$. In both the latter cases, α is called the *path* of $\boldsymbol{q}$, denoted by $\mathsf{path}(\boldsymbol{q})$.

A *conjunctive query* (CQ) $\boldsymbol{q}$ is a CRPQ such that, for each atom $\alpha(t, t') \in body(\boldsymbol{q})$, $\alpha \in (\mathsf{P} \cup \mathsf{A})$. Intuitively, a CQ has as body a conjunction of atoms whose predicates are in $\mathsf{A} \cup \mathsf{P}$ (without regular expressions). Given a C(2)RPQ $\boldsymbol{q}$ with answer variables $\vec{x} = x_1, \ldots, x_n$ and an n-tuple of individuals $\boldsymbol{a} = (a_1, \ldots, a_n)$, we use $\boldsymbol{q}(\boldsymbol{a})$ to refer to the Boolean C(2)RPQ obtained from $\boldsymbol{q}$ by replacing x_i with a_i in $body(\boldsymbol{q})$, for every $1 \leqslant i \leqslant n$. An *instance query* (IQ) takes one of the following two forms: *(i)* $q(x) \leftarrow A(x)$, where $A \in \mathsf{A}$ (*concept instance query*); or *(ii)* $q(x, y) \leftarrow P(x, y)$, where $P \in \mathsf{P}$ (*role instance query*).

Semantics of C2RPQs. We now define the semantics of C2RPQs [38]. Given individual names a and b, an interpretation $\mathcal{I}$, and a regular language α over the alphabet $\mathsf{P} \cup \mathsf{A}$, we say that b *α-follows* a in $\mathcal{I}$, denoted by $\mathcal{I} \models a \xrightarrow{\alpha} b$, if and only if there is some $w = u_1 \ldots u_n \in L(\alpha)$ and some sequence $e_0, \ldots, e_n$ with $e_i \in \Delta^{\mathcal{I}}$, $0 \leqslant i \leqslant n$, such that $e_0 = a^{\mathcal{I}}$ and $e_n = b^{\mathcal{I}}$, and for all $1 \leqslant i \leqslant n$: *(a)* if $u_i = A \in \mathsf{A}$, then $e_{i-1} = e_i \in A^{\mathcal{I}}$; *(b)* if $u_i = P \in \mathsf{P}$, then $(e_{i-1}, e_i) \in P^{\mathcal{I}}$. A *match* for a Boolean C2RPQ $\boldsymbol{q}$ in an interpretation $\mathcal{I}$ is a mapping π from the terms in $body(\boldsymbol{q})$ to the elements in I such that:

(a) $\pi(c) = c$ if $c \in \mathsf{I}$;

(b) $\mathcal{I} \models \pi(t) \xrightarrow{\alpha} \pi(t')$ for each atom $\alpha(t, t')$ in $\boldsymbol{q}$.

To simplify the notation, we do not allow unary atoms in the body of the query, since each atom of the form $A(t)$, where $A \in \mathsf{A}$ and $t \in \mathsf{V} \cup \mathsf{I}$, can be always replaced by a binary atom $A(t, z)$, where z is a fresh variable (that is, newly invented and not appearing elsewhere). However, we shall use unary atoms in some examples throughout the thesis, whenever this improves the legibility. It is easy to see that a query with all unary atoms replaced by binary atoms as above is equivalent to the original query. Given an interpretation $\mathcal{I}$ and a C2RPQ q, we say that $\mathcal{I} \models \boldsymbol{q}$ if there is a match for $\boldsymbol{q}$ in $\mathcal{I}$, and that $\mathcal{K} \models \boldsymbol{q}$ if $\mathcal{I} \models \boldsymbol{q}$ for every model $\mathcal{I}$ of the KB $\mathcal{K}$. Also, we use $\boldsymbol{q}^{\mathcal{I}}$ to denote:

$$\boldsymbol{q}^{\mathcal{I}} := \{t \mid \mathcal{I} \models \boldsymbol{q}(t)\}$$

and $\boldsymbol{q}^{\mathcal{K}}$ to denote:

$$\boldsymbol{q}^{\mathcal{K}} := \{t \mid t \in \boldsymbol{q}^{\mathcal{I}} \text{ for every model } \mathcal{I} \text{ of } \mathcal{K}\}.$$

Given an ABox $\mathcal{A}$, we use $\mathcal{A} \models \boldsymbol{q}$ as a shorthand for $(\varnothing, \mathcal{A}) \models \boldsymbol{q}$, where $(\varnothing, \mathcal{A})$ is a knowledge base with an empty TBox.

Given a C2RPQ $\boldsymbol{q}$ of arity n, a tuple of individual names $\boldsymbol{a} = (a_1, \ldots, a_n)$ is a *certain answer* for $\boldsymbol{q}$ with respect to a KB $\mathcal{K}$ if and only if $\mathcal{K} \models \boldsymbol{q}(\boldsymbol{a})$.

2.2 Related Work

2.2.1 Frameworks for SPARQL query rewriting

Several works in the literature address data integration via SPARQL query rewriting. Very close to our work is [42] which proposes an algorithm to rewrite SPARQL queries with the aim of achieving RDF data integration. The approach is based on the encoding of rewriting rules called *Entity Alignments*, which express semantic mappings between two datasets and can be interpreted as definite *Horn clauses*[7] in First-Order (FO) logic where only the *triple* predicate is used. The rewriting is based on the GAV data integration approach. The main limitation of this work is that it is not applicable when a more expressive formalism is needed to map between two schemas, for example, when the relations in the sources need to be specified as views over the mediated schema, i.e., as LAV mappings. One interesting aspect is that the framework deals with *entity resolution* - the task of disambiguating manifestations of real world entities in various records or mentions - by including functional dependencies in the mapping rules.

Other SPARQL rewriting approaches are proposed by Makris et al. [80, 81] and in the work of Thiéblin et al. [104], who define mapping frameworks based on Description Logics, where a term from one vocabulary is mapped to a Description Logic expression over another vocabulary. These *one-to-many* mapping approaches can support either a GAV or a LAV formalism, that is, a combination of GAV and LAV rules are not permitted in the same setting. A similar limitation is found the work of Lopes et al. [77] who adopt a rule-based mapping language in which single atoms are mapped to a conjunction of atoms. Other approaches on data mediation are discussed in [84, 98, 106]. Specifically, [84] proposes a LAV approached called *SemLAV*, an alternative technique to process SPARQL queries without gen-

[7] A clause (i.e., a disjunction of literals) is called a Horn clause if it contains at most one positive literal. Horn clauses are usually written as $\neg L_1 \vee \neg L_2 \vee \cdots \vee \neg L_n \vee L \equiv L_1, L_2, \ldots, L_n \rightarrow L$.

erating rewritings. SemLAV executes the query against a partial instance of the global schema which is built on-the-fly with data from the relevant views. Work in [98] addresses rewriting techniques that consider only co-reference resolution in the rewriting process. [106] adopts a small set of mapping axioms defined only by those RDF triples whose predicate is one of the following OWL or RDFS terms: `sameAs`, `subClassOf`, `subPropertyOf`, `equivalentClass`, and `equivalentProperty`. Other SPARQL rewriting approaches are [73], which is based on SPARQL views specifying GAV rules, and [97], which is limited to RDFS axioms as mapping language.

All the above-mentioned frameworks address query answering assuming a two-tiered schema architecture, while we wish to explore a peer-to-peer architecture to encompass the general case, where the mapping topologies are arbitrary.

2.2.2 Peer-to-peer Systems

Peer-to-peer systems based on the relational model have been widely addressed in the literature. We begin this section with a description of the logical model of the Piazza *Peer Data Management System* (PDMS) [58], to illustrate the commonly adopted approach of interpreting such systems using a first-order semantics, and we compare it with an alternative approach which relies on epistemic logics [36]. We then discuss the complexity of the query answering problem in PDMSs, restrictions to ensure decidability, and the relationship between these results and our own work, which is based on the RDF data model.

The Piazza PDMS [58, 102, 101, 57, 59] consists of a set of data sources (also known as *peers*) which are related through semantic mappings, also called *peer mappings*. Each peer defines its own relational peer schema whose relations are called *peer relations*. A query in a PDMS is posed over the relations of a specific peer schema.

Peers contribute data to the system in the form of *stored relations*. Peer relations

are mapped to stored relations via *storage descriptions*. All queries submitted to a peer will be reformulated in terms of stored relations that may be stored locally or at other peers.

The goal of the Piazza PDMS is to preserve the features of both the GAV and LAV formalisms, but also to extend them from a two-tiered architecture to a more general network of interrelated peer and source relations. Peer mappings in a PDMS are specified between pairs of peer relations. Ultimately, a query over a given peer schema may be reformulated over source relations stored at any peer reachable from the transitive closure of the peer mappings.

We now discuss the complexity of the query answering problem in the Piazza PDMS. Two extreme cases of query answering are characterised [58]:

1. The problem of answering a conjunctive query, Q, for a given PDMS, N, is undecidable.

2. If N includes only acyclic mappings, then a conjunctive query can be answered in polynomial time with respect to data complexity.

These results are based on an open-world assumption in which peers have incomplete rather than full information. The closed-world assumption, which is necessary for supporting negation in mappings and queries, is known to make the problem of finding all certain answers much harder (co-NP hard in the size of the data [1]) even for two peers (and, in fact, even without negation). The proof of these results is in [60].

There is also an additional special case where query answering in a PDMS is tractable and two more cases when it is decidable. Note that, in general, decidability is ensured only when the peer mappings are acyclic.

Similarly to the Piazza PDMS, in [52] the authors give logical and computational characterisations of peer-to-peer database systems. Their formalisation is *robust* since, unlike other formalisations, it allows for local inconsistencies in some node of the P2P network: if some rule in the local database schema is not satisfied it will not result in the entire database being inconsistent. They define a model-theoretic semantics of a peer-to-peer system which allows for local inconsistency handling, and then characterise the general computational properties for the problem of answering queries in such a peer-to-peer system.

A special case of the PDMS framework, called *peer data exchange* is addressed in [53]; in this framework, peer mappings can be defined in a single direction, that is, from a *source* peer to a *target* peer. Instead, in a full-fledged PDMS such distinction of peers does not exist, and mappings can be expressed in either direction (from one peer to another, and vice versa).

Calvanese and De Giacomo [36] compare the commonly adopted approach of interpreting peer-to-peer systems using a first-order logic with an alternative approach based on epistemic semantics which, in their setting, can be considered as a well-behaved, sound approximation of the first-order semantics where only the certain answers are exported from peer to peer. They show that in systems in which peer mappings are arbitrarily interconnected, the first-order approach may lead to undecidability of query answering, while the epistemic approach always preserves decidability.

In [96] is outlined a major distinction between PDMS frameworks, according to whether peer mappings are interpreted under *global* or *local reasoning*. Global reasoning means that peer mappings are interpreted as a single (global) first-order theory. Under local reasoning, each peer is modeled as a distinct (local) theory, and inter-peer mappings are interpreted as exchanging certain facts between such theories only. Of the approaches mentioned above, the work of Calvanese and De

Giacomo is an example of PDMS interpreted under local reasoning, as opposed to the Piazza PDMS which is under global reasoning.

Several P2P systems for the RDF data model have been proposed in the literature. For instance, in [23, 24] the authors describe a distributed RDF metadata storage, querying and subscription service, as a structured P2P network. Similarly, work in [88] proposes routing strategies for RDF-based P2P networks. Similar approaches are described in [86, 87, 71, 105, 100]. However, these are non-database-oriented tools that have little support for semantic integration of highly heterogeneous RDF data.

We conclude this section with a brief overview of other papers on peer-to-peer (P2P) data integration for the relational model that we have reviewed for this thesis. [82, 83] extend an existing approach to data integration, called both-as-view, to be an effective mechanism for defining peer-to-peer integration at the schema level. [13] identifies particular database problems introduced by P2P computing and proposes the Local Relational Model (LRM) to solve some of them. [69] addresses semantic and algorithmic issues related to the use of mapping tables, arguing that mapping tables are appropriate for data mapping in a P2P environment and then discussing alternative semantics for these tables. [95] presents a decentralised strategy that guides peers in their decision over which further mappings and data a query should be executed. The strategy uses statistics of the peer's own data and statistics of mappings to neighbouring peers to predict whether it is worthwhile to send or prune the query at a specific point in the peer network. [107] presents a distributed solution to process and optimise queries in a PDBMS for multi-way join queries. This approach first processes a multi-way join query based on an initial query evaluation plan (generated using statistical data that may be obsolete or inaccurate); as the query is being processed, statistics obtained on-the-fly are used to refine the current plan dynamically into a more effective one.

2.2.3 Query Rewriting Under Description Logics

Query rewriting has been extensively employed in conjunctive query answering under different types of ontologies [54, 90, 85]. The first query rewriting algorithm proposed in DLs is the one presented in [34] where the authors address tractable query rewriting (in NLOGSPACE) for conjunctive queries to unions of conjunctive queries under the FO-rewritable DL-Lite family. For the non-FO-rewritable $\mathcal{EL}$ family of languages [6], Rosati [94] uses a rewriting algorithm similar to the one in [34] to show that query answering in $\mathcal{EL}$ is PTIME-complete in data complexity. Instance checking (i.e., answering single-atom boolean queries) under DL-Lite$_\mathcal{R}$ via query rewriting is addressed in [17]. Another similar work is [91] which presents a resolution-based query rewriting algorithm for DL-Lite+ ontologies; in this case, a more expressive language, Linear Datalog [55], is adopted as the target language. Similar approaches are used when the ontology language is TGDs. For instance, in [9] the authors use a resolution-based mechanism to identify classes of tuple-generating dependencies for which reasoning tasks (such as conjunctive query answering or entailment) are decidable.

In [40] the authors address the problem of how to recover FO-rewritability of SPARQL lost because of `owl:sameAs` statements. Other works [15, 16] study FO-rewritability of conjunctive queries in the presence of ontologies formulated in a description logic lying between $\mathcal{EL}$ and Horn-$\mathcal{SHIF}$ (which subsumes $\mathcal{EL}$ in expressiveness), along with related query containment problems. In [63, 62] the authors propose an algorithm for computing FO rewritings of concept queries (i.e., single-atom queries where the atom relation is a concept name) under $\mathcal{EL}$ TBoxes that is tailored towards efficient implementation.

As we shall see in Chapter 4, the query rewriting algorithm adopted in this thesis is derived from that of [70, 72], which address query rewriting over $\mathcal{EL}$, $\mathcal{QL}$ and $\mathcal{RL}$, and propose techniques to rewrite conjunctive queries under $\mathcal{QL}$. Similarly

to our approach, the authors in [20] exploit the SPARQL 1.1 language in query rewriting. They find that SPARQL 1.1 is powerful enough to encode a schema-agnostic rewriting under $\mathcal{QL}$. Using additional SPARQL 1.1 features, they develop a new method of query rewriting, where arbitrary conjunctive queries over $\mathcal{QL}$ are rewritten into equivalent SPARQL 1.1 queries in a way that is fully independent of the actual schema.

The complexity of answering CRPQs under DL-Lite and $\mathcal{EL}$ families is studied in [18]. To the best of our knowledge, there are no works in the literature that address the rewriting of CQs (nor CRPQs) into CRPQs under DL ontologies, nor under other dependency rules for the relational model such as TGDs. In this regard, we present in Chapters 4 and 5 a novel rewriting technique, based on non-deterministic finite-state automata, for rewriting CQs into CRPQs under a fragment of the DL comprising DL-Lite+ of [91] plus the additional feature of inverse role inclusions.

2.3 Discussion

In this chapter we have introduced the necessary background theory and a discussion of the literature related to the work proposed in this thesis. This review of the literature raises some open questions, that are investigated in the following chapters of the thesis:

Q1 How do we formalise an RDF-based peer-to-peer architecture for SPARQL query answering where the number of peers and the mapping topologies are arbitrary?

Q2 How can we generate the rewriting of SPARQL queries with respect to the peer mappings, interpreted so as to preserve decidability?

Q3 Is the expressiveness of SPARQL sufficient for such rewritings?

Q4 In case that the above rewritings require a more expressive language than SPARQL (or SPARQL 1.1) a restriction on the mappings is needed to preserve decidability. How do we restrict the mapping language while keeping it as expressive as possible?

Q5 Which Description Logic is suitable as the mapping language according to the task of **Q4**?

Q1 and **Q2** are addressed in Chapter 3 where we we introduce our framework for peer-to-peer RDF semantic data integration, the RPS. Our goal is to leverage the techniques for specifying semantic mappings between RDF sources, extending them beyond a two-tiered architecture and preserving decidability of query answering with arbitrary mapping topologies.

Q3 is addressed in Section 3.5 of Chapter 3, where we discuss the rewritability of the mapping language and propose a rewriting algorithm that outputs a union of conjunctive queries for a restricted version of the mapping language and a Datalog query for the general case. Finally, **Q4** and **Q5** are addressed in Chapters 4 and 5, where we introduce the description logic $\mathcal{ELH}^{\ell in}$, and a novel ontology language, named $\mathcal{ELHI}_h^{\ell in}$, which strictly extends the known ontology languages DL-Lite$_\mathcal{R}$ and $\mathcal{ELH}^{\ell in}$. Specifically, in Chapter 5 we show that it is possible to rewrite CQs into C2RPQs (and therefore into SPARQL 1.1) under $\mathcal{ELHI}_h^{\ell in}$.

Chapter 3

Peer-to-Peer Semantic Integration of Linked Data

In this chapter, we introduce our framework for peer-to-peer RDF semantic data integration, the RDF Peer System (RPS). Our goal is to leverage techniques for specifying semantic mappings between RDF sources, extending them beyond a two-tiered architecture. In our framework, each peer is represented by its peer schema, comprising the set of IRIs adopted by the peer to model its data. Integration is achieved by means of peer-to-peer mappings between these sets of URIs. To formally specify the problem of query answering, we generalize the notion of *certain answers* [1] to our context. The chapter is structured as follows. In Section 3.1 we define the syntax of RPSs and in Section 3.2 we introduce their semantics. The problem of SPARQL query answering under RPSs is addressed in Section 3.3, where we show that the semantics of the mappings preserves tractability. In Section 3.4 we illustrate how our technique can be applied in a real-world case study. The chapter concludes with Section 3.5, where we address the problem of query rewriting under RPSs.

3.1 RDF Peer Systems

In this section, we introduce our framework for peer-to-peer RDF semantic data integration. We present a new peer mapping language designed to be suitable for the RDF model. In this setting, users can instantiate a peer-to-peer system by defining sets of BGP SPARQL queries and IRIs arbitrarly mapped in a peer-to-peer fashion, where each peer schema is simply represented by the set of IRIs adopted by the RDF datasource.

An *RDF Peer System* (RPS) $\mathcal{P}$ constitutes a set of peers and a set of mappings that specify the semantic relationships between peers. Formally, an RPS $\mathcal{P}$ is defined as a tuple $\mathcal{P} = (P, G, E, \Pi)$, where:

- P is the set of the *peers* in $\mathcal{P}$. Each peer $p \in P$ has its *peer schema*, denoted by $\Pi(p)$, which is the set of all the constants $u \in I$ (where I is the set of all the IRIs in Linked Data) adopted by p to describe data in the form of RDF triples. Informally, a peer schema is a subset of I comprising only the IRIs adopted by the peer. Two peer schemas need not be disjoint sets: this is in accordance with real Linked Data sources, where two different RDF databases may share some IRIs in the RDF triples.

- G is a set of *graph mapping assertions*, each of which is an expression of the form $Q \rightsquigarrow Q'$, where Q and Q' are *BGP SPARQL queries* of the same arity, expressed over the schemas $\Pi(p)$ and $\Pi(p')$, respectively, of two peers $p, p' \in P$. Formally, the graph pattern BGP_Q in the query Q contains triple patterns from $(\Pi(p) \cup L \cup V) \times (\Pi(p) \cup V) \times (\Pi(p) \cup L \cup V)$, and the graph pattern $BGP_{Q'}$ in the query Q' contains triple patterns from $(\Pi(p') \cup L \cup V) \times (\Pi(p') \cup V) \times (\Pi(p') \cup L \cup V)$.

- E is a set of *equivalence mappings* of the form $c \equiv_e c'$, where c and c' are in some $\Pi(p)$, with $p \in P$.

- $\Pi : P \rightarrow 2^I$ is a function from the set of peers to the set of peer schemas.

3.2 Semantics of RDF Peer Systems

We assume that data stored in the peers is in the form of a set of RDF triples for each peer in the system. Formally, for each peer $p \in P$ in $\mathcal{P}$, we have a database d, that is, a set of triples $\big(s, p, o\big) \in \big(\Pi(p) \cup B\big) \times \Pi(p) \times \big(\Pi(p) \cup B \cup L\big)$. Consequently, the *stored database* D of an RPS $\mathcal{P}$ is the union of all the peer databases d of all the peers in $\mathcal{P}$. A *peer-to-peer database* of an RPS $\mathcal{P}$ is simply an arbitrary RDF database containing triples $\big(s, p, o\big) \in \big(\Pi(p_1) \cup \cdots \cup \Pi(p_n) \cup B\big) \times \big(\Pi(p_1) \cup \cdots \cup \Pi(p_n)\big) \times \big(\Pi(p_1) \cup \cdots \cup \Pi(p_n) \cup B \cup L\big)$, where $p_1, \ldots, p_n \in P$ are the peers in $\mathcal{P}$.

We also denote by $subjQ(c)$, $predQ(c)$ and $objQ(c)$ three special BGP SPARQL queries:

- $subjQ(c) := x_{pred}, x_{obj} \leftarrow (c, x_{pred}, x_{obj})$
- $predQ(c) := x_{subj}, x_{obj} \leftarrow (x_{subj}, c, x_{obj})$
- $objQ(c) := x_{subj}, x_{pred} \leftarrow (x_{subj}, x_{pred}, c)$

where $c \in \big(\Pi(p_1) \cup \cdots \cup \Pi(p_n) \cup L\big)$.

The evaluation of $subjQ(c)$ over an RDF dataset is the set of pairs of the form $(t.pred, t.obj)$ containing the *predicate* and *object* of all triples in the dataset where the constant c occurs as the *subject*. The queries $predQ(c)$ and $objQ(c)$ are defined similarly, with the constant c now occurring as the *predicate* and the *object* of an RDF triple, respectively.

Below we give formal definitions for a *solution* of an RPS $\mathcal{P}$ and for the set of *certain answers* for a query posed against $\mathcal{P}$. Informally, a peer-to-peer database is

a solution of an RPS $\mathcal{P}$ if it contains the stored database of $\mathcal{P}$, as well as all triples inferred by the mappings of $\mathcal{P}$. The certain answers to a query against $\mathcal{P}$ are those which appear in all possible solutions of $\mathcal{P}$.

Definition 3.1. A peer-to-peer database I is said to be a *solution* for an RPS $\mathcal{P}$ *based on* a stored database D if:

1. For every peer database $d \in D$, we have that $d \subseteq I$.

2. For every graph mapping assertion in G of the form $Q \rightsquigarrow Q'$, we have that $Q^I_{\downarrow} \subseteq {Q'}^I_{\downarrow}$, with

$$\begin{aligned} Q^I_{\downarrow} &:= \{\, \boldsymbol{t} \mid \boldsymbol{t} = (t_1, \ldots, t_n) \in Q^I \text{ and } t_1, \ldots, t_n \in (I \cup L) \,\}, \text{ and} \\ {Q'}^I_{\downarrow} &:= \{\, \boldsymbol{t} \mid \boldsymbol{t} = (t_1, \ldots, t_n) \in {Q'}^I \text{ and } t_1, \ldots, t_n \in (I \cup L) \,\}. \end{aligned}$$

3. For every equivalence mapping in E of the form $c \equiv_e c'$, all of the following hold:

$$\begin{aligned} subjQ(c)^I &= subjQ(c')^I \\ predQ(c)^I &= predQ(c')^I \\ objQ(c)^I &= objQ(c')^I \end{aligned}$$

As we can see from the above definition, graph mapping assertions "drop" the tuples containing some elements in B (blank nodes). Blank nodes are used in RDF triples as placeholders for unknown resources [65]; in other words, they denote variables which may take values in the set of IRIs and literals ($I \cup L$). In this regard, graph mapping assertions exchange only full information between peers, dropping all the tuples containing partial information. We can say that, in the context of graph

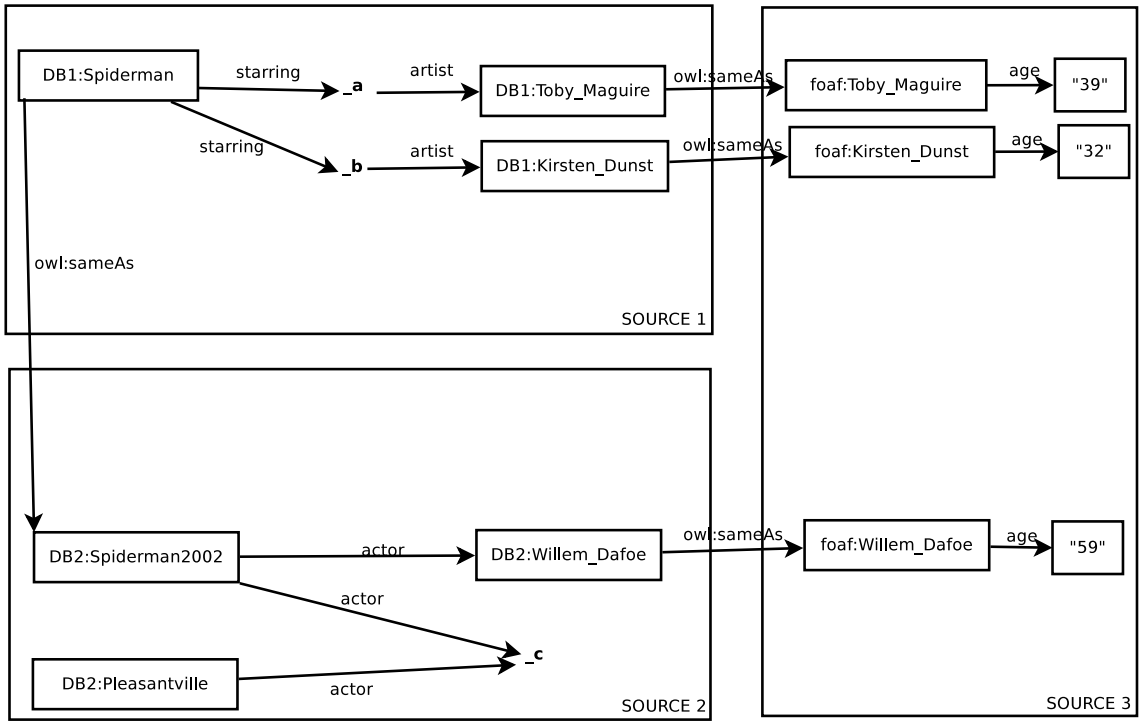

Figure 3.1: Example of an RDF graph from three data sources.

mapping assertions, blank nodes are treated as *labelled nulls* in the relational model, which are placeholders for unknown values and are not included in query results.

Definition 3.2. We define the *certain answers* $\texttt{RPS-ans}(q, D, \mathcal{P})$ of an arbitrary BGP SPARQL query q of arity n, based on a stored database D of an RPS $\mathcal{P}$, as the set of n-tuples $\boldsymbol{t}$ of constants in $\big(\Pi(p_1) \cup \cdots \cup \Pi(p_n) \cup L\big)$ such that, for every peer-to-peer database I that is a solution for the system $\mathcal{P}$ based on D, we have that $\boldsymbol{t} \in q^I$.

The query answering problem is defined as follows: given an RPS $\mathcal{P}$, a stored database D and a BGP SPARQL query q, find the certain answers $\texttt{RPS-ans}(q, D, \mathcal{P})$.

Example 3.2.1 illustrates how the semantics of certain answers under RPSs can support integration of multiple RDF sources in a typical Linked Data scenario.

Example 3.2.1. Figure 3.1 illustrates an RDF graph containing triples from three different sources. Sources 1 and 2 contain data about films, while Source 3 describes

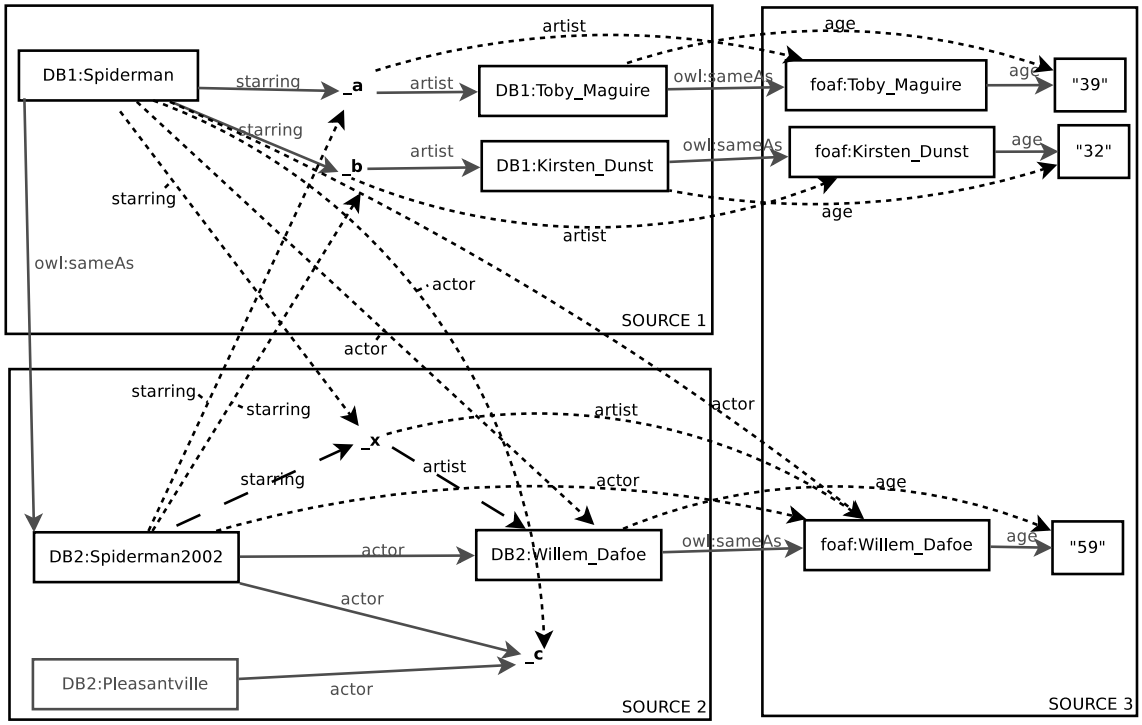

Figure 3.2: RDF graph of a universal model for the peer system of Example 3.2.1. Dotted arrows and dashed arrows represent triples inferred by the equivalence mappings and the graph mapping assertions, respectively.

people and their properties. We can see that URIs representing the same entities (e.g., `DB1:Spiderman` and `DB2:Spiderman2002`, for the film *Spiderman*) are linked by the OWL property `sameAs`, which states that the linked URIs represent the same real-world entity (best practices for `owl:sameAs` are given in [61]). It is clear that there is a semantic equivalence mapping between URIs linked by `sameAs`. We can also see that there is a semantic equivalence mapping between pairs of triples of the form `(a starring _z)` and `(_z artist b)` in Source 1 and triples of the form `(a actor b)` in Source 2; both represent the relationship that "*actor b acted in the film a*".

We define an RPS $\mathcal{P} = (P, G, E, \Pi)$ as follows:

- $P := \{p_1, p_2, p_3\}$ are the peers, i.e., the three sources.
- G is composed of a graph mapping assertions of the form $Q_2 \rightsquigarrow Q_1$, where:

- $Q_1 := x, y \leftarrow (x, starring, z)$ AND $(z, artist, y)$,
- $Q_2 := x, y \leftarrow (x, actor, y)$.

- E contains an equivalence mapping $c \equiv_e c'$ for each triple of the form $(c, sameAs, c')$.
- Π is defined so that $\Pi(p_i)$ is the set of IRIs in the i^{th} source. For example, we have that

$$\Pi(p_2) = \{\text{DB2:Spiderman2002}, \text{DB2:Willem\_Dafoe}, \text{DB2:Pleasantville}, \text{actor}\}.$$

Note that, for simplicity, here we do not consider other potential peer mappings - for instance, the mapping $Q_1 \rightsquigarrow Q_2$. Now assume that a user poses the following BGP SPARQL query:

```
SELECT ?x ?y
WHERE { DB1:Spiderman starring ?z . ?z artist ?x .
?x age ?y }
```

This query returns an empty result on the data of Figure 3.1, since the `sameAs` property is missing from the query, and SPARQL does not automatically exploit semantic mappings between RDF resources.

Figure 3.2 illustrates an RDF database which is a universal model for $\mathcal{P}$. Let us consider again the SPARQL query illustrated above. Now, evaluating the query over the universal model, we obtain the result in Listing 3.1. It is important to observe that the user poses a query over Sources 1 and 3 but retrieves additional information also from Source 2 in a transparent way. The RPS, in fact, not only captures the semantics of the `owl:sameAs` property, but also performs integration of similar sources in order to return additional answers to the user. This integration

```
#Query

SELECT ?x ?y
WHERE { DB1:Spiderman starring ?z .
?z artist ?x .
?x age ?y }

#Result
DB1:Toby_Maguire "39"
foaf:Toby_Maguire "39"
DB1:Kirsten_Dunst "32"
foaf:Kirsten_Dunst "32"
DB2:Willem_Dafoe "59"
foaf:Willem_Dafoe "59"

#Result without redundancy

DB1:Toby_Maguire "39"
DB1:Kirsten_Dunst "32"
DB2:Willem_Dafoe "59"
```

Listing 3.1: SPARQL query over the universal model.

can be performed dynamically as new data sources appear, and requires no input from the user. □

3.3 Query answering

To investigate the complexity of the query answering problem, we show that the problem of finding $\texttt{RPS-ans}(q, D, \mathcal{P})$ can be reduced to CQ answering under a certain set of TGDs. In this setting, we will see that the set of TGDs retains the *decidability* property; this is stated in Theorem 3.1, whose proof shows that the chase terminates in polynomial time with respect to the size of the data.

