Analyse statique de requête pour le Web sémantique

Melisachew Wudage Chekol Chekol

UNIVERSITÉ DE GRENOBLE

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE GRENOBLE

Spécialité : **Informatique**

Arrêté ministérial : 7 août 2006

Présentée par

Melisachew Wudage CHEKOL

Thèse dirigée par **Jérôme Euzenat**
et codirigée par **Nabil Layaïda**

préparée au sein **Institut national de recherche en informatique et en automatique, laboratoire d'informatique de Grenoble**
et de **l'Ecole Doctorale Mathématiques, Sciences et Technologies de l'Information, Informatique**

Static Analysis of Semantic Web Queries

Thèse soutenue publiquement le **19 décembre 2012**,
devant le jury composé de :

Mme. Marie-Christine ROUSSET
Professeur Université de Grenoble, Présidente
M. Diego Calvanese
Professeur Free University Bozen-Bolzano, Rapporteur
M. Axel Polleres
Professeur Siemens AG, Rapporteur
Mme. Marie-Laure Mugnier
Professeur University of Montpellier II , Examinatrice
M. Jérôme Euzenat
Directeur de recherche INRIA, Directeur de thèse
M. Nabil Layaïda
Chargé de recherche INRIA, Co-Directeur de thèse
M., Pierre Genevès
Chargé de recherche CNRS, Invité

Static Analysis of Semantic Web Queries

Abstract: Query containment is defined as the problem of determining if the result of a query is included in the result of another query for any given dataset. It has major applications in query optimization and knowledge base verification. The main objective of this thesis is to provide sound and complete procedures to determine containment of SPARQL queries under expressive description logic axioms. Further, we implement these procedures to support theoretical results by experimentation.

To date, testing query containment has been performed using different techniques: containment mapping, canonical databases, automata theory techniques and through a reduction to the validity problem in logic. In this thesis, we use the later technique to test containment of SPARQL queries using an expressive logic called μ-calculus. In doing so, RDF graphs are encoded as transition systems which preserves its characteristics, and queries and schema axioms are encoded as μ-calculus formulae. Thereby, query containment can be reduced to the validity test in the logic.

This thesis identifies various fragments of SPARQL (and PSPARQL) and description logic schema languages for which containment is decidable. Additionally, it provides theoretically and experimentally proven procedures to check containment of those decidable fragments. Finally, this thesis proposes a benchmark for containment solvers. This benchmark is used to test and compare the current state-of-the-art containment solvers.

Keywords: Containment, static analysis, SPARQL, PSPARQL, entailment regimes, OWL, RDF

Analyse Statique de Requête pour le Web Sémantique

Résumé: L'inclusion de requête est un problème bien étudié durant plusieurs décennies de recherche. En règle générale, il est défini comme le problème de déterminer si le résultat d'une requête est inclus dans le résultat d'une autre requête pour tout ensemble de données. Elle a des applications importantes dans l'optimisation des requêtes et la vérification de bases de connaissances. L'objectif principal de cette thèse est de fournir des procédures correctes et complètes pour déterminer l'inclusion des requêtes SPARQL en vertu d'axiomes exprimés en logiques de description. De plus, nous mettons en œuvre ces procédures à l'appui des résultats théoriques par l'expérimentation.

À ce jour, l'inclusion de requête a été testée à l'aide de différentes techniques: homomorphisme de graphes, bases de données canoniques, les techniques de la théorie des automates et réduction au problème de la validité d'une logique. Dans cette thèse, nous utilisons la derniere technique pour tester l'inclusion des requêtes SPARQL utilisant une logique expressive appelée le μ-calcul. Pour ce faire, les graphes RDF sont codés comme des systèmes de transitions, et les requêtes et les axiomes du schéma sont codés comme des formules de μ-calcul. Ainsi, l'inclusion de requêtes peut être réduite au test de la validité d'une formule logique.

Dans cette thèse j'identifier les divers fragments de SPARQL (et PSPARQL) et les langages de description logique de schéma pour lequelle l'inculsion est décidable. En outre, afin de fournir théoriquement et expérimentalement des procédures éprouvées pour vérifier l'inclusion de ces fragments décidables. Enfin, cette thèse propose un point de repère pour les solveurs d'inclusion. Ce benchmark est utilisé pour tester et comparer l'état actuel des solveurs d'inclusion.

Mots clés: Inclusion, analyse statique, SPARQL, PSPARQL, entailment regimes, OWL, RDF

Acknowledgments

I thank Marie-Laure Mugnier and Marie-Christine Rousset for accepting to be part of my thesis committee.

I express my heartfelt gratitude to my reviewers, Diego Calvanese and Axel Polleres, who provided me encouraging and constructive feedback. They took the time to very precisely read and criticize my work and I am grateful for their thoughtful and detailed comments. They greatly helped me improve the quality of this thesis.

I thank my supervisors Jérôme Euzenat and Nabil Layaïda for directing my work. They gave me the opportunity to join EXMO and WAM teams at INRIA as their student. I also thank them for always having had confidence in me, and letting me freely choose my research directions and the way of investigating them. In the WAM team, I was fortunate to meet Pierre Genevès. From the beginning to the end, this thesis benefited from his guidance. I would like to thank Pierre for his invaluable availability and support.

Special thanks to the members of EXMO and WAM teams, in particular to Cássia Trojahn dos Santos, Manuel Atencia, Jérôme David, and Everardo Barcenas, thank you for providing me a very pleasant and friendly atmosphere at INRIA. To my friends Besira Mekonnen, Fasil Terefe, and Zerihun Zewdu, thanks for all the support you have given me.

On a more personal note, I would like to thank my family for their unconditional support. Thank you all.

Contents

Chapter 1

Introduction

Contents

1.1 Motivation and Objectives

The semantic web is an extension of the world wide web (WWW) aiming to create machine readable content on the web by giving well-defined syntax and semantics to web resources. Over a decade of research has resulted in the development of various semantic web frameworks: ontology development languages, query languages, rule languages, and others. In the semantic web, semantically rich datasets can be created using World Wide Web Consortium (W3C) recommended languages such as the Resource Description Framework (RDF) and the Web Ontology Language (OWL). RDF is a language used to express structured information on the web as graphs. An RDF document is a set of triples $(subject, predicate, object)$ that can be represented by a directed labeled graph (hence the name, RDF graph). It has a model theoretic semantics [Hayes 2004]. RDF has a lightweight schema language called RDFS (RDF Schema). RDFS allows one to express subclass and property hierarchies, with domain and range definitions of these properties. RDF and RDFS allow for the representation of some ontological knowledge. The main modelling primitives of RDF/RDFS concern the organization of vocabularies in typed hierarchies: subclass and subproperty relationships, domain and range restrictions, and instances of classes. However a number of other features (such as negation, cardinality restrictions, and boolean connectives) are missing, which led to the development of OWL. OWL, built on top of RDF and RDFS, defines a family of knowledge representation languages for creating ontologies on the semantic web. It brings the expressive and reasoning power of description logics to the semantic web. RDF and OWL have been developed through years of research and standardized. The same goes for the query languages: prominently SPARQL (SPARQL Protocol And RDF Query Language) which is a W3C recommended query language for RDF. Since its creation it has been a source of research from various directions, mostly in terms of

extending the language: adding paths (regular expression patterns), negation, querying modulo schema, subqueries and the likes. Querying RDF graphs with SPARQL amounts to matching graph patterns that are given as sets of triples of subjects, predicates and objects. These triples, also known as triple patterns, are usually connected to form graphs by means of joins expressed using several occurrences of the same variable. It allows variables to be bound to components in the input RDF graph. In addition, operators akin to relational joins, unions, left outer joins, selections, and projections can be combined to build more expressive queries. Queries are formed from graph patterns which in turn are defined inductively from triple patterns. SPARQL is more than a query language, it is also a protocol for sending a request and receiving the answers in different formats.