We first define the relational schema $\boldsymbol{R}_{RDF} := \{triple, resource\}$, where *triple* is a ternary relational symbol and *resource* is a unary relational symbol. Now, for each RDF database D we define the relational instance $D^{\boldsymbol{R}_{RDF}}$ over the relational

schema $\boldsymbol{R}_{RDF}$, such that for each RDF triple $(s, p, o) \in D$ there exists an atom $triple(s, p, o) \in D^{\boldsymbol{R}_{RDF}}$ and for each $r \in dom(D)$ there exists an atom $resource(r) \in D^{\boldsymbol{R}_{RDF}}$. Note that $dom(D)$ does not include blank nodes contained in D.

Given a BGP SPARQL query q of the form $\boldsymbol{X} \leftarrow BGP_q$ we define the conjunction of atoms $\phi_q(\boldsymbol{X}, \boldsymbol{Y})$ as follows:

$$\phi_q(\boldsymbol{X}, \boldsymbol{Y}) := \bigwedge_{(x,y,z) \text{ is a triple pattern in } q} triple(x, y, z),$$

where $\boldsymbol{Y} = y_1, \ldots, y_m \in var(BGP_q)$ are the existentially quantified variables of q. For example, given the following BGP SPARQL query

$$q := \quad x_1, x_2 \leftarrow (x_1, \texttt{father}, y) \text{ AND } (y, \texttt{father}, x_2),$$

where $x_1, x_2, y \in var(BGP_q)$ and $\texttt{father} \in I$, then $\phi_q((x_1, x_2), y)$ is the conjunction of atoms

$$triple(x_1, \texttt{father}, y) \wedge triple(y, \texttt{father}, x_2),$$

Now, we derive from a BGP SPARQL query q the CQ q_{CQ} defined as follows:

$$q_{CQ} := ans_{q_{CQ}}(\boldsymbol{X}) \leftarrow \phi_q(\boldsymbol{X}, \boldsymbol{Y}),$$

From this formalisation, it follows that $\boldsymbol{t} \in q^D$ if and only if $ans_{q_{CQ}}(\boldsymbol{t}) \in q_{CQ}(D^{\boldsymbol{R}_{RDF}})$.

Given an RPS $\mathcal{P} = (P, G, E, \Pi)$, we are now ready to define a set of TGDs $\mathcal{P}^{\boldsymbol{R}_{RDF}}$ such that $\texttt{RPS-ans}(q, D, \mathcal{P}) = \texttt{ans}(q_{CQ}, D^{\boldsymbol{R}_{RDF}}, \mathcal{P}^{\boldsymbol{R}_{RDF}})$. For each graph mapping assertion $Q \rightsquigarrow Q' = \boldsymbol{X} \leftarrow BGP_Q \rightsquigarrow \boldsymbol{X'} \leftarrow BGP_{Q'} \in G$, we have the following *graph mapping* TGD in $\mathcal{P}^{\boldsymbol{R}_{RDF}}$:

$$\phi_Q(\boldsymbol{X}, \boldsymbol{Y}) \wedge resource(x_1) \wedge \cdots \wedge resource(x_n) \rightarrow \exists \boldsymbol{Y'}\ h_{(\boldsymbol{X'},\boldsymbol{X})}\big(\phi_{Q'}(\boldsymbol{X'}, \boldsymbol{Y'})\big).$$

where $h_{(\boldsymbol{X'},\boldsymbol{X})} : S_1 \rightarrow S_2$ is the substitution $\{x'_1 \rightarrow x_1,\ x'_2 \rightarrow x_2, \ldots,\ x'_n \rightarrow x_n\}$, $\boldsymbol{X} = x_1, x_2 \ldots, x_n$ and $\boldsymbol{X'} = x'_1, x'_2 \ldots, x'_n$.

For each equivalence mapping $c \equiv_e c' \in E$, we have the following set of *equivalence mapping* TGDs in $\mathcal{P}^{\boldsymbol{R}_{RDF}}$:

$$\begin{aligned}
&triple(c, y, z) \rightarrow triple(c', y, z),\\
&triple(c', y, z) \rightarrow triple(c, y, z),\\
&triple(x, c, z) \rightarrow triple(x, c', z),\\
&triple(x, c', z) \rightarrow triple(x, c, z),\\
&triple(x, y, c) \rightarrow triple(x, y, c'),\\
&triple(x, y, c') \rightarrow triple(x, y, c).
\end{aligned}$$

From this definition, it follows that $\texttt{RPS-ans}(q, D, \mathcal{P})$ and $\texttt{ans}(q_{CQ}, D^{\boldsymbol{R}_{RDF}}, \mathcal{P}^{\boldsymbol{R}_{RDF}})$ yield the same result. Thus, to assess the complexity of answering a BGP SPARQL query q over RPSs, we consider the equivalent problem of finding the certain answers of the CQ q_{CQ} under $D^{\boldsymbol{R}_{RDF}}$ and the set of TGDs $\mathcal{P}^{\boldsymbol{R}_{RDF}}$.

The set of certain answers for CQ answering under TGDs is computed by evaluating queries over the so-called *universal model* [44] (also referred to in the literature as the *universal solution*). To generate a universal model, a source database is "chased" using the set of dependencies. Each step of the chase "extends" the database so that the chosen dependency is satisfied. As described in Chapter 2, given a TGD $\phi(\boldsymbol{X}, \boldsymbol{Z}) \rightarrow \exists \boldsymbol{Y}\ \psi(\boldsymbol{X}, \boldsymbol{Y})$ and a mapping h (from the variables in $\phi(\boldsymbol{X}, \boldsymbol{Z})$ to constants) for which the dependency is not satisfied, the chase step generates new atoms in the instance in order to satisfy the dependency. The new atoms are generated by: (a) extending h to h' such that each existentially quantified variable in $\psi(\boldsymbol{X}, \boldsymbol{Y})$ is assigned a freshly created constant, a labelled null, followed by: (b) taking the

image of the atoms of ψ under h' (see Chapter 2.1.1 for more details of the chase procedure). An "RDF" version of the chase procedure that has as its input the mappings in an RPS and an RDF stored database is listed in Algorithm 1; this version gives a more direct way to implement the chase procedure as it does not require the translation of RPS mappings into TGDs.

In our set of TGDs $\mathcal{P}^{\boldsymbol{R}_{RDF}}$, there are no atoms of type $resource(x)$ in the head of any dependency such that the variable x is existentially quantified. Therefore, the set of IRIs and literals remains constant during the chase procedure. Thus, the chase generates new blank nodes as labelled nulls. Without loss of generality, we will use the term *newly created blank nodes* when we want to denote labelled nulls.

Graph mapping TGDs are the only dependencies in $\mathcal{P}^{\boldsymbol{R}_{RDF}}$ that contain existentially quantified variables in the head, therefore they are the only dependencies for which the chase may generate newly created blank nodes. Newly created blank nodes cannot trigger any of the graph mapping TGDs, so the chase sequence is bounded by a finite number of steps: this is because, following the semantics of the RPS, tuples containing elements in B (blank nodes) are dropped from the evaluation of the BGP SPARQL queries in both the head and the body of the graph mapping assertions. We now show that, in fact, the number of steps in the chase is bounded by a polynomial in the size of the set of constants, i.e., $I \cup L$.

Theorem 3.1. *The problem of finding all certain answers* $\texttt{RPS-ans}(q, D, \mathcal{P})$ *to a BGP SPARQL query q, for a given RDF Peer System $\mathcal{P}$ and a stored database D, has PTIME data complexity.*

Proof. We know that $\texttt{RPS-ans}(q, D, \mathcal{P}) = \texttt{ans}(q_{CQ}, D^{\boldsymbol{R}_{RDF}}, \mathcal{P}^{\boldsymbol{R}_{RDF}})$ by the definitions of q_{CQ}, $D^{\boldsymbol{R}_{RDF}}$ and $\mathcal{P}^{\boldsymbol{R}_{RDF}}$. The answer to q_{CQ} with respect to $D^{\boldsymbol{R}_{RDF}}$ and

Algorithm 1: Using the chase to compute the certain answers `RPS-ans`$(q, D, \mathcal{P})$.

Data: Graph pattern query q, RPS $\mathcal{P} = (P, G, E, \Pi)$, stored instance D.
Result: The set $\vec{t}$ of the certain answers `RPS-ans`$(q, D, \mathcal{P})$.
Initialize instance $J = D$;
`/* Chase step to generate a universal model */`
while *some of the mappings of* $\mathcal{P}$ *are not satisfied in* J **do**
  Let m be such a mapping;
  case m *of* $Q \rightsquigarrow Q' \in G$*:* **do**
    for *each tuple* $t \in Q^J \setminus Q'^J$ **do**
      generate the boolean query bQ' by substituting t in the free variables Q';
      add triples to J generating new blank nodes, such that $bQ'^J = true$;
  case m *of* $c \equiv_e c' \in E$ **do**
    if $subjQ(c)^{*J} \neq subjQ(c')^{*J}$ **then**
      for *each tuple* $(p, o) \in (subjQ(c)^{*J} \setminus subjQ(c')^{*J})$ **do**
        add the triple (c', p, o) to J;
      for *each tuple* $(p, o) \in (subjQ(c')^{*J} \setminus subjQ(c)^{*J})$ **do**
        add the triple (c, p, o) to J;
    if $predQ(c)^{*J} \neq predQ(c')^{*J}$ **then**
      for *each tuple* $(s, o) \in (predQ(c)^{*J} \setminus predQ(c')^{*J})$ **do**
        add the triple (s, c', o) to J;
      for *each tuple* $(s, o) \in (predQ(c')^{*J} \setminus predQ(c)^{*J})$ **do**
        add the triple (s, c, o) to J;
    if $objQ(c)^{*J} \neq objQ(c')^{*J}$ **then**
      for *each tuple* $(s, p) \in (objQ(c)^{*J} \setminus objQ(c')^{*J})$ **do**
        add the triple (s, p, c') to J;
      for *each tuple* $(s, p) \in (objQ(c')^{*J} \setminus objQ(c)^{*J})$ **do**
        add the triple (s, p, c) to J;
`/* End of chase */`
compute the certain answers $\vec{t} := q^J$;
`/* The certain answers are generated by evaluating the query over the universal solution */`
return $\vec{t}$;

$\mathcal{P}^{\boldsymbol{R}_{RDF}}$ is computed by generating a universal model J for $D^{\boldsymbol{R}_{RDF}}$ with respect to $\mathcal{P}^{\boldsymbol{R}_{RDF}}$ and then evaluating q_{CQ} over J.

Let us consider first the chase sequence that generates J. We first check the number of chase steps that are triggered by the graph mapping TGDs. Since labelled nulls cannot trigger any such rules, the number of steps is bounded by a function $Q(n)$ over the number of constants n in $I \cup L$. Let us now consider $Q(n)$. The relation $triple(x, y, z)$ is a 3-ary predicate, therefore the number of tuples that can trigger one atom in the body of a rule is at most n^3. Let d be the maximum number of atoms of the form $triple(x, y, z)$ appearing in the body of these rules, and s the number of graph mapping TGDs in $\mathcal{P}^{\boldsymbol{R}_{RDF}}$. Then, number of chase steps that are triggered by the graph mapping TGDs is bounded by $Q(n) = (n^3)^d \times s$. Since s and d are constants, $Q(n)$ is a polynomial.

Consider now the number of chase steps triggered by equivalence mapping TGDs and let r be the number of equivalence mapping TGDs in $\mathcal{P}^{\boldsymbol{R}_{RDF}}$. This number of steps is bounded by the function $T(n) = (n + n_b)^3 \times r$ over the number of constants n in the original source instance and the number of newly created labelled nulls n_b. Equivalence mappings do not contain existentially quantified variables in the head, so they do not generate newly created blank nodes during the chase. Thus, n_b depends only on the chase sub-sequence under graph mapping assertions, so is bounded by a polynomial function, say $P(m)$, over the maximum number of chase steps under graph mapping assertions, i.e., $Q(n)$. Thus, $T(n) = \Big(n + P(Q(n))\Big)^3 \times r$.

The number of chase steps is then bounded by

$$T(n) + Q(n) = \Big[\Big(n + P(Q(n))\Big)^3 \times r\Big] + Q(n),$$

since, in a worst-case scenario, one chase step is triggered by a single rule application. Therefore, a universal model J can be produced in polynomial time. In particular,

since n is finite, the chase always terminates.

It now remains to generate the set of certain answers by posing the CQ q_{CQ} over J. Since CQ answering is in AC^0 for data complexity [2], the claim follows. □

3.4 Case Study

We now present a larger-scale, real-world case study that illustrates the practical usage of our approach. The case study illustrated here is part of the INSPIRE project [92], which addresses the domain of career guidance, particularly *career transitions*. Although many sources of careers advice exist (school and university careers advisors; websites such as Jobserve, Prospects, UK National Careers service; friends, relatives, peers and colleagues), when a person is at a particular career point they may be unaware of all the possible options and may not see longitudinal paths. For example, it may be hard to gauge the relevance of available information, the diversity of information sources may create a sense of 'being lost', it may take a long time to find useful information, and it may be hard to elicit longer-term career trajectories as opposed to just immediate possible next steps.

The aim of the INSPIRE project is to develop an interactive tool that can aid users' exploration of Linked Data graphs representing job and learning opportunities by assisting them in identifying paths in the data that will be beneficial for expanding their awareness of career options. Such a tool would have the potential to motivate users to explore short- or longer-term career paths, through a combination of personalised ontological querying and reasoning over large volumes of real careers data, stored as Linked Data graphs. So far we are using two databases: the L4All RDF/S ontology arising from the L4All and MyPlan projects, capturing the work and educational experiences of lifelong learners [93]; and a new RDF/S ontology

we have designed based on the information published on *LinkedIn*[1], a business and employment-oriented service that operates via websites and mobile apps. To integrate data from both the L4All ontology and the LinkedIn web service, we define an RDF Peer System.

In the following sections, we illustrate the steps we followed for the case study: in Section 3.4.1 we show how we designed a new ontology based on the LinkedIn profile pages; in Section 3.4.2 we give an overview of the L4All ontology arising from the L4All and MyPlan projects; finally, in Section 3.4.3 we show how we exploited the RPS mapping language to materialise an integrated version of the two heterogeneous data sources.

3.4.1 LinkedIn Ontology design

Based on the LinkedIn members' pages, we have designed an RDF/S ontology, which we call here the *LinkedIn ontology*. Its main concepts are as follows:

- CVItem (represents the various events of a LinkedIn user's CV);
- EducationItem (represents all CV events related to qualification, including degree and self-study);
- CertificationItem (represents all CV events leading to a certified qualification, including professional development courses and MOOCs);
- WorkItem (represents all CV events related to work experience with some role);
- OtherItem (represents additional CV events, e.g. travel, voluntary experience);

[1] `https://uk.linkedin.com/`

- Member (represents information about the person, including skills, hobbies).

Figure 3.3 contains a class diagram of the Linkedin ontology.

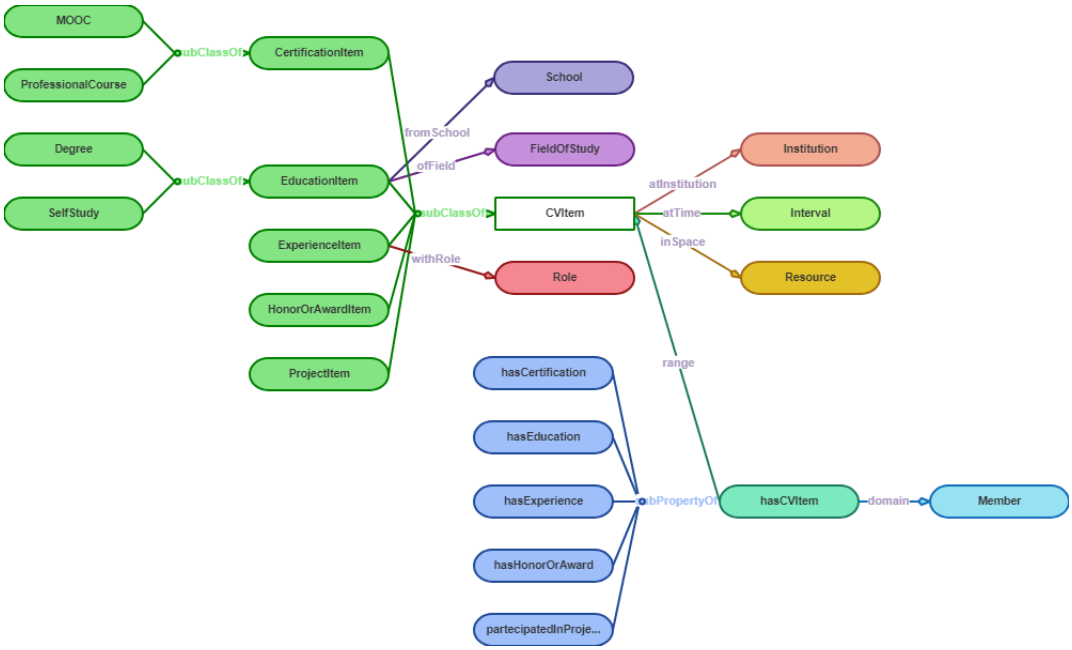

Figure 3.3: *NeighborGram*^{TM} of the Linkedin ontology.

To obtain a set of test data to populate the LinkedIn ontology, we developed a Java tool that parses LinkedIn HTML profile pages into anonymised RDF triples. To implement the tool we performed the following steps.

Firstly, we inspected the underlying HTML structure of a LinkedIn profile page. This process allowed us to identify which HTML tree elements in the LinkedIn pages (e.g., branches and leaves) contain the information needed to populate the LinkedIn ontology. Figure 3.4 illustrates the design process we followed for the page inspection. We first identify the HTML branch containing data regarding education experience; the root of this branch is the *section* element with the id "education-section", at the top of the image. Then, this branch is traversed to identify the sub-branches corresponding to single education items; these branches are called the *education item branches*. In the figure, an education item branch root is pointed

to by a purple arrow. Finally, the information related to the EducationItem class in the LinkedIn ontology is extracted as text from a set of candidate leaves of each education item branch: *1)* the school, highlighted in red; *2)* the degree title, highlighted in yellow; *3)* the field of study, highlighted in green; and *4)* the time, highlighted in blue.

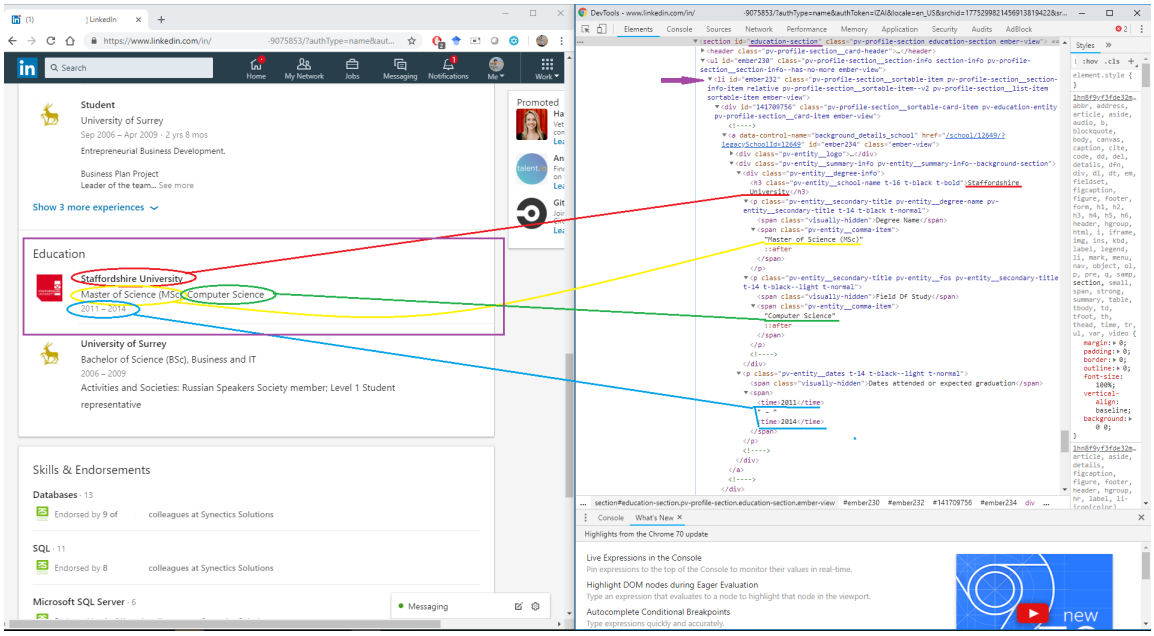

Figure 3.4: Analysis of LinkedIn profile HTML structure.

Following from the page inspection, we developed a Java application that automatically extracts data from the LinkedIn profile pages. The application takes as input a set of HTML files and outputs an RDF dataset in one of the RDF standard serialisation formats[2]. The parsing logic was implemented with the Jsoup library[3] following the approach described in the example above.

A sample output of the parser is depicted in Figure 3.5 which includes a snapshot of the RDF data extracted from a LinkedIn HTML profile page, serialised in Turtle format.

To generate a test database instance for the LinkedIn ontology, we selected a set

[2]https://www.w3.org/TR/rdf-syntax-grammar/
[3]https://jsoup.org/

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX time: <http://www.w3.org/2006/time#>
PREFIX data: <http://dcs.bbk.ac.uk/LinkedIn/data/>
PREFIX linkedIn: <http://dcs.bbk.ac.uk/LinkedIn/ontology#>


data:bcb1d76d-c61c-4e1f-8bd0-73809966b4ab linkedIn:Summary "A postgraduate in MSc Computer Science, having first class honours BSc in Business and IT,
who has worked in public and private sector companies. Fluent in Russian and English, able to work well both independently on solitary tasks or as part
of a team. Highly motivated and adaptable. A meticulous problem solver, who ensures that tasks and projects are completed within deadlines. Particularly
interested in working with an entrepreneurial company. Excellent analytical skills. Enjoys challenges." ;

        linkedIn:memberSubtitle "Developer at Synectics Solutions Ltd" ;

        linkedIn:hasExperience _:BX2D79af5035X3A159126645d1X3AX2D7ea2 , _:BX2D79af5035X3A159126645d1X3AX2D7d77 ;

        linkedIn:hasEducation _:BX2D79af5035X3A159126645d1X3AX2D7f1a ;

        linkedIn:hasSkill linkedIn:Database_design , linkedIn:MI_analysis , linkedIn:SQL ,
linkedIn:Agile_methodologies , linkedIn:C++ , linkedIn:Database_design , linkedIn:System_administrator .


_:BX2D79af5035X3A159126645d1X3AX2D7d77 rdfs:label "Admin / Finance Assistant Tiger Beer UK LtdAugust 2007 – May 2009 (1 year 10 months) Key
Responsibilities: reception, customer service, order processing and accounting administration. I gained valuable experience in communicating effectively
with distributors and customers, handling customer complaints and facilitating office systems." ;

        linkedIn:withRole linkedIn:Finance_Assistant ;

        linkedIn:atCompany linkedIn:Adam_Tech ;

        linkedIn:atTime _:BX2D79af5035X3A159126645d1X3AX2D7e04 .


_:BX2D79af5035X3A159126645d1X3AX2D7e04 rdf:type time:Interval ;

        time:hasBeginning "2007-07-31T23:00:00Z"^^<http://www.w3.org/2001/XMLSchema#dateTime> ;