A recent effort from the semantic web community has seen the emergence of linked data (a.k.a. web of data). It is a vision aimed at dismantling data silos: semantically enriching web resources that will allow computers to automatically reason about the underlying data and enable them to make reliable and logically founded conclusions. With linked data large datasets containing billions of triples are now available across the web.

There are three important challenges regarding querying in the semantic web. Firstly, queries in the semantic web are evaluated over large datasets, optimizations are indispensable in order to find minimal queries to avoid the computational cost of query evaluation. Secondly, queries can be evaluated at remote endpoints, satisfiability test of queries is necessary to avoid high communication costs between the sender and receiver. Thirdly, if one query is known to be included in another, then materialized query evaluation can be used. To tackle these challenges, static analysis of queries is absolutely essential. Let us elaborate the second challenge, SPARQL is equipped with a protocol that allows it to send queries to endpoints of datasets and receive the answers at a sender's premises. Because a given SPARQL endpoint may be an interface to a triplestore or a relational data store [Levandoski & Mokbel 2009, Fernández *et al.* 2010] or Hadoop, the ability to query several endpoints with one query is a very useful feature. To take advantage of this valuable feature, federation has been recently added to SPARQL1.1[1]. The idea of federated queries is that parts of a query are sent to different endpoints on the web and the answers to those parts are collected and summarized or merged by the sender. This feature is highly important in linked data where datasets are spread across the web.

Erroneous queries often return the empty set for any input, i.e., they are unsatisfiable. Thus, an unsatisfiable query is a hint for an error in the query. Checking if a query is unsatisfiable and warning the user in the case of an unsatisfiable query therefore helps to debug a program containing the query, that overall leads to more stable programs. Static program analysis of embedded semantic web data and query languages can increase the detection of errors already at compile time, which is long before the application is really executed. A static program analysis can surely detect errors, which – without a static program analysis – may only be detected after running

[1] http://www.w3.org/TR/sparql11-federated-query/

a huge amount of test cases, as the static program analysis considers every branch in an application. Thus, these are major evidences why static analysis of SPARQL queries should be studied.

The fact that SPARQL is used to query large datasets (sometimes billion of triples) and that queries can be executed at remote endpoints (distributed query evaluation) opens new challenges such as optimization (is one able to find an equivalent query that runs faster), satisfiability (a query may have a solution or not), materialization (if the results of query q is contained in the results of q', then q can be evaluated in the materialized view of q'). Consequently, addressing these challenges is essential.

1.1.1 State of the Art

Containment, equivalence and satisfiability problems are well studied for relational database query languages. Their study started with a pioneering paper from Chandra and Merlin [Chandra & Merlin 1977], which studied optimization of conjunctive queries. They observed that minimal conjunctive queries are unique up to isomorphism, which also means that the unique minimal conjunctive query for a given conjunctive query q can be produced by the following simple procedure: start with q and repeatedly remove atoms that are redundant in the sense that dropping them preserves equivalence; the order in which atoms are dropped is irrelevant and the only non-trivial part is checking equivalence, implemented as two query containment checks [Bienvenu 2012]. In general, query containment is the problem of checking whether the result of one query is included in the result of another one for any given dataset. Also in [Chandra & Merlin 1977], it is proved that union of conjunctive query containment and equivalence problems are NP-complete. These problems, containment, equivalence, and satisfiability, are collectively called static analysis of queries.

Equivalence and satisfiability problems can be derived from containment, thus, we mainly concentrate on the containment problem. In relational databases, union of conjunctive query containment has been studied using containment mappings also known as graph homomorphism and canonical databases. It is known that, for (union of) conjunctive queries, query answering and containment are equivalent problems since query containment can be reduced to query answering [Chandra & Merlin 1977]. Unfortunately, to apply these techniques to a query language equipped with ontologies and regular expressions is not entirely possible. That is why for semistructured data query languages (referred as regular path queries) automata theoretic notions are often employed to address containment and other problems [Calvanese *et al.* 2000]. In addition to using automata, containment has been addressed by a reduction to satisfiability test. In this direction, queries are translated into formulas in a particular logic that supports the query languages features and then the overall problem is reduced into satisfiability test. Several works exist that developed and used this technique [Calvanese *et al.* 2000, Genevès *et al.* 2007, Calvanese *et al.* 2008] which also inspired this thesis. Specifically, a study in [Genevès *et al.* 2007] has developed a tree-logic from the alternation-free fragment of the μ-calculus and applied it to encode XPath queries and perform static analysis tasks. Their attempt has been successful and put to practice.

Their implementation allows one to perform containment, satisfiability and equivalence of XPath queries. The study has been extended to several other languages namely XQuery, CSS and JSON.

All the aforementioned results were obtained under the assumption that queries are evaluated under set semantics. This means that the database relations given as inputs to queries are sets (i.e., no duplicate tuples are allowed) and that queries return sets as answers. In real database systems, however, queries are usually evaluated under bag semantics, not set semantics: input database relations may be bags (multisets), and queries may return bags as answers. The same holds for SPARQL. In particular, SPARQL queries are evaluated under bag semantics, since duplicate tuples are not eliminated unless explicitly specified in the syntax using the SELECT DISTINCT construct. The reason for not eliminating duplicate tuples in SPARQL is that the values of aggregate operators, such as AVG and COUNT, depend on the multiplicities of the tuples in the graph. Most of the studies on query containment use set semantics rather than bag semantics. This is due to very high complexity: for instance, containment becomes undecidable (even for) union of conjunctive queries under bag semantics [Ioannidis & Ramakrishnan 1995]. Thus, this thesis relies on the set semantics of SPARQL to obtain decidability results.

The importance of the containment problem goes beyond the field of databases. It has attracted a lot of attention from the description logic community. In this regard, many of the works concentrated on the problem of conjunctive query answering as containment follows from it. All these works have sound theoretical and mathematical foundations, but they fail to provide an implementation (or experimentation results) of their approaches.

Deviating from conjunctive queries in the database and description logic worlds, we have regular path queries (RPQs): query languages used to query arbitrary length paths in graph databases of semistructured data. Like conjunctive queries, they have been used and studied widely. They are different from conjunctive queries in that, they allow recursion by using regular expression patterns. The problem of containment has been addressed for several extensions of these languages: CRPQs, P2RPQs, and ECRPQs [Florescu *et al.* 1998, Calvanese *et al.* 2000, Barceló *et al.* 2010] . One prominent language used in querying semi-structured data is XPath. This language has been studied extensively over 2 decades. One study which attracted our attention is the one from [Genevès *et al.* 2007], where the static analysis of XPath queries has been investigated using a graph logic and providing an implementation which has been put to practice.

A comparison of SPARQL and relational algebra has already been made, to figure out important similarities and use the results of studies for relational algebra. Consequently, the results from [Polleres 2007] and [Angles & Gutierrez 2008] taken together imply that SPARQL has exactly the same expressive power as relational algebra. From early results on query containment in relational algebra and first-order logic, one can infer that containment in relational algebra is undecidable [Abiteboul *et al.* 1995]. Therefore, containment of SPARQL queries is also undecidable [Letelier *et al.* 2012]. Hence, in this thesis, we study the containment problem for various

fragments of SPARQL to retain decidability.

Given the subgraph matching nature of SPARQL and its evaluation over graphs, it can be encoded in a logic where the interpretation of that logic is over a graph. One logic that has these characteristics is the μ-calculus: it is an extension of modal logic which is highly expressive and has well-behaved computational complexity. It provides fixpoint operators which allows to perform navigation in a graph in an intricate way. Furthermore, μ-calculus is suited for system verification, reachability problems, reasoning on graphs, and game problems. For its complexity, it is expressive enough to encompass PDL, LTL, CTL and CTL* [Blackburn *et al.* 2007]. The satisfiability problem in the μ-calculus is decidable in exponential amount of time. It has attractive model-theoretic properties namely finite and tree model properties. Beyond all, the availability of satisfiability solvers for the logic, makes it a very good choice for this study.

1.2 Contributions

Having introduced the global context and the basic ideas behind containment and equivalence of semantic web queries, we now summarize the work that will be presented in this thesis and highlight the major contributions. Our focus will be on the static analysis of SPARQL and PSPARQL queries. Specifically, for this study, we consider the union of conjunctive SPARQL (resp. PSPARQL) queries. The contributions of this thesis are sixfold:

- We provide an encoding of RDF graphs as transition systems that can be used for determining the containment of SPARQL queries.

- We propose a technique to determine the containment of union of conjunctive SPARQL queries under $\mathcal{ALCH}$ schema axioms. To do so, we encode queries and schema axioms as μ-calculus formulas, thereby reducing containment into unsatisfiability test. We prove the soundness and completeness of our approach. We show that the containment problem can be determined in a double exponential amount of time. Beyond this, we provide an implementation and experimentation for the proposed approach.

- We address the study of the containment of PSPARQL queries. For that purpose, we encode PSPARQL queries as μ-calculus formulas and then reduce containment test into the validity problem in the μ-calculus. The complexity of determining containment of path SPARQL queries (also under $\mathcal{ALCH}$ axioms) is double exponential. This complexity is an upper bound for the problem.

- We provide three different approaches to check containment of SPARQL queries under, a meaning and substantial fragment of, the RDFS entailment regime (that we call simpleRDFS entailment regime). We prove that this problem can be solved in an exponential amount time. Moreover, we support our theoretical results with an implementation. We also investigate the containment problem

under, a fragment of, the OWL direct semantics entailment regime[2], specifically under $\mathcal{ALCH}$ entailment regime. To do so, we identify decidable fragments of the ontology and query languages.

- We experimentally show that under currently available RDF data repositories and SPARQL endpoints, 87% of real world SPARQL queries are tree-structured. We propose an encoding procedure to determine containment for the mentioned type of queries using a reduction to the modal logic K_n, and compare experimentally the performance of containment solvers using the proposed method.

- We present a first benchmark for statically analysing semantic web queries. We design test suites (with and without ontology axioms) that test and compare the performance and correctness of containment solvers. In addition, the benchmark is tested on the currently available tools.

1.3 Summary of Publications

In the following we summarize the publications that are related to this thesis.

All of the author's publications focus on the area of statically analysing semantic web queries. The paper [Chekol *et al.* 2011a] studies the containment problem for PSPARQL (Path SPARQL) queries. All the results of this paper and additionally others (containment of PSPARQL under constraints, two-way path queries) are discussed in Chapter 4 of this thesis. An extended version of this paper is also published as a research report in [Chekol *et al.* 2011b].

The results of Chapter 3 on the containment and equivalence of SPARQL queries under highly expressive description logic axioms are published in
[Chekol *et al.* 2012b] where a double exponential upper bound is proved for the problem. The optimality of this complexity bound is investigated in this thesis. We show that this complexity bound is complete for the description logic $\mathcal{SHI}$.

Further, W3C is working towards extending SPARQL as a query language for ontologies (beyond subgraph matching) also known as SPARQL entailment regimes. To this end, it is natural to study the satisfiability, containment and equivalence problems of SPARQL under entailment regimes (RDF and OWL). In Chapter 5 we have addressed these issues in detail and some of the results have been published in [Chekol *et al.* 2012a].

1.4 Thesis Outline

In Chapter 2, we present the preliminaries that are used in the development of this thesis. We present the foundations of the semantic web, followed by a brief overview of modal logics, and finally we conclude this chapter with a broad survey of related works.

[2] http://www.w3.org/TR/sparql11-entailment/

Chapter 3 unveils a procedure to translate RDF graphs into transition systems. In doing so, RDF graphs become bipartite graphs with two sets of nodes: triple nodes and subject, predicate, and object nodes where navigation can be done using transition programs. The next task requires encoding axioms and queries as μ-calculus formulas and then, consequently, reducing query containment test into validity test in the logic. We then prove that this reduction is sound and complete.

In Chapter 4, we extend the procedures of Chapter 3 to study the containment problem for PSPARQL queries. We divide this task into three: (i) we investigate the problem for PSPARQL queries (equivalent to conjunctive RPQs), (ii) next we add inverse to paths of PSPARQL (equivalent to two-way conjunctive RPQs) and study containment for this fragment, finally (iii) we address the problem under description logic schema axioms.

Three approaches to determine containment of SPARQL queries under RDFS entailment regime are discussed in Chapter 5 borrowing some of the encoding procedures from Chapter 3 and 4. Furthermore, we show how these approaches can be extended for the OWL Direct Semantics entailment regime[3].

For tree-structured queries, that constitute a large class of real world SPARQL queries, we consider a less expressive logic than μ-calculus to study static analysis as discussed in Chapter 6. We provide experimental results for testing containment with a number of benchmark queries.

In Chapter 7, we present the first-of-its-kind containment benchmark for SPARQL queries. We propose several test suites that can be used to test containment solvers. Accordingly, we have carried out experiments to test and compare current state-of-the-art containment solvers. In Chapter 8, we briefly present the study of containment for negated SPARQL queries i.e., queries constructed from MINUS graph patterns, before concluding a summary of the results of this thesis and perspectives in Chapter 9.

In summary, in Figure 1.1, we show briefly what has been discussed in each chapter. Our approach relies on encoding graphs, queries and schemas into a logic as shown below:

RDF graphs G	(P)SPARQL queries q	Schema axioms $\mathcal{S}$
$\downarrow \sigma$	$\downarrow \mathcal{A}$	$\downarrow \eta$
Transition systems $\sigma(G)$	μ-calculus formulae $\mathcal{A}(q)$	$\eta(\mathcal{S})$

We reduce *query containment* to the problem of *unsatisfiability* in μ-calculus :

$$q \sqsubseteq_{\mathcal{S}} q'$$
$$\downarrow$$
$$\models \eta(\mathcal{S}) \wedge \mathcal{A}(q) \wedge \neg\mathcal{A}(q', m)$$