        time:hasEnd "2009-04-30T23:00:00Z"^^<http://www.w3.org/2001/XMLSchema#dateTime> .
```

Figure 3.5: Snapshot of RDF data for a LinkedIn profile in Turtle serialisation format.

of LinkedIn profiles comprised of 31 profiles of users who obtained a BSc in the field of Computer Science, and 25 profiles of users who obtained a MSc in the field of Computer Science. Each profile includes a mixture of both education and work experience items, as well as a set of skills and additional information about the person, including the profile summary, current job title and hobbies. After parsing the selected profiles, we obtained an RDF database containing a total of 3190 RDF statements.

3.4.2 L4All ontology overview

In our case study, we aim at integrating the LinkedIn ontology with the ontology developed by the L4All project [93]. The L4All system allows users to create and maintain a chronological record — a timeline — of their learning and work episodes. This data and metadata are encoded as RDF/S, as shown in Figure 3.6. In particular, each instance of the Episode class is: linked to a subclass of Episode by an

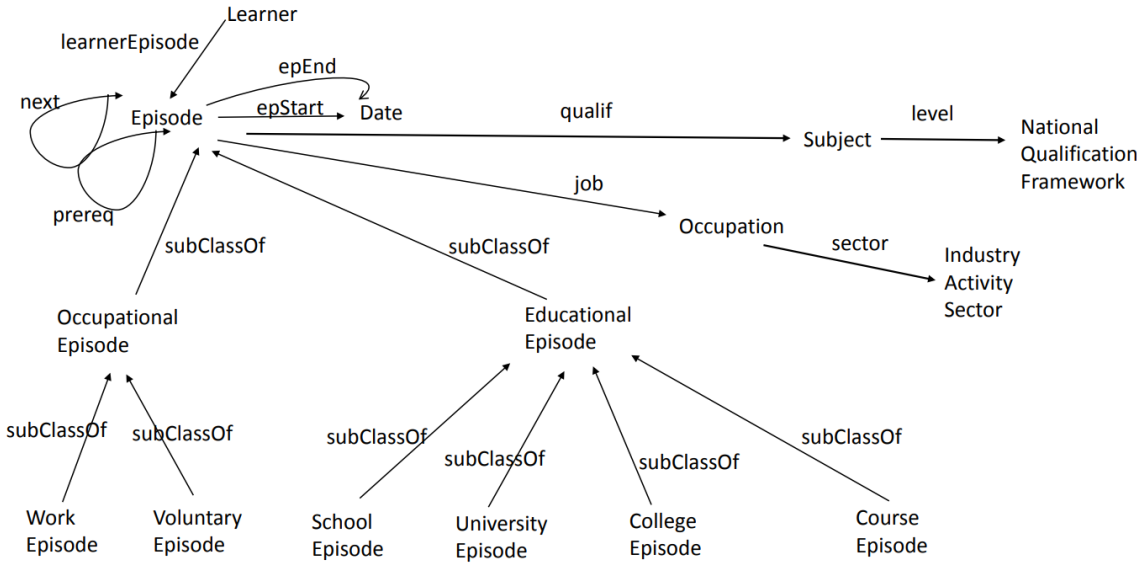

Figure 3.6: L4All ontology fragment (from [92]).

edge labelled rdf:type; linked to other episode instances by edges labelled 'next' or 'prereq' (indicating whether the earlier episode simply preceded, or was necessary in order to be able to proceed to, the later episode); linked either to an occupation or to an educational qualification by means of an edge labelled 'job' or 'qualif'. Each occupation is linked to a subclass of the Occupation class by an edge labelled rdf:type, and to an instance of the Industry Activity Sector class by an edge labelled 'sector'. Each qualification is linked to a subclass of the Subject class by an edge labelled rdf:type and to an instance of the National Qualification Framework[4] class by an edge labelled 'level'. The dataset used in our case study comprises 17 timelines derived from the L4All project — 16 from real users, and 1 demonstration timeline. Similarly to the LinkedIn ontology, each L4All timeline contains both educational and occupational episodes, and they vary in terms of the number of episodes contained within them, as well as the classification of each episode. The dataset contains a total of RDF 195 triples, with an average of 5.3 episodes per timeline and a total of 91 episodes.

[4] `http://www.cedefop.europa.eu/en/events-and-projects/projects/national-qualifications-framework-nqf`

In addition to the above dataset, we translated a subset of the LinkedIn ontology database into a set of L4All timelines to be added to the existing 17 L4All timelines so as to create a semantic overlap between the two test databases. To achieve this, we used the information of 5 LinkedIn profiles from the LinkedIn ontology in Section 3.4.1 and manually generated a set of 5 corresponding L4All timelines. The outcome of this step produced a total of 31 additional L4All episodes and 67 additional RDF triples in the L4All dataset.

In order to align the 5 user profiles contained in both the L4All and the LinkedIn ontology, we defined a set of 36 RPS equivalence mappings; such mappings indicate that two IRI or blank node references (one in L4All and one in the LinkedIn dataset) actually refer to the same entity, i.e., the individuals have the same identity. These equivalence mappings, along with both the source data and the graph mapping assertions in the following section, form the RPS for this case study.

3.4.3 Mappings in RPS and materialised data

To align the two ontologies illustrated in the previous sections, we defined a set of mappings adopting the syntax introduced for the RDF peer system in Section 3.1. Firstly, we linked classes and properties via *1-to-1 mappings*, i.e., mappings from a single LinkedIn term to a single L4All term (and vice versa) when both terms represent the same ontological concepts. An example is the mapping between the CVItem class from LinkedIn and the Episode class from L4All, in that both encapsulate episodes in a user's career path, whether an educational episode or an experience one. For each 1-to-1 mapping, we defined a pair of RPS graph mapping assertions, one to translate data from the LinkedIn to the L4All schema and one for translation in the opposite direction. For example, the following mappings constitute the alignment of the CVItem and Episode classes and of the hasCVItem and

learnerEpisode relationships:

$$
\begin{aligned}
x \leftarrow (x, \texttt{rdf:type}, \texttt{CVItem}) &\rightsquigarrow x' \leftarrow (x', \texttt{rdf:type}, \texttt{Episode}) \\
x \leftarrow (x, \texttt{rdf:type}, \texttt{Episode}) &\rightsquigarrow x' \leftarrow (x', \texttt{rdf:type}, \texttt{CVItem}) \\
x, y \leftarrow (x, \texttt{hasCVItem}, y) &\rightsquigarrow x', y' \leftarrow (x', \texttt{learnerEpisode}, y') \\
x, y \leftarrow (x, \texttt{learnerEpisode}, y) &\rightsquigarrow x', y' \leftarrow (x', \texttt{hasCVItem}, y')
\end{aligned}
$$

In most cases, identifying a potential link between equivalent ontological concepts was straightforward (as in the examples above) and the expressive power of 1-to-1 mappings was sufficient. However, some alignment cases appeared to be ambiguous and required a thorough analysis of the different scenarios, and a more expressive mapping definition. Examples of this are the RPS graph mapping assertions between the relationship 'epStart' in L4All and the relationship 'atTime' in the LinkedIn Ontology:

$$
\begin{aligned}
&x, y \leftarrow (x, \texttt{epStart}, y) \rightsquigarrow x', y' \leftarrow (x', \texttt{atTime}, z') \text{ AND } (z', \texttt{hasBeginning}, y') \\
&x, y \leftarrow (x, \texttt{atTime}, z) \text{ AND } (z, \texttt{hasBeginning}, y) \rightsquigarrow x', y' \leftarrow (x', \texttt{epStart}, y')
\end{aligned}
$$

The above assertions define the mapping between the L4All property 'epStart' and the path composed of 'atTime' and 'hasBeginning' in the LinkedIn ontology. We also introduced two additional graph mapping assertions to provide alignment of LinkedIn CV items that are linked to a specific date instead of an interval:

$$
\begin{aligned}
x, y \leftarrow (x, \texttt{atTime}, y) &\rightsquigarrow x', y' \leftarrow (x', \texttt{epStart}, y') \\
x, y \leftarrow (x, \texttt{atTime}, y) &\rightsquigarrow x', y' \leftarrow (x', \texttt{epEnd}, y')
\end{aligned}
$$

Ultimately, the RPS defined for this case study contained 22 graph mapping assertions (the full set is listed in Appendix A.1.1) and 36 equivalence mappings (which we do not include here for reasons of privacy protection). We then generated the universal model over the source data, the graph mapping assertions and the equiv-

alence mappings. In order to do so, we implemented the chase step of Algorithm 1 as a Java application and we ran it over the set of mappings illustrated here and the RDF dataset composed of the union of the triples obtained from LinkedIn and L4All. The execution of the algorithm outputted a total of 29603 materialised RDF triples, compared to 10049 RDF triples in both the original sources combined. By answering queries over the materialised data, we were able to retrieve more information about the two sources, as illustrated in the example below.

We note that, the above four graph mapping assertions introduce a cycle: this is because the triple $(x, \texttt{epStart}, y)$ is present on the left hand side and right hand side of two different assertions (modulo variable renaming). In such scenarios, the commonly adopted approach of interpreting peer-to-peer systems using a first-order logic may lead to undecidability of query answering; in this case, in fact, the chase algorithm does not terminate (see Section 2.2.2). An example is shown in Figure 3.7, which shows the triples added to the universal model at each step of a chase procedure under the above four graph mapping assertions, interpreted under RPS semantics and FO semantics. Under RPS semantics, the chase stops at step 1, whereas under FO semantics it runs indefinitely. This is because *_:bNode1* does not trigger the mapping $x, y \leftarrow (x, \texttt{atTime}, y) \rightsquigarrow x', y' \leftarrow (x', \texttt{epStart}, y')$, since *_:bNode1* is a blank node and y is a free variable of the query on the left hand side of the axiom.

Consider now the following two SPARQL queries:

a)
```
SELECT DISTINCT ?learner
WHERE
  { ?learner L4All:learnerEpisode ?occupEpisode .
   ?occupEpisode L4All:job ?job .
   ?job a L4All:Science_and_Technology_Professionals
  }
```

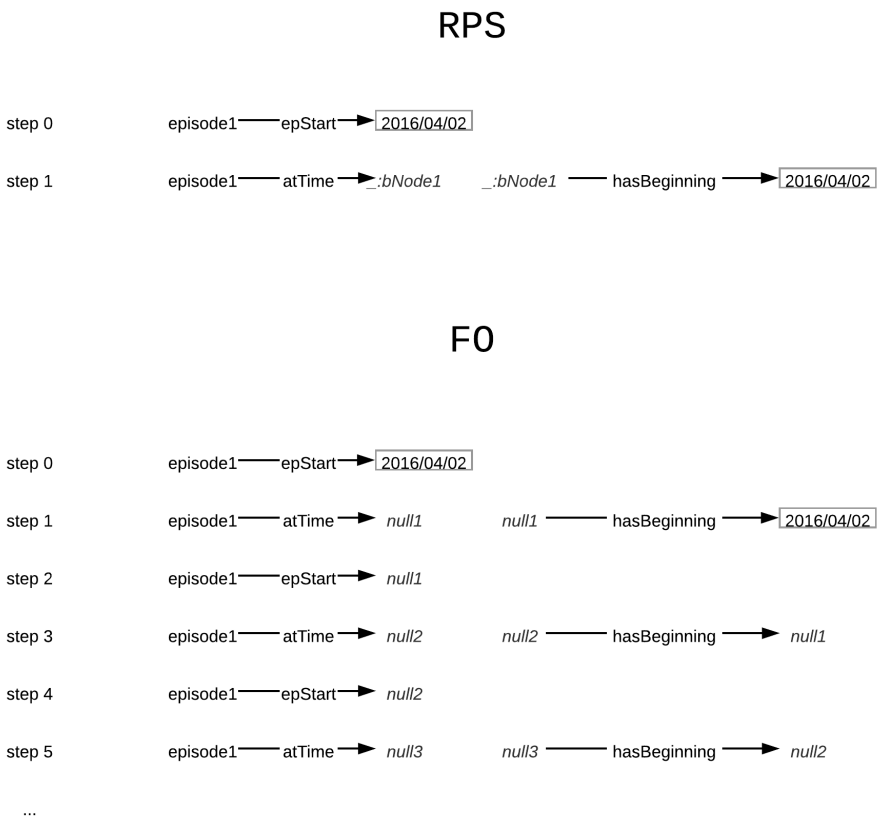

Figure 3.7: Materialisation of RDF triples via the chase algorithm, under RPS semantics and FO semantics.

b)
```
SELECT DISTINCT ?learner
WHERE
  { ?member linkedIn:hasCVitem ?occupEpisode .
       ?occupEpisode linkedIn:role ?role .
       ?role a linkedIn:Science_and_Technology
  }
```

Query a) asks for all the L4All learners with work experience as a Science and Technology professional. We ran Query a) over the materialised data and we obtained 27 results, as opposed to 3 results when the same query was posed over just the original L4All data. Thus, the RPS mappings enable L4All users to retrieve both L4All and LinkedIn data in a transparent way, without needing to be familiar with

the LinkedIn ontology. Query b) is equivalent to Query a) but is expressed using terms of the LinkedIn vocabulary. Query b) produced the same results as Query a) over the materialised data and no results over just the original LinkedIn data.

3.5 SPARQL query rewriting for RPSs

In this section, we first check in Section 3.5.1 whether the TGDs for the RPS peer mappings enjoy the FO-rewritability property, which is a property that ensures a perfect rewriting in a language that is not more expressive than FO queries. As we will see later in this section, the TGDs for general RPSs mappings are not FO-rewritable, and thus it is not possible to compute a perfect rewriting in SPARQL 1.0. In Section 3.5.2 we identify some restrictions on the peer mapping language that do allow us to formulate such a rewriting. Based on these restrictions, in Section 3.5.3 we describe an empirical evaluation of our prototype of an RPS framework.

3.5.1 FO-rewritable TGDs

With the following proposition, we show that the TGDs for general RPSs mappings are not FO-rewritable.

Proposition 3.1. *Let $\mathcal{P}$ be an RDF Peer System. The set of TGDs $\mathcal{P}^{\boldsymbol{R}_{RDF}}$ is not FO-rewritable.*

Proof. Let D be a stored database and let $\mathcal{P}^{\boldsymbol{R}_{RDF}}$ be defined only by the following TGD σ:

$$triple(x, A, z), triple(z, A, y), resource(x), resource(y) \rightarrow triple(x, A, y),$$

We assume without loss of generality that D does not contain blank nodes. Now let us drop the atoms $resource(x)$, $resource(y)$ in the body of the TGD resulting in σ^- defined as follows:

$$triple(x, A, z), triple(z, A, y) \rightarrow triple(x, A, y),$$

Since D does not contain blank nodes, we have that

$$chase(D^{\boldsymbol{R}_{RDF}}, \{\sigma\}) = chase(D^{\boldsymbol{R}_{RDF}}, \{\sigma^-\}).$$

Now consider the following set σ_{trans} of TGDs:

$$triple(x, A, y) \rightarrow A(x, y)$$
$$A(x, z), A(z, y) \rightarrow A(x, y)$$
$$A(x, y) \rightarrow triple(x, A, y).$$

Given a relational instance I, let A^I be the set of atoms in I of the form $A(t_1, t_2)$. It is easy to see that $chase(D, \sigma_{\text{trans}})/A^{chase(D,\sigma_{\text{trans}})} = chase(D, \sigma^-)$. Thus, for a given BGP query q, we have that $q_{CQ}\big(chase(D, \sigma_{\text{trans}})\big) = q_{CQ}\big(chase(D, \sigma^-)\big)$ as the relation A is not a member of the relational schema $\boldsymbol{R}_{RDF}$. It follows that

$$q_{CQ}\big(chase(D, \sigma_{\text{trans}})\big) = q_{CQ}\big(chase(D, \sigma)\big),$$

since $chase(D^{\boldsymbol{R}_{RDF}}, \{\sigma\}) = chase(D^{\boldsymbol{R}_{RDF}}, \{\sigma^-\})$. Now, assume there exists a FO-rewriting ${q_{CQ}}^*$ of q_{CQ} with respect to the σ. Then ${q_{CQ}}^*$ is also a FO-rewriting of q_{CQ} with respect to the σ_{trans} because $q_{CQ}\big(chase(D, \sigma_{\text{trans}})\big) = q_{CQ}\big(chase(D, \sigma)\big)$, which means that σ_{trans} is a set of FO-rewritable TGDs. This is not possible since the TGD $A(x, z), A(z, y) \rightarrow A(x, y)$ in σ_{trans} is not FO-rewritable because it captures the transitive closure of the relation A, which cannot be done using a finite number of first-order queries. □

As a result of this, given an RPS and a BGP SPARQL query, a language of higher expressiveness than FO is required in order to formulate a perfect rewriting; in practical terms, it is not in general possible to formulate a rewriting in SPARQL 1.0. This issue motivates our exploration in Chapters 4 and 5 of the problem of identifying ontological constraints that allow a rewriting in SPARQL 1.1. In the next section, we study query rewriting for restricted forms of RPSs which guarantee a query reformulation in FO-queries, and therefore in SPARQL 1.0 queries.

3.5.2 SPARQL 1.0 rewriting for restricted RPSs

Proposition 3.2. *Given an RPS $\mathcal{P} = (\mathcal{S}, G, E)$ and a Boolean BGP SPARQL query q, if G can be translated into either a linear, sticky, or sticky-join set of TGDs, then we can generate a set of Boolean BGP SPARQL queries $\{q_{\mathcal{P}1}, q_{\mathcal{P}2}, \ldots, q_{\mathcal{P}n}\}$, such that $q_{\mathcal{P}1}^D \cup q_{\mathcal{P}2}^D \cup \cdots \cup q_{\mathcal{P}n}^D = q^J$, where D is a stored database and J is the universal solution for $\mathcal{P}$ based on D.*

Proof. This follows directly from the fact that: *i)* we can rewrite a BCQ into a union of BCQs under either a linear, sticky, or sticky-join set of TGDs [31], and *ii)* the set of TGDs for the equivalence mappings E is linear. □

According to Proposition 3.2, given a BGP SPARQL query q and an RPS $\mathcal{P}$ such that the TGDs for graph mapping assertions of $\mathcal{P}$ that enjoy FO-rewritability, we are able to formulate the perfect rewriting q' of q with respect to $\mathcal{P}$ such that q' can be posed over the stored data and the set of certain answers will be retrieved. This is illustrated by the following example:

Example 3.5.1. Consider again the RPS $\mathcal{P}$ and the SPARQL query in Example 3.2.1. The set G of graph mapping assertions is linear as it is composed of a TGD with a single atom in the head. Hence, following from Proposition 3.2, we can

generate an FO-rewriting of a given Boolean query to entail the mapping assertions of the RPS. Listing 3.2 shows the rewriting based on the SPARQL query and the RPS of Example 3.2.1. To compute the set of certain answers of the given query, first we generate the set of all the possible 2-tuples from the stored database, because the query has two answer variables. Then we iterate over each 2-tuple t and decide whether or not t is in the set of certain answers, by substituting t into the original SPARQL query q to obtain a Boolean query q_b (note that this is a polynomial-time reduction of the problem, since there are polynomially many k-tuples from the source database).

We now show how the algorithm rewrites the Boolean query

$$q : q() \leftarrow \mathit{triple}(\mathit{DB1{:}Spiderman}, \mathit{starring}, z) \wedge \mathit{triple}(z, \mathit{artist}, \mathit{DB1{:}Toby\_Maguire}) \\ \wedge \mathit{triple}(\mathit{DB1{:}Toby\_Maguire}, \mathit{age}, \text{"39"})$$

obtained by substituting $\langle \mathit{DB1{:}Toby\_Maguire}, \text{"39"} \rangle$ in place of the answer variables of the query in Example 3.2.1. For simplicity, we show the steps of the rewriting algorithm given as input only the following TGD

$$\sigma : \mathit{triple}(\mathit{foaf{:}Toby\_Maguire}, y, z) \rightarrow \mathit{triple}(\mathit{DB1{:}Toby\_Maguire}, y, z)$$

which is one of the six TGDs that arise from the equivalence mapping

$$\mathit{mboxfoaf{:}Toby\_Maguire} \equiv_e \mathit{DB1{:}Toby\_Maguire}.$$

During the rewriting step, the following query is produced

$$q' : \mathit{ans}_{q'}() \leftarrow \mathit{triple}(\mathit{DB1{:}Spiderman}, \mathit{starring}, z) \\ \wedge \mathit{triple}(z, \mathit{artist}, \mathit{DB1{:}Toby\_Maguire}) \wedge \mathit{triple}(\mathit{foaf{:}Toby\_Maguire}, \mathit{age}, \text{"39"})$$

By posing the SPARQL query q `UNION` q', we retrieve the certain answers.

```
#Original query
SELECT ?x ?y
WHERE { DB1:Spiderman starring ?z .
?z artist ?x .
?x age ?y }

#Boolean query: ask if the tuple
#(DB1:Toby_Maguire,"39") is in the query result.
ASK { DB1:Spiderman starring ?z .
?z artist DB1:Toby_Maguire .
DB1:Toby_Maguire age "39" }

Answer: <false>

#Rewritten query
ASK {{ DB1:Spiderman starring ?z .
?z artist DB1:Toby_Maguire .
DB1:Toby_Maguire age "39" }
UNION
{ DB1:Spiderman starring ?z .
?z artist DB1:Toby_Maguire .
foaf:Toby_Maguire age "39" }}