[3]http://www.w3.org/TR/sparql11-entailment/

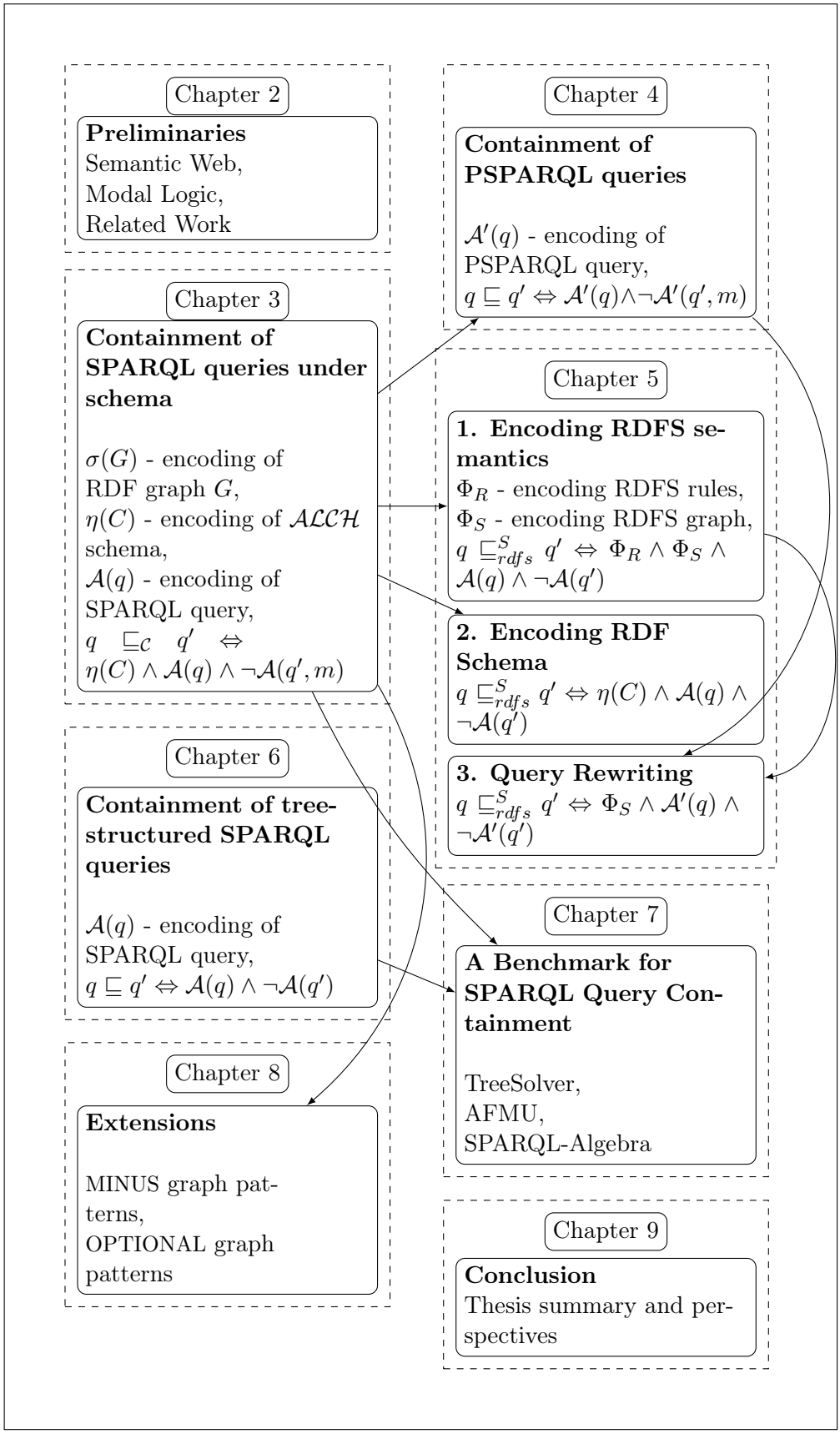


Figure 1.1: Summary of thesis.

Chapter 2

Preliminaries

Contents

2.1 Semantic Web

The semantic web has been around for more than a decade now. In an article published in May 2001, Tim Berners-Lee describes it as, creating machine-readable content on the web [Berners-Lee & Hendler 2001]. It is not a separate Web but an extension of the current one, in which information is given "well-defined meaning, better enabling computers and people to work in cooperation". The semantic web has grown reasonably with a well established and "dynamic" architecture. It has languages that enable to create machine understandable data and to query it. More recently, linked data (a.k.a web of data) has emerged and has attracted a strong attention outside the research world since. It is a vision that started with the aim of dismantling data silos [Heath & Bizer 2011]. The ambiguities that surfaced over the difference and/or similarity of the semantic web and linked data have been clarified in [Bizer *et al.* 2009]. There, it is mentioned that the semantic web, or web of data, is the goal or the end result of creating meaningful content with respect to machines. "Over time, with linked data as a foundation, some of the more sophisticated proposals associated with the semantic web vision, such as intelligent agents, may become a reality".

The vision was that the semantic annotation of web resources would allow computers to automatically reason about the underlying data and enable them to make reliable and logically founded conclusions. At a global scale, the semantic web thus can

be understood as a large knowledge base that links information from different sources together, eases the access to information, and ultimately improves search and knowledge discovery in the Internet. In an effort to bring the vision of the semantic web to a reality, necessary measures are being taken. A strong force behind this are the W3C working groups, highly involved with standardization and creating a means for researchers to communicate their innovations and to share ideas.

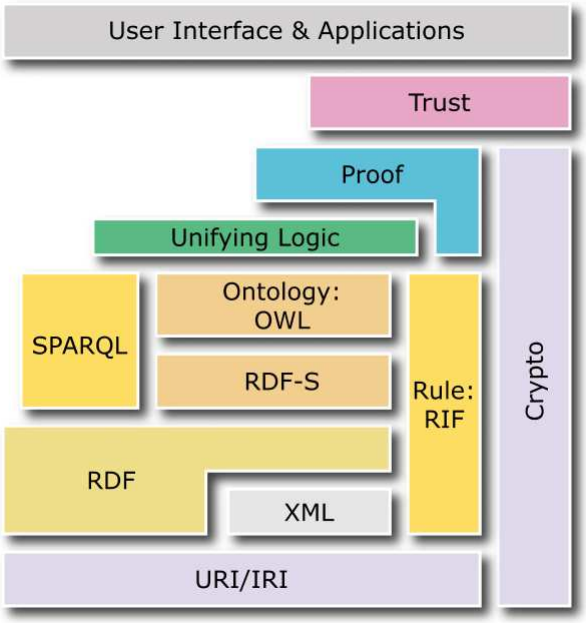


Figure 2.1: Semantic web layer cake [Gerber *et al.* 2008].

The diagram in Figure 2.1, is known as the semantic web layer cake (a.k.a semantic web architecture), shows the effort being undertaken by W3C in order to bring the semantic web to reality. At the bottom of this architecture is, the encoding mechanism or standard used for the data. As per the vision of Berners-Lee, this layer is about representing every item on the web using IRIs (Internationalized Resource Identifiers)–which provide ways to construct names for globally identifying physical or logical resources. The second layer is engaged in the syntactic representation of the data using XML and RDF. A core data representation format for semantic web is the Resource Description Framework (RDF). RDF is a framework for representing information about resources in terms of graphs. It was primarily intended for representing metadata about WWW resources, such as the title, author, and modification date of a web page, but it can be used for storing any other data. It is based on triples subject-predicate-object that form a graph. RDF is the primary representation language with a normative XML serialization syntax. Formal semantics of RDF is defined as well. A more detailed presentation is provided in Section 2.1.1.

In the third layer, a Simple Protocol and RDF Query Language (SPARQL) is available for querying RDF data as well as RDFS and OWL ontologies with knowledge bases. SPARQL is an SQL-like language, but uses RDF triples and resources for both

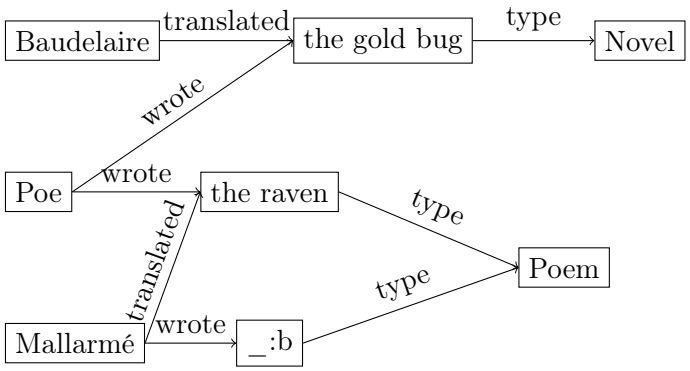


Figure 2.2: Example RDF graph G about writers and their works.

matching part of the query and for returning results of the query. Since both RDFS and OWL are built on RDF, SPARQL can be used for querying ontologies and knowledge bases directly as well. SPARQL is not only a query language, it is also a protocol for accessing RDF data.

In this thesis, we concentrate mainly on the third layer, a detailed discussion on the rest of the layers of the semantic web layer cake, Figure 2.1, can be found in [Gerber *et al.* 2008].

2.1.1 RDF

RDF is a language used to express structured information on the Web as graphs. We present a compact formalization of RDF [Hayes 2004]. Let U, B, and L be three disjoint infinite sets denoting the set of URIs (identifying a resource), blank nodes (denoting an unidentified resource) and literals (a character string or some other type of data) respectively. We abbreviate any union of these sets as for instance, UBL = U∪B∪L. We refer to U, B, and L as *terms* of the graph. A triple of the form $(s, p, o) \in \text{UB} \times U \times \text{UBL}$ is called an *RDF triple*. s is the *subject*, p is the *predicate*, and o is the *object* of the triple. Each triple can be thought of as an edge between the subject and the object labelled by the predicate, hence a set of RDF triples is often referred to as an *RDF graph*.

Example 1 (RDF Graph)**.** *Here are 8 triples of an RDF graph about writers and their works: (all identifiers correspond to URIs, _:b is a blank node):*

{ (Poe,wrote,thegoldbug), (Baudelaire,translated,thegoldbug),
(Poe,wrote,theraven), (Mallarmé,translated,theraven),
(theraven,type,Poem), (Mallarmé,wrote,_:b),
(_:b,type,Poem), (thegoldbug,type,Novel) }

These triples can also be represented graphically as shown in Figure 2.2.

RDF has a model theoretic semantics [Hayes 2004], that defines the notion of consequence between two RDF graphs, i.e., does an RDF graph G entails an RDF graph

H (known as RDF entailment). This semantics builds on the notion of interpretations. Informally, the interpretation of an RDF graph contains all the triples that are logically implied according to a set of rules.

2.1.1.1 RDF Schema

RDF Schema (RDFS) [Hayes 2004] may be considered as a simple ontology language expressing subsumption relations between classes or properties. Technically, this is an RDF vocabulary used for expressing axioms constraining the interpretation of graphs. Hence, schemas are themselves RDF graphs. The RDFS vocabulary and its semantics are given in [Hayes 2004]. The W3C specifications introduce two standard namespaces: the *RDF namespace* `http://www.w3.org/1999/02/22-rdf-syntax-ns#` (prefix *rdf*) and the *RDF Schema namespace* `http://www.w3.org/2000/01/rdf-schema#` (prefix *rdfs*). These namespaces comprise a set of URIs with predefined meaning. Below, we present some of the predefined vocabulary terms, the notation in the parenthesis is the vocabulary syntax used in this thesis:

- The predefined URI *rdf:type (type)* can be used for typing entities.
- *rdfs:subClassOf (sc)* and *rdfs:subPropertyOf (sp)* are used to describe subclass and subproperty relationships between classes and properties, respectively.
- The two classes *rdfs:Class (Class)* and *rdf:Property (Property)* can be used to assign a logical type to URIs.
- The URIs *rdfs:domain (dom)* and *rdfs:range (range)* can be used to specify the domain and range of properties.
- All things described by RDF are called *resources*, and are instances of the class *rdfs:Resource (Resource)*.

In [Hayes 2004], rules are given which allow to deduce or infer new triples using RDF Schema triples. For our purposes, we consider the RDFS inference or deduction rules of [Gutierrez *et al.* 2004, Muñoz *et al.* 2007] shown below.

- Subclass (sc)

$$\frac{(a, sc, b)\ (b, sc, c)}{(a, sc, c)} \qquad \frac{(a, sc, b)\ (x, type, a)}{(x, type, b)} \tag{2.1}$$

- Subproperty (sp)

$$\frac{(a, sp, b)\ (b, sp, c)}{(a, sp, c)} \qquad \frac{(a, sp, b)\ (x, a, y)}{(x, b, y)} \tag{2.2}$$

- Typing (dom, range)

$$\frac{(a, dom, b)\ (x, a, y)}{(x, type, b)} \qquad \frac{(a, range, b)\ (x, a, y)}{(y, type, b)} \tag{2.3}$$

- Implicit Typing

$$\frac{(a, dom, b)\ (c, sp, a)\ (x, c, y)}{(x, type, b)} \qquad \frac{(a, range, b)\ (c, sp, a)\ (x, c, y)}{(y, type, b)} \tag{2.4}$$

- Subclass reflexivity

$$\frac{(a, type, Class)}{(a, sc, a)} \qquad \frac{(a, sc, b)}{(a, sc, a)\ (b, sc, b)} \tag{2.5}$$

- Subproperty reflexivity

$$\frac{(a, type, Property)}{(a, sp, a)} \qquad \frac{(x, a, y)}{(a, sp, a)} \tag{2.6}$$

- Resource

$$\frac{(a, b, c)}{(a, type, Resource)} \qquad \frac{(a, b, c)}{(c, type, Resource)} \qquad \frac{(a, type, Class)}{(a, sc, Resource)} \tag{2.7}$$

- Property

$$\frac{(a, b, c)}{(b, type, Property)} \tag{2.8}$$

- Class

$$\frac{(a, b, c)}{(a, type, Class)} \qquad \frac{(a, type, c)}{(c, type, Class)} \tag{2.9}$$

In [Ter Horst 2005], it is shown that the standard set of entailment rules for RDFS, is incomplete and that this can be corrected by allowing blank nodes in predicate position. The rules shown above, taken from [Muñoz *et al.* 2007], fix this problem. For our purposes, we consider only a subset of these rules (see Chapter 5).

Example 2. *This example shows the usage of RDFS inference rules, consider the graph:*

$$\begin{aligned} G = \{&(john, child, mary), (child, sp, ancestor), \\ &(ancestor, dom, Person), (ancestor, range, Person)\} \end{aligned}$$

By applying either both typing (2.3) and subproperty (2.2) rules or implicit typing rule (2.5), it can be inferred that {(john,type,Person), (mary,type,Person), (john,ancestor,mary)}.

Hence, the deductive closure of the graph G, denoted as $cl(G)$, is:

$$\begin{aligned} cl(G) = \{&(john, child, mary), (child, sp, ancestor), (ancestor, dom, Person) \\ &(john, type, Person), (mary, type, Person), \\ &(john, ancestor, mary)\} \end{aligned}$$

Note that additional triples that can be derived by reflexivity and other rules are not displayed in $cl(G)$, it contains only a part of the closure graph.

2.1.1.2 Entailment

Here we present simple and RDFS entailment in RDF graphs, a more detailed discussion can be found in [Hayes 2004, Ter Horst 2005].