Answer: <true>
```

Listing 3.2: SPARQL Boolean query rewriting.

3.5.3 Peer-based Linked Data integration system

For the case study in Section 3.4 we leveraged the semantics of the RPS mapping assertions by materialising an integrated instance of the sources through the implementation and the execution of Algorithm 1. Here, we undertake a preliminary empirical evaluation of how the RPS behaves when the peer-to-peer integration is achieved by adopting a query rewriting approach instead of a data materialisation approach.

This section describes the main components of a middleware for LOD integration

that is based on SPARQL query rewriting as described above. In overview, our middleware exposes a unified view of heterogeneous RDF sources which are semantically linked with the RPS mapping assertions. A unified SPARQL endpoint accepts queries expressed in any source vocabulary. A SPARQL query rewriting engine rewrites the queries with respect to the semantic mappings of the RPS. The rewritten query is evaluated over the sources in a federated approach and the query result is presented to the user.

In this system, the query rewriting engine is composed of two sub-engines: *(i)* the *semantic integration* module generates a perfect rewriting of the user's query, that is, a query that returns, once evaluated, a sound and complete answer of the original query based on the semantic mappings in the RPS; *(ii)* the *query federation* module executes a second rewriting step exploiting the `SERVICE` clause of SPARQL 1.1, generating a federated query to be evaluated over multiple RDF sources.

The system design provides for *automated alignment* of the peer schemas. It extracts structural information from the sources, such as the sets of entities, predicates, classes etc. Then, it performs schema alignment and coreference resolution by: *(i)* retrieving mappings between sources, such as `owl:sameAs` or `VoID`[5] triples, and other semantic links between sources; *(ii)* generating new mappings, using existing ontology matching and instance linkage techniques, such as *Falcon-AO* [67]; *(iii)* translating these alignments into our peer mapping language; and *(iv)* storing the mappings in the RPS.

For an empirical evaluation, we implemented the algorithm TGD-rewrite taken from [54] as a Java application. The algorithm rewrites SPARQL queries under FO-rewritable sets of peer mappings. In addition, we implemented two optimisations of the rewriting application. The first optimisation executes a pruning of all the SPARQL disjuncts with triple patterns that cannot provide a successful graph pattern match,

[5] `http://www.w3.org/TR/void/`

that is, triple patterns whose IRIs are not contained in one single source. With the second optimisation, the rewriting application ignores the equivalence mappings during the rewriting steps, since they lead to a production of SPARQL disjuncts that grow exponentially with respect to the number of mapping assertions. Instead, we treat equivalence mappings as `sameAs` triples which are stored externally on a Virtuoso[6] server and are accessed through a SPARQL endpoint. We then perform a reflexive, symmetric and transitive closure on the stored `sameAs` triples. The SPARQL query is then rewritten in order to leverage the stored `sameAs` triples to retrieve a complete answer with respect to the equivalence mappings.

Regarding query federation, triple patterns in the body of the query are grouped with respect to the RDF sources that can provide a successful graph pattern match. Then, the groups are assigned to the endpoints of the related sources, and evaluated using the SPARQL 1.1 `SERVICE` clause. Finally, the results are presented to the user. An example of the whole rewriting procedure is shown below.

Empirical Evaluation

The goal of the empirical evaluation is to provide a study of the behaviour of the current version of the framework with the aim of *(i)* ensuring that our framework can be used in its restricted version, *(ii)* analysing basic performance in terms of query execution time, and *(iii)* detecting current weaknesses of our framework to suggest future developments. We select three large-scale datastores with overlapping vocabularies in the domain of movies: *DBpedia*, *Linked Movie Database* and *Fact Forge*. The current version of our middleware is a Java application that takes as input a BGP SPARQL query, generates the SPARQL rewriting and executes the rewritten federated query over the selected datastores using Apache Jena[7].

[6] `https://en.wikipedia.org/wiki/Virtuoso_Universal_Server`
[7] `https://jena.apache.org/`

We performed a partial semantic alignment of the DBpedia, Linked Movie Database and Fact Forge schemas, defining a set of FO-rewritable one-to-one mappings for similar classes and predicates, adopting the RPS mapping language. For instance, we define a mapping of the form

$$x, y \leftarrow (x, \texttt{linkedmdb:actor}, y) \rightsquigarrow x, y \leftarrow (x, \texttt{dbpedia:starring}, y)$$

to express a 1:1 *predicate mapping* from the IRI `linkedmdb:actor` to the IRI `dbpedia:starring`, and a mapping of the form

$$x \leftarrow (x, \texttt{rdf:type}, \texttt{ff:Person}) \rightsquigarrow x \leftarrow (x, \texttt{rdf:type}, \texttt{foaf:Person})$$

to express a 1:1 *class mapping* from the IRI `ff:Person` to the IRI `foaf:Person`, leveraging the semantics of the built-in RDF predicate `rdf:type` for the class mappings. Also, we retrieved a subset of the `sameAs` triples from the sources and we generated new triples so as to encode the reflexive, symmetric and transitive closure of the `sameAs` binary relation; this provides co-reference resolution of IRIs as explained in the previous section. The peer mappings obtained present arbitrary topologies and include some mapping cycles.

To conduct our tests, we generated a set of SPARQL queries with up to three triple patterns in the body. We then evaluated the queries over the three endpoints of the datastores in order to obtain our baselines. Finally, we executed the queries on our middleware, and we compared the number of results retrieved and the query execution time. The results are shown in Figure 3.8 and allow us to derive two main insights. As expected, the amount of information retrieved increases significantly by adopting our system, due to its provision of interoperability between heterogeneous vocabularies. In addition, the approach does not compromise query execution time, since overall the response time of our system can be seen as an average of the query

response time over the single datastores. In fact, using the RPS can sometimes be faster than using just one single source endpoint. This may be due to reduction of the number of joins performed by Jena, by submitting smaller queries to each endpoint.

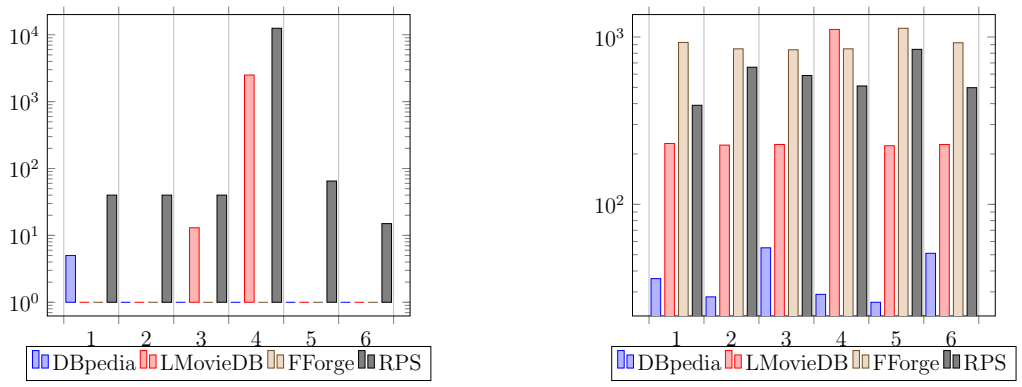

Figure 3.8: Number of results on the left and query execution time in milliseconds on the right (logarithmic scales). Queries 1 - 6 shown on the x axes.

3.6 Discussion

In this chapter we have proposed a formalisation of the notion of a P2P semantic integration system that achieves decidability of the query answering problem when reasoning over a set of P2P mappings. The mappings between peers comprise schema-level mappings as well as equality constraints which entail the semantics of the OWL property `sameAs`. We have shown that the semantics of the mappings preserve tractability of the conjunctive SPARQL query answering problem over the set of peer mappings. We have also illustrated how our technique can be applied in a real-world case study relating to obtaining careers guidance. The key novelty of the RPS is to achieve a tractable semantics for the integration of multiple RDF sources with arbitrary mapping topologies, whereas existing techniques applied on the same set of RDF sources may give rise to a set of undecidable rules.

Next, we have addressed the problem of SPARQL query rewriting under RPSs. We have firstly compared the problem with CQ rewriting under TGDs and we have seen that is not possible to generate a SPARQL 1.0 query as the perfect rewriting of an input BGP SPARQL query under general RPSs, as the RPS peer mappings are in general not FO-rewritable rules. Following this, we have taken into account well-known FO-rewritable sets of TGDs and compared them to the peer mappings; thus we have outlined some restricted forms of RPSs for which it is possible to generate a SPARQL 1.0 query as perfect rewriting.

Finally, we have presented a middleware system based on this rewriting technique and we have undertaken an empirical evaluation of its behaviour. When SPARQL queries are rewritten according to our algorithm and posed over multiple RDF sources, the amount of information retrieved increases significantly due to its provision of interoperability between heterogeneous vocabularies. In addition, the approach does not seem to compromise query execution time, since overall the response time of our system was seen to be not greater than the maximum query response time over the single datastores. However, clearly further empirical evaluation is necessary, and this is an area of future work.

In this chapter we have focussed on the FO-rewritability of the mapping rules. However, with the adoption of the more recent standard SPARQL 1.1 and its property paths we are able to extend the expressivity beyond FO by including regular expressions in the body of the target SPARQL queries. Following this direction, in the next two chapters we will step away from the language of RPSs to conduct study on C2RPQ-rewritability under a broader ontology language. Ultimately, the results of Chapter 4 and 5 could be used to potentially find larger subsets of SPARQL 1.1-rewritable RPSs than those illustrated here.

Chapter 4

Rewriting of IQs to C2RPQs under harmless linear $\mathcal{ELHI}$ Description Logic

In this chapter we define the description logic *harmless linear* $\mathcal{ELHI}$, an ontology language that generalises both DL-Lite$_\mathcal{R}$ and linear $\mathcal{ELH}$. Extending DL-Lite$_\mathcal{R}$ with qualified existential quantification on the left-hand side of concept inclusion axioms is equivalent to allowing inverse roles in role inclusion axioms in linear $\mathcal{ELH}$ ($\mathcal{ELH}^{\ell in}$), resulting in $\mathcal{ELHI}^{lin}$. Allowing inverse roles in $\mathcal{ELH}^{\ell in}$ is shown in [91] to result in PTIME-completeness of CQ answering with respect to data complexity; therefore a rewriting in C2RPQs for this language is not feasible — if, as normally assumed, NLOGSPACE is a proper subclass of PTIME — since the data complexity of answering C2RPQs is in NLOGSPACE. In fact, inverse roles allow the encoding of a conjunction of concepts on the left hand side of axioms, which is known to lead to PTIME-hardness [91].

The intuition behind the *harmlessness* property is that, given two role names R_1 and R_2, whenever $\exists R_2.A_2$ is on the left-hand side of an axiom, if there is another

axiom where R_1 appears on the left-hand side and A_2 on the right-hand side, then the harmlessness property prevents the entailment of either $R_1 \sqsubseteq R_2^-$ or $R_2 \sqsubseteq R_1^-$. This is achieved by preventing any sequence of role inclusions between R_1 and R_2 where an even number of inverse roles appear.

In this chapter we show that

(a) the harmlessness property prevents the simulation of conjunctions of concepts on the left hand side of concept inclusion axioms;

(b) the harmlessness property allows the possibility of answering IQs by query rewritings into 2RPQs.

This chapter is structured as follows. In Section 4.1 we give an introduction to the work presented in this second part of the thesis, with motivations and examples. In Section 4.2 we introduce the DL $\mathcal{ELHI}^{\ell in}$ and preliminaries. In Section 4.3 we define the harmless linear $\mathcal{ELHI}$ ($\mathcal{ELHI}_h^{\ell in}$), an ontology language that generalises both DL-Lite$_\mathcal{R}$ and linear $\mathcal{ELH}$. Then, in Section 4.4 we prove the rewritability of *instance queries* (queries with a single atom in their body) under $\mathcal{ELHI}_h^{\ell in}$ knowledge bases with C2RPQs as the target language, presenting a query rewriting algorithm that makes use of non-deterministic finite-state automata. The chapter ends with a discussion of our contributions in Section 4.5.

4.1 Motivations

We recall from Chapter 2 that a DL knowledge base consists of a TBox (the *terminological* component) and an ABox (the *assertional* component). The former is a conceptual representation of the schema, while the latter is an instance of the schema. A common assumption in the context of DLs is the so-called *open-world*

assumption, namely that the information in the ABox is sound but not complete; the TBox, in particular, specifies how the ABox can be expanded with additional information in order to answer queries. Answers to a query in this context are called *certain answers*, as they correspond to the answers that are true in all models of the theory constituted by the knowledge base [74]. Informally, this corresponds to *cautious* reasoning, and contrasts with *bold* reasoning where an answer is returned if it is entailed by at least one model. The set of all models (which is not necessarily finite) is represented by the so-called *expansion* (or *chase* [33], see Section 2.1.1) of an ABox $\mathcal{A}$ according to a TBox $\mathcal{T}$; this is illustrated in the following example.

Example 4.1.1. Consider the TBox $\mathcal{T}$ comprising the assertions $C \sqsubseteq A$ and $A \sqsubseteq \exists S.C$, where C and A are concepts. The concept $\exists S.C$ denotes the objects connected via the role S to some object belonging to the concept C; in other words, it contains all x such that $S(x, y)$ and $C(y)$ for some y. The first assertion means that every object in the class C is also in A; the second means that every object in the class A is also in the class represented by $\exists S.C$. Now suppose we have the ABox $\mathcal{A} = \{A(a)\}$; we can *expand* $\mathcal{A}$ according to the TBox $\mathcal{T}$ so as to add to it all atoms entailed by $(\mathcal{T}, \mathcal{A})$; we therefore add $S(a, z_0)$ and $C(z_0)$, where z_0 is a so-called *labelled null*, that is, a placeholder for an unknown value of which we know the existence (note that, with this approach, $\mathcal{A}$ can be expanded further). Given the query $\boldsymbol{q}$ defined as $q(x) \leftarrow S(x, y)$, the answer to $\boldsymbol{q}$ under $(\mathcal{T}, \mathcal{A})$ is $\{a\}$ because $S(a, z_0)$ is entailed by $(\mathcal{T}, \mathcal{A})$; in fact, the certain answers to $\boldsymbol{q}$ are obtained by evaluating $\boldsymbol{q}$ on the expansion and by considering answers that do not contain nulls. If we consider the query $\boldsymbol{q}_1$ defined as $q_1(x) \leftarrow C(x)$, the answer is empty because z_0, though known to exist, is not known.

Answers to queries over DL knowledge bases can be computed, for certain languages, by *query rewriting* [54]. In query rewriting, a new query $\boldsymbol{q}'$ is computed (rewritten) from the given query $\boldsymbol{q}$ according to the TBox $\mathcal{T}$, such that the answers to $\boldsymbol{q}$ on

$\mathcal{K} = (\mathcal{T}, \mathcal{A})$ are obtained by evaluating $\boldsymbol{q}'$ on $\mathcal{A}$; it is said that $\boldsymbol{q}$ is *rewritten* into $\boldsymbol{q}'$ and that $\boldsymbol{q}'$ is the *perfect rewriting of* $\boldsymbol{q}$ *with respect to* $\mathcal{T}$. The language of $\boldsymbol{q}'$, called the *target language*, can be more expressive than that of $\boldsymbol{q}$. A common rewriting technique for DLs and other knowledge representation formalisms, inspired by resolution in Logic Programming, has as the target language *unions of conjunctive queries* [33].

Example 4.1.2. Let us consider again the knowledge base of Example 4.1.1. The perfect rewriting of query $\boldsymbol{q}$ is the query $\boldsymbol{q}'$ defined as $q(x) \leftarrow A(x) \cup S(x, y)$; intuitively, $\boldsymbol{q}'$ captures the fact that, to search for objects from which some other object is connected via the role S, we need also to consider objects in A, because the TBox might infer the former from the latter objects. The evaluation of $\boldsymbol{q}'$ on $\mathcal{A}$ returns the correct answer.

The OWL 2 QL profile of the OWL 2 Web Ontology Language — which is based on the family of description logics called DL-Lite$_{\mathcal{R}}$ [5] — is expressly designed so that query answering can be performed via query rewriting. Data (assertions) that are stored in a standard relational database can be queried through an ontology by rewriting the query into an SQL query that is then answered by the RDBMS, without any changes to the data (for example, a tractable rewriting was presented in [34]).

Extending the expressivity of DL-Lite$_{\mathcal{R}}$ may lead to the need for a more expressive target language than SQL, i.e. than first order (FO) queries. This occurs, for example, when *qualified existential quantification* is allowed on the left hand side (LHS) of axioms, i.e., formulae of the form $\exists R.D$ where R is a role and D a concept. In this case, we say that the language is not *FO-rewritable*. The following example illustrates this issue.

Example 4.1.3. Consider the TBox $\mathcal{T} = \{\exists R.A \sqsubseteq A\}$ and the query $\boldsymbol{q}$ defined as $q(x) \leftarrow A(x)$. Note that an expression of the form $\exists R.A$ is forbidden on the

left hand of the axioms in DL-Lite$_\mathcal{R}$. It is easy to see that the query rewriting technique described earlier produces an infinite union of conjunctive queries: $q(x) \leftarrow A(x)$, $q(x) \leftarrow R(x, y), A(y)$ and all conjunctive queries of the form $q(x) \leftarrow R(x, y_1), \ldots, R(y_k, y_{k+1}), A(y_{k+1})$, with $k \geqslant 1$. This cannot be captured by an FO-rewriting.

However, as mentioned in Chapter 3, by adopting the semantic web query language SPARQL 1.1 [64], database systems should be able to answer queries that are more expressive than FO queries since the *property paths* of SPARQL 1.1 are able to express navigational queries by defining regular expressions on predicates. In particular, every conjunctive two-way regular path query (C2RPQ) [38] can be translated to a SPARQL 1.1 query. Building on this, in this thesis we propose a language that extends DL-Lite$_\mathcal{R}$ but still allows query answering via a simple rewriting mechanism, with C2RPQs instead of SQL queries as the target language. We allow qualified existential quantification on the LHS of axioms and identify a property of the resulting language that allows a rewriting into C2RPQs. The description logic resulting from this extension, which we call *harmless linear* $\mathcal{ELHI}$, denoted by $\mathcal{ELHI}_h^{lin}$, is a generalisation of both DL-Lite$_\mathcal{R}$ [5] and linear $\mathcal{ELH}$ [91].

Example 4.1.4. Recall the issue in the previous example, where a finite FO-rewriting was not feasible. In order to capture the infinite FO-rewriting, we can produce a rewriting into a C2RPQ $\boldsymbol{q'}$ defined as $q(x) \leftarrow R^*(x, y), A(y)$, where R^* is a regular expression denoting all finite compositions of R with itself.

The work described in this chapter of the thesis extends our recent work [48] where we first proposed exploiting the capabilities of navigational queries in order to allow query rewriting of conjunctive queries into CRPQs (not C2RPQs) under a more restrictive DL, namely linear $\mathcal{ELH}$. We also give here a complete theoretical development and full proofs.

4.2 Preliminaries

We first introduce the DL $\mathcal{ELHI}^{\ell in}$ [94, 90], from which the logic that we propose in this thesis is derived. $\mathcal{ELHI}^{\ell in}$ is itself derived from the $\mathcal{EL}$ language (which is the core of the OWL 2 EL profile), extended with the additional features of inverse roles ($\mathcal{I}$) and role inclusion axioms ($\mathcal{H}$), but disallowing conjunction of concepts on the left-hand side of concept inclusion axioms.

The syntax of $\mathcal{ELHI}^{\ell in}$ is as follows. The alphabet contains three pairwise disjoint and countably infinite sets of *concept names* A, *role names* R, and *individual names* I. The alphabet also contains a set of *roles* P, such that each $P \in \mathsf{P}$ is either a role name R or its *inverse*, denoted by R^-. A *complex concept* C is constructed from a special primitive concept $\top$ ('top'), concept names and role names using the following production rules:

$$C ::= \quad A \mid \exists P.C \mid \exists P.\top$$

where $A \in \mathsf{A}$ and $P \in \mathsf{P}$. The set of complex concepts is denoted by C. The alphabet includes two additional sets of *negated complex concepts* E and *negated roles* Q constructed using the following production rules:

$$\begin{aligned} D ::= &\quad A \mid \exists P.\top \\ E ::= &\quad D \mid \neg D \\ Q ::= &\quad P \mid \neg P \end{aligned}$$

Here a restricted form of complex concept is defined as part of the definition of a negated complex concept. Negation on concepts is only applied to elements in D as complex concepts of the form $\exists P.A$ are not allowed on the right-hand side of inclusions. In $\mathcal{ELHI}^{\ell in}$, a TBox $\mathcal{T}$ is a finite set of *concept* and *role inclusion axioms* of the form

$$C \sqsubseteq E \qquad \text{and} \qquad P \sqsubseteq Q.$$

We call *positive inclusions* (PIs) assertions of the form $C \sqsubseteq D$ or $P_1 \sqsubseteq P_2$ and *negative inclusions* (NIs) assertions of the form $C \sqsubseteq \neg D$ or $P_1 \sqsubseteq \neg P_2$.

This language could be enhanced with the capability of allowing qualified existential quantification on the right-hand side of inclusion assertions on concepts. However, this can be simulated by making use of inclusions between roles and unqualified existential quantification of concepts in inclusions between concepts. For instance, the assertion $A \sqsubseteq \exists R.B$ can be simulated by $A \sqsubseteq \exists R'$, $\exists R'^{-} \sqsubseteq B$ and $R' \sqsubseteq R$, where R' is a new role name. Therefore, in this thesis, we do not explicitly consider qualified existential quantification on the RHS of assertions.

Here we adopt the semantics of DLs defined in terms of interpretations [94, 90]. An *interpretation* $\mathcal{I}$ is a pair $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ that consists of a non-empty countable infinite *domain of interpretation* $\Delta^{\mathcal{I}}$ and an *interpretation function* $\cdot^{\mathcal{I}}$ which assigns *(i)* an element $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$ to each individual name a, *(ii)* a subset $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ to each concept name $A \in \mathsf{A}$ and *(iii)* a binary relation $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ to each role name $R \in \mathsf{R}$. We adopt the *unique name assumption* (UNA), so distinct individuals are assumed to be interpreted by distinct elements in $\Delta^{\mathcal{I}}$. The interpretation function $\cdot^{\mathcal{I}}$ for $\mathcal{ELHI}^{\ell in}$ is extended inductively to complex concepts with the following definitions.

$$\begin{aligned}
(R^{-})^{\mathcal{I}} &= \{(v,u) \mid (u,v) \in R^{\mathcal{I}}\} \\
(\neg P)^{\mathcal{I}} &= (\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}) \setminus P^{\mathcal{I}} \\
\top^{\mathcal{I}} &= \Delta^{\mathcal{I}} \\
(\exists P.\top)^{\mathcal{I}} &= \{u \mid \text{there is a } v \text{ such that } (u,v) \in P^{\mathcal{I}}\} \\
(\neg D)^{\mathcal{I}} &= \Delta^{\mathcal{I}} \setminus D^{\mathcal{I}} \\
(\exists P.C)^{\mathcal{I}} &= \{u \mid \text{there is a } v \in C^{\mathcal{I}} \text{ such that } (u,v) \in P^{\mathcal{I}}\}
\end{aligned}$$

The satisfaction relation $\models$ for inclusions and assertions is defined similarly to the

more general case of the $\mathcal{ALCHI}$ DL discussed in Section 2.1.3:

$$\begin{aligned}
\mathcal{I} \models C \sqsubseteq E \quad &\text{if and only if} \quad C^{\mathcal{I}} \subseteq E^{\mathcal{I}} \,, \\
\mathcal{I} \models P \sqsubseteq Q \quad &\text{if and only if} \quad P^{\mathcal{I}} \subseteq Q^{\mathcal{I}} \,, \\
\mathcal{I} \models C(a) \quad &\text{if and only if} \quad a^{\mathcal{I}} \in C^{\mathcal{I}} \,, \\
\mathcal{I} \models P(a,b) \quad &\text{if and only if} \quad (a^{\mathcal{I}}, b^{\mathcal{I}}) \in P^{\mathcal{I}} \,.
\end{aligned}$$

An interpretation $\mathcal{I}$ is a *model* of a knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, written $\mathcal{I} \models \mathcal{K}$, if it satisfies all concept and role inclusions of $\mathcal{T}$ and all concept and role assertions of $\mathcal{A}$. A knowledge base is *satisfiable* if admits at least one model.

Now, we first convert $\mathcal{ELHI}^{lin}$ TBoxes to a normal form, taken from [6]:

Definition 4.1. An $\mathcal{ELHI}^{lin}$ TBox is said to be in *normal form* if each of its concept inclusions and role inclusions is of one of the following forms:

$$\begin{aligned}
&A_1 \sqsubseteq A_2, \quad A_1 \sqsubseteq \neg A_2, \quad \exists R.\top \sqsubseteq A, \quad \exists R.A_1 \sqsubseteq A_2, \\
&A \sqsubseteq \exists R.\top, \quad R_1 \sqsubseteq R_2, \quad R_1 \sqsubseteq \neg R_2, \quad R_1 \sqsubseteq R_2^-,
\end{aligned}$$

where $A, A_1, A_2 \in \mathsf{A}$ and $R, R_1, R_2 \in \mathsf{R}$.

The inclusions excluded by the normal form are the following: *(i)* concept inclusion axioms with $\exists R^-.A_1$ on LHS; *(ii)* concept inclusion axioms with $\exists R^-.\top$ either on LHS or RHS; *(iii)* role inclusion axioms with R^- on LHS; *(iv)* negative inclusions with either $\neg R^-$ or $\neg\exists R^-.\top$ on RHS; *(v)* concept inclusion axioms with complex concepts of the form $\exists P_1.P_2.P_3.\ldots.P_n.\top$ or $\exists P_1.P_2.P_3.\ldots.P_n.A$ on LHS, where P_i is ether R_i or R_i^-.

Theorem 4.1. *Every $\mathcal{ELHI}^{lin}$ TBox $\mathcal{L}$ can be transformed into a TBox $\mathcal{L}'$ in normal form such that the size of $\mathcal{L}'$ is linear in the size of $\mathcal{L}$, and $\mathcal{L}$ and $\mathcal{L}'$ are equivalent in the following sense:*

- *every model of $\mathcal{L}$ can be extended to a model of $\mathcal{L}'$ by defining interpretations of the role and concept names that are in $\mathcal{L}'$ but not in $\mathcal{L}$,*
- *every model of $\mathcal{L}'$ is a model of $\mathcal{L}$.*

Proof. The claim follows by showing that each axiom that is not in normal form can be encoded by a set of axioms of linear size. A concept inclusion axiom of the form $\exists R_1.R_2.R_3.A_3 \sqsubseteq B$ is encoded by the following concept inclusion axioms in normal form: $\exists R_3.A_3 \sqsubseteq A_2$, $\exists R_2.A_2 \sqsubseteq A_1$ and $\exists R_1.A_1 \sqsubseteq B$. Every concept inclusion axiom that presents a role R^-, with $R \in \mathsf{R}$, can be encoded with a concept inclusion axiom and a role inclusion axiom. The concept inclusion axiom is obtained by replacing R^- with a fresh role name R_*, and the role inclusion axiom is of the form $R \sqsubseteq R_*^-$. For instance, a concept inclusion axiom of the form $A \sqsubseteq \exists S^-.B$ is encoded by $A \sqsubseteq \exists S_*.B$ and $S \sqsubseteq S_*^-$. Every role inclusion axiom of the form $R_1^- \sqsubseteq R_2$ is equivalent to the role inclusion axiom of the form $R_1 \sqsubseteq R_2^-$. □

Now we need to introduce the notion of a *canonical model* of a KB. Given an $\mathcal{ELHI}^{lin}$ knowledge base $(\mathcal{T}, \mathcal{A})$ with $\mathcal{T}$ in normal form, it is possible to find all answers to a CQ $\boldsymbol{q}$ over this KB by evaluating $\boldsymbol{q}$ over the (possibly infinite) canonical model which can be constructed using the chase procedure. We begin by defining the *base model* of a given ABox, following from [72].

Definition 4.2 (Base model)**.** The *base model* $\mathcal{I}_\mathcal{A}$ of the ABox $\mathcal{A}$ is defined as follows:

(1) $\Delta^{\mathcal{I}_\mathcal{A}} = \mathsf{ind}(\mathcal{A})$;

(2) $a^{\mathcal{I}_\mathcal{A}} = a$, for $a \in \mathsf{ind}(\mathcal{A})$;

(3) $A^{\mathcal{I}_\mathcal{A}} = \{a \mid A(a) \in \mathcal{A}\}$, for each concept name A;

(4) $R^{\mathcal{I}_\mathcal{A}} = \{(a, b) \mid R(a, b) \in \mathcal{A}\}$, for each role name R.

We then use the base model to generate the canonical model of a KB, again following from [72].

Definition 4.3 (Canonical model)**.** To build the canonical model for an $\mathcal{ELHI}^{\ell in}$ KB $\mathcal{K} = (\mathcal{A}, \mathcal{T})$, where $\mathcal{T}$ is in normal form, we take the base model $\mathcal{I}_\mathcal{A}$ as $\mathcal{I}_0$ and the following rules are applied inductively to obtain $\mathcal{I}_{k+1}$ from $\mathcal{I}_k$:

(i) if $d \in A^{\mathcal{I}_k}$ then d is added to $A^{\mathcal{I}_{k+1}}$;

(ii) if $(d, d') \in R^{\mathcal{I}_k}$ then (d, d') is added to $R^{\mathcal{I}_{k+1}}$;

(iii) if $d \in A_1^{\mathcal{I}_k}$ and $A_1 \sqsubseteq A_2 \in \mathcal{T}$, then d is added to $A_2^{\mathcal{I}_{k+1}}$;

(iv) if $(d, d') \in R_1^{\mathcal{I}_k}$ and $R_1 \sqsubseteq R_2 \in \mathcal{T}$, then (d, d') is added to $R_2^{\mathcal{I}_{k+1}}$;

(v) if $(d, d') \in R_1^{\mathcal{I}_k}$ and $R_1 \sqsubseteq R_2^- \in \mathcal{T}$, then (d', d) is added to $R_2^{\mathcal{I}_{k+1}}$;

(vi) if $d \in (R.D)^{\mathcal{I}_k}$ and $\exists R.D \sqsubseteq A \in \mathcal{T}$, where D is a concept name or $\top$, then d is added to $A^{\mathcal{I}_{k+1}}$;

(vii) if $d \in A^{\mathcal{I}_k}$, $A \sqsubseteq \exists R.\top \in \mathcal{T}$ and $d \notin (\exists R.\top)^{\mathcal{I}_k}$then (d, d') is added to $R^{\mathcal{I}_{k+1}}$, where d' is a *fresh* labelled null.

(viii) if $d \in A^{\mathcal{I}_k}, d \in B^{\mathcal{I}_k}$ and $A \sqsubseteq \neg B \in \mathcal{T}$ then HALT;

(ix) if $(d, d') \in R^{\mathcal{I}_k}, (d, d') \in S^{\mathcal{I}_k}$, and $R \sqsubseteq \neg S \in \mathcal{T}$ then HALT.

We then take a fixpoint interpretation, as $k \to \infty$. If HALT is reached, then the KB is not *satisfiable*, which means $\mathcal{K}$ does not admit any model. If HALT is not reached, then $\mathcal{K}$ is satisfiable and therefore it admits at least one model. If $\mathcal{K}$ is satisfiable, then the resulting interpretation satisfies all the inclusions in $\mathcal{T}$ and all the assertions in $\mathcal{A}$ - i.e., it is a model for $\mathcal{K}$ - and is called the *canonical model* of $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, denoted by $\mathcal{J}_\mathcal{K}$.

A useful property of the canonical model is that, given a KB $(\mathcal{T}, \mathcal{A})$ a query $\boldsymbol{q}$ and a tuple $\boldsymbol{a}$, we have that $(\mathcal{T}, \mathcal{A}) \models \boldsymbol{q}(\mathbf{a})$ if and only if $\mathcal{J}_\mathcal{K} \models \boldsymbol{q}(\mathbf{a})$ ([33]). We point out that this procedure is extended from [72], by adding the rule *(vii)* for the axioms of the form $A \sqsubseteq \exists R.\top$. Also, we note that the above procedure is a reformulation of the chase [33] tailored for $\mathcal{ELHI}^{\ell in}$ axioms.

Treating negative inclusions

If $\mathcal{T}$ contains some NIs, query answering can be undertaken by treating NIs and PIs in $\mathcal{T}$ separately. Suppose that $\mathcal{T}_n$ is the set of all the NIs in $\mathcal{T}$ and $\mathcal{T}_p = \mathcal{T} \setminus \mathcal{T}_n$, then answering a CQ $\boldsymbol{q}$ over $(\mathcal{T}, \mathcal{A})$ is equivalent to answering $\boldsymbol{q}$ over $(\mathcal{T}_p, \mathcal{A})$ and then additionally checking that each $\sigma \in \mathcal{T}_n$ is satisfied in $(\mathcal{T}_p, \mathcal{A})$ [29]. Checking whether $\sigma \in \mathcal{T}_n$ is satisfied in $(\mathcal{T}_p, \mathcal{A})$ is done by checking that the following boolean conjunctive queries q_σ evaluate to false in $(\mathcal{T}_p, \mathcal{A})$:

$$q_\sigma() \leftarrow \exists x\ \gamma(x) \wedge \delta(x) \text{ if } \sigma \text{ is of the form } C \sqsubseteq \neg D \text{ and}$$
$$q_\sigma() \leftarrow \exists x, y\ \rho_1(x, y) \wedge \rho_2(x, y), \text{ if } \sigma \text{ is of the form } P_1 \sqsubseteq \neg P_2$$

where:

(a) $\gamma(x) = A(x)$ if $C = A$,
$\gamma(x) = \exists y R(x, y)$ if $C = \exists R.\top$,
$\gamma(x) = \exists y R(y, x)$ if $C = \exists R^-.\top$,
$\gamma(x) = \exists y R(x, y) \wedge A(y)$ if $C = \exists R.A$,
$\gamma(x) = \exists y R(y, x) \wedge A(y)$ if $C = \exists R^-.A$,

(b) $\delta(x) = A(x)$ if $D = A$,
$\delta(x) = \exists y R(x, y)$ if $D = \exists R.\top$,
$\delta(x) = \exists y R(y, x)$ if $D = \exists R^-.\top$,

(c) $\rho_i(x, y) = R_i(x, y)$ if $P_i = R$, and
$\rho_i(x, y) = R_i(y, x)$ if $P_i = R^-$.

It is easy to see that, given an interpretation $\mathcal{I}$, we have that $\mathcal{I} \models \sigma$ if and only if $\mathcal{I} \not\models q_\sigma$. As an immediate consequence, we have that NIs do not increase the complexity of CQ answering under $\mathcal{ELHI}^{\ell in}$, therefore NIs are not taken into account for the rewriting procedures presented in the rest of the thesis.

4.3 Harmless $\mathcal{ELHI}^{\ell in}$ Description Logic

Extending DL-Lite$_\mathcal{R}$ with qualified existential quantification on the left-hand side of concept inclusion axioms is equivalent to allowing inverse roles in role inclusion axioms in $\mathcal{ELH}^{\ell in}$, resulting in $\mathcal{ELHI}^{lin}$, defined in Section 4.2. Allowing inverse roles in $\mathcal{ELH}^{\ell in}$ is shown in [91] to result in PTIME-completeness of CQ answering with respect to data complexity; therefore a rewriting in C2RPQs for this language is not feasible — if, as normally assumed, NLOGSPACE is a proper subclass of PTIME — since the data complexity of answering C2RPQs is in NLOGSPACE. In fact, inverse roles allow the encoding of a conjunction of concepts on the left hand side of

axioms (as shown in the example below), which is known to lead to PTIME-hardness ([91], Theorem 4.3).

Example 4.3.1. Consider the ABox $\{C_1(c), C_2(c)\}$ and the TBox $\{C_1 \sqsubseteq \exists R.\top$, $\exists R^-.C_2 \sqsubseteq D, \exists R.D \sqsubseteq C_3\}$. Using the chase procedure, the first inclusion axiom generates the assertion $R(c, d_0)$, where d_0 is a fresh labelled null. Since c is now connected to d_0 via R and c is of type C_2, the second rule generates the assertion $D(d_0)$. Now, we have that c is connected to an element of type D via the role R, so the third rule generates the assertion $C_3(c)$. Thus, the TBox encodes the axiom $C_1 \sqcap C_2 \sqsubseteq C_3$.

In this chapter, our aim is to investigate the possibility of finding a sub-language of $\mathcal{ELHI}^{\ell in}$ whose CQ answering problem has NLOGSPACE data complexity. We do so by identifying a syntactic property that does not allow the above encoding of rules of the type $C_1 \sqcap C_2 \sqsubseteq C_3$. We then show that the resulting sublanguage allows a rewriting from CQs to C2RPQs, thus avoiding a polynomial blow-up.

We now define a syntactic property of $\mathcal{ELHI}^{\ell in}$ that identifies what we call *harmless* $\mathcal{ELHI}^{\ell in}$ TBoxes, denoted by $\mathcal{ELHI}_h^{\ell in}$. We first introduce some auxiliary definitions.

Definition 4.4. Let R and R' be two role names appearing in an $\mathcal{ELHI}^{\ell in}$ TBox $\mathcal{L}$ that is in normal form. If R, R' are two roles in $\mathcal{L}$ and there exist $R_0, \ldots, R_n$ such that *(i)* $R = R_0$, $R' = R_n$ and *(ii)* for $1 \leqslant i \leqslant n$ either $R_{i-1} \sqsubseteq R_i \in \mathcal{L}$ or $R_{i-1} \sqsubseteq R_i^- \in \mathcal{L}$, then:

(a) if the number of inverse roles R_i^- is even, we write $R \rightharpoonup_{\mathcal{L}} R'$;

(b) if the number is odd, we write $R \rightharpoonup_{\mathcal{L}} R'^-$.

The syntactic property defined above is equivalent to the semantic property of role inclusion with respect to an $\mathcal{ELHI}^{\ell in}$ TBox. This is stated in the following proposition.

Proposition 4.1. *Given two role names R, R' appearing in an $\mathcal{ELHI}^{\ell in}$ TBox $\mathcal{L}$ in normal form, we have:*

(a) $\mathcal{L} \models R \sqsubseteq R'$ if and only if $R \rightharpoonup_{\mathcal{L}} R'$, and

(b) $\mathcal{L} \models R \sqsubseteq R'^{-}$ if and only if $R \rightharpoonup_{\mathcal{L}} R'^{-}$.

The proof of the above proposition follows from the observation that the only way to infer a role inclusion in an $\mathcal{ELHI}^{\ell in}$ TBox is through the closure of role inclusion axioms of the form $R_1 \sqsubseteq R_2$.

We now define the *harmless* condition for two given roles appearing in an $\mathcal{ELHI}^{\ell in}$ TBox in normal form:

Definition 4.5. Let R and R' be two role names appearing in an $\mathcal{ELHI}^{\ell in}$ TBox $\mathcal{L}$ in normal form. If neither $R \rightharpoonup_{\mathcal{L}} R'^{-}$ nor $R' \rightharpoonup_{\mathcal{L}} R^{-}$, then we say that R and R' are *mutually harmless* roles with respect to $\mathcal{L}$.

We are now ready to define the class of *harmless* $\mathcal{ELHI}^{\ell in}$ TBoxes:

Definition 4.6. Given an $\mathcal{ELHI}^{\ell in}$ TBox $\mathcal{L}$ in normal form, $A_1, A_2 \in \mathsf{A}$, $R_1, R_2 \in \mathsf{R}$, we say that $\mathcal{L}$ is *harmless* if, whenever there is some $\exists R_2.A_2$ on the left-hand side of an axiom in $\mathcal{L}$, if there exists some axiom $\exists R_1.\top \sqsubseteq A_2$ or $\exists R_1.A_1 \sqsubseteq A_2$ in $\mathcal{L}$, then we have that R_1 and R_2 are mutually harmless roles with respect to $\mathcal{L}$. The language of all harmless $\mathcal{ELHI}^{\ell in}$ TBoxes is denoted by $\mathcal{ELHI}_h^{\ell in}$.

We note that each DL-Lite$_{\mathcal{R}}$ KB is also an $\mathcal{ELHI}_h^{\ell in}$ KB, as atoms of the form $\exists R.D$ are forbidden on the left hand side of DL-Lite$_{\mathcal{R}}$ TBoxes. Also, each $\mathcal{ELH}^{\ell in}$ KB is an $\mathcal{ELHI}_h^{\ell in}$ KB, since inverse roles are not included in $\mathcal{ELH}^{\ell in}$ and therefore roles are always harmless. Thus, $\mathcal{ELHI}_h^{\ell in}$ is a generalisation of both DL-Lite$_{\mathcal{R}}$ and $\mathcal{ELH}^{\ell in}$.

We note that $\mathcal{ELHI}_h^{\ell in}$ can also express rules with complex concepts of the form $\forall R.A$ on the right hand side (RHS) of concept inclusion axioms. For instance, we

can encode $B \sqsubseteq \forall R.A$ with the TBox $\mathcal{L} := \{\exists P.B \sqsubseteq A, R \sqsubseteq P^-\}$, where P is a fresh role name, and R and P are harmless roles with respect to $\mathcal{L}$.

In the following sections we show that

(a) the harmlessness property prevents the simulation of conjunctions of concepts on the left hand side of concept inclusion axioms;

(b) the harmlessness property allows the possibility of answering IQs by query rewritings into 2RPQs.

4.4 Rewriting Instance Queries into 2RPQs under $\mathcal{ELHI}_h^{\ell in}$

As shown in Example 4.1.3, it is not possible to generate a first order query as a perfect rewriting if we allow qualified existential quantification on the left hand side of concept inclusion axioms, even in the case where the input query is an instance query (IQ). In this section we present a technique that uses the expressive power of NFAs in order to rewrite IQs into 2RPQs under $\mathcal{ELHI}_h^{\ell in}$ TBoxes.

We first describe in Section 4.4.1 a query rewriting technique that is widely adopted in the Knowledge Representation and Databases literature [26, 33, 34], and inspired by Partial Evaluation in Logic Programming [76]. It produces the correct (*perfect* [33]) rewritings that compute exactly the set of certain answers to a given CQ.

However, such a technique in our case is not guaranteed to produce finite rewritings, and is therefore not directly usable. To address this issue, we propose in Section 4.4.2 a novel algorithm, which makes use of NFAs, that is able to rewrite IQs into 2RPQs under $\mathcal{ELHI}_h^{\ell in}$ TBoxes by encoding the possibly infinite steps of the above "basic" rewriting algorithm.

4.4.1 Rewriting Conjunctive Queries for $\mathcal{ELHI}_h^{lin}$ via resolution

In this section, we present a rewriting algorithm for CQs on $\mathcal{ELHI}_h^{lin}$ knowledge bases, which relies on a resolution-like procedure widely adopted in the literature [33, 26, 54].

We first formalise the notion of perfect rewriting of a CQ under a TBox into a C2RPQ. Given a CQ $\boldsymbol{q}$, we use the assertions of the TBox $\mathcal{T}$ to rewrite $\boldsymbol{q}$ into a C2RPQ $\mathbf{p}$ that returns, when evaluated over the data instance (ABox) $\mathcal{A}$, all the certain answers of $\boldsymbol{q}$ with respect to $(\mathcal{T}, \mathcal{A})$. The rewriting $\mathbf{p}$ only depends on the TBox $\mathcal{T}$ and the given query $\boldsymbol{q}$; it is independent of the ABox $\mathcal{A}$. In query processing, therefore, we use $\mathcal{A}$ only in the final step, when the rewriting is evaluated on it.

We call a CQ $\boldsymbol{q}$ and a TBox $\mathcal{T}$ *C2RPQ-rewritable* if there exists a C2RPQ $\mathbf{p}$ such that, for any ABox $\mathcal{A}$ and any tuple $\mathbf{a}$ of individuals in $\mathsf{ind}(\mathcal{A})$, we have

$$(\mathcal{T}, \mathcal{A}) \models \boldsymbol{q}(\mathbf{a}) \text{ if and only if } \mathcal{A} \models \mathbf{p}(\mathbf{a}).$$

In this case, we say that $\mathbf{p}$ is the *perfect C2RPQ rewriting* of $\boldsymbol{q}$ with respect to $\mathcal{T}$.

The algorithm described below, and listed as Algorithm 2, generates the perfect rewriting of a CQ under $\mathcal{ELHI}_h^{lin}$ as a set of CQs. This set of CQs is then interpreted as a *union of conjunctive queries* which can be evaluated over the ABox. The rewriting technique is based on two steps: a *reduction* step, which eliminates atoms in the query that are more specific than some other atom, and the actual *rewriting* step, which is similar to the resolution step in logic programming. We note that the algorithm might not terminate for $\mathcal{ELHI}_h^{lin}$ and we will show in Section 4.4.2 that our NFA-based rewriting captures all the rewriting branches produced by the algorithm, including infinite ones.

Following the approach of [34], a term of an atom in a query is said to be *bound* if it corresponds to (i) an answer variable, (ii) a *shared variable*, that is, a variable occurring at least twice in the query body, or (iii) a *constant*, that is, an element in I. Conversely, a term of an atom in a query is *unbound* if it corresponds to a non-shared existentially quantified variable. As is customary, we adopt the symbol '_' to represent an unbound term[1].

A set of atoms $A = \{a_1, \ldots, a_n\}$, where $n \geqslant 2$, *unifies* if there is a function $\phi : (\mathsf{V} \cup \mathsf{I}) \to (\mathsf{V} \cup \mathsf{I})$, called a *unifier* for A, such that (i) if $t \in \mathsf{I}$, then $\phi(t) = t$, and (ii) $a_1(\phi(t_1), \ldots, \phi(t_m)) = \cdots = a_n(\phi(t_1), \ldots, \phi(t_m))$. A *most general unifier* (MGU) for A is a unifier for A, denoted by γ_A, such that for each other unifier γ for A, there exists a substitution γ' such that $\gamma = \gamma' \circ \gamma_A$. The Reduce function in Algorithm 2 takes as input a conjunctive query q and a set of atoms S occurring in the body of q and returns a conjunctive query obtained by applying to q the *most general unifier* between the atoms of S. We note that, in unifying a set of atoms, each occurrence of the _ symbol is considered to be a different unbound variable.

We now recall from [34] the definition of when concept and role inclusion axioms are *applicable* to atoms in a query, as used in the rewriting step of Algorithm 2.

An axiom I is *applicable to an atom* $A(x_1, x_2)$ for $A \in \mathsf{A}$ if the RHS of I is A. An axiom I is *applicable to an atom* $R(x_1, x_2)$ for $R \in \mathsf{R}$ if (1) $x_2 =$ _ and the RHS of I is $\exists R.\top$; or (2) the RHS of I is either R or R^-. Informally, an axiom I is applicable to an atom g if the predicate of g is equal to the predicate in the right-hand side of I and, in the case when I is an inclusion assertion between concepts, if g has at most one bound argument and corresponds to the object that is implicitly referred to by the inclusion I.

Below we introduce a set of rewriting rules for the atoms in the body of a query.

[1]The underscore symbol '_' is commonly used in Logic Programming, where it is named "don't care". In the presence of multiple occurrences of "don't care" symbols in a formula, such symbols are to be considered as *distinct* existentially quantified variables.

Since the algorithm in [34] deals with DL-Lite TBoxes, we include here an additional rule, (c), so that the rewriting algorithm is capable of managing concept inclusion axioms where qualified existential quantifications appear on the left-hand side (we recall that $\mathcal{ELHI}_h^{lin}$ extends DL-Lite$_\mathcal{R}$ by allowing such existential quantification). Proof of correctness is shown in [34] when I is of the form $A_1 \sqsubseteq A_2$ — i.e., for rule (a) — and is analogous for the other forms listed below, including rule (c) [34]. On the other hand, adding rule (c) does not retain termination, which is guaranteed when the maximum number of atoms in the body of a conjunctive query generated by the algorithm is equal to the length of the initial query. It is easy to see that this condition is violated by applying rule (c), since it may lead to generating an infinite number of atoms, as we have seen above. We show in Section 4.4.2 in which cases the expressive power of NFAs finitely captures "regular patterns" of possibly infinite rewriting steps.

Let I be an inclusion axiom that is applicable to an atom g. The set of atoms obtained from g by applying I, denoted by $gr(g, I)$, is defined as follows:

(a) If $g = A_2(x_1, x_2)$ and $I = A_1 \sqsubseteq A_2$, then $gr(g, I) = \{A_1(x_1, x_2)\}$;

(b) If $g = A(x_1, x_2)$ and $I = \exists R.\top \sqsubseteq A$, then $gr(g, I) = \{R(x_1, \_)\}$;

(c) If $g = A_1(x_1, x_2)$ and $I = \exists R.A_2 \sqsubseteq A_1$, then $gr(g, I) = \{R(x_1, z), A_2(z, \_)\}$, where z is a fresh variable;

(d) If $g = R(x_1, \_)$ and $I = A_1 \sqsubseteq \exists R.\top$ then $gr(g, I) = \{A_1(x_1, \_)\}$;

(e) If $g = R_2(x_1, x_2)$ and $I = R_1 \sqsubseteq R_2$, then $gr(g, I) = \{R_1(x_1, x_2)\}$;

(f) If $g = R_2(x_1, x_2)$ and $I = R_1 \sqsubseteq R_2^-$, then $gr(g, I) = \{R_1(x_2, x_1)\}$.

The rewriting procedure that generates the perfect rewriting of q with respect to $\mathcal{T}$, denoted by $\mathsf{Rewrite}(q, \mathcal{T})$, is given by Algorithm 2. As described above, two steps (reduction and rewriting) are applied to each query in the set of rewritten queries

Algorithm 2: Algorithm $\mathsf{Rewrite}(q, \mathcal{T})$