Simple RDF Entailment: simple entailment depends only on the basic logical form of RDF graphs and therefore holds for any vocabulary. Given two RDF graphs G_1 and G_2, a *map* from G_1 to G_2 is a function h from terms of G_1 to terms of G_2, preserving URIs and literals, such that for each triple $(a, b, c) \in G_1$ we have $(h(a), h(b), h(c)) \in G_2$. An RDF graph G_1 *simply entails* G_2, denoted $G_1 \models_s G_2$, if and only if there exists a map from G_2 to G_1.

RDFS entailment: RDFS entailment captures the semantics added by the RDFS vocabulary. We write that $G_1 \models_{rule} G_2$ if G_2 can be derived from G_1 by iteratively applying rules in groups (*subclass*), (*subproperty*) and (*typing*). The *closure* of a graph G, denoted as $cl(G)$, is the graph obtained from it by iteratively applying the RDFS inference rules. We have that $G_1 \models_{rule} G_2$ if and only if $G_2 \in cl(G1)$. It turns out that G_1 RDFS-entails G_2, written $G_1 \models_{rdfs} G_2$, iff there is a graph G derived from G_1 by exhaustively applying the RDFS rules such that $G_1 \models_{rule} G$ and $G \models_s G_2$ [Ter Horst 2005]. Alternatively, $G_1 \models_{rdfs} G_2$ iff $cl(G_1, G_2) \models_s G_2$ where $cl(G_1, G_2)$ is the union of the closure of G_1 and G_2 obtained by exhaustively applying the RDFS inference rules.

2.1.2 Description Logics

Description Logics (DLs) are a family of knowledge representation (KR) formalisms that represent the knowledge of an application domain by first defining the relevant concepts of the domain (its terminology), and then using these concepts to specify properties of objects and individuals occurring in the domain (the world description) [Baader & Nutt 2003, Baader *et al.* 2007]. Alternatively, DLs are fragments of first-order logic that model a domain of interest in terms of concepts and roles denoting unary and binary predicates, respectivley [Baader *et al.* 2007]. DLs are equipped with a feature that allow for reasoning in a knowledge base. Reasoning enables one to infer implicitly represented knowledge from the knowledge that is explicitly contained in the knowledge base. A knowledge base (KB) comprises two components, the TBox and

the ABox. The TBox introduces the terminology, i.e., the vocabulary for classes and properties in an application domain, while the ABox contains assertions about named individuals in terms of this vocabulary.

There are various types of description logics with differing expressivity, DL-Lite and its extensions, $\mathcal{SROIQ}(\mathbf{D})$ and its fragments, and $\mathcal{ALC}$ and its extensions [Baader *et al.* 2007, Horrocks *et al.* 2006]. In this thesis, for our purposes, we use the well-studied DL $\mathcal{SROIQ}(\mathbf{D})$ [Horrocks *et al.* 2006] that underlies the foundations of the web ontology language (OWL 2). We consider various fragments of this logic, mainly the $\mathcal{ALCH}$ fragment. $\mathcal{SROIQ}(\mathbf{D})$ KB satisfiability is 2NExpTime-complete [Horrocks *et al.* 2006, Kazakov 2008] with respect to combined complexity.

2.1.2.1 $\mathcal{SROIQ}(\mathbf{D})$

We consider here the following constructs occurring in expressive description logics: role hierarchy $\mathcal{H}$, role *transitivity* $\mathcal{S}$, role *composition* $\mathcal{R}$, *nominals* $\mathcal{O}$, role *inverse* $\mathcal{I}$, *qualified number restrictions* $\mathcal{Q}$, and *datatypes* $\mathbf{D}$. Recently, OWL 2 has become a W3C recommended ontology language. The logic underlying this ontology language is $\mathcal{SROIQ}(\mathbf{D})$. The syntax and semantics of this logic is presented in Table 2.1 and 2.2. Notice that, even though we detail here its constructs, in this thesis we are interested in its fragments, mainly in $\mathcal{ALCH}$.

Syntax In $\mathcal{SROIQ}(\mathbf{D})$ concepts and roles are formed according to the syntax presented in Table 2.1, where R denotes an atomic role or its inverse, A represents an atomic concept, C denotes a complex concept, o refers to a nominal, and n is a non-negative integer. Additionally, the following abbreviations are used:

$$\begin{aligned} C_1 \sqcup C_2 &= \neg(\neg C_1 \sqcap \neg C_2) \\ \top &= \neg(\bot) \\ \forall R.C &= \neg(\exists R.\neg C) \end{aligned}$$

$\mathcal{SROIQ}(\mathbf{D})$ Axioms: The TBox is a finite set of axioms consisting of *concept inclusions*, *role inclusion*, *role transitivity*, and *role chain* axioms:

$$\begin{aligned} C_1 \sqsubseteq C_2 \qquad & R \sqsubseteq S \\ R_1 \circ \cdots \circ R_n \sqsubseteq S & \end{aligned}$$

Two concepts C_1 and C_2 are said to be equivalent, denoted as $C_1 \equiv C_2$, if and only if $C_1 \sqsubseteq C_2$ and $C_2 \sqsubseteq C_1$. Likewise, $R_1 \equiv R_2$ iff $R_1 \sqsubseteq R_2$ and $R_2 \sqsubseteq R_1$.

Example 3. $\mathcal{SROIQ}(\mathbf{D})$ *TBox axioms modeling a university domain.*

$$
\begin{aligned}
\mathit{PostgradStudent} &\sqsubseteq \mathit{Student} \\
\mathit{UndergradStudent} &\sqsubseteq \mathit{Student} \\
\mathit{Department} &\sqsubseteq \mathit{Faculty} \\
\mathit{Faculty} &\sqsubseteq \mathit{University} \\
\mathit{Staff} \sqcup \mathit{Student} &\sqsubseteq \bot \\
\mathit{Professor} &\sqsubseteq \mathit{Person} \\
\mathit{Student} &\sqsubseteq \mathit{Person} \\
\mathit{Chair} &\equiv \mathit{Person} \sqcap \exists \mathit{headOf}.\mathit{Department} \\
\mathit{Student} &\equiv \mathit{Person} \sqcap \exists \mathit{takesCourse}.\mathit{Course} \\
\mathit{Professor} &\equiv \mathit{Person} \sqcap \exists \mathit{givesCourse}.\mathit{Course} \\
\exists \mathit{headOf}.\top &\sqsubseteq \mathit{Professor} \\
\mathit{takesCourse} &\equiv \mathit{givesCourse}^{-}
\end{aligned}
$$

Here, we provide a small textual explanation for some of the TBox axioms shown above. The first axiom states that every postgraduate student is a student, the sixth axiom defines that every professor is a person, the ninth axiom asserts that every student is a person and takes a certain course and vice versa.

Semantics An interpretation, $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, consists of a non-empty domain $\Delta^{\mathcal{I}}$ and an interpretation function $\cdot^{\mathcal{I}}$ that assigns to each object name o an element $o^{\mathcal{I}} \in \Delta^{\mathcal{I}}$, to each atomic concept A a subset $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ of the domain, and to each atomic role R a binary relation $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ over the domain. The role and concept constructs can be interpreted in $\mathcal{I}$ as depicted in Tables 2.1.

An interpretation $\mathcal{I}$ *satisfies* an inclusion $C \sqsubseteq D$ iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$, and it satisfies an equality $C \equiv D$ iff $C^{\mathcal{I}} = D^{\mathcal{I}}$. If $\mathcal{T}$ is a set of axioms, then $\mathcal{I}$ satisfies $\mathcal{T}$ iff $\mathcal{I}$ satisfies each element of $\mathcal{T}$. If $\mathcal{I}$ satisfies an axiom (resp. a set of axioms), then we say that it is a *model* of this axiom (resp. set of axioms). Two axioms or two sets of axioms are *equivalent* if they have the same models.