```
Data: Conjunctive query q, TBox T.
Result: Union of conjunctive queries Q.
Q := {⟨q, 1⟩};
repeat
    Q' := Q ;
    foreach ⟨qr, x⟩ ∈ Q' do
        /* Reduction step                                          */
        if there exists I ∈ T such that I is not applicable to qr then
            foreach set of atoms S ⊆ body(qr) do
                if S unify then
                    Q := Q ∪ {⟨Reduce(qr, S), 0⟩}
        /* Rewriting step                                          */
        foreach axiom I ∈ T do
            if I is applicable to qr then
                qr' := rewrite qr according to I ;
                Q := Q ∪ {⟨qr', 1⟩}
until Q' = Q;
Q_fin := {q | ⟨q, 1⟩ ∈ Q} ;
return Q_fin
```

until a fixed point is reached. Note that the reduction step produces a query marked with '0' whilst the rewriting step marks queries with '1', and only queries marked with '1' are added to the output set. We adopt this approach to avoid redundancy in the output set, since a query marked with '0' is always semantically contained in a query marked with '1'.

Example 4.4.1. Consider applying the Rewrite algorithm to the query $\boldsymbol{q}$ defined by $q(x) \leftarrow R(x, y), R(\_, y)$ over the TBox $\{A \sqsubseteq \exists R.\top\}$, where $A \in \mathsf{A}$ and $R \in \mathsf{R}$. The atoms $R(x, y)$ and $R(\_, y)$ in $\boldsymbol{q}$ unify, so executing $\mathsf{Reduce}(\boldsymbol{q}, \{R(x, y), R(\_, y)\})$ yields the atom $R(x, y)$. The variable y is now unbound, so can be replaced by "$\_$". Now, the axiom $\{A \sqsubseteq \exists R.\top\}$ can be applied to $R(x, \_)$, whereas, before the reduction process, it could not be applied to any atom of the query. Following this, the rewriting step reformulates the query to $q(x) \leftarrow A(x, \_)$ which is added to the output set. For more details on the rewriting procedure, we refer readers to [34, 54].

Theorem 4.2. *Let $\mathcal{T}$ be an $\mathcal{ELHI}_h^{lin}$ TBox in normal form and $\boldsymbol{q}$ a conjunctive query over $\mathcal{T}$. Assume that $\mathsf{Rewrite}(\boldsymbol{q}, \mathcal{T})$ terminates and let PR be the union of conjunctive queries returned by $\mathsf{Rewrite}(q, \mathcal{T})$. Then for every ABox $\mathcal{A}$, $\boldsymbol{q}^{(\mathcal{T},\mathcal{A})} = \bigcup_{S \in PR} S^{(\varnothing,\mathcal{A})}$.*

Proof. From Lemma 39 in [34] we know that for every CQ $\boldsymbol{q}'$, DL-Lite TBox $\mathcal{T}'$ and ABox A', $\boldsymbol{q}'^{(\mathcal{T}',\mathcal{A}')} = \bigcup_{S \in \mathsf{Rewrite}(\boldsymbol{q}',\mathcal{T}')} S^{(\varnothing,\mathcal{A}')}$. The claim follows since the proof of correctness in [34] is shown for inclusion axioms of the form $A_1 \sqsubseteq A_2$ — i.e., for rule (a) — and is analogous for the other forms listed above, including rule (c). □

4.4.2 Rewriting Concept Instance Queries to 2RPQs for $\mathcal{ELHI}_h^{lin}$

We now show how to encode rewritings for concept IQs under an $\mathcal{ELHI}_h^{lin}$ TBox, by means of a finite-state automaton; intuitively, the automaton is able to encode infinite sequences of rewriting steps executed according to Algorithm 2. We focus here on the rewriting of concept IQs, since in the case of role IQs the rewriting can be computed in polynomial time by a simple check on sequences of role inclusions in the TBox. We state this with the following corollary of Theorem 4.2.

Corollary 4.4.1. *Let $\mathcal{L}$ be an $\mathcal{ELHI}^{lin}$ TBox in normal form and $\boldsymbol{q}_R$ a role IQ of the form $q(x, y) \leftarrow R(x, y)$ and $\boldsymbol{q}_{R^-}$ a role IQ of the form $q(x, y) \leftarrow R(y, x)$. Let $R_\mathcal{T}$ and $R_\mathcal{T}^-$ be two sets of roles such that $R' \in R_\mathcal{T}$ if and only if $R' \rightharpoonup_\mathcal{L} R$ and $R'^- \in R_\mathcal{T}^-$ if and only if $R' \rightharpoonup_\mathcal{L} R^-$. Then for every ABox A, it holds that*

$$\boldsymbol{q}_R^{(\mathcal{T},\mathcal{A})} = \boldsymbol{q}_R^{(\varnothing,\mathcal{A})} \cup \Big(\bigcup_{P \in (R_\mathcal{T} \cup R_\mathcal{T}^-)} \boldsymbol{q}_P^{(\varnothing,\mathcal{A})} \Big),$$
$$\boldsymbol{q}_{R^-}^{(\mathcal{T},\mathcal{A})} = \boldsymbol{q}_{R^-}^{(\varnothing,\mathcal{A})} \cup \Big(\bigcup_{P \in (R_\mathcal{T} \cup R_\mathcal{T}^-)} \boldsymbol{q}_{P^-}^{(\varnothing,\mathcal{A})} \Big)$$

Proof. We know that, given two role names R, R' appearing in an $\mathcal{ELHI}^{lin}$ TBox $\mathcal{L}$ in normal form, we have that $\mathcal{L} \models R \sqsubseteq R'$ if and only if $R \rightharpoonup_\mathcal{L} R'$, and $\mathcal{L} \models R \sqsubseteq R'^-$

if and only if $R \rightharpoonup_{\mathcal{L}} R'^{-}$. Then the claim immediately follows from Theorem 4.2. □

We now present the construction of an NFA based on an $\mathcal{ELHI}_h^{lin}$ TBox $\mathcal{T}$ and a concept name A. Theorem 4.3 proves that this NFA is capable of encoding the (possibly infinite) rewriting steps of Algorithm 2 for concept instance IQs.

Definition 4.7. Let $\mathcal{T}$ be an $\mathcal{ELHI}_h^{lin}$ TBox in normal form, Σ be the alphabet $\mathsf{P} \cup \mathsf{A}$, and A be a concept name appearing in $\mathcal{T}$. The *NFA-rewriting* of A with respect to $\mathcal{T}$, denoted by $\mathsf{NFA}^{-}_{A,\mathcal{T}}$, is the NFA $(Q, \Sigma, \delta, S_A, F)$ defined as follows:

(1) states S_A, SF_A and $S_\top$ are in Q, SF_A and $S_\top$ are in F, and transition (S_A, A, SF_A) is in δ; S_A is the initial state;

(2) for each $B \in \mathsf{A}$ that appears in at least one concept or role inclusion axiom of $\mathcal{T}$, states S_B and SF_B are in Q, SF_B is in F, and transition (S_B, B, SF_B) is in δ;

(3) for each concept inclusion axiom $\rho \in \mathcal{T}$:

(3.1) if ρ is of the form $B \sqsubseteq C$, where $B, C \in \mathsf{A}$, the transition (S_C, ϵ, S_B) is in δ;

(3.2) if ρ is of the form $B \sqsubseteq \exists R.\top$, where $B \in \mathsf{A}$ and $R \in \mathsf{R}$, for each transition $(S_X, R, S_\top) \in \delta$, the transition (S_X, ϵ, S_B) is in δ;

(3.3) if ρ is of the form $\exists R.\top \sqsubseteq B$, where $B \in \mathsf{A}$ and $R \in \mathsf{R}$, the transition $(S_B, R, S_\top)$ is in δ;

(3.4) if ρ is the form $\exists R.D \sqsubseteq C$, where $C, D \in \mathsf{A}$ and $R \in \mathsf{R}$, the transition (S_C, R, S_D) is in δ;

(4.1) for each role inclusion axiom $T \sqsubseteq S \in \mathcal{T}$ and each transition of the form $(S_C, S, S_B) \in \delta$, the transition (S_C, T, S_B) is in δ.

(4.2) for each role inclusion axiom $T \sqsubseteq S^{-} \in \mathcal{T}$ and each transition of the form $(S_C, S, S_B) \in \delta$ or $(S_C, S^{-}, S_B) \in \delta$, the transition (S_C, T^{-}, S_B) or (S_C, T, S_B) is in δ, respectively.

We now define a subclass of CQs called a *two-way simple path conjunctive queries.*

Definition 4.8. A conjunctive query $\boldsymbol{q}$ is a two-way simple path conjunctive query (2SPCQ) if it is of one of the following forms:

1. $q(x_1) \leftarrow A(x_1, x_2)$,
2. $q(x_1) \leftarrow \tau_{R_1}(x_1, y_1), \tau_{R_2}(y_1, y_2), \ldots, \tau_{R_n}(y_{n-1}, x_2)$, or
3. $q(x_1) \leftarrow \tau_{R_1}(x_1, y_1), \tau_{R_2}(y_1, y_2), \ldots, \tau_{R_n}(y_{n-1}, y_n), A(y_n, x_2)$,

where:

- $\tau_R(x, y) ::= R(x, y) \mid R(y, x)$;
- x_1, x_2 are answer variables, with $x_1 \neq x_2$;
- for each i, y_i is an existentially quantified variable and, for each j, $y_i \neq y_j$;
- for each i, $y_i \neq x_1$ and $y_i \neq x_2$;
- $n \geqslant 1$;
- $A \in \mathsf{A}$ and $R_1, \ldots, R_n \in \mathsf{R}$.

A 2SPCQ $\mathit{head}(\boldsymbol{q}) \leftarrow Z_1(x_0, x_1), \ldots, Z_n(x_{n-1}, x_n)$ is equivalent to a 2RPQ of the form $\mathit{head}(\boldsymbol{q}) \leftarrow Z_1 \cdots Z_n(x_0, x_n)$; thus, throughout the thesis we will use either the 2RPQ form or the CQ form of a 2SPCQ, whichever is more natural in the given context. Thus, given a 2SPCQ $\boldsymbol{q}$, $\mathit{path}(\boldsymbol{q})$ is $Z_1 \cdots Z_n$. For example, if $\boldsymbol{q}$ is $q(x_1) \leftarrow P(x_1, y_2), T(y_3, y_2), B(y_3, x_2)$, where P, T are role names and B is a concept, then $\mathit{path}(\boldsymbol{q})$ is $PT^- B$ and if $\boldsymbol{q}$ is $q(x_1) \leftarrow P(x_1, y_2), T(y_3, y_2)$, then $\mathit{path}(\boldsymbol{q})$ is $PT^- \top$.

As we illustrated in Example 4.1.3, Algorithm 2 may generate an infinite set of CQs. After the following example, we show that, given an input of a concept instance query and an $\mathcal{ELHI}_h^{lin}$ TBox, each output CQ is a 2SPCQ and the NFA-rewriting encodes all the possible outputs.

Example 4.4.2. Consider the TBox $\mathcal{T}$ defined by the following inclusion axioms: $\exists R.C \sqsubseteq \exists P.\top$, $\exists P.\top \sqsubseteq A$, $\exists P.\top \sqsubseteq B$, $\exists T.B \sqsubseteq C$, $\exists S.A \sqsubseteq A$ and $V \sqsubseteq T^-$, where P, R, S, T, V are role names and A, B, C are concept names. Consider now the query $\boldsymbol{q} = q(x) \leftarrow A(x, y)$. First, we transform $\mathcal{T}$ into normal form, $\mathcal{T}'$, by adding a fresh concept name X and by replacing $\exists R.C \sqsubseteq \exists P.\top$ by $\exists R.C \sqsubseteq X$ and $X \sqsubseteq \exists P.\top$. It is easy to see that $\mathsf{Rewrite}(\boldsymbol{q}, \mathcal{T}')$ runs indefinitely (for instance, we have an infinite loop when rule *(c)* is applied to the atom $A(x, y)$).

Let us consider the NFA rewriting of A with respect to $\mathcal{T}'$. We construct $\mathsf{NFA}^-_{A,\mathcal{T}'}$ (shown in Figure 4.1) as follows: by *(2)* in Definition 4.7 we have the transitions $(S_A, A, SF_A), (S_B, B, SF_B), (S_C, C, SF_C)$ and (S_X, X, SF_X); by *(3.3)* in Definition 4.7 and the inclusion assertions $\exists P.\top \sqsubseteq A$ and $\exists P.\top \sqsubseteq B$, we have the transitions $(S_A, P, S_\top)$ and $(S_B, P, S_\top)$; by *(3.2)* in Definition 4.7 and the inclusion assertion $X \sqsubseteq \exists P.\top$, we have the transitions (S_A, ϵ, S_X) and (S_B, ϵ, S_X); by *(3.4)* in Definition 4.7 and the inclusion assertions $\exists R.C \sqsubseteq X$, $\exists T.B \sqsubseteq C$ and $\exists S.A \sqsubseteq A$, we have the transitions (S_X, R, S_C), (S_C, T, S_B) and (S_A, S, S_A); finally, by *(4.2)* in Definition 4.7 and the inclusion assertion $V \sqsubseteq T^-$ we have the transition (S_C, V^-, S_B).

The language accepted by $\mathsf{NFA}^-_{A,\mathcal{T}'}$ can be described by the following regular expression: $S^*((A|P|X)|(((R(T|V^-))^*(P|B|X|RC))))$. It can be verified that all the infinite outputs of $\mathsf{Rewrite}(\boldsymbol{q}, \mathcal{T}')$ are of the form $q(x) \leftarrow \mathsf{NFA}^-_{A,\mathcal{T}'}(x, y)$. For instance, some rewritings of $\boldsymbol{q}$ are:

$$
\begin{aligned}
&q(x) \leftarrow P(x, y)\\
&q(x) \leftarrow S(x, z_1), A(z_1, y)\\
&q(x) \leftarrow S(x, z_1), S(z_1, z_2), A(z_2, y)\\
&q(x) \leftarrow S(x, z_1), S(z_1, z_2), P(z_2, y)\\
&q(x) \leftarrow R(x, z_1), T(z_1, z_2), R(z_2, z_3), C(z_3, y)\\
&q(x) \leftarrow R(x, z_1), V(z_2, z_1), R(z_2, z_3), C(z_3, y)
\end{aligned}
$$

It is easy to verify that each of these output queries is a 2SPCQ and that each path is in $L(\mathsf{NFA}^-_{A,\mathcal{T}'})$.

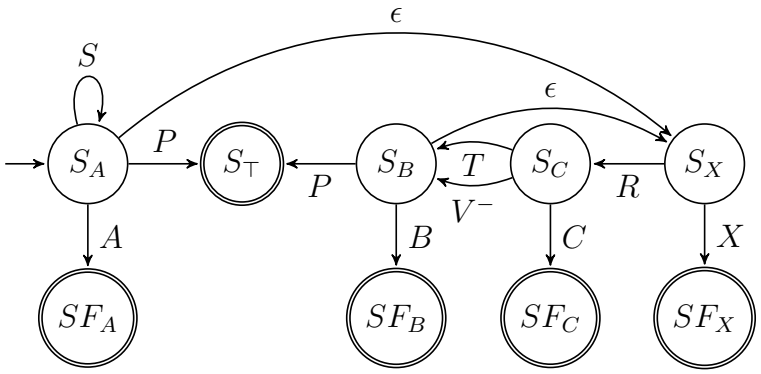

Figure 4.1: NFA for Example 4.4.2.

We now introduce two lemmata that are needed for the proof of correctness.

Lemma 4.1. *Let $\mathcal{L}$ be an $\mathcal{ELHI}^{lin}$ TBox in normal form. Given an 2SPCQ $\boldsymbol{q}$ and an axiom $\rho \in \mathcal{L}$ that is not applicable to an atom $a \in body(\boldsymbol{q})$, if the atom a unifies with a set of atoms $S \subseteq body(\boldsymbol{q})$, then ρ is also not applicable to a', where a' is the atom resulting from applying to $\{a\} \cup S$ the most general unifier for $\{a\} \cup S$.*

Proof. We consider all the possible cases of ρ.

Case 1: ρ is of the type $B \sqsubseteq A$, $\exists P.\top \sqsubseteq A$, $\exists R.B \sqsubseteq A$, $R \sqsubseteq P$, $R \sqsubseteq P^-$. ρ is not applicable to S if S does not contain $A(x_1, x_2)$ nor $P(x_1, x_2)$, and the claim follows.

Case 2: ρ is of the type $A \sqsubseteq \exists R.\top$. ρ is not applicable to a if a is not an atom of the type $R(x_1, x_2)$, or if a is an atom of the type $R(x_1, x_2)$ and $x_2 \neq \_$. If a is not of the type $R(x_1, x_2)$, then clearly no unification can produce an atom to which ρ can be applied. If a is of the type $R(x_1, x_2)$ and $x_2 \neq \_$, since $\boldsymbol{q}$ is a 2SPCQ this happens only if $R(x_1, x_2)$ is not the last atom of the path, or if it is the last atom of the path and we have $body(q) = ...\tau_R(x_1, x_2), R(x_3, x_2)$, since x_2 in this case is not $\_$. In the latter case, there are two ways to unify the atoms $R(x_3, x_2)$ with S:

if the atom directly to the left of $R(x_3, x_2)$ is of the form $R(x_2, x_1)$, which produces $R(x_2, x_2)$ and still we have $x_2 \neq \_$, or the case that the atom directly to the left is of the form $P(x_1, x_2)$, which contradicts the condition of $\mathcal{L}$ being an $\mathcal{ELHI}_h^{lin}$ TBox in normal form. In the case where the atom $R(x_1, x_2)$ is not at the end of the path, there are two ways to have $x_2 = \_$. In case *(i)*, $R(x_1, x_2)$ unifies with the atom on its RHS, which has to be of the form $\tau_R(x_2, x_3)$. If it is of the form $R(x_2, x_3)$, the unification produces an atom $R(x_2, x_2)$, with $x_2 \neq \_$. If it is of the form $R(x_3, x_2)$, then we have that $\mathcal{L}$ is not an $\mathcal{ELHI}_h^{lin}$ TBox in normal form. In case *(ii)*, $R(x_1, x_2)$ unifies with the atom on its LHS, which has to be of the form $\tau_R(x_0, x_1)$. If it is of the form $R(x_0, x_1)$, the unification produces an atom $R(x_1, x_1)$, with $x_1 \neq \_$. If it is of the form $R(x_1, x_1)$, then we have that $\mathcal{L}$ is not an $\mathcal{ELHI}_h^{lin}$ TBox. The statement of the lemma therefore follows. □

Lemma 4.2. *Let $\mathcal{L}$ be an $\mathcal{ELHI}^{lin}$ TBox in normal form. Let $\boldsymbol{q}$ be a concept IQ of the form $q(x) \leftarrow A(x, y)$. If $\boldsymbol{q}_{rew} \in \mathsf{Rewrite}(\boldsymbol{q}, \mathcal{T})$, then $\boldsymbol{q}_{rew}$ is a 2SPCQ.*

Proof. From Lemma 4.1 we know that queries produced by the reduction step are never processed by the rewriting step, and so they are never marked with '1'. So queries produced by the reduction step are never in the output set, and thus we can ignore the reduction step. The proof is then by induction on the set of queries that are produced after each rewriting step. We denote by $Q^{[i]}$ the set of the queries produced after the i-th iteration of the repeat loop in Algorithm 2.

Base Step. $Q^{[1]}$ contains $\{q(x) \leftarrow A(x, y)\}$ together with the queries obtained by the first rewriting step. The possible cases are the rewriting rules (a), (b) and (c) which generate 2SPCQs.

Inductive Step. If $q \in Q^{[i+1]}$, then q has been computed by applying a rewriting rule to a query in $Q^{[i]}$. The claim follows by induction if, for each rewriting q to q', we have that q' is a 2SPCQ. If q is a 2SPCQ then we can identify a fixed set of

possible rewriting cases, according to the rewriting rules (a)-(f). For each possible rewriting case, when q is rewritten to q', it is easy to see that if q is a 2SPCQ, then q' is also a 2SPCQ. □

Lemma 4.2 leads to the following claim, which states that $\mathcal{ELHI}_h^{lin}$ TBoxes cannot encode conjunctions on the left hand side of concept inclusion axioms.

Claim 1. *Given an $\mathcal{ELHI}_h^{lin}$ TBox $\mathcal{L}$ in normal form, $\mathcal{L}$ cannot encode a concept of the form $C_1 \sqcap C_2$ on the LHS of concept inclusion axioms, where $C_1, C_2 \in \mathsf{A}$*

Proof. Suppose that $\mathcal{L}$ can encode the assertion $C_1 \sqcap C_2 \sqsubseteq C_3$. Now consider the concept IQ $\boldsymbol{q} =: q(x) \leftarrow C_3(x)$. Since the Rewrite algorithm generates the perfect rewriting, then $\mathsf{Rewrite}(\boldsymbol{q}, \mathcal{L})$ contains the CQ $q(x) \leftarrow C_1(x), C(x)$ which contradicts Lemma 4.2. □

Theorem 4.3. *Let $\mathcal{L}$ be an $\mathcal{ELHI}_h^{lin}$ TBox in normal form, with $A \in \mathsf{A}$. We have that $\boldsymbol{q} \in \mathsf{Rewrite}(q(x) \leftarrow A(x,y), \mathcal{T})$ if and only if $path(\boldsymbol{q}) \in L(\mathsf{NFA}^{-}_{A,\mathcal{T}})$.*

Proof. ($\Rightarrow$) The proof is by induction on the set of queries that are marked with '1' after each rewriting step of Algorithm 2, as the queries marked with '0' are not returned by the algorithm. We denote by $Q^{[i]}$ the set of queries marked with '1' after the i-th application of the rewriting step.

Base Step. $Q^{[0]} = \{\boldsymbol{q}\}$. By *(1)* in Definition 4.7 we have that S_A is the initial state q_0 and by *(2)* we have the transition (S_A, A, SF_A); therefore $A \in L(\mathsf{NFA}^{-}_{A,\mathcal{T}})$ and the claim follows trivially.

Inductive Step. From Lemma 4.1 we have that, if an axiom $\rho \in \mathcal{T}$ is not applicable to $body(\boldsymbol{q})$, for each set of atoms $S \subseteq body(\boldsymbol{q})$ that unify, ρ is also not applicable to $body(\mathsf{Reduce}(\boldsymbol{q}, S))$. It follows that if a query q' is marked with '0' then there is no axiom in $\mathcal{T}$ that is applicable to q'. Thus, if $q \in Q^{[i+1]}$, then q has

been computed by applying a rewriting rule to a query that is marked with '1' at the i-th application of the rewriting step, which is a query in $Q^{[i]}$. Suppose that for each $q \in Q^{[i]}$ we have that $\mathsf{path}(q) \in L(\mathsf{NFA}^-_{A,\mathcal{T}})$. Then the claim follows by induction if for each rewriting of q to q', we have that $\mathsf{path}(q') \in L(\mathsf{NFA}^-_{A,\mathcal{T}})$. From Lemma 4.2 it follows that the body of each query marked with '1' is a two-way simple path, thus we can identify a fixed set of possible rewriting cases. For each form of axiom in $\mathcal{T}$ we have the following rewriting cases, and we show for each of them that $\mathsf{path}(q') \in L(\mathsf{NFA}^-_{q_{in},\mathcal{T}})$:

- An axiom of the form $B \sqsubseteq A$ is applicable to q only if $\mathsf{body}(q)$ contains an atom of the form $A(x_1, x_2)$. By applying rule (a) to $\mathsf{body}(q)$ we obtain the same set of atoms in $\mathsf{body}(q)$ except with an atom of the form $B(x_1, x_2)$ in the place of the atom of the form $A(x_1, x_2)$. Then $\mathsf{path}(q') \in L(\mathsf{NFA}^-_{q_{in},\mathcal{T}})$ as from (3.1) and (1) we have that $(S_A, \epsilon, S_B), (S_B, B, SF_B) \in \delta$.

- An axiom of the form $B \sqsubseteq \exists R.\top$ is applicable to q only if $\mathsf{body}(q)$ contains an atom of the form $R(x_1, \_)$. By applying rule (d) to $\mathsf{body}(q)$ we obtain the same set of atoms in $\mathsf{body}(q)$ except with an atom of the form $B(x_1, \_)$ in the place of the atom of the form $R(x_1, \_)$. Then $\mathsf{path}(q') \in L(\mathsf{NFA}^-_{q_{in},\mathcal{T}})$ as from (1) and (3.2) we have that $(S_B, B, SF_B) \in \delta$ and for each transition $(S_X, R\ , S_\top) \in \delta$ there is a transition $(S_X, \epsilon, S_B) \in \delta$.

- An axiom of the form $\exists R.\top \sqsubseteq A$ is applicable to q only if $\mathsf{body}(q)$ contains an atom of the form $A(x_1, x_2)$. Applying rule (b) we we obtain the same set of atoms in $\mathsf{body}(q)$ except with an atom of the form $R(x_1, \_)$ in the place of the atom of the form $A(x_1, x_2)$. Then $\mathsf{path}(q') \in L(\mathsf{NFA}^-_{q_{in},\mathcal{T}})$ as from (3.3) we have the transition $(S_A, R\ , S_\top) \in \delta$.

- An axiom of the form $\exists R.B \sqsubseteq A$ is applicable to q only if $\mathsf{body}(q)$ contains an atom of the form $A(x_1, x_2)$. Applying rule (c) we obtain the same set of atoms

in $\mathsf{body}(q)$ except with a pair of atoms of the form $R(x_0, x_1), B(x_1, x_2)$ in the place of the atom of the form $A(x_1, x_2)$. Then $\mathsf{path}(q') \in L(\mathsf{NFA}^-_{q_{in},\mathcal{T}})$, as from (1) and (3.4) we have that $(S_B, B, SF_B), (S_A, R, S_B) \in \delta$.

- An axiom of the form $R_1 \sqsubseteq R_2$ ($R_1 \sqsubseteq R_2^-$) is applicable to q only if $\mathsf{body}(q)$ contains an atom of the form $R_2(x_k, x_{k+1})$. Applying rule (e) (resp. (f)) we obtain $body(q')$ from $\mathsf{body}(q)$ by replacing $R_2(x_k, x_{k+1})$ with $R_1(x_k, x_{k+1})$ (resp. $R_1(x_{k+1}, x_k)$). Thus, it is sufficient to show that, if a sequence of symbols w contains the symbol R_2 and $w \in L(\mathsf{NFA}^-_{q_{in},\mathcal{T}})$ then, if we replace an occurrence of R_2 with the symbol R_1 (resp. R_1^-) in w, we have that w is still contained in $L(\mathsf{NFA}^-_{q_{in},\mathcal{T}})$. This follows from (4.1) (resp. (4.2)).

($\Leftarrow$) Since each rule to construct $\mathsf{NFA}^-_{A,\mathcal{T}}$ corresponds to one of the rewriting steps (a)-(f), the claim follows by induction on the construction rules of the $\mathsf{NFA}^-_{A,\mathcal{T}}$, starting from A. □

Corollary 4.4.2. *Given an $\mathcal{ELHI}_h^{lin}$ TBox $\mathcal{T}$, concept A and a complex concept B, we have that $\mathcal{T} \models B \sqsubseteq A$ if and only if, for each ABox $\mathcal{A}$ and for each individual $a \in \mathsf{ind}(\mathcal{A})$, it holds that $\mathcal{I}_\mathcal{A} \models B(a)$ only if $\mathcal{I}_\mathcal{A} \models q() \leftarrow \mathsf{NFA}^-_{A,\mathcal{T}}(a, y)$, where y is a fresh variable.*

Proof. From Theorem 4.3 we have that $q(x) \leftarrow \mathsf{NFA}^-_{A,\mathcal{T}}(x, y)$ is the perfect rewriting of $q(x) \leftarrow A(x, y)$ with respect to $\mathcal{T}$. Therefore, for each ABox $\mathcal{A}$ and for each individual a, it holds that $(\mathcal{T}, \mathcal{A}) \models \boldsymbol{q}(a) \leftarrow A(a, y)$ if and only if $\mathcal{I}_A \models q(a) \leftarrow \mathsf{NFA}^-_{A,\mathcal{T}}(a, y)$. Since $\mathcal{T} \models B \sqsubseteq A$, for each ABox $\mathcal{A}$ and for each individual a we have that if $\mathcal{I}_\mathcal{A} \models B(a)$ then $\mathcal{K} = (\mathcal{T}, \mathcal{A}) \models q(a) \leftarrow A(a, y)$ and therefore $\mathcal{I}_\mathcal{A} \models q(a) \leftarrow \mathsf{NFA}^-_{A,\mathcal{T}}(a, y)$. □

4.5 Discussion

The work presented in this chapter extends our recent work [48] where we first proposed exploiting the capabilities of navigational queries in order to allow query rewriting of conjunctive queries into C2RPQs under the DL known in the literature as linear $\mathcal{ELH}$. In this chapter, we have undertaken the first steps to extend the above technique to a more expressive DL, namely harmless linear $\mathcal{ELHI}$.

In more detail, we have defined $\mathcal{ELHI}_h^{\ell in}$ (harmless linear $\mathcal{ELHI}$), an ontology language that generalises both DL-Lite$_\mathcal{R}$ and linear $\mathcal{ELH}$; and we have proved the rewritability of instance queries (queries with a single atom in their body) under $\mathcal{ELHI}_h^{\ell in}$ knowledge bases with C2RPQs as the target language, presenting a query rewriting algorithm that makes use of non-deterministic finite-state automata.

In the following chapter, we will use the NFA-based technique proposed here to obtain a rewriting of CQs into C2RPQs under $\mathcal{ELHI}_h^{\ell in}$; this is achieved by extending the rewriting algorithm known in the literature as *tree witness* rewriting [72] with the NFAs presented in this chapter. We will then undertake a complexity analysis for query answering under $\mathcal{ELHI}_h^{\ell in}$; we will formally prove the correctness of our algorithms, and also that they comply with the upper complexity bounds.

Chapter 5

Rewriting of CQs to C2RPQs under harmless linear $\mathcal{ELHI}$ Description Logic

In this chapter, we develop a rewriting of CQs following the approach of [70, 72] which splits the problem of rewriting CQs under DL-Lite$_{\mathcal{R}}$ in two. Firstly, it deals separately with the part of the TBox that does not have existential quantifications on the right-hand side of assertions (that is, the part that when expanded does not produce any labelled null), also called the *flat* part of the TBox. Informally, this is obtained by leveraging the NFAs presented in the previous chapter (Section 4.4), which allow us to finitely encode all the (possibly infinite) rewritings of concept instance queries under $\mathcal{ELHI}_h^{lin}$ TBoxes; this approach is described in Section 5.1. In Section 5.2, we show how the rewriting of the flat part of the TBox is "merged" with the rewriting of the rest of the TBox to generate the perfect rewriting of CQs. Then, in Section 5.3, we undertake a complexity analysis for query answering under $\mathcal{ELHI}_h^{lin}$.

5.1 Rewriting for Flat $\mathcal{ELHI}_h^{lin}$

We now show how to rewrite CQs under the special case of *flat* $\mathcal{ELHI}_h^{lin}$ TBoxes, i.e., those that do not contain existential quantifiers on the right-hand side of concept inclusions. In other words, flat $\mathcal{ELHI}_h^{lin}$ in normal form can only contain concept and role inclusions of the form $A_1 \sqsubseteq A_2$, $\exists R.D \sqsubseteq A$, $R_1 \sqsubseteq R_2$ or $R_1 \sqsubseteq R_2^-$, for concept names A, A_1, A_2, role names R_1, R_2, and D a concept name or $\top$.

Let $\mathcal{T}$ be a flat $\mathcal{ELHI}_h^{lin}$ TBox, $\boldsymbol{q}$ a conjunctive query and $\mathbf{a}$ a tuple of individuals. Since $\mathcal{J}_\mathcal{K}$ is the canonical model for $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, we have that $(\mathcal{T}, \mathcal{A}) \models \boldsymbol{q}(\mathbf{a})$ if and only if $\boldsymbol{q}(\mathbf{a})$ is true in the canonical model $\mathcal{J}_\mathcal{K}$. If the TBox is flat, the canonical model $\mathcal{J}_\mathcal{K}$ contains no labelled nulls, and so, from the definition of $\mathcal{J}_\mathcal{K}$ and Corollary 4.4.2, we have that:

- $\text{fr}\mathcal{J}_\mathcal{K} \models A(a)$ if and only if $\mathcal{I}_\mathcal{A} \models q() \leftarrow B(a, y)$ and $B \in L(\mathsf{NFA}^-_{A,\mathcal{T}})$, for some B,
- $\mathcal{J}_\mathcal{K} \models P(a, b)$ if and only if $\mathcal{I}_\mathcal{A} \models R(a, b)$ and $\mathcal{T} \models R \sqsubseteq P$, for some R.

Following from this observation, we are now able to define a C2RPQ $\boldsymbol{q}_{\mathcal{T}-ext}$ such that, for any CQ $\boldsymbol{q}$ and any flat $\mathcal{ELHI}_h^{lin}$ TBox $\mathcal{T}$, $\boldsymbol{q}_{\mathcal{T}-ext}$ is the perfect rewriting of $\boldsymbol{q}$ with respect to $\mathcal{T}$.

Definition 5.1. Given a CQ $\boldsymbol{q}$ and an $\mathcal{ELHI}_h^{lin}$ TBox $\mathcal{T}$, we construct a C2RPQ $\boldsymbol{q}_{\mathcal{T}-ext}$ by replacing every atom $A(z_1, z_2)$ in $\boldsymbol{q}$ with $A_{\mathcal{T}-ext}(z_1, z_2)$ and every atom $P(z_1, z_2)$ in $\boldsymbol{q}$ with $P_{\mathcal{T}-ext}(z_1, z_2)$, where $A_{\mathcal{T}-ext}(z_1, z_2)$ and $P_{\mathcal{T}-ext}(z_1, z_2)$ are the following formulas:

$$A_{\mathcal{T}-ext}(u_1, u_2) = \alpha(u_1, u_2),$$

where α is a regular expression denoting $L(\mathsf{NFA}^-_{A,\mathcal{T}})$, and

$$P_{\mathcal{T}-ext}(u_1, u_2) = \bigcup_{\mathcal{T} \models R \sqsubseteq P} R(u_1, u_2) \cup \bigcup_{\mathcal{T} \models R \sqsubseteq P^-} R(u_2, u_1),$$

Note that $P_{\mathcal{T}-ext}(u_1, u_2)$ can be expressed as a single RPQ, as shown in Example 5.1.1.

Example 5.1.1. Consider the flat $\mathcal{ELHI}_h^{lin}$ TBox $\mathcal{T} = \{\exists R.A \sqsubseteq A,\ P \sqsubseteq R\}$ and the CQ $\boldsymbol{q} = q(x, y) \leftarrow \exists z\ A(x, z), R(x, y)$. Following from Definition 5.1 we construct $\boldsymbol{q}_{\mathcal{T}-ext}$ as follows:

$$q(x, y) \leftarrow \exists z\ (R|P)^* A(x, z), R|P(x, y)$$

Now we show that, for any CQ $\boldsymbol{q}$ and any flat $\mathcal{ELHI}_h^{lin}$ TBox $\mathcal{T}$, $\boldsymbol{q}_{\mathcal{T}-ext}$ is the C2RPQ rewriting of $\boldsymbol{q}$ with respect to $\mathcal{T}$.

Proposition 5.1. *For every $\mathcal{ELHI}_h^{lin}$ KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, every concept name A, role name P and individual names a and b we have:*

- $\mathcal{J}_\mathcal{K} \models A(a)$ *if and only if* $\mathcal{I}_\mathcal{A} \models q() \leftarrow A_{\mathcal{T}-ext}(a, y)$, *where y is a fresh variable,*
- $\mathcal{J}_\mathcal{K} \models P(a, b)$ *if and only if* $\mathcal{I}_\mathcal{A} \models q() \leftarrow P_{\mathcal{T}-ext}(a, b)$.

Proof. From Corollary 4.4.2 we have that $q() \leftarrow A_{\mathcal{T}-ext}(a, y)$ is the perfect rewriting of $q() \leftarrow A(a)$ with respect to $\mathcal{T}$. The fact that $\mathcal{J}_\mathcal{K} \models P(a, b)$ if and only if $q() \leftarrow P_{\mathcal{T}-ext}(a, b)$ follows from the observation that the only way to infer a role inclusion in an $\mathcal{ELHI}^{lin}$ TBox is through the closure of role inclusion axioms of the form $R_1 \sqsubseteq R_2$ and $R_1 \sqsubseteq R_2^-$. The claim follows. □

Proposition 5.2. *For any CQ $\boldsymbol{q}$ and any flat $\mathcal{ELHI}_h^{lin}$ TBox $\mathcal{T}$, $\boldsymbol{q}_{\mathcal{T}-ext}$ is the C2RPQ rewriting of $\boldsymbol{q}$ with respect to $\mathcal{T}$.*