Description logic datatype syntax and semantics Datatypes restrict the interactions between concrete and "abstract" parts of a knowledge base so as to avoid problems of undecidability and to simplify implementation, and are widely used in ontology languages, including OWL and OWL 2. The syntax and semantics of datatypes is summarised in Table 2.2, where D is a datatype name, T is a concrete role, v is a data value and n is a non-negative integer. An interpretation, $\mathcal{I} = (\Delta_D^{\mathcal{I}}, \cdot^{\mathcal{I}})$, consists of a non-empty concrete domain $\Delta_D^{\mathcal{I}}$ and an interpretation function $\cdot^{\mathcal{I}}$.

Assertions about individuals In an ABox (Assertional Box), one describes a specific state of affairs of an application domain in terms of concepts and roles. Some of the concept and role atoms in the ABox may be defined names of the TBox. In the

Construct Name	Syntax	Semantics	
top concept	$\top$	$\top^{\mathcal{I}} = \Delta^{\mathcal{I}}$	$\mathcal{ALC}$
atomic concept	A	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$	
atomic role	R	$R^{\mathcal{I}} \sqsubseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$	
conjunction	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$	
disjunction	$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$	
negation	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$	
exists restriction	$\exists R.C$	$\{x \mid \exists y.\langle x, y\rangle \in R^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}}\}$	
value restriction	$\forall R.C$	$\{x \mid \forall y.\langle x, y\rangle \in R^{\mathcal{I}} \text{ implies } y \in C^{\mathcal{I}}\}$	
concept hierarchy	$C \sqsubseteq D$	$C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$	
role hierarchy	$R \sqsubseteq S$	$R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$	$\mathcal{H}$
inverse role	R^-	$\{\langle x, y\rangle \mid \langle y, x\rangle \in R^{\mathcal{I}}\}$	$\mathcal{I}$
transitive role	$R \in R_+$	$R^{\mathcal{I}} = (R^{\mathcal{I}})^+$	$\mathcal{S}$
role chains	$R_1 \circ \cdots \circ R_n \sqsubseteq S$	$R_1^{\mathcal{I}} \circ \cdots \circ R_n^{\mathcal{I}} \subseteq S^{\mathcal{I}}$	$\mathcal{R}$
nominal	$\{o\}$	$\{o^{\mathcal{I}}\}$	$\mathcal{O}$
number restriction	$\geq n\ R$	$\{x \mid \sharp\{y.\langle x, y\rangle \in R^{\mathcal{I}}\} \geq n\}$	$\mathcal{N}$
	$\leq n\ R$	$\{x \mid \sharp\{y.\langle x, y\rangle \in R^{\mathcal{I}}\} \leq n\}$	
qualified number restriction	$\geq n\ R.C$	$\{x \mid \sharp\{y.\langle x, y\rangle \in R^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}}\} \geq n\}$	$\mathcal{Q}$
	$\leq n\ R.C$	$\{x \mid \sharp\{y.\langle x, y\rangle \in R^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}}\} \leq n\}$	

Table 2.1: Syntax and semantics of the $\mathcal{ALC}$ and $\mathcal{S}$ families of description Logics (courtesy of [Baader *et al.* 2007, Horrocks & Patel-Schneider 2010]).

ABox, one introduces individuals, by giving them names, and one asserts properties of these individuals. We denote individual names by IRIs such as $o, o_1, o_2, \ldots, o_n$. Using concepts C and roles R, in an ABox, one can make two kinds of assertions:

$$o : C \qquad (o_1, o_2) : R$$

The first one, $o : C$, is called *concept assertion*: it states that o belongs to (the interpretation of) C, formally, $o^{\mathcal{I}} \subseteq C^{\mathcal{I}}$. The second one, $(o_1, o_2) : R$, is called *role assertion*: it states that o_1 is related by the role R with o_2, formally, $(o_1^{\mathcal{I}}, o_2^{\mathcal{I}}) \subseteq R^{\mathcal{I}}$. An ABox, denoted as $\mathcal{A}$, is a finite set of such assertions.

Example 4. *This example shows a set of ABox assertions that model a university domain.*

$$\begin{aligned}\mathcal{A} = \{&Jerome : Professor,\\ &SemanticWeb : Course,\\ &(Jerome, SemanticWeb) : givesCourse\}\end{aligned}$$

2.1.3 OWL

The Web Ontology Language (OWL) is a W3C recommendation which defines a family of knowledge representation languages for creating ontologies on the semantic web. The OWL language provides three increasingly expressive sublanguages (i.e., OWL

Construct Name	Syntax	Semantics
datatype	D	$D^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}_{\mathbf{D}}$
data value	v	$v^{\mathcal{I}} \in \Delta^{\mathcal{I}}_{\mathbf{D}}$
concrete role	T	$T^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}_{\mathbf{D}}$
enumerated datatype	$\{v_1, \ldots, v_n\}$	$\{v_1^{\mathcal{I}}, \ldots, v_n^{\mathcal{I}}\}$
exists restriction	$\exists T.D$	$\{x \mid \exists y.\langle x, y\rangle \in T^{\mathcal{I}} \text{ and } y \in D^{\mathcal{I}}\}$
value restriction	$\forall T.D$	$\{x \mid \forall y.\langle x, y\rangle \in T^{\mathcal{I}} \text{ implies } y \in D^{\mathcal{I}}\}$
number restriction	$\geq n\ T$	$\{x \mid \sharp\{y.\langle x, y\rangle \in T^{\mathcal{I}}\} \geq n\}$
	$\leq n\ T$	$\{x \mid \#\{y.\langle x, y\rangle \in T^{\mathcal{I}}\} \leq n\}$
qualified number restriction	$\geq n\ T.D$	$\{x \mid \sharp\{y.\langle x, y\rangle \in T^{\mathcal{I}} \text{ and } y \in D^{\mathcal{I}}\} \geq n\}$
	$\leq n\ T.D$	$\{x \mid \sharp\{y.\langle x, y\rangle \in T^{\mathcal{I}} \text{ and } y \in D^{\mathcal{I}}\} \leq n\}$

Table 2.2: Syntax and Semantics of Description Logics Datatypes (taken from [Horrocks & Patel-Schneider 2010])

comes with three dialects) namely, OWL Lite, OWL DL, and OWL full. OWL Lite is a syntactic subset of OWL DL that prohibits and/or restricts the use of certain constructors and axioms with the aim of making the language easier to understand and implement. OWL Lite is expressively equivalent to $\mathcal{SHIF}(\mathbf{D})$ – stands for the DL $\mathcal{S}$ with role hierarchy $\mathcal{H}$, inverse $\mathcal{I}$ and functionality $\mathcal{F}$, while OWL DL is based on the description logic $\mathcal{SHOIN}(\mathbf{D})$, shown in Table 2.1, [Horrocks & Patel-Schneider 2003]. OWL Full allows all RDF documents to be interpreted as OWL ontologies, which can only be done at the expense of decidability. Further, with OWL Full, one gets all the syntactic freedom of RDF. For instance, statements about statements (RDF reification) and meta-modelling are possible. On the other hand, OWL DL offers maximal expressiveness while maintaining decidability.