Proof. Since $\mathcal{T}$ is flat, no axioms contain existential quantifiers on the right-hand side, so no labelled nulls appear during the chase procedure. Therefore we can construct the perfect rewriting of a conjunctive query q by splitting q into its atoms, generating the perfect rewriting of the single atomic queries and taking the conjunction of the resulting set of atoms. More specifically, we can substitute every atom $A(z_1, z_2)$ in $\boldsymbol{q}$ with $A_{\mathcal{T}-ext}(z_1, z_2)$ and every atom $P(z_1, z_2)$ in $\boldsymbol{q}$ with $P_{\mathcal{T}-ext}(z_1, z_2)$ which gives rise to $\boldsymbol{q}_{\mathcal{T}-ext}$. □

5.2 Rewriting for Full $\mathcal{ELHI}_h^{lin}$

To generate rewritings for full $\mathcal{ELHI}_h^{lin}$ TBoxes, we need a closer look at the construction of the canonical model through the application of rules *(i)-(vii)* in Definition 4.3 to the base model $\mathcal{I}_\mathcal{A}$. There are two key observations: first, fresh labelled nulls can only be added by applying rule *(vii)*, and, second, if two labelled nulls, d_1 and d_2, are introduced by applying rule *(vii)* using the same concept inclusion $A \sqsubseteq \exists R.\top$, then the same rules will be applicable to d_1 and d_2 in the continuation of the chase procedure. In addition, from Proposition 5.1, we know that for every $\mathcal{ELHI}_h^{lin}$ KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, every concept name A and individual name a, we have $\mathcal{J}_\mathcal{K} \models A(a)$ if and only if $\mathcal{I}_\mathcal{A} \models q() \leftarrow A_{\mathcal{T}-ext}(a, y)$, where y is a fresh variable. Thus, for each labelled null d' resulting from applying *(vii)* we can identify an individual name $d \in \mathsf{ind}(A)$ that "triggers" an application of rule *(vii)* and a subsequent series of applications of *(vii)* that leads to the generation of d'.

To formalise this, we first introduce a symbol $w_{\exists R}$ as a *witness* for $\exists R.\top$ and we define a *generating relation* $\rightsquigarrow_{\mathcal{T},\mathcal{A}}$ on the set of these witnesses together with $\mathsf{ind}(\mathcal{A})$ by taking:

(α) $a \rightsquigarrow_{\mathcal{T},\mathcal{A}} w_{\exists R}$, if $a \in \mathsf{ind}(\mathcal{A})$, $\mathcal{I}_\mathcal{A} \models q(a), \leftarrow \mathsf{NFA}^-_{A,\mathcal{T}}(a, y)$, $A \sqsubseteq \exists R.\top \in \mathcal{T}$ and $\mathcal{I}_\mathcal{A} \not\models \exists R.\top(a)$,

(β) $w_{\exists S} \rightsquigarrow_{\mathcal{T},\mathcal{A}} w_{\exists R}$ if $S^- \in L(\mathsf{NFA}^-_{A,\mathcal{T}})$ and also that $A \sqsubseteq \exists R.\top \in \mathcal{T}$, where S is a role name.

(α) and (β) reflect the two key observations above, respectively. Note that, if a labelled null d_1 is introduced by applying *(vii)* via some concept inclusion $B \sqsubseteq \exists S.\top$ and $S^- \in L(\mathsf{NFA}^-_{A,\mathcal{T}})$, then, at some point in the chase, the concept assertion $A(d_1)$ is generated; thus, (β) is necessary in the definition of the generating relation.

A *path* σ on the generating relation $\rightsquigarrow_{\mathcal{T},\mathcal{A}}$ is a finite concatenation $aw_{\exists R_1} \dots w_{\exists R_n}$, $n \geqslant 0$, such that $a \in \mathsf{ind}(\mathcal{A})$ and, if $n > 0$, then $a \rightsquigarrow_{\mathcal{T},\mathcal{A}} w_{\exists R_1}$ and $w_{\exists R_i} \rightsquigarrow_{\mathcal{T},\mathcal{A}} w_{\exists R_{i+1}}$, for $i < n$. Thus, a path of the form $\sigma w_{\exists R}$ also denotes the fresh labelled null introduced by applying *(vii)* to some $A \sqsubseteq \exists R.\top$ on the individual σ. For brevity, we denote by "$\rightsquigarrow_{\mathcal{T},\mathcal{A}}$*-path* σ" a path σ on the generating relation $\rightsquigarrow_{\mathcal{T},\mathcal{A}}$.

We point out that the approach of defining a generating relation for labelled nulls is originally introduced in [70, 72]. Here, we extend it by introducing $\mathsf{NFA}^-_{A,\mathcal{T}}$ in the above formulas, allowing us to follow a similar approach and define a finite generating relation $\rightsquigarrow_{\mathcal{T},\mathcal{A}}$ when $\mathcal{T}$ is an $\mathcal{ELHI}_h^{lin}$ TBox.

Following from the observations above, we now show that for any $\mathcal{ELHI}^{lin}$ TBox $\mathcal{T}$ and ABox $\mathcal{A}$ we are able to express each labelled null generated during the chase procedure as a $\rightsquigarrow_{\mathcal{T},\mathcal{A}}$-path σ.

Proposition 5.3. *Consider an $\mathcal{ELHI}_h^{lin}$ KB $\mathcal{K} = (\mathcal{A}, \mathcal{T})$ and an element $d_0 \in \mathsf{ind}(\mathcal{A})$. Suppose that $A_0^{\mathcal{I}_0} = \{d_0\}$ and that there is a sequence of applications of rules* (i)-(vii) *of Definition 4.3 during the chase procedure as follows:*

- *the rule* (vii) *is applied to $A_0 \sqsubseteq \exists P_0.\top$, so the fresh labelled null d_1 is generated and (d_0, d_1) is added to $P_0^{\mathcal{I}_1}$;*
- *for $1 \leqslant i \leqslant n$ we have $d_i \in A_i^{\mathcal{I}_{(i+\sum_{j=1}^{i} k_j)}}$, and* (vii) *is applied to $A_i \sqsubseteq \exists P_i.\top$, so the fresh labelled null d_{i+1} is generated and (d_i, d_{i+1}) is added to $P_i^{\mathcal{I}_{(i+1+\sum_{j=1}^{i} k_j)}}$;*

with $k_1, \ldots, k_n \geqslant 1$. *Such a sequence of rule applications is performed during the chase if and only if it holds that:*

$$d_0 \rightsquigarrow_{\mathcal{T},\mathcal{A}} w_{\exists P_1} \text{ and } w_{\exists P_i} \rightsquigarrow_{\mathcal{T},\mathcal{A}} w_{\exists P_{i+1}}, \text{ for } i < n.$$

Proof. We show this by the induction on the i-th application of the rule *(vii)*.
($\Rightarrow$) Base step. For $i = 0$ we have that the rule *(vii)* is applied according to an axiom $A_0 \sqsubseteq \exists P_0.\top$ which means that $\mathcal{I}_0 \not\models \exists P_0.\top(a)$. For rule (α) in the definition of the generating relation we have that $d_0 \rightsquigarrow_{\mathcal{T},\mathcal{A}} w_{\exists P_0}$.

Inductive step. For $i = l$, by induction hypothesis it holds that

$$d_0 \rightsquigarrow_{\mathcal{T},\mathcal{A}} w_{\exists P_1} \text{ and } w_{\exists P_i} \rightsquigarrow_{\mathcal{T},\mathcal{A}} w_{\exists P_{i+1}}, \text{ with } 1 \leqslant i < l.$$

The proof follows if it also holds that $w_{\exists P_l} \rightsquigarrow_{\mathcal{T},\mathcal{A}} w_{\exists P_{l+1}}$. By the induction hypothesis, we have that $d_{l+1} \in A_{l+1}^{\mathcal{I}_{(l+1+\sum_{j=1}^{l+1} K_j)}}$. Since d_{l+1} is generated by applying $A_l \sqsubseteq \exists P_l.\top$ it holds that $P_l^- \in L(\mathsf{NFA}^-_{A_{l+1},\mathcal{T}})$ and also that $A_{l+1} \sqsubseteq \exists P_{l+1}.\top \in \mathcal{T}$. Thus, by rule (β) in the generating relation it also holds that $w_{\exists P_l} \rightsquigarrow_{\mathcal{T},\mathcal{A}} w_{\exists P_{l+1}}$ and the claim follows.

($\Leftarrow$) Base step. For $i = 0$ we have

$$d_0 \rightsquigarrow_{\mathcal{T},\mathcal{A}} w_{\exists P_1}$$

Thus, by the rule (α) in definition of the generating relation we have that there is concept name A_0 such that $a \in \mathsf{ind}(\mathcal{A})$, $\mathcal{I}_\mathcal{A} \models q(a), \leftarrow \mathsf{NFA}^-_{A_0,\mathcal{T}}(a, y)$, $A_0 \sqsubseteq \exists P_1.\top \in \mathcal{T}$ and $\mathcal{I}_\mathcal{A} \not\models P_1.\top(a)$. Therefore the rule *(vii)* is applied according to $A_0 \sqsubseteq \exists P_0.\top$, so a fresh labelled null , say d_1, is generated and (a, d_1) is added to $P_0^{\mathcal{I}_1}$.

Inductive step. Now assume that the claim holds for $i = l$. For $i = l + 1$ we have that $w_{\exists P_l} \rightsquigarrow_{\mathcal{T},\mathcal{A}} w_{\exists P_{l+1}}$, and therefore, by the rule (β) in the definition of the

generating relation, there is a concept name, say A_{l+1}, such that $P_l^- \in L(\mathsf{NFA}^-_{A_{l+1},\mathcal{T}})$ and $A_{l+1} \sqsubseteq \exists P_{l+1}.\top \in \mathcal{T}$. Thus, at a certain point in the chase, say at the $(l + 1 + \sum_{j=1}^{l+1} K_j)$-th cycle, the assertion $A_{l+1}(d_{l+1})$ is generated, since by the induction hypothesis it holds that $\exists P_l.\top(d_{l+1})^{\mathcal{I}_{(l+1+\sum_{j=1}^{l} K_j)}}$. Therefore, $d_{l+1} \in A_{l+1}^{\mathcal{I}_{(l+1+\sum_{j=1}^{l+1} K_j)}}$ and *(vii)* is applied according to $A_{l+1} \sqsubseteq \exists P_{l+1}.\top$ and the fresh labelled null, say d_{l+2}, is generated and (d_{l+1}, d_{l+2}) is added to $P_{l+1}^{\mathcal{I}\left((l+2)+\sum_{j=1}^{l+1} K_j\right)}$. The claim follows. □

Following from Proposition 5.3, we are now able to generate the rewriting for $\mathcal{ELHI}_h^{lin}$ TBoxes, similarly to [70, 72] for DL-Lite$_\mathcal{R}$. For each $\mathcal{ELHI}^{lin}$ TBox $\mathcal{T}$ and ABox $\mathcal{A}$ we can express each labelled null generated during the chase procedure as a $\rightsquigarrow_{\mathcal{T},\mathcal{A}}$-path σ. Thus, we are now able to construct the canonical model in a top-down fashion by "unravelling" σ. Let us denote by $\mathsf{tail}(\sigma)$ the last element in σ. Following from Proposition 5.3 we know that the last element in the $\rightsquigarrow_{\mathcal{T},\mathcal{A}}$-path σ uniquely determines all the subsequent rule applications. Therefore, we can construct a canonical model $\mathcal{C}_{\mathcal{T},\mathcal{A}}$ as follows.

Definition 5.2. Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ be an $\mathcal{ELHI}_h^{lin}$ KB, and $\Delta^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$ be the set of all $\rightsquigarrow_{\mathcal{T},\mathcal{A}}$-paths. The canonical model $\mathcal{C}_{\mathcal{T},\mathcal{A}}$ is defined by taking:

(1) $a^{\mathcal{C}_{\mathcal{T},\mathcal{A}}} = a$, for $a \in \mathsf{ind}(\mathcal{A})$;

(2) $A^{\mathcal{C}_{\mathcal{T},\mathcal{A}}} = \{a \in \mathsf{ind}(A) \mid \mathcal{I}_\mathcal{A} \models B(a) \text{ and } B \in L(\mathsf{NFA}^-_{A,\mathcal{T}})\} \cup$
$\{\sigma w_{\exists R} \mid R^- \in L(\mathsf{NFA}^-_{A,\mathcal{T}})\}$, for each concept name A;

(3) $P^{\mathcal{C}_{\mathcal{T},\mathcal{A}}} = \{(a, b) \mid \mathcal{I}_\mathcal{A} \models R(a, b) \text{ and } \mathcal{T} \models R \sqsubseteq P\} \cup$
$\{(\sigma w_{\exists R}, \sigma) \mid \mathsf{tail}(\sigma) \rightsquigarrow_{\mathcal{T},\mathcal{A}} w_{\exists R}, \mathcal{T} \models R \sqsubseteq P^-\} \cup$
$\{(\sigma, \sigma w_{\exists R}) \mid \mathsf{tail}(\sigma) \rightsquigarrow_{\mathcal{T},\mathcal{A}} w_{\exists R}, \mathcal{T} \models R \sqsubseteq P\}$, for each role name P.

In the following theorem, we show that $\mathcal{C}_{\mathcal{T},\mathcal{A}}$ is in fact a canonical model for $\mathcal{K} = (\mathcal{T}, \mathcal{A})$.

Theorem 5.1. *For every $\mathcal{ELHI}_h^{lin}$ KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, any CQ $q(\vec{x})$ of arity n and any n-tuple $\vec{a} \subseteq \mathsf{ind}(A)^n$, $\mathcal{K} \models q(\vec{a})$ if and only if $\mathcal{C}_{\mathcal{T},\mathcal{A}} \models q(\vec{a})$.*

Proof. By the definition of rule *(iii)* and *(vi)* and Corollary 4.4.2, an ABox individual a belongs to $A^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$ just in the case $\mathcal{I}_A \models q() \leftarrow \mathsf{NFA}^-_{A,\mathcal{T}}(a, y)$. Similarly, by the definition of rules *(iii)* and *(vii)* and by Proposition 5.3, a labelled null of the form $\sigma w_{\exists R}$ belongs to $A^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$ just in the case $R^- \in L(\mathsf{NFA}^-_{A,\mathcal{T}})$. For a role name P, rules *(v)* and *(vii)* provide an analogous argument. More precisely, by the definition of rule *(v)*, a pair (d, d') of domain elements belongs to $P^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$ just in the case $(d, d') \in R^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$ and $\mathcal{T} \models R \sqsubseteq P$ for some P (note that checking $\mathcal{T} \models R \sqsubseteq P$ can be done in linear time). It then follows from the definition of rule *(vi)* and from Proposition 5.3 that a pair (d, d') belongs to $P^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$ just in three cases: for some R,

- both elements of the pair are ABox individuals with $\mathcal{I}_\mathcal{A} \models R(d, d')$ and $\mathcal{T} \models R \sqsubseteq P$,
- the first component of the pair is created by an application of the rule *(vii)* to the second component of the pair: $d = \sigma w_{\exists R}, d' = \sigma$ and $\mathcal{T} \models R \sqsubseteq P^-$,
- the second component of the pair is created by an application of the rule *(vii)* to the first component of the pair: $d = \sigma, d' = \sigma w_{\exists R}$ and $\mathcal{T} \models R \sqsubseteq P$.

These three cases are reflected in the three sets in the union in Definition 5.2 of $P^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$. Thus, $\mathcal{C}_{\mathcal{T},\mathcal{A}}$ is a canonical model for $\mathcal{K}$ and the claim follows. □

Suppose $\mathcal{T}$ is an $\mathcal{ELHI}_h^{lin}$ TBox in normal form. To compute the certain answers to a conjunctive query $\boldsymbol{q}$ over $(\mathcal{T}, \mathcal{A})$, for some A, it is enough to find answers to $\boldsymbol{q}$ in the canonical model $\mathcal{C}_{\mathcal{T},\mathcal{A}}$. To do so, we have to check, for every tuple of elements in $\mathsf{ind}(\mathcal{A})$, whether there exists a homomorphism from $\boldsymbol{q}$ to $\mathcal{C}_{\mathcal{T},\mathcal{A}}$. The answer variables take values only from $\mathsf{ind}(\mathcal{A})$, while the existentially quantified variables in $\boldsymbol{q}$ can

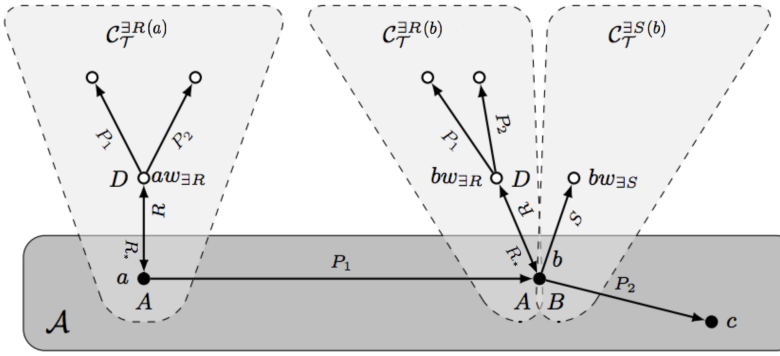

Figure 5.1: The canonical model $\mathcal{C}_{\mathcal{T},\mathcal{A}}$ from Example 5.2.1.

be mapped either to $\mathsf{ind}(\mathcal{A})$ or to the labelled nulls in $\mathcal{C}_{\mathcal{T},\mathcal{A}}$. In order to define a rewriting that is independent from a particular ABox, we now look more closely at the structure of the canonical models $\mathcal{C}_{\mathcal{T},\mathcal{A}}$ with fixed $\mathcal{T}$ and varying $\mathcal{A}$.

Let $\mathcal{T}$ be an $\mathcal{ELHI}^{lin}$ TBox and $A \sqsubseteq \exists R.\top$ an axiom in $\mathcal{T}$. For an arbitrary individual name a, we define the tree $\mathcal{C}_{\mathcal{T}}^{\exists R(a)}$ as the canonical model of the KB $(\mathcal{T}, \{A(a)\})$. Now, take any ABox $\mathcal{A}$ and any $a \in \mathsf{ind}(\mathcal{A})$. By the definition of the canonical model, if $a \rightsquigarrow_{\mathcal{T},\mathcal{A}} \omega_{\exists R}$ then $\mathcal{C}_{\mathcal{T},\mathcal{A}}$ contains a subset that is isomorphic to $\mathcal{C}_{\mathcal{T}}^{\exists R(a)}$. Moreover, such subsets may intersect only on their common root a. It is easy to see that, for an individual name $a \in \mathsf{ind}(\mathcal{A})$, $\mathcal{C}_{\mathcal{T}}^{\exists R(a)}$ is the restriction of the canonical model $\mathcal{C}_{\mathcal{T},\mathcal{A}}$ to only the domain that consists of a and all the labelled nulls with the prefix $a\omega_{\exists R}$.

Example 5.2.1. Consider a TBox $\mathcal{T}$ with the following concept inclusions: $A \sqsubseteq \exists R.\top$, $R \sqsubseteq R_*^-$, $\exists T \sqsubseteq D$ $D \sqsubseteq \exists P_1.\top$, $D \sqsubseteq \exists P_2.\top$ and $B \sqsubseteq \exists S.\top$, and suppose that an ABox $\mathcal{A}$ contains $A(a)$, $P_1(a,b)$, $A(b)$, $B(b)$ and $P_2(b,c)$. The canonical model $\mathcal{C}_{\mathcal{T},\mathcal{A}}$ is depicted in Fig. 5.1. The individual a in this canonical model has a single tree $\mathcal{C}_{\mathcal{T}}^{\exists R(a)}$. The individual b has trees $\mathcal{C}_{\mathcal{T}}^{\exists R(b)}$ and $\mathcal{C}_{\mathcal{T}}^{\exists S(b)}$. These two trees intersect only at their common root b.

We now show that in the case of $\mathcal{ELHI}_h^{lin}$ KBs, we can follow the approach of [70, 72] for DL-Lite$_{\mathcal{R}}$, which shows how the process of constructing FO-rewritings can be

split into two steps: the first step considers only the flat part of the TBox and uses the formulas $A_{\mathcal{T}-ext}(u_1, u_2)$ and $P_{\mathcal{T}-ext}(u_1, u_2)$ defined in the previous section; the second step (described below) takes account of the remaining part of the TBox, that is, inclusions of the form $A \sqsubseteq \exists R.\top$. We first need some preliminary definitions (adapted from [70, 72]).

Definition 5.3. (H-completeness) Let $\mathcal{T}$ be a (not necessarily flat) $\mathcal{ELHI}_h^{lin}$ TBox. A simple ABox $\mathcal{A}$ is said to be *H-complete with respect to* $\mathcal{T}$ if, for all concept names A and role names P, we have:

- $A(a) \in \mathcal{A}$ if $\mathcal{I}_\mathcal{A} \models q() \leftarrow \mathsf{NFA}^-_{A,\mathcal{T}}(a, y)$, where y is a fresh variable;
- $P(a, b) \in \mathcal{A}$ if $\mathcal{I}_\mathcal{A} \models R(a, b)$ and $\mathcal{T} \models R \sqsubseteq P$, for some R.

Our definition of H-completeness differs from that in [72] in that it includes the NFAs defined in Section 4.4, which allow us to finitely encode all the (possibly infinite) complex concept expressions B that entail a concept name $A \in \mathcal{A}$ under $\mathcal{ELHI}_h^{lin}$ TBoxes. We say that a C2RPQ p is a *perfect rewriting* of q with respect to $\mathcal{T}$, if, for any ABox $\mathcal{A}$ and any tuple a from $\mathsf{ind}(A)$, the following holds:

$$(\mathcal{T}, \mathcal{A}) \models q(a) \text{ if and only if } \mathcal{I}_\mathcal{A} \models p(a).$$

In the case that the above formula holds only if $\mathcal{A}$ is H-complete with respect to $\mathcal{T}$, then we say that p is a *perfect rewriting* of q and $\mathcal{T}$ *over H-complete ABoxes* [72]. The authors in [72] observe that, if an ABox A is H-complete with respect to $\mathcal{T}$, then the ABox part of $\mathcal{C}_{\mathcal{T},\mathcal{A}}$, the part that does not contain labelled nulls, coincides with $\mathcal{I}_\mathcal{A}$. Thus, if $\mathcal{T}$ is flat then q itself is clearly the perfect rewriting of q and T over H-complete ABoxes. Following from this observation, we show that we can easily obtain rewritings (over arbitrary ABoxes) from rewritings over H-complete ABoxes.

We now introduce two additional lemmata that are needed for the proof of correctness.

Lemma 5.1. *Let q be a conjunctive query of arity n, $\mathcal{A}$ an H-complete ABox, $\mathcal{T}$ an $\mathcal{ELHI}_h^{lin}$ TBox and $t = (a_1, \ldots, a_n)$, with $a_1, \ldots, a_n \in \mathsf{ind}(A)$. Then $t \in q^{\mathcal{I}_\mathcal{A}}$ if and only if $t \in {q_{\mathcal{T}-ext}}^{\mathcal{I}_\mathcal{A}}$.*

Proof. ($\Rightarrow$) It follows immediately from the definition of $q_{\mathcal{T}-ext}$.
($\Leftarrow$) Since $\mathcal{A}$ is H-complete, then $A(a) \in \mathcal{A}$ if $\mathcal{I}_\mathcal{A} \models q() \leftarrow B(a, y)$ and $B \in \mathsf{NFA}^-_{A,\mathcal{T}}$. In similar way, $P(a, b) \in \mathcal{A}$ if $I_\mathcal{A} \models R(a, b)$ and $\mathcal{T} \models R \sqsubseteq P$, for some R. Thus, $A(a) \in \mathcal{A}$ if $\mathcal{I}_\mathcal{A} \models q() \leftarrow A_{\mathcal{T}-ext}(a, y)$ and $P(a, b) \in \mathcal{A}$ if $\mathcal{I}_\mathcal{A} \models q() \leftarrow P_{\mathcal{T}-ext}(a, y)$ (with $q() \leftarrow P_{\mathcal{T}-ext}(a, y)$ expressed as a single RPQ). Since $q_{\mathcal{T}-ext}$ is composed only by replacing every atom $A(z_1, z_2)$ in q with $A_{\mathcal{T}-ext}(z_1, z_2)$ and every atom $P(z_1, z_2)$ in $\boldsymbol{q}$ with $P_{\mathcal{T}-ext}(z_1, z_2)$, the claim follows. □

Corollary 5.2.1. *If p is a perfect rewriting of q and $\mathcal{T}$ over H-complete ABoxes, then $p_{\mathcal{T}-ext}$ is a perfect rewriting of q and $\mathcal{T}$ over H-complete ABoxes.*

Proof. The proof follows immediately from Lemma 5.1. □

Definition 5.4. Let $\mathcal{A}_H$ be an ABox constructed as follows. We take the base model $\mathcal{I}_\mathcal{A}$ as $\mathcal{I}_0$ and the following rules are applied inductively to obtain $\mathcal{I}_{k+1}$ from $\mathcal{I}_k$:

(i) if $d \in A^{\mathcal{I}_k}$ then d is added to $A^{\mathcal{I}_{k+1}}$;

(ii) if $(d, d') \in R^{\mathcal{I}_k}$ then (d, d') is added to $R^{\mathcal{I}_{k+1}}$;

(iii) if $d \in A_1^{\mathcal{I}_k}$ and $A_1 \sqsubseteq A_2 \in \mathcal{T}$, then d is added to $A_2^{\mathcal{I}_{k+1}}$;

(iv) if $(d, d') \in R_1^{\mathcal{I}_k}$ and $R_1 \sqsubseteq R_2 \in \mathcal{T}$, then (d, d') is added to $R_2^{\mathcal{I}_{k+1}}$;

(v) if $(d, d') \in R_1^{\mathcal{I}_k}$ and $R_1 \sqsubseteq R_2^- \in \mathcal{T}$, then (d', d) is added to $R_2^{\mathcal{I}_{k+1}}$;

(vi) if $d \in (R.D)^{\mathcal{I}_k}$ and $\exists R.D \sqsubseteq A \in \mathcal{T}$, where D is a concept name or $\top$, then d is added to $A^{\mathcal{I}_{k+1}}$;

We then take a fixpoint interpretation, as $k \to \infty$, and we denote the resulting interpretation with $\mathcal{I}_{\mathcal{A}_H}$. We take $\mathcal{A}_H$ as the ABox of which the base model is $\mathcal{I}_{\mathcal{A}_H}$. Note that, by definition, $\mathcal{A}_H$ is H-complete. We call such an ABox, $\mathcal{A}_H$, an a *H-complete extension of* $\mathcal{A}$.

Lemma 5.2. *Let q be a conjunctive query of arity n, $\mathcal{A}$ an ABox, $\mathcal{T}$ an $\mathcal{ELHI}_h^{lin}$ TBox and $t = (a_1, \ldots, a_n)$, with $a_1, \ldots, a_n \in \mathsf{ind}(A)$. Then $t \in {q_{\mathcal{T}-ext}}^{\mathcal{I}_{\mathcal{A}_H}}$ if and only if $t \in {q_{\mathcal{T}-ext}}^{\mathcal{I}_{\mathcal{A}}}$.*

Proof. ($\Rightarrow$) Let us assume that $t \in {q_{\mathcal{T}-ext}}^{\mathcal{I}_{\mathcal{A}_H}}$ and $t \notin {q_{\mathcal{T}-ext}}^{\mathcal{I}_{\mathcal{A}}}$. Then, from the definition of $q_{\mathcal{T}-ext}$ this is possible when either of the following two conditions is satisfied:

- for some concept name A appearing in q and some $a \in \mathsf{ind}(\mathcal{A}_H)$ we have that $\mathcal{I}_{\mathcal{A}_H} \models q() \leftarrow A_{\mathcal{T}-ext}(a, y)$ and $\mathcal{I}_{\mathcal{A}} \not\models q() \leftarrow A_{\mathcal{T}-ext}(a, y)$, where y is a fresh variable.
- for some role name P appearing in q and some $a, b \in \mathsf{ind}(\mathcal{A}_H)$ we have that $\mathcal{I}_{\mathcal{A}_H} \models q() \leftarrow P_{\mathcal{T}-ext}(a, b)$ and $\mathcal{I}_{\mathcal{A}} \not\models q() \leftarrow P_{\mathcal{T}-ext}(a, b)$.

Now, without loss of generality, let us assume that $\mathcal{T}$ is in normal form and let us consider the TBox $\mathcal{T}_H$ defined as follows:

$$\mathcal{T}_H = \{\, r \mid r \in \mathcal{T},\ r \text{ is not of the form } A \sqsubseteq \exists R.\top \}.$$

From the definition of $\mathcal{T}_H$ we have that $\mathcal{J}_{(\mathcal{A},\mathcal{T}_H)} = \mathcal{I}_{\mathcal{A}_H}$. Then, following from Proposition 5.1 we have that $\mathcal{I}_{\mathcal{A}_H} \models q() \leftarrow A(a, y)$ if and only if $\mathcal{I}_{\mathcal{A}} \models q() \leftarrow A_{\mathcal{T}-ext}(a, y)$ and $\mathcal{I}_{\mathcal{A}_H} \models q() \leftarrow P(a, b)$ if and only if $\mathcal{I}_{\mathcal{A}} \models q() \leftarrow P_{\mathcal{T}-ext}(a, b)$. Since $q() \leftarrow$

$A(a, y)$ is contained in $q() \leftarrow A_{\mathcal{T}-ext}(a, y)$ and $q() \leftarrow P(a, b)$ is contained in $q() \leftarrow P_{\mathcal{T}-ext}(a, b)$, neither of the two conditions above can be satisfied and the claim follows.

($\Leftarrow$) The claim follows since C2RPQs are monotonic[1] and $\mathcal{A} \subseteq \mathcal{A}_H$. □

Theorem 5.2. *If p is the perfect rewriting of q and $\mathcal{T}$ over H-complete ABoxes, then $p_{\mathcal{T}-ext}$ is the perfect rewriting of q with respect to $\mathcal{T}$.*

Proof. We know that $\mathcal{A} \subseteq \mathcal{A}_H$ from the definition of $\mathcal{A}_H$ and that ${p_{\mathcal{T}-ext}}^{\mathcal{I}_{\mathcal{A}_H}} = {p_{\mathcal{T}-ext}}^{\mathcal{I}_{\mathcal{A}}}$ from Lemma 5.2. Since p is the perfect rewriting of q and $\mathcal{T}$ over H-complete ABoxes, from Corollary 5.2.1 we know that also $p_{\mathcal{T}-ext}$ is a perfect rewriting of q and $\mathcal{T}$ over H-complete ABoxes. The claim follows since $\mathcal{A}_H$ is H-complete by definition. □

So, to generate a C2RPQ rewriting we can now focus on constructing rewritings over H-complete ABoxes.

Tree Witnesses. Consider a CQ q and a knowledge base $(\mathcal{T}, \mathcal{A})$. Suppose that, for some tuple a in $\mathsf{ind}(A)$, there is a homomorphism h from $q(a)$ to $\mathcal{C}_{\mathcal{T},\mathcal{A}}$. Then h partitions $q(a)$ into the atoms mapped by h to the ABox part and atoms mapped by h to the trees $\mathcal{C}_{\mathcal{T}}^{\exists R(a)}$ of the anonymous part of $\mathcal{C}_{\mathcal{T},\mathcal{A}}$. The tree-witness rewriting of q and $\mathcal{T}$ we are going to present now lists all possible partitions of the atoms of q into such subsets. We begin with an example which illustrates this idea.

[1] In database theory, a query is monotonic if for each pair of databases I and J over the same schema, $I \subseteq J$ implies $q(I) \subseteq q(J)$

Example 5.2.2. Consider the TBox $\mathcal{T}$ with the concept and role inclusions

$$\begin{aligned} A &\sqsubseteq \exists R.\top, \\ R &\sqsubseteq S^-, \\ \exists S.\top &\sqsubseteq B, \\ B &\sqsubseteq \exists T.\top, \\ B &\sqsubseteq \exists P.\top \end{aligned}$$

and the CQ

$$q(x) \leftarrow \exists y, z\ R(x, y) \wedge T(y, z).$$

We recall that if the canonical model $\mathcal{C}_{\mathcal{T},\mathcal{A}}$ for some ABox $\mathcal{A}$ contains some individuals $a \in A^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$ and $b \in B^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$, then $\mathcal{C}_{\mathcal{T},\mathcal{A}}$ must also contain the trees $\mathcal{C}_{\mathcal{T}}^{\exists R(a)}$ and $\mathcal{C}_{\mathcal{T}}^{\exists T(b)}$. Let us consider all the different ways of obtaining certain answers to the query by checking all possible homomorphisms from atoms of $q(x)$ to $\mathcal{C}_{\mathcal{T},\mathcal{A}}$ so that the answer variable x is mapped to $\mathsf{ind}(A)$. First, the variables x, y and z can be mapped to ABox individuals. Also, x and y can be mapped to ABox individuals, a and b, and if b is in $B^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$, then there is a homomorphism h_1 from the last atom of $q(a)$ to the anonymous part; this is shown in Fig. 5.2. Another way to obtain a homomorphism is to map only x to an ABox individual, a, and if a is in $A^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$ then the whole of $q(a)$ can be homomorphically mapped to the anonymous part; see h_2 in Fig. 5.2. The possible ways of mapping subsets of a query to the anonymous part of the canonical model are called *tree witnesses* [72]. The three tree witnesses for $q(x)$ and $\mathcal{T}$ found above give rise to the rewriting $q_{tw}(x)$ of $q(x)$ and $\mathcal{T}$ over H-complete

ABoxes as the union of the following conjunctive queries:

$$q_{1tw}(x) \leftarrow \exists y, z\ R(x,y) \wedge T(y,z),$$
$$q_{2tw}(x) \leftarrow \exists y\ R(x,y) \wedge B(y),$$
$$q_{3tw}(x) \leftarrow A(x).$$

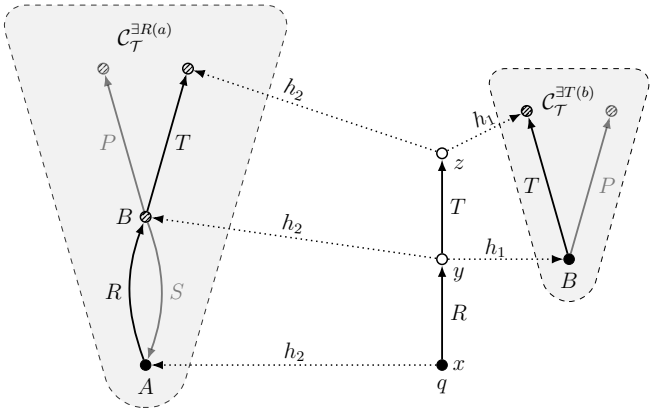

Figure 5.2: Homomorphisms from subsets of q to $\mathcal{C}_{\mathcal{T}}^{\exists R(a)}$ and $\mathcal{C}_{\mathcal{T}}^{\exists T(b)}$.

We now recall from [72] a general definition of the tree-witness rewriting over H-complete ABoxes. Let $\mathcal{T}$ be an $\mathcal{ELHI}_h^{lin}$ TBox in normal form and q a CQ with at least one existentially quantified variable in its body. Consider a pair $\mathbf{t} = (\mathbf{t}_r, \mathbf{t}_i)$ of disjoint sets of terms in q, where $\mathbf{t}_i$ is non-empty and contains only existentially quantified variables ($\mathbf{t}_r$, on the other hand, can be empty and can contain answer variables and individual names). Let

$$q_{\mathbf{t}} = \{S(\boldsymbol{z}) \in q \mid \boldsymbol{z} \subseteq \mathbf{t}_r \cup \mathbf{t}_i \text{ and } \boldsymbol{z} \not\subseteq \mathbf{t}_r\}.$$

Then $\mathbf{t}$ is defined as a *tree witness* for q and $\mathcal{T}$ generated by $\exists R.\top$ if the following two conditions are satisfied:

(a) there exists a homomorphism h from $q_{\mathbf{t}}$ to $\mathcal{C}_{\mathcal{T}}^{\exists R(a)}$, for some a, such that $\mathbf{t}_r =$

$\{z \mid h(z) = a\}$ and $\mathbf{t}_i$ contains the remaining variables in $q_\mathbf{t}$,

(b) $q_\mathbf{t}$ is a minimal subset of q such that, for any $y \in \mathbf{t}_i$, every atom in q containing y belongs to $q_\mathbf{t}$.

Note that unary atoms with arguments in $\mathbf{t}_r$ or binary atoms with both arguments in $\mathbf{t}_r$ do not belong to $q_\mathbf{t}$ and, therefore, condition (a) does not require them to be homomorphically mapped into $\mathcal{C}_\mathcal{T}^{\exists R(a)}$. The terms in $\mathbf{t}_r$ (if any) are called the *roots* of $\mathbf{t}$ and the (existentially quantified) variables in $\mathbf{t}_i$ the *interior* of $\mathbf{t}$ [72]. The homomorphism h in condition (a) is not necessarily unique; however, it is important that all roots are mapped to a and all variables of the interior are not mapped to a. Thus, $q_\mathbf{t}$ can contain at most one individual name, a; if $q_\mathbf{t}$ does not contain an individual name then the choice of a is irrelevant. Condition (b) reflects the fact that if a homomorphism from q to the canonical model of $(\mathcal{T}, \mathcal{A})$, for some $\mathcal{A}$, maps a variable y of an atom $R(y, z)$ to a non-root of a tree $\mathcal{C}_\mathcal{T}^{\exists R(a)}$ then the other variable of the atom must be mapped to the same tree. Let $\mathbf{t} = (\mathbf{t}_r, \mathbf{t}_i)$ be a tree witness for q and $\mathcal{T}$. Consider the following formula from [72]:

$$tw_\mathbf{t} = \exists u \, [\bigwedge_{x \in \mathbf{t}_r} (x = u) \; \wedge \bigvee_{\substack{B \sqsubseteq \exists R.\top \in \mathcal{T} \\ \mathbf{t} \text{ generated by } \exists R.\top}} B(u)],$$

whose free variables are the roots, $\mathbf{t}_r$, of $\mathbf{t}$. The formula $tw_\mathbf{t}$ describes the ABox individuals that root the trees in the anonymous part of $\mathcal{C}_{\mathcal{T},\mathcal{A}}$ into which the atoms $q_\mathbf{t}$ of the tree witness $\mathbf{t}$ can be homomorphically mapped. More formally, if $\mathcal{I}_A \models tw_\mathbf{t}(a, \ldots, a)$, for some $a \in \mathsf{ind}(A)$, then $\mathcal{C}_{\mathcal{T},\mathcal{A}}$ contains the tree $\mathcal{C}_\mathcal{T}^{\exists R(a)}$, and so there is a homomorphism from $q_\mathbf{t}$ to $\mathcal{C}_{\mathcal{T},\mathcal{A}}$ that maps all the roots of $\mathbf{t}$ to a. Conversely, if there is a homomorphism from $q_\mathbf{t}$ to $\mathcal{C}_{\mathcal{T},\mathcal{A}}$ such that all the roots of $\mathbf{t}$ are mapped to a (but all the variables from the interior, $\mathbf{t}_i$, of $\mathbf{t}$ are mapped to labelled nulls) then $\mathcal{I}_A \models tw_\mathbf{t}(a, \ldots, a)$.

Continuing with the terminology of [72], let $\Theta_{\mathcal{T}}^q$ be the set of tree witnesses for q and $\mathcal{T}$. Tree witnesses $\mathbf{t}$ and $\mathbf{t}'$ are said to be *conflicting* if $q_{\mathbf{t}} \cap q_{\mathbf{t}'} \neq \varnothing$ (in other words, the interior of one tree witness, say, $\mathbf{t}$, contains a root or an interior variable of the other, $\mathbf{t}'$, or the other way round, which makes it impossible to have both tree witnesses mapped into the anonymous part of $\mathcal{C}_{\mathcal{T},\mathcal{A}}$ at the same time). A set $\Theta \subseteq \Theta_{\mathcal{T}}^q$ of tree witnesses is said to be *independent* if any two distinct tree witnesses in Θ are non-conflicting. If Θ is independent then we can 'cut' the query q into independent subqueries in the following way. Consider a homomorphism that, for each $\mathbf{t} \in \Theta$, maps the subset $q_{\mathbf{t}}$ of q to the tree $\mathcal{C}_{\mathcal{T}}^{\exists R(a)}$, for some a, (provided that $\mathbf{t}$ is generated by $\exists R.\top$) and maps the remaining atoms in q to the ABox part of $\mathcal{C}_{\mathcal{T},\mathcal{A}}$. Such a homomorphism is possible if there is a tuple a in $\mathsf{ind}(A)$ such that the formula

$$q_{cut}^{\Theta} = \exists \mathbf{y} \Big((q \backslash q_{\Theta}) \ \wedge \bigwedge_{\mathbf{t} \in \Theta} tw_{\mathbf{t}} \Big)$$

holds in $\mathcal{I}_{\mathcal{A}}$ on a, where $q \backslash q_{\Theta}$ is the conjunction of all the atoms in q that do not belong to $q_{\mathbf{t}}$, for any $\mathbf{t} \in \Theta$. Conversely, if there is a homomorphism from $q(a)$ to $\mathcal{C}_{\mathcal{T},\mathcal{A}}$ then there exists an independent set Θ of tree witnesses such that $\mathcal{I}_{\mathcal{A}} \models q_{cut}^{\Theta}(a)$.

The following formula q_{tw} is called the *tree-witness rewriting* of q and $\mathcal{T}$ over H-complete ABoxes:

$$q_{tw} = \bigvee_{\Theta \in \Theta_{\mathcal{T}}^q \text{ independent}} q_{cut}^{\Theta}.$$

Example 5.2.3. Consider a $\mathcal{ELHI}_h^{\ell in}$ TBox in normal form with the following

concept inclusions

$$A_0 \sqsubseteq \exists R.\top \qquad A \sqsubseteq \exists T.\top$$

$$B_0 \sqsubseteq \exists U.\top \qquad B \sqsubseteq \exists S.\top$$

$$U \sqsubseteq U_1^- \qquad R \sqsubseteq R_1^-$$

$$\exists U_1.\top \sqsubseteq B \qquad \exists R_1.\top \sqsubseteq A$$

and the following CQ q

$$q(x, y'') \leftarrow \exists y, z, y', x', z', x'' \; R(x, y) \wedge T(y, z) \wedge T(y', z), \\ R(x', y') \wedge S(x', z') \wedge S(x'', z') \wedge U(y'', x'')$$

There are four tree witnesses for q and $\mathcal{T}$:

- $\mathbf{t}^1 = (\mathbf{t}_r^1, \mathbf{t}_i^1)$ generated by $\exists T.\top$ with $\mathbf{t}_r^1 = \{y, y'\}$ and $\mathbf{t}_i^1 = \{z\}$ and $q_{\mathbf{t}}^1 = \{T(y, z), T(y', z)\}$;
- $\mathbf{t}^2 = (\mathbf{t}_r^2, \mathbf{t}_i^2)$ generated by $\exists S.\top$ with $\mathbf{t}_r^2 = \{x', x''\}$ and $\mathbf{t}_i^2 = \{z'\}$ and $q_{\mathbf{t}}^2 = \{S(x', z'), S(x'', z')\}$;
- $\mathbf{t}^3 = (\mathbf{t}_r^3, \mathbf{t}_i^3)$ generated by $\exists R.\top$ with $\mathbf{t}_r^3 = \{x, x'\}$ and $\mathbf{t}_i^3 = \{y, y', z\}$ and $q_{\mathbf{t}}^3 = \{R(x, y), T(y, z),$ $T(y', z), R(x', y')\}$;
- $\mathbf{t}^4 = (\mathbf{t}_r^4, \mathbf{t}_i^4)$ generated by $\exists U.\top$ with $\mathbf{t}_r^4 = \{y', y''\}$ and $\mathbf{t}_i^4 = \{x', x'', z'\}$ and $q_{\mathbf{t}}^4 = \{R(x', y'), S(x', z'),$ $S(x'', z'), U(x'', y'')\}$;

Clearly, $\mathbf{t}^3$ and $\mathbf{t}^4$ are conflicting since, for example, $y' \in \mathbf{t}_i^3$ and $y' \in \mathbf{t}_r^4$; $\mathbf{t}^3$ is also conflicting with $\mathbf{t}^1$, since $y' \in \mathbf{t}_r^1$, but not with $\mathbf{t}^2$ because $\mathbf{t}_i^3 \cap (\mathbf{t}_r^2 \cup \mathbf{t}_i^2) = \varnothing$ and

$\mathbf{t}_i^2 \cap (\mathbf{t}_r^3 \cup \mathbf{t}_i^3) = \varnothing$. Also, $\mathbf{t}^4$ is conflicting with $\mathbf{t}^2$ since, for example, $z' \in \mathbf{t}_i^4$ and $\mathbf{t}_i^2 = \{z'\}$. Thus, we have the following 8 independent sets of tree witnesses:

$$\varnothing, \{\mathbf{t}^1\}, \{\mathbf{t}^2\}, \{\mathbf{t}^3\}, \{\mathbf{t}^4\}, \{\mathbf{t}^1, \mathbf{t}^2\}, \{\mathbf{t}^1, \mathbf{t}^4\}, \{\mathbf{t}^2, \mathbf{t}^3\},$$

which result in a tree-witness rewriting of 8 subqueries with the following tree witness formulas:

$$tw_{\mathbf{t}^1}(y, y') = \exists u\ ((u = y) \wedge (u = y') \wedge A(u)),$$
$$tw_{\mathbf{t}^2}(x', x'') = \exists u\ ((u = x') \wedge (u = x'') \wedge B(u)),$$
$$tw_{\mathbf{t}^3}(x, x') = \exists u\ ((u = x) \wedge (u = x') \wedge A_0(u)),$$
$$tw_{\mathbf{t}^4}(y', y'') = \exists u\ ((u = y') \wedge (u = y'') \wedge B_0(u).$$

Proposition 5.4. *Let $\mathcal{T}$ be an $\mathcal{ELHI}_h^{\ell in}$ TBox and $\boldsymbol{q}$ a CQ. For any H-complete ABox $\mathcal{A}$ and any tuple a in $\mathsf{ind}(\mathcal{A})$, we have $\mathcal{C}_{\mathcal{T},\mathcal{A}} \models \boldsymbol{q}(a)$ if and only if $\mathcal{I}_{\mathcal{A}} \models \boldsymbol{q}_{tw}(a)$.*

Proof. Proposition 27 in [70] shows that $\mathcal{C}_{\mathcal{T},\mathcal{A}} \models \boldsymbol{q}(a)$ if and only if $\mathcal{I}_{\mathcal{A}} \models \boldsymbol{q}_{tw}(a)$, for any H-complete ABox $\mathcal{A}$ and any tuple a in $\mathsf{ind}(\mathcal{A})$, where $\mathcal{T}$ is a QL TBox. Since $\mathcal{A}$ is H-complete, the proof involves only the structure of the anonymous part of $\mathcal{C}_{\mathcal{T},\mathcal{A}}$, which is isomorphic to the union of the trees $\mathcal{C}_{\mathcal{T}}^{\exists R(a)}$. Now, say that $\mathcal{T}$ is an $\mathcal{ELHI}_h^{\ell in}$ TBox and that ${\mathcal{C}_{\mathcal{T},\mathcal{A}}}^{\ominus}$ is the anonymous part of $\mathcal{C}_{\mathcal{T},\mathcal{A}}$, that is, the subset of the graph of $\mathcal{C}_{\mathcal{T},\mathcal{A}}$ where all the edges are attached to at least one null. From Definition 5.2 we know that:

- $A^{{\mathcal{C}_{\mathcal{T},\mathcal{A}}}^{\ominus}} = \{\sigma w_{\exists R} \mid R^- \in L(\mathsf{NFA}^-_{A,\mathcal{T}})\}$, for each concept name A;
- $P^{{\mathcal{C}_{\mathcal{T},\mathcal{A}}}^{\ominus}} = \{(\sigma w_{\exists R}, \sigma) \mid \mathsf{tail}(\sigma) \rightsquigarrow_{\mathcal{T},\mathcal{A}} w_{\exists R}, \mathcal{T} \models R \sqsubseteq P^-\} \cup$
  $\{(\sigma, \sigma w_{\exists R}) \mid \mathsf{tail}(\sigma) \rightsquigarrow_{\mathcal{T},\mathcal{A}} w_{\exists R}, \mathcal{T} \models R \sqsubseteq P\}$, for each role name P.

Thus, every node in ${\mathcal{C}_{\mathcal{T},\mathcal{A}}}^{\ominus}$ is a path $\sigma w_{\exists R}$, i.e., ${\mathcal{C}_{\mathcal{T},\mathcal{A}}}^{\ominus}$ is isomorphic to a subset of

the union of the $\mathcal{C}_{\mathcal{T}}^{\exists R(a)}$ trees, for some $a \in \mathsf{ind}(A)$. Therefore, the proof in [70] is analogous for $\mathcal{ELHI}_h^{lin}$ TBoxes. □

As stated above, a tree-witness rewriting p is a perfect rewriting of q and $\mathcal{T}$ over H-complete ABoxes. Consider an ABox $\mathcal{A} := \{R(a,b), R(b,c), A_0(c)\}$, the TBox $\mathcal{T}$ in Example 5.2.3, and the query

$$q(y) \leftarrow \exists y', z\ T(y,z) \wedge T(y',z).$$

There is one (independent) tree witness for $\boldsymbol{q}$, which is $\mathbf{t}^1$, and therefore, one tree-witness rewriting

$$p(y) \leftarrow \exists u, y', z\ ((u = y) \wedge (u = y') \wedge A(u)),$$

which by variable renaming can be simplified to

$$p(y) \leftarrow A(y).$$

Clearly, $\mathcal{A}$ is not H-complete with respect to $\mathcal{T}$ since it does not contain the atoms $A_0(a), A_0(b)$, and thus the answer of p over the ABox only, $p^{\mathcal{A}}$, holds an incomplete answer. In fact, $p^{\mathcal{A}} = \{(c)\}$ while $q^{(\mathcal{T},\mathcal{A})} = \{(a),(b),(c)\}$. To generate a rewriting over arbitrary ABoxes, the rewriting p needs to be rewritten into $p_{\mathcal{T}-ext}$ (see Definition 5.1) which replaces every atom in p with the formulas $A_{\mathcal{T}-ext}$ and $P_{\mathcal{T}-ext}$. This reformulation produces the following (C2)RPQ:

$$p_{\mathcal{T}-ext}(y) \leftarrow \exists p\ R^* A(y,p).$$

Now, ${p_{\mathcal{T}-ext}}^{\mathcal{A}} = \{(a),(b),(c)\}$.

Theorem 5.3. *Let $\mathcal{T}$ be an $\mathcal{ELHI}_h^{lin}$ TBox and $\boldsymbol{q}$ a CQ. For any ABox $\mathcal{A}$ and any tuple a in $\mathsf{ind}(\mathcal{A})$, we have $\mathcal{C}_{\mathcal{T},\mathcal{A}} \models \boldsymbol{q}(a)$ if and only if $\mathcal{I}_{\mathcal{A}} \models \boldsymbol{q}_{tw\mathcal{T}-ext}(a)$.*

Proof. The proof follows directly from Theorem 5.2 and Proposition 5.4. □

Corollary 5.2.2. *Let $\mathcal{T}$ be an $\mathcal{ELHI}_h^{lin}$ TBox and $\boldsymbol{q}$ a CQ. $\mathcal{T}$ is C2RPQ-rewritable with respect to $\boldsymbol{q}$.*

5.3 Complexity Analysis

In this section we establish results on the computational complexity of the problem of conjunctive query answering for $\mathcal{ELHI}_h^{lin}$ knowledge bases. The results are summarised in Figure 5.5. Note that the results on DL-lite$_\mathcal{R}$ and $\mathcal{ELH}$ are known. We present our complexity results in terms of query answering problems (as is common practice [35]), although technically the results refer to the decision versions of the problems.

Here we think of a CQ as a labelled directed multigraph $\langle N, E, \ell_N, \ell_E \rangle$ with N a set of nodes, $E \subseteq N \times N$ a set of edges, $\ell_N : N \to \mathsf{A}$ a function assigning labels to nodes from the set of concept names, and $\ell_E : E \to \mathsf{R}$ a function assigning labels to edges from the set of role names. For a given CQ q, the *graph of* q is composed as follows: *(i)* the set of nodes N is the set of terms in q; *(ii)* for each atom in the body of q of the form $A(x)$ there is a label assignment $x \to A \in \ell_N$; and *(iii)* for each atom of the form $R(x_1, x_2)$ there is an edge $(x_1, x_2) \in E$ and a label assignment $(x_1, x_2) \to R \in \ell_E$. For example, the graph of the query

$$q(x_1, x_2, x_3) \leftarrow \exists y_1, y_2, y_3 A_1(x_1) \wedge A_3(x_3) \wedge B_3(y_3) \wedge T(x_1, y_1) \wedge R(x_1, x_2) \\ \wedge T(x_2, x_3) \wedge S(x_3, y_2) \wedge S(y_3, y_2)$$

is illustrated in Figure 5.3. Also, in this section we adopt the notion of a *polytree*, which is simply a directed acyclic graph with the property that ignoring the directions on edges yields a graph with no cycles [43]. Another way of checking if

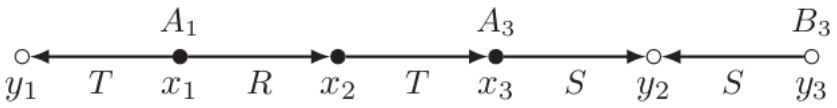

Figure 5.3: Conjunctive query as a labelled directed multigraph. The answer variables are filled in black.

a directed graph is a polytree is by checking that its symmetric closure does not contain cycles.

Definition 5.5. Given a CQ q and a set of terms $\boldsymbol{t}$ in q, we say that q is *polytree-transformable* with respect to $\boldsymbol{t}$ if there is a substitution h from the terms of q to terms of q, such that: *i)* for each $t_r \in \boldsymbol{t}$, $h(t_r) = \mathsf{root}_h$, where root_h is either a constant or a variable, and denotes the *root of* h; *ii)* for each term in q, t_i, such that $t_i \notin \boldsymbol{t}$, we have that $h(t_i) \neq \mathsf{root}_h$; *iii)* the graph of $h(q)$ is a polytree.

Example 5.3.1. An example of polytree-transformable query is given by the CQ q

$$q(x) \leftarrow \exists y_1, y_2, y_3, y_4, y_5\ R(x, y_1) \wedge R(x, y_2) \wedge S(y_1, y_3) \\ \wedge T(y_1, y_4) \wedge T(y_2, y_4) \wedge R(y_5, y_2),$$

which is not a polytree but is polytree-transformable with respect to $\{x, y_5\}$ via the substitution $h = \{y_1 \to y_2, y_5 \to x\}$ where $\mathsf{root}_h = x$. The transformation results in the query $h(q)$

$$h(q)(x) \leftarrow \exists y_2, y_3, y_4\ \wedge R(x, y_2) \wedge S(y_2, y_3) \wedge T(y_2, y_4).$$

Figure 5.4 shows the graphs of q and $h(q)$.

Definition 5.6. Consider a query q such that the graph of q is a polytree, a term root in q, a constant a and a TBox $\mathcal{T}$. We say that q *tree-maps* $\mathcal{C}_{\mathcal{T}}^{\exists R(a)}$ *on* root if there is a homomorphism h from the atoms of q to $\mathcal{C}_{\mathcal{T}}^{\exists R(a)}$ such that h is an injective function and $h(t) = a$ only if $t = \mathsf{root}$.

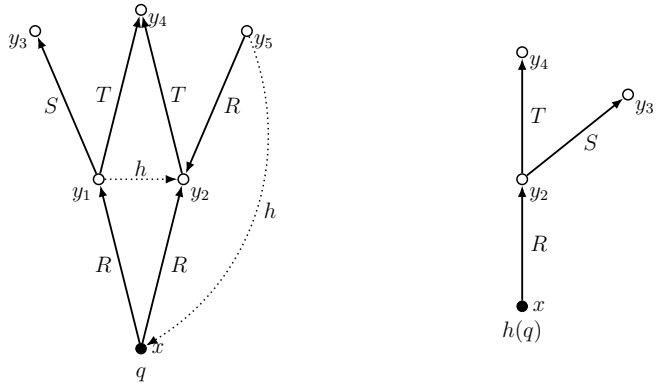

Figure 5.4: The graphs of q and $h(q)$ of Example 5.3.1

Proposition 5.5. *Consider a CQ q, a set of terms $\mathbf{t}$ in q, a TBox $\mathcal{T}$ and a pair $\mathbf{t} = (\mathbf{t}_r, \mathbf{t}_i)$ of disjoint sets of terms in q, where $\mathbf{t}_i$ is non-empty and contains only existentially quantified variables, and $\mathbf{t}_r$ contains the remaining terms of q. Then, there exists a homomorphism h from the atoms of q to $\mathcal{C}_{\mathcal{T}}^{\exists R(a)}$, for some a, such that $\mathbf{t}_r = \{z \mid h(z) = a\}$ if and only if q is polytree-transformable with respect to $\mathbf{t}_r$ via a homomorphism $\overline{h}$, and $\overline{h}(q)$ tree-maps $\mathcal{C}_{\mathcal{T}}^{\exists R(a)}$ on $\mathsf{root}_{\overline{h}}$.*

Proof. ($\Rightarrow$) From the definition of h, we know that $h(z) = a$ if and only if $z \in \mathbf{t}_r$, and therefore conditions *i)*, *ii)* in Definition 5.5 are satisfied. From Theorem 5.1 we know that, for every ABox $\mathcal{A}$, each null in $\mathcal{C}_{\mathcal{T}}^{\exists R(a)}$ is also a path σ on the generating relation $\rightsquigarrow_{\mathcal{T},\mathcal{A}}$, therefore the graph of $\mathcal{C}_{\mathcal{T}}^{\exists R(a)}$ (ignoring the direction on edges) does not contain cycles, and thus $h(q)$ is a polytree and condition *iii)* in Definition 5.5 is also satisfied. Now, we know that q is polytree-transformable with respect to $\mathbf{t}_r$ via $h(q)$ with $\mathsf{root}_h = a$ and that $h(q)$ is a polytree. It follows that $h(q)$ tree-maps $\mathcal{C}_{\mathcal{T}}^{\exists R(a)}$ on a via the identity function i, since h is a homomorphism from the atoms of q to $\mathcal{C}_{\mathcal{T}}^{\exists R(a)}$ and $i(t) = a$ only if $t = \mathsf{root}_h = a$. Also, the identity function i is injective by definition, and the claim follows.

($\Leftarrow$) We know that q is polytree-transformable with respect to $\mathbf{t}_r$ via a homomorphism $\overline{h}$, and $\overline{h}(q)$ tree-maps $\mathcal{C}_{\mathcal{T}}^{\exists R(a)}$ on $\mathsf{root}_{\overline{h}}$ via a homomorphism h^*. Now, set

$h = \overline{h} \circ h^*$. Then, h is a homomorphism from the atoms of q to $\mathcal{C}_{\mathcal{T}}^{\exists R(a)}$. From Definition 5.6 we know $h^*(t) = a$ only if $t = \mathsf{root}_{\overline{h}}$. Also, from Definition 5.6 we know that *i)* for each $t_r \in \mathbf{t}_r$, $\overline{h}(t_r) = \mathsf{root}_{\overline{h}}$, and that *ii)* for each term in q, t_i, such that $t_i \notin \mathbf{t}_r$, we have that $\overline{h}(t_i) \neq \mathsf{root}_h$. Thus, $\mathbf{t}_r = \{z \mid h(z) = a\}$ and the claim follows. □

Definition 5.7. Given an $\mathcal{ELHI}_h^{\ell in}$ TBox $\mathcal{T}$, an arbitrary individual name a and a complex concept of the form $\exists R$ appearing on the LHS of an axiom in $\mathcal{T}$. We define $\mathcal{C}_{\mathcal{T}_n}^{\exists R(a)}$ as the maximum subset of $\mathcal{C}_{\mathcal{T}}^{\exists R(a)}$ such that each null in $\mathcal{C}_{\mathcal{T}_n}^{\exists R(a)}$ is a sequence $a\omega_{\exists R}\omega_{\exists T_1}...\omega_{\exists T_{l-2}}$ of length $l \leqslant n+1$.

Lemma 5.3. *Given a polytree query q with root root, where each path starting from root is of length $l \leqslant n$, an $\mathcal{ELHI}_h^{\ell in}$ TBox $\mathcal{T}$ and an individual name a, q tree-maps $\mathcal{C}_{\mathcal{T}}^{\exists R(a)}$ on root only if q tree-maps $\mathcal{C}_{\mathcal{T}_n}^{\exists R(a)}$ on root.*

Proof. We prove this lemma by showing that, for each $i \geqslant n$, if q does not tree-map $\mathcal{C}_{\mathcal{T}_i}^{\exists R(a)}$ on root, then q does not tree-map $\mathcal{C}_{\mathcal{T}_{i+1}}^{\exists R(a)}$ on root. Let us now assume that q does not tree-map $\mathcal{C}_{\mathcal{T}_i}^{\exists R(a)}$ on root but it tree-maps $\mathcal{C}_{\mathcal{T}_{i+1}}^{\exists R(a)}$ on root. This is possible only if:

- $\mathcal{C}_{\mathcal{T}_{i+1}}^{\exists R(a)}$ contains at least one null x that is a sequence $a\omega_{\exists R}\omega_{\exists T_1}...\omega_{\exists T_i}$ of length $i+2$;
- one variable of q is mapped to x, since the mapping function is injective by definition;

Thus, n variables of q are mapped to $i+1$ nulls, $a\omega_{\exists R}, a\omega_{\exists R}\omega_{\exists T_1}, \ldots, a\omega_{\exists R}\omega_{\exists T_1}...\omega_{\exists T_i}$, for the definition of $\mathcal{C}_{\mathcal{T}}^{\exists R(a)}$ in Definition 5.2, item (3), and because the mapping function is injective. This is not possible since $i \geqslant n$. Therefore, for each $i \geqslant n$, if q does not tree-map $\mathcal{C}_{\mathcal{T}_i}^{\exists R(a)}$ on root, then q does not tree-map $\mathcal{C}_{\mathcal{T}_{i+1}}^{\exists R(a)}$ on root, and by

	IQs		CQs	
	Data	**Comb.**	**Data**	**Comb.**
DL-lite$_{\mathcal{R}}$	in AC_0	NL	in AC_0	NP
$\mathcal{ELHI}_h^{lin}$	NL	in P	NL	NP
$\mathcal{ELH}$	P	P	P	NP

Figure 5.5: Summary of complexity results of $\mathcal{ELHI}_h^{lin}$. All bounds are tight, unless otherwise stated. Our results are in the grey box.

induction, if q does not tree-map $\mathcal{C}_{\mathcal{T}_n}^{\exists R(a)}$ on root, then q does not tree-map $\mathcal{C}_{\mathcal{T}}^{\exists R(a)}$ on root. The claim follows. □

Theorem 5.4. *Answering ICs and CQs on $\mathcal{ELHI}_h^{lin}$ knowledge bases is* NLogSpace*-complete with respect to data complexity.*

Proof. It is known that the data complexity of answering IQs in $\mathcal{ELH}^{lin}$ [35] is NLogSpace-hard, so the same holds for $\mathcal{ELHI}_h^{lin}$, which is a proper extension of $\mathcal{ELH}^{lin}$. Membership in NLogSpace for IQ answering follows from the fact that we can rewrite instance queries to 2RPQs, and answering 2RPQs is in NLogSpace [11]. For CQs, the upper bound follows from our algorithm to compute a perfect rewriting of $\boldsymbol{q}$ for $\mathcal{T}$ as C2RPQ queries, and the fact that the data complexity of C2RPQ answering is NLogSpace-complete [11]. For both IQs and CQs, the rewriting algorithm relies solely on the query and the TBox. Since both the query and the TBox are considered fixed in the definition of data complexity, then producing the rewriting is done in constant time; therefore, the rewriting algorithm does not use more than logarithmic space. The lower bound follows from hardness for IQs. We also need to consider the cost of satisfiability of NIs, which is done by treating NIs and PIs separately, that is, removing the NIs from the TBox and checking the resulting knowledge base against a set of Boolean CQs (see Section 4.2) of linear

size with respect to the TBox. Since the TBox is fixed, then checking satisfiability of NIs is also done in logarithmic space. □

Theorem 5.5. *Answering CQs on $\mathcal{ELHI}_h^{lin}$ knowledge bases is* NP*-complete with respect to combined complexity.*

Proof. In this case, satisfiability check of NIs does not carry extra cost, as it is done by answering an additional set of Boolean CQs which can be given as input for the rewriting algorithm. Since the above set of BCQs is of linear size with respect to the TBox, then the cost of satisfiability check is included in the cost of answering CQs. For the upper bound for CQs, we devise a non-deterministic version of the rewriting algorithm for a given CQ $\boldsymbol{q}$ and a TBox $\mathcal{T}$. The algorithm is as follows:

`[step 1]` guess a $q_{\mathbf{t}} = \{S(\boldsymbol{z}) \in q \mid \boldsymbol{z} \subseteq \mathbf{t}_r \cup \mathbf{t}_i \text{ and } \boldsymbol{z} \not\subseteq \mathbf{t}_r\}$, i.e., a possible tree-witness. Since $q_{\mathbf{t}}$ is a subset of atoms in q, the guess is done in polynomial time;

`[step 2]` guess a homomorphism $\overline{h}$ from the atoms of $q_{\mathbf{t}}$ to the atoms of $q_{\mathbf{t}}$; this step is performed in non-deterministic polynomial time, as the space of guesses is of size 2^n, where n is the number of nodes in the graph of $q_{\mathbf{t}}$;

`[step 3]` check that each term $t \in \mathbf{t}_r$ maps to the same term, i.e., $\mathsf{root}_{\overline{h}}$;

`[step 4]` check if the graph of $\overline{h}(q_{\mathbf{t}})$ is a tree via a graph traversal (in polynomial time with respect to the size of $q_{\mathbf{t}}$);

`[step 5]` guess an $\exists R$ and generate $\mathcal{C}_{\mathcal{T}_m}^{\exists R(a)}$, where $a = \mathsf{root}_{\overline{h}}$ if $\mathsf{root}_{\overline{h}}$ is a individual name (or for an arbitrary individual a otherwise), and m is the length of the longest path in $\overline{h}(q_{\mathbf{t}})$ starting from $\mathsf{root}_{\overline{h}}$; the cost of generating $\mathcal{C}_{\mathcal{T}_m}^{\exists R(a)}$ is boned by $m \times |\mathcal{T}|$;

`[step 6]` check if $\overline{h}(q_{\mathbf{t}})$ tree-maps $\mathcal{C}_{\mathcal{T}_m}^{\exists R(a)}$ on $\mathsf{root}_{\overline{h}}$:

`[step 6.1]` guess an injective function h from the terms in $\overline{h}(q_{\mathbf{t}})$ to the terms in $\mathcal{C}_{\mathcal{T}_m}^{\exists R(a)}$, where the spaces of guesses is $(m \times |\mathcal{T}|)!$;

`[step 6.2]` check if $h(\overline{h}(q_{\mathbf{t}})) \subseteq \mathcal{C}_{\mathcal{T}_m}^{\exists R(a)}$;

`[step 7]` check if $q_{\mathbf{t}}$ is a minimal subset of q such that, for any $y \in \mathbf{t}_i$, every atom in q containing y belongs to $q_{\mathbf{t}}$;

`[step 8]` rewrite $\boldsymbol{q}'$ accordingly and generate ${\boldsymbol{q}'}_{\mathcal{T}-ext}$;

`[step 9]` check if ${\boldsymbol{q}'}_{\mathcal{T}-ext}$ is true when evaluated on the ABox.

Since $\boldsymbol{q}'_{\mathcal{T}-ext}$ is a C2RPQ and the data complexity of answering C2RPQs over a plain database is in NLogSpace, the problem of answering $\boldsymbol{q}$ is in NP. This NP bound is optimal, since CQ answering is already NP-hard over an ABox alone. □

Theorem 5.6. *Answering IQs on $\mathcal{ELHI}_h^{\ell in}$ knowledge bases is in* PTime *with respect to combined complexity.*

Proof. To answer IQs, for a concept instance query we can build in polynomial time the NFA, and then answer the path query resulting from the rewriting over the ABox. For a role instance query, the rewriting can be constructed in linear time and the resulting query is a 2RPQ. We also need to consider the cost of satisfiability of NIs, which is done by checking the knowledge base against a set of boolean CQs of linear size with respect to the TBox. By definition, those boolean CQs contain at most two atoms and two variables. Thus, we can generate the rewriting of each BCQ in polynomial time by substituting the non-deterministic guesses of the algorithm above with the following deterministic steps:

`[step 1]` generate all the possible tree-witnesses $q_{\mathbf{t}}$, which are at most three;

`[step 2]` generate the homomorphisms $\overline{h}$ from the atoms of $q_{\mathbf{t}}$ to the atoms of $q_{\mathbf{t}}$; the number of homomorphisms is bounded by 2^2;

`[step 5]` generate the set of all $\mathcal{C}_{\mathcal{T}_m}^{\exists R(a)}$, whose size is bounded by $|\mathcal{T}|$; note that m is at most 1;

`[step 6]` check if $\overline{h}(q_{\mathbf{t}})$ tree-maps $\mathcal{C}_{\mathcal{T}_m}^{\exists R(a)}$ on $\mathsf{root}_{\overline{h}}$; by definition, this is possible only if the term in $\overline{h}(q_{\mathbf{t}})$ that is not $\mathsf{root}_{\overline{h}}$ is mapped via h to a term in $\mathcal{C}_{\mathcal{T}_m}^{\exists R(a)}$ that is not a, and then $h(\overline{h}(q_{\mathbf{t}})) \subseteq \mathcal{C}_{\mathcal{T}_m}^{\exists R(a)}$; since m is at most 1, then the number of injective functions h is bounded by $|\mathcal{T}|$.

At this point, we only need to check each rewritten query against the ABox alone, thus not using more than logarithmic space for each query, since they are boolean C2RPQs. □

5.4 Discussion

In Chapter 4, we introduced a new ontology language, named $\mathcal{ELHI}_h^{lin}$, which strictly extends the known ontology languages DL-Lite$_\mathcal{R}$ and linear $\mathcal{ELH}$. Following from that, in this chapter we have proposed a query rewriting algorithm for answering conjunctive queries under $\mathcal{ELHI}_h^{lin}$ knowledge bases, with C2RPQs as target language. This algorithm extends the *tree witness* rewriting of [72] and uses the NFA-based rewriting technique presented in Chapter 4. Since C2RPQs can be straightforwardly expressed in SPARQL 1.1 by means of property paths, our approach is therefore directly applicable to real-world querying settings.

Lastly, we have undertaken a complexity analysis for query answering under $\mathcal{ELHI}_h^{lin}$. We have analysed the computational cost of query answering in terms of both data complexity and combined complexity. We have shown that answering instance queries under $\mathcal{ELHI}_h^{lin}$ is NLogSpace-complete for data complexity and in PTime for combined complexity; we have also shown that answering CQs under $\mathcal{ELHI}_h^{lin}$ is NLogSpace-complete for data complexity and NP-complete for combined complexity.

Chapter 6

Conclusion

6.1 Summary of Thesis Contributions

The research in this thesis was motivated by the problem of integrating and querying multiple heterogeneous Linked Data sets through ontological rules. In the first part of this thesis, we have proposed a formalisation of the notion of a peer-to-peer Linked Data integration system, where the mappings between peers comprise schema-level mappings and equality constraints between different IRIs; we call this formalism an *RDF Peer System* (RPS). We have shown that the semantics of the mappings preserve tractability of the *conjunctive SPARQL query answering problem over RPSs*, that is, answering queries expressed in the conjunctive fragment of the SPARQL query language against the data stored in the RDF sources and the set of constraints given by the RPSs mappings. In more detail, we have shown that answering Basic Graph Pattern (BGP) SPARQL queries on an RPS can be done in polynomial time in terms of data complexity. The key novelty of the RPS is to achieve a tractable semantics for the integration of multiple RDF sources with arbitrary mapping topologies, whereas previous techniques applied on the same set of RDF sources may give rise to a set of undecidable rules.

To illustrate the practical usage of our approach, we have presented a larger-scale real-world case study, undertaken as part of the INSPIRE project [92], which addresses the domain of career guidance, particularly *career transitions*. We have undertaken the integration of two databases: the L4All RDF/S ontology arising from the L4All and MyPlan projects, capturing the work and educational experiences of lifelong learners [93]; and a new RDF/S ontology we have designed based on the information published on *LinkedIn*, a business and employment-oriented service that operates via websites and mobile apps. Through the case study, we have shown how to exploit the RPS mapping language to materialise an integrated version of the two heterogeneous data sources.

Next, we have addressed the problem of SPARQL query rewriting under RPSs. We have firstly compared the problem with CQ rewriting under TGDs and we have seen that is not possible to generate a SPARQL 1.0 query as the perfect rewriting of an input BGP SPARQL query under general RPSs, as the RPS peer mappings are generally not FO-rewritable rules; this is a major drawback of general RPSs since data materialisation is required to exploit their full semantics. Following this, we have taken into account well-known FO-rewritable sets of TGDs and compared them to our peer mappings; thus we have outlined some restricted forms of RPSs for which it is possible to generate a SPARQL 1.0 query as a perfect rewriting.

Next, we have presented a middleware system based on these restricted forms of RPSs and we have undertaken an empirical evaluation of its behaviour. We have seen that, when BGP SPARQL queries are rewritten according to our algorithm and posed over multiple RDF sources, the amount of information retrieved increases significantly due to the provision of interoperability between heterogeneous vocabularies. In addition, the approach does not seem to compromise query execution time since, overall, the response time of our system was seen to be not greater than the maximum query response time over the single datastores. However, further

empirical evaluation is necessary, and this is an area of future work.

With the adoption of the more recent standard SPARQL 1.1 and its property paths we are able to extend the expressivity of the target language beyond FO by including regular expressions in the body of the target SPARQL queries. Following this idea, in the second part of the thesis we have stepped away from the language of RPSs to conduct a study on C2RPQ-rewritability under a broader ontology language. We have defined $\mathcal{ELHI}_h^{lin}$ (harmless linear $\mathcal{ELHI}$), an ontology language that generalises both the DL-Lite$_\mathcal{R}$ and linear $\mathcal{ELH}$ description logics. We have proved the rewritability of instance queries (queries with a single atom in their body) under $\mathcal{ELHI}_h^{lin}$ knowledge bases with C2RPQs as the target language, presenting a query rewriting algorithm that makes use of non-deterministic finite-state automata. Following from that, we have proposed a query rewriting algorithm for answering conjunctive queries under $\mathcal{ELHI}_h^{lin}$ knowledge bases, with C2RPQs as the target language. This algorithm extends the *tree witness* rewriting of [72] and uses the above NFA-based rewriting technique. Since C2RPQs can be straightforwardly expressed in SPARQL 1.1 by means of property paths, we believe that our approach is directly applicable to real-world querying settings. In addition, the complexity of answering C2RPQs is in the highly tractable class NLOGSPACE with respect to data complexity. It follows that, under C2RPQ-rewritable rules, the query answering problem would also be highly tractable. Ultimately, these results could be used to potentially find larger subsets of SPARQL 1.1-rewritable RPSs than those identified in the first part of this thesis.

Lastly, we have undertaken a complexity analysis for query answering under $\mathcal{ELHI}_h^{lin}$. We have analysed the computational cost of query answering in terms of both data complexity (where the ontology and the query are fixed and the data alone is a variable input) and combined complexity (where query, ontology and data all constitute the variable input). We have shown that answering instance queries under

$\mathcal{ELHI}_h^{\ell in}$ is NLogSpace-complete for data complexity and in PTime for combined complexity; we have also shown that answering CQs under $\mathcal{ELHI}_h^{\ell in}$ is NLogSpace-complete for data complexity and NP-complete for combined complexity.

6.2 Future Work

As future work, we intend to verify if $\mathcal{ELHI}_h^{\ell in}$ enjoys the C2RPQ-rewritability property when C2RPQs - instead of CQs - are given as input queries. The results in [19] demonstrate that, in data complexity, the cost of answering C2RPQs is NLogSpace-complete under DL-Lite$_\mathcal{R}$. Upon first analysis, we predict that NLogSpace-completeness is retained when we extend DL-Lite$_\mathcal{R}$ with a restricted form of inverse roles, as in $\mathcal{ELHI}_h^{\ell in}$; if our prediction is correct, then a rewriting of C2RPQs to C2RPQs under $\mathcal{ELHI}_h^{\ell in}$ might be feasible.

We also plan to investigate more expressive ontology languages that may lie within the scope of C2RPQ-rewritability. In particular, it would be interesting to investigate more general ways of integrating inverse roles into $\mathcal{ELHI}_h^{\ell in}$, and to include complex role chains and unions in the language, as in [85].

We point out that studying C2RPQ-rewritability of DLs may lead to finding new languages that are candidates for future standards recommendations. For instance, in OWL 2 - a new version of the OWL ontology language that is currently a W3C candidate recommendation - scalability requirements are addressed by *profiles* which are subsets of the language that enjoy desirable computational properties. In particular, the OWL 2 QL profile[1], based on the DL-Lite$_\mathcal{R}$ description logic, was expressly designed for query answering via a pure query rewriting approach; however it is restricted by the FO-rewritability of the underpinning logic. Thus, finding C2RQP-rewritable DLs may result in finding more expressive profiles for which

[1] `https://www.w3.org/TR/owl2-profiles/#OWL_2_QL`

query answering can also be done via query rewriting.

Future work includes also an empirical evaluation of our rewriting algorithms on real-world databases. Since query answering is not tractable in the size of queries (i.e., NP-complete for CQs), we need to consider what shape and size of rewritings we should aim at to make query answering under $\mathcal{ELHI}_h^{lin}$ efficient. In particular, we would like to investigate what causes exponentially long rewritings of CQs over $\mathcal{ELHI}_h^{lin}$ ontologies and to check empirically whether those scenarios occur in real-world queries and ontologies.

Finally, future work on our proposed RPS framework includes developing a full end-to-end reference implementation of the RPS system design, to encompass all the components and functionalities of the peer-based Linked Data integration system discussed in Section 3.5.3. We also aim to optimise the evaluation of RPS mapping rules in the general case, which require (full) data materialisation. In particular, we intend to investigate query answering techniques that follow a "hybrid" approach, i.e., partial data materialisation and partial query rewriting. In addition, we plan to use our findings in the second part of this thesis to find larger subsets of RPSs that support a "pure" rewriting approach. An immediate step in this direction could be comparing the set of graph mapping assertions to $\mathcal{ELHI}_h^{lin}$ axioms and implementing the NFA-based algorithm introduced in Chapter 5, and thus leveraging the expressive power of SPARQL 1.1's property paths in the rewriting.

Bibliography

[1] S. Abiteboul and O. M. Duschka. Complexity of answering queries using materialized views. In *Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 254–263. ACM, 1998.

[2] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of databases: the logical level*. Addison-Wesley Longman Publishing Co., Inc., 1995.

[3] D. Allemang and J. Hendler. *Semantic web for the working ontologist: effective modeling in RDFS and OWL*. Elsevier, 2011.

[4] M. Arenas, C. Gutierrez, and J. Pérez. On the semantics of sparql. In *Semantic Web Information Management*, pages 281–307. Springer, 2010.

[5] A. Artale, D. Calvanese, R. Kontchakov, and M. Zakharyaschev. The DL-Lite family and relations. *Journal of Artificial Intelligence Research*, 36(1):1–69, 2009.

[6] F. Baader, S. Brandt, and C. Lutz. Pushing the el envelope. In *IJCAI*, volume 5, pages 364–369, 2005.

[7] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, New York, NY, USA, 2003.

[8] F. Baader and W. Nutt. Basic description logics. In *Description logic handbook*, pages 43–95, 2003.

[9] J.-F. Baget, M. Leclère, M.-L. Mugnier, and E. Salvat. On rules with existential variables: Walking the decidability line. *Artificial Intelligence*, 175(9):1620–1654, 2011.

[10] P. Barceló, L. Libkin, A. W. Lin, and P. T. Wood. Expressive languages for path queries over graph-structured data. *ACM Transactions on Database Systems (TODS)*, 37(4):31, 2012.

[11] P. Barceló Baeza. Querying graph databases. In *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGAI symposium on Principles of database systems*, pages 175–188. ACM, 2013.

[12] C. Beeri and M. Y. Vardi. *The implication problem for data dependencies*, pages 73–85. Springer Berlin Heidelberg, Berlin, Heidelberg, 1981.

[13] P. A. Bernstein, F. Giunchiglia, A. Kementsietsidis, J. Mylopoulos, L. Serafini, and I. Zaihrayeu. Data management for peer-to-peer computing: A vision. Technical report, University of Trento, 2002.

[14] G. Berry and R. Sethi. From regular expressions to deterministic automata. *Theoretical computer science*, 48:117–126, 1986.

[15] M. Bienvenu, P. Hansen, C. Lutz, and F. Wolter. First order-rewritability and containment of conjunctive queries in horn description logics. In *DLOG*, 2016.

[16] M. Bienvenu, C. Lutz, and F. Wolter. First-order rewritability of atomic queries in horn description logics. In *IJCAI*, 2013.

[17] M. Bienvenu and M. Ortiz. Ontology-mediated query answering with data-tractable description logics. In *Reasoning Web International Summer School*, pages 218–307. Springer, 2015.

[18] M. Bienvenu, M. Ortiz, and M. Simkus. Conjunctive regular path queries in lightweight description logics. In *IJCAI*, 2013.

[19] M. Bienvenu, M. Ortiz, and M. Simkus. Regular path queries in lightweight description logics: Complexity and algorithms. *Journal of Artificial Intelligence Research*, 53:315–374, 2015.

[20] S. Bischof, M. Krötzsch, A. Polleres, and S. Rudolph. Schema-agnostic query rewriting in sparql 1.1. In *International Semantic Web Conference*, pages 584–600. Springer International Publishing, 2014.

[21] C. Buil-Aranda, M. Arenas, and O. Corcho. Semantics and optimization of the sparql 1.1 federation extension. In *Proceedings of the 8th extended semantic web conference on The semanic web: research and applications - Volume Part II*, ESWC'11, pages 1–15, Berlin, Heidelberg, 2011. Springer-Verlag.

[22] L. Cabibbo. The expressive power of stratified logic programs with value invention. *Information and Computation*, 147(1):22–56, 1998.

[23] M. Cai and M. Frank. RDFPeers: a scalable distributed RDF repository based on a structured peer-to-peer network. In *Proc. of WWW*, pages 650–657, 2004.

[24] M. Cai, M. Frank, B. Yan, and R. MacGregor. A subscribable peer-to-peer RDF repository for distributed metadata management. *Web Semantics*, 2(2):109–130, 2004.

[25] A. Calì. Reasoning in data integration systems: why lav and gav are siblings. In *International Symposium on Methodologies for Intelligent Systems*, pages 562–571. Springer, 2003.

[26] A. Calì, D. Calvanese, G. De Giacomo, and M. Lenzerini. Accessing data integration systems through conceptual schemas. In *Proc. of the 20th International Conference on Conceptual Modeling (ER 2001)*, pages 270–284, 2001.

[27] A. Calì, G. Gottlob, and M. Kifer. Taming the infinite chase: Query answering under expressive relational constraints. *Journal of Artificial Intelligence Research*, 48:115–174, 2013.

[28] A. Calì, G. Gottlob, and T. Lukasiewicz. Datalog±: a unified approach to ontologies and integrity constraints. In *Proceedings of the 12th International Conference on Database Theory*, pages 14–30. ACM, 2009.

[29] A. Calì, G. Gottlob, and T. Lukasiewicz. A general datalog-based framework for tractable query answering over ontologies. *Web Semantics: Science, Services and Agents on the World Wide Web*, 14:57–83, 2012.

[30] A. Calì, G. Gottlob, and A. Pieris. Advanced processing for ontological queries. *PVLDB*, 3(1):554–565, 2010.

[31] A. Calì, G. Gottlob, and A. Pieris. Query answering under non-guarded rules in Datalog±. In *Proc. of RR*, pages 1–17, 2010.

[32] A. Calı, G. Gottlob, and A. Pieris. New expressive languages for ontological query answering. In *Proc. of AAAI*, volume 2011, 2011.

[33] A. Calì, D. Lembo, and R. Rosati. Query rewriting and answering under constraints in data integration systems. In *Proc. of the 18th Int. Joint Conference on Artificial Intelligence*, pages 16–21, 2003.

[34] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Tractable reasoning and efficient query answering in description logics: The DL-lite family. *Journal of Automated reasoning*, 39(3):385–429, 2007.

[35] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Data complexity of query answering in description logics. In *Proc. of the 24th Int. Joint Conference on Artificial Intelligence*, 2015.

[36] D. Calvanese, G. De Giacomo, M. Lenzerini, and R. Rosati. Logical foundations of peer-to-peer data integration. In C. Beeri and A. Deutsch, editors, *PODS*, pages 241–251. ACM, 2004.

[37] D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Y. Vardi. Rewriting of regular expressions and regular path queries. In *Proceedings of the eighteenth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 194–204. ACM, 1999.

[38] D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Y. Vardi. Query processing using views for regular path queries with inverse. In *Proc. of the 19th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS 2000)*, pages 58–66, 2000.

[39] D. Calvanese, G. D. Giacomo, and M. Lenzerini. Conjunctive query containment and answering under description logic constraints. *ACM Transactions on Computational Logic (TOCL)*, 9(3):22, 2008.

[40] D. Calvanese, M. Giese, D. Hovland, and M. Rezk. Ontology-based integration of cross-linked datasets. In *International Semantic Web Conference*, pages 199–216. Springer, 2015.

[41] A. K. Chandra and P. M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *Proceedings of the Ninth Annual ACM Symposium on Theory of Computing*, STOC '77, pages 77–90, New York, NY, USA, 1977. ACM.

[42] G. Correndo, M. Salvadores, I. Millard, H. Glaser, and N. Shadbolt. SPARQL query rewriting for implementing data integration over Linked Data. In *Proc. of EDBT/ICDT Wksp*, 2010.

[43] S. Dasgupta. Learning polytrees. In *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*, pages 134–141. Morgan Kaufmann Publishers Inc., 1999.

[44] A. Deutsch, A. Nash, and J. Remmel. The chase revisited. In *Proceedings of the Twenty-seventh ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '08, pages 149–158, New York, NY, USA, 2008. ACM.

[45] M. M. Dimartino. Peer-based query rewriting in SPARQL for semantic integration of linked data. In *Proceedings of the ISWC 2015 Doctoral Consortium Co-located with the 14th International Semantic Web Conference (ISWC 2015), Bethlehem, Pennsylvania, USA, October 12th, 2015.*, 2015.

[46] M. M. Dimartino, A. Calì, A. Poulovassilis, and P. T. Wood. Implementing peer-to-peer semantic integration of linked data. In *Data Science - 30th British International Conference on Databases, BICOD 2015, Edinburgh, UK, July 6-8, 2015, Proceedings*, pages 41–45, 2015.

[47] M. M. Dimartino, A. Calì, A. Poulovassilis, and P. T. Wood. Peer-to-peer semantic integration of Linked Data. In *Proc. of EDBT/ICDT Workshops*, 2015.

[48] M. M. Dimartino, A. Calì, A. Poulovassilis, and P. T. Wood. Query rewriting under linear *EL* knowledge bases. In *Web Reasoning and Rule Systems - 10th International Conference, RR 2016, Aberdeen, UK, September 9-11, 2016, Proceedings*, pages 61–76, 2016.

[49] M. M. Dimartino, A. Calì, A. Poulovassilis, and P. T. Wood. Efficient ontological query answering by rewriting into graph queries. In *International Conference on Flexible Query Answering Systems*, 2019.

[50] M. Duerst and M. Suignard. RFC 3987: Internationalized Resource Identifiers (IRIs). RFC 3987 (Proposed Standard), see `http://www.ietf.org/rfc/rfc3987.txt`, January 2005.

[51] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data exchange: semantics and query answering. *Theoretical Computer Science*, 336(1):89–124, 2005.

[52] E. Franconi, G. Kuper, A. Lopatenko, and L. Serafini. A robust logical and computational characterisation of peer-to-peer database systems. In K. Aberer, M. Koubarakis, and V. Kalogeraki, editors, *Databases, Information Systems, and Peer-to-Peer Computing*, volume 2944 of *Lecture Notes in Computer Science*, pages 64–76. Springer Berlin Heidelberg, 2004.

[53] A. Fuxman, P. G. Kolaitis, R. J. Miller, and W.-C. Tan. Peer data exchange. *ACM Transactions on Database Systems (TODS)*, 31(4):1454–1498, 2006.

[54] G. Gottlob, G. Orsi, and A. Pieris. Ontological queries: Rewriting and optimization (extended version). *CoRR*, abs/1112.0343, 2011.

[55] G. Gottlob and C. Papadimitriou. On the complexity of single-rule datalog queries. *Information and Computation*, 183(1):104–122, 2003.

[56] C. Gutierrez, C. Hurtado, and A. O. Mendelzon. Foundations of semantic web databases. In *Proceedings of the Twenty-third ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '04, pages 95–106, New York, NY, USA, 2004. ACM.

[57] A. Halevy, Z. G. Ives, D. Suciu, I. Tatarinov, A. Y. Halevy, Z. G. Ives, D. Suciu, and I. Tatarinov. Schema mediation in peer data management systems. In *In Proc. of ICDE*, 2003.

[58] A. Y. Halevy, Z. G. Ives, J. Madhavan, P. Mork, D. Suciu, and I. Tatarinov. The piazza peer data management system. *IEEE Transactions on Knowledge and Data Engineering*, 16(7):787–798, 2004.

[59] A. Y. Halevy, Z. G. Ives, P. Mork, and I. Tatarinov. Piazza: Data management infrastructure for semantic web applications. In *Proceedings of the 12th International Conference on World Wide Web*, WWW '03, pages 556–567, New York, NY, USA, 2003. ACM.

[60] Y. Halevy, G. Ives, D. Suciu, and I. Tatarinov. Schema mediation for large-scale semantic data sharing. *The VLDB Journal*, 14(1):68–83, Mar. 2005.

[61] H. Halpin, P. J. Hayes, J. P. McCusker, D. L. McGuinness, and H. S. Thompson. When owl: sameas isn't the same: An analysis of identity in linked data. In *International semantic web conference*, pages 305–320. 2010.

[62] P. Hansen, C. Lutz, I. Seylan, and F. Wolter. Query rewriting under EL TBoxes: Efficient algorithms. In *Description Logics*, 2014.

[63] P. Hansen, C. Lutz, I. Seylan, and F. Wolter. Efficient query rewriting in the description logic EL and beyond. In *IJCAI*, 2015.

[64] S. Harris and A. Seaborne. SPARQL 1.1 Query Language, W3C Recommendation 21 March 2013, 2013.

[65] P. Hayes and B. McBride. RDF semantics. W3C recommendation, 2004. *Online: http://www. w3. org/TR/2004/REC-rdf-mt-20040210*.

[66] T. Heath and C. Bizer. Linked data: Evolving the web into a global data space. *Synthesis lectures on the semantic web: theory and technology*, 1(1):1–136, 2011.

[67] W. Hu, Y. Qu, and G. Cheng. Matching large ontologies: A divide-and-conquer approach. *Data & Knowledge Engineering*, 67(1):140–160, 2008.

[68] D. S. Johnson and A. Klug. Testing containment of conjunctive queries under functional and inclusion dependencies. In *Proceedings of the 1st ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, PODS '82, pages 164–169, New York, NY, USA, 1982. ACM.

[69] A. Kementsietsidis, M. Arenas, and R. J. Miller. Mapping data in peer-to-peer systems: Semantics and algorithmic issues. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, SIGMOD '03, pages 325–336, New York, NY, USA, 2003. ACM.

[70] S. Kikot, R. Kontchakov, and M. Zakharyaschev. Conjunctive Query Answering with OWL 2 QL. In *KR*, 2012.

[71] G. Kokkinidis and V. Christophides. Semantic query routing and processing in P2P database systems: The ICS-FORTH SQPeer middleware. In *Proc. of EDBT*, pages 486–495, 2004.

[72] R. Kontchakov and M. Zakharyaschev. An introduction to description logics and query rewriting. *Reasoning Web. Reasoning on the Web in the Big Data Era*, pages 195–244, 2014.

[73] W. Le, S. Duan, A. Kementsietsidis, F. Li, and M. Wang. Rewriting queries on SPARQL views. In *Proc. of WWW*, pages 655–664, 2011.

[74] M. Lenzerini. Data integration: a theoretical perspective. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '02, pages 233–246, New York, NY, USA, 2002. ACM.

[75] A. Y. Levy, A. O. Mendelzon, Y. Sagiv, and D. Srivastava. Answering queries using views. In *PODS*, volume 95, pages 95–104. Citeseer, 1995.

[76] J. W. Lloyd and J. C. Shepherdson. Partial evaluation in logic programming. *J. of Logic Programming*, 11(3&4):217–242, 1991.

[77] F. L. R. Lopes, E. R. Sacramento, and B. F. Lóscio. Using heterogeneous mappings for rewriting SPARQL queries. In *DEXA Workshops*, pages 267–271, 2012.

[78] D. Maier, A. O. Mendelzon, and Y. Sagiv. Testing implications of data dependencies. *ACM Trans. Database Syst.*, 4(4):455–469, Dec. 1979.

[79] D. Mailharro. A classification and constraint-based framework for configuration. *AI EDAM*, 12(4):383–397, 1998.

[80] K. Makris, N. Bikakis, N. Gioldasis, and S. Christodoulakis. SPARQL-RW: transparent query access over mapped RDF data sources. In *Proc. of EDBT*, pages 610–613, 2012.

[81] K. Makris, N. Gioldasis, N. Bikakis, and S. Christodoulakis. Ontology mapping and SPARQL rewriting for querying federated RDF data sources. *OTM*, pages 1108–1117, 2010.

[82] P. McBrien and A. Poulovassilis. Defining peer-to-peer data integration using both as view rules. In *International Workshop on Databases, Information Systems, and Peer-to-Peer Computing*, pages 91–107. Springer, 2003.

[83] P. Mcbrien and A. Poulovassilis. P2p query reformulation over both-as-view data transformation rules. In *Proceedings of the 2005/2006 International Conference on Databases, Information Systems, and Peer-to-peer Computing*, DBISP2P'05/06, pages 310–322, Berlin, Heidelberg, 2007. Springer-Verlag.

[84] G. Montoya, L.-D. Ibáñez, H. Skaf-Molli, P. Molli, and M.-E. Vidal. SemLAV: local-as-view mediation for SPARQL queries. *TLDKS Journal XIII*, pages 33–58, 2014.

[85] M. Mosurovic, N. Krdzavac, H. Graves, and M. Zakharyaschev. A decidable extension of SROIQ with complex role chains and unions. *J. Artif. Intell. Res.(JAIR)*, 47:809–851, 2013.

[86] W. Nejdl. Design issues and challenges for RDF- and schema-based peer-to-peer systems. In *Proc. of DBISP2P*, 2003.

[87] W. Nejdl, B. Wolf, C. Qu, S. Decker, M. S. A. Naeve, M. Nilsson, M. Palmer, and T. Risch. Edutella: A P2P networking infrastructure based on RDF. In *Proc. of WWW*, 2002.

[88] W. Nejdl, M. Wolpers, W. Siberski, C. Schmitz, M. Schlosser, I. Brunkhorst, and A. Löser. Super-peer-based routing strategies for RDF-based peer-to-peer networks. *Web Semantics: Science, Services and Agents on the World Wide Web*, 1(2):177–186, 2004.

[89] J. Pérez, M. Arenas, and C. Gutierrez. Semantics and complexity of sparql. *ACM Trans. Database Syst.*, 34(3):16:1–16:45, Sept. 2009.

[90] H. Pérez-Urbina, I. Horrocks, and B. Motik. Efficient query answering for OWL 2. *The Semantic Web-ISWC 2009*, pages 489–504, 2009.

[91] H. Pérez-Urbina, B. Motik, and I. Horrocks. Rewriting conjunctive queries over description logic knowledge bases. In *Semantics in Data and Knowledge Bases*, pages 199–214. Springer, 2008.

[92] A. Poulovassilis, M. Al-Tawil, R. Frosini, M. Dimartino, and V. Dimitrova. Combining flexible queries and knowledge anchors to facilitate the exploration of knowledge graphs. In *Proceedings of the 5th International Workshop on Intelligent Exploration of Semantic Data (IESD 2016) co-located with the 15th International Semantic Web Conference (ISWC 2016)*. Leeds, 2016.

[93] A. Poulovassilis, P. Selmer, and P. T. Wood. Flexible querying of lifelong learner metadata. *IEEE Transactions on Learning Technologies*, 5(2):117–129, 2011.

[94] R. Rosati. On conjunctive query answering in EL. In *20th International Workshop on Description Logics DL'07*, 2007.

[95] A. Roth and F. Naumann. Benefit and cost of query answering in PDMS. In *Proceedings of the 2005/2006 International Conference on Databases, Information Systems, and Peer-to-peer Computing*, DBISP2P'05/06, pages 50–61, Berlin, Heidelberg, 2007. Springer-Verlag.

[96] A. Roth and S. Skritek. Peer data management. In *Dagstuhl Follow-Ups*, volume 5. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2013.

[97] G. Schenner, S. Bischof, A. Polleres, and S. Steyskal. Integrating distributed configurations with RDFS and SPARQL. In *16th International Configuration Workshop*, 2014.

[98] K. Schlegel, F. Stegmaier, S. Bayerl, M. Granitzer, and H. Kosch. Balloon fusion: SPARQL rewriting based on unified co-reference information. In *Proc. of ICDEW Workshops*, 2014.

[99] M. Schmachtenberg, C. Bizer, and H. Paulheim. Adoption of the Linked Data best practices in different topical domains. In *The Semantic Web–ISWC 2014*, pages 245–260. Springer, 2014.

[100] M.-A. Sicilia, S. Sánchez-Alonso, and E. García-Barriocanal. Sharing linked open data over peer-to-peer distributed file systems: The case of ipfs. In *Metadata and Semantics Research: 10th International Conference, MTSR 2016, Göttingen, Germany, November 22-25, 2016, Proceedings*, pages 3–14. Springer, 2016.

[101] I. Tatarinov and A. Halevy. Efficient query reformulation in peer data management systems. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, SIGMOD '04, pages 539–550, New York, NY, USA, 2004. ACM.

[102] I. Tatarinov, Z. Ives, J. Madhavan, A. Halevy, D. Suciu, N. Dalvi, X. L. Dong, Y. Kadiyska, G. Miklau, and P. Mork. The piazza peer data management project. *ACM Sigmod Record*, 32(3):47–52, 2003.

[103] H. J. Ter Horst. Completeness, decidability and complexity of entailment for rdf schema and a semantic extension involving the owl vocabulary. *Web Semantics: Science, Services and Agents on the World Wide Web*, 3(2-3):79–115, 2005.

[104] É. Thiéblin, F. Amarger, O. Haemmerlé, N. Hernandez, and C. Trojahn. Rewriting SELECT SPARQL queries from 1: n complex correspondences. *Ontology Matching*, 2016.

[105] D. Tomaszuk. RDF event stream processing based on the publish-subscribe pattern. In *Multimedia and Network Information Systems*, pages 369–378. Springer, 2017.

[106] A. I. Torre-Bastida, J. Bermúdez, A. Illarramendi, E. Mena, and M. González. Query rewriting for an incremental search in heterogeneous Linked Data sources. *Flexible Query Answering Systems*, pages 13–24, 2013.

[107] S. Wu, Q. H. Vu, J. Li, and K.-L. Tan. Adaptive multi-join query processing in PDBMS. In *Proceedings of the 2009 IEEE International Conference on Data Engineering*, ICDE '09, pages 1239–1242, Washington, DC, USA, 2009. IEEE Computer Society.

Appendices

Appendix A

RPS mappings for the case study in Chapter 3

A.1 Graph mapping assertions

A.1.1 Mapping of classes

$$x \leftarrow (x, \texttt{rdf:type}, \texttt{CVItem}) \rightsquigarrow x' \leftarrow (x', \texttt{rdf:type}, \texttt{Episode})$$
$$x \leftarrow (x, \texttt{rdf:type}, \texttt{Episode}) \rightsquigarrow x' \leftarrow (x', \texttt{rdf:type}, \texttt{CVItem})$$
$$x \leftarrow (x, \texttt{rdf:type}, \texttt{EducationItem}) \rightsquigarrow x' \leftarrow (x', \texttt{rdf:type}, \texttt{Educational\_Episode})$$
$$x \leftarrow (x, \texttt{rdf:type}, \texttt{Educational\_Episode}) \rightsquigarrow x' \leftarrow (x', \texttt{rdf:type}, \texttt{EducationItem})$$
$$x \leftarrow (x, \texttt{rdf:type}, \texttt{Degree}) \rightsquigarrow x' \leftarrow (x', \texttt{rdf:type}, \texttt{University\_Episode})$$
$$x \leftarrow (x, \texttt{rdf:type}, \texttt{University\_Episode}) \rightsquigarrow x' \leftarrow (x', \texttt{rdf:type}, \texttt{Degree})$$
$$x \leftarrow (x, \texttt{rdf:type}, \texttt{ExperienceItem}) \rightsquigarrow x' \leftarrow (x', \texttt{rdf:type}, \texttt{Occupational\_Episode})$$
$$x \leftarrow (x, \texttt{rdf:type}, \texttt{Occupational\_Episode}) \rightsquigarrow x' \leftarrow (x', \texttt{rdf:type}, \texttt{ExperienceItem})$$
$$x \leftarrow (x, \texttt{rdf:type}, \texttt{Member}) \rightsquigarrow x' \leftarrow (x', \texttt{rdf:type}, \texttt{Learner})$$
$$x \leftarrow (x, \texttt{rdf:type}, \texttt{Learner}) \rightsquigarrow x' \leftarrow (x', \texttt{rdf:type}, \texttt{Member})$$
$$x \leftarrow (x, \texttt{rdf:type}, \texttt{FieldOfStudy}) \rightsquigarrow x' \leftarrow (x', \texttt{rdf:type}, \texttt{Subject})$$
$$x \leftarrow (x, \texttt{rdf:type}, \texttt{Subject}) \rightsquigarrow x' \leftarrow (x', \texttt{rdf:type}, \texttt{FieldOfStudy})$$
$$x \leftarrow (x, \texttt{rdf:type}, \texttt{Role}) \rightsquigarrow x' \leftarrow (x', \texttt{rdf:type}, \texttt{Occupation})$$
$$x \leftarrow (x, \texttt{rdf:type}, \texttt{Occupation}) \rightsquigarrow x' \leftarrow (x', \texttt{rdf:type}, \texttt{Role})$$

A.1.2 Mapping of properties

$$
\begin{aligned}
x, y \leftarrow (x, \texttt{hasCVItem}, y) &\rightsquigarrow x', y' \leftarrow (x', \texttt{learnerEpisode}, y') \\
x, y \leftarrow (x, \texttt{learnerEpisode}, y) &\rightsquigarrow x', y' \leftarrow (x', \texttt{hasCVItem}, y') \\
x, y \leftarrow (x, \texttt{withRole}, y) &\rightsquigarrow x', y' \leftarrow (x', \texttt{job}, y') \\
x, y \leftarrow (x, \texttt{job}, y) &\rightsquigarrow x', y' \leftarrow (x', \texttt{withRole}, y') \\
x, y \leftarrow (x, \texttt{atTime}, y) &\rightsquigarrow x', y' \leftarrow (x', \texttt{epStart}, y') \\
x, y \leftarrow (x, \texttt{atTime}, y) &\rightsquigarrow x', y' \leftarrow (x', \texttt{epEnd}, y') \\
x, y \leftarrow (x, \texttt{epStart}, y) &\rightsquigarrow x', y' \leftarrow (x', \texttt{atTime}, z') \text{ AND } (z', \texttt{hasBeginning}, y') \\
x, y \leftarrow (x, \texttt{atTime}, z) \text{ AND } (z, \texttt{hasBeginning}, y) &\rightsquigarrow x', y' \leftarrow (x', \texttt{epStart}, y')
\end{aligned}
$$