OWL 2 is an extension and revision of OWL 1 augmented with rich expressiveness and has been set as a standard since 2009 by W3C. Due to expressivity limitations (qualified cardinality restrictions, relational expressivity, datatype expressivity, and keys), syntax issues, meta-modeling, imports and versioning, annotations, and others has led to the extension of OWL 1 [Grau *et al.* 2008]. OWL 2 DL is based on the description logic $\mathcal{SROIQ}(\mathbf{D})$ which is more expressive than $\mathcal{SHOIN}(\mathbf{D})$ (OWL 1 DL). Where reasoning in $\mathcal{SROIQ}(\mathbf{D})$ is 2NExpTime-complete [Kazakov 2008] whereas in $\mathcal{SHOIN}(\mathbf{D})$ it is NExpTime-complete [Tobies 2001]. OWL 2 comes with lightweight ontology languages with differing expressive powers called OWL profiles (namely OWL 2 EL, OWL 2 QL, and OWL 2 RL). Here, we present a brief discussion regarding OWL 2, for a detailed presentation, we refer the interested reader to [Motik *et al.* 2009, Grau *et al.* 2008, Kazakov 2008] .

2.1.3.1 Syntax

The OWL 2 specification defines a Functional Style Syntax [Motik *et al.* 2009], among others, as a simple encoding of the metamodel which is different from the OWL 1 Abstract Syntax [Grau *et al.* 2008]. Here, we present the function-style syntax for OWL 2 constructs that is used as a syntax for the query language SPARQL-OWL (see

Section 5.2.2). Table 2.3 summarizes the basic concept and role constructs of OWL 2 where A is a class name (an IRI), C (possibly subscripted) is an arbitrary class, P is an object property name (an IRI), R (possibly subscripted) is an arbitrary object property, T is a data property name (an IRI).

OWL 2 Axioms OWL 2 axioms provide information about classes, properties, data ranges, keys and individuals, as shown in Table 2.4.

Functional style Syntax	DL Syntax
Descriptions (C)	
A	A
owl : Thing	$\top$
owl : Bottom	$\bot$
ObjectUnionOf(C_1 ... C_n)	$C_1 \sqcup \cdots \sqcup C_n$
ObjectIntersectionOf(C_1 ... C_n)	$C_1 \sqcap \cdots \sqcap C_n$
ObjectComplementOf(C)	$\neg C$
ObjectOneOf(o_1 ... o_n)	$\{o_1\} \sqcup \cdots \sqcup \{o_n\}$
ObjectSomeValuesFrom(R C)	$\exists R.C$
ObjectAllValuesFrom(R C)	$\forall R.C$
ObjectHasValue(R o)	$\exists R.\{o\}$
ObjectMinCardinality(n R C)	$\geq nR.C$
ObjectMaxCardinality(n R C)	$\leq nR.C$
ObjectExactCardinality(n R C)	$= nR.C$
ObjectHasSelf(R)	$\exists R.self$
DataSomeValuesFrom(U D)	$\exists U.D$
DataAllValuesFrom(U D)	$\forall U.D$
DataHasValue(U v)	$R : o$
DataMinCardinality(n U D)	$\geq nU.D$
DataMaxCardinality(n U D)	$\leq nU.D$
DataExactCardinality(n U D)	$= nU.D$
DataRanges(D)	
D	D
DataIntersectionOf(D_1 ... D_n)	$D_1 \sqcap \cdots \sqcap D_n$
DataUnionOf(D_1 ... D_n)	$D_1 \sqcup \cdots \sqcup D_n$
DataComplementOf(D)	$\neg D$
DataOneOf($v_1, \ldots, v_n$)	$\{v_1\} \sqcup \cdots \sqcup \{v_n\}$
DatatypeRestriction(D F_i v_i ... F_n v_n)	
ObjectProperties(R)	
owl : topObjectProperty	$\top_2$
owl : bottomObjectProperty	$\bot_2$
R	R
ObjectInverseOf(R)	R^-
Individuals (o)	
o	o
_:*anonymous*	_:*anonymous*
Datatype Properties (U)	
U	U
owl : topDataProperty	$\top_u$
owl : bottomDataProperty	$\bot_u$
Data values (v)	
v	v

Table 2.3: OWL Descriptions, Data Ranges, Properties, Individuals, and Data Values [Horrocks & Patel-Schneider 2010]

Functional style syntax	DL syntax
SubClassOf(C_1 C_2)	$C_1 \sqsubseteq C_2$
EquivalentClasses(C_1 ... C_n)	$C_1 \equiv \cdots \equiv C_n$
DisjointClasses(C_1 ... C_n)	$C_i \sqcap C_j \sqsubseteq \bot, i \neq j$
DisjointUnion(C C_1 ... C_n)	$C \equiv C_1 \sqcup \cdots \sqcup C_n$, $C_i \sqcap C_j \sqsubseteq \bot, i \neq j$
SubObjectPropertyOf(R_1 R_2)	$R_1 \sqsubseteq R_2$
EquivalentObjectProperties(R_1 ... R_n)	$R_1 \equiv \cdots \equiv R_n$
DisjointObjectProperties(R_1 ... R_n)	$R_i \sqcap R_j \sqsubseteq \bot, i \neq j$
InverseObjectProperties(R_1 R_2)	$R_1 \equiv R_2^-$
ObjectPropertyDomain(R C)	$\exists R.\top \sqsubseteq C$
ObjectPropertyRange(R C)	$\top \sqsubseteq \forall R.C$
FunctionalObjectProperty(P)	$\top \sqsubseteq \leq 1P$
InverseFunctionalObjectProperty(P)	$\top \sqsubseteq \leq 1P^-$
ReflexiveObjectProperty(P)	$\top \sqsubseteq \exists P.self$
IrreflexiveObjectProperty(P)	$\exists P.self \sqsubseteq \bot$
SymmetricObjectProperty(P)	$P \equiv P^-$
AsymmetricObjectProperty(P)	
TransitiveObjectProperty(P)	$P \circ P \sqsubseteq P$
SubDataPropertyOf(U_1 U_2)	$U_1 \sqsubseteq U_2$
EquivalentDataProperties(U_1 U_2)	$U_1 \equiv U_2$
EquivalentDataProperties(U_1 U_2)	$U_1 \equiv U_2$
EquivalentDataProperties($U_1 \ldots U_n$)	$U_i \equiv U_{i+1}$ for $1 \leq i < n$
DisjointDataProperties($U_1 \ldots U_n$)	$U_i \sqsubseteq \neg U_j$ for $1 \leq i < j \leq n$
DataPropertyDomain(U C)	$\exists U.\top_u \sqsubseteq C$
DataPropertyRange(U D)	$\top \sqsubseteq \forall U.D$
FunctionalDataProperty(U)	$\top \sqsubseteq \leq 1U$
HasKey(C U1...U_n)	$U_1 \ldots U_n$ key for C
SameIndividual($o_1 \ldots o_n$)	$o_i = o_{i+1}$ for $1 \leq i < n$
DifferentIndividuals($o_1 \ldots o_n$)	$o_i \neq o_j$ for $1 \leq i < j \leq n$
ClassAssertion(C o)	$o : C$
ObjectPropertyAssertion(R o_1 o_2)	$(o_1, o_n) : R$
NegativeObjectPropertyAssertion(R o_1 o_2)	$o_1 : (\neg \exists R.\{o_2\})$

Table 2.4: OWL axioms [Horrocks & Patel-Schneider 2010]

Example 5. *We represent Example 3 using functional style syntax.*

```
SubClassOf(PostgradStudent Student)
SubClassOf(UndergradStudent Student)
SubClassOf(Department Faculty)
SubClassOf(Faculty University)
DisjointClasses(Staff Student)
SubClassOf(Professor Person)
SubClassOf(Student Person)
EquivalentClasses(Chair
 ObjectIntersectionOf(Person
  ObjectSomeValuesFrom(headOf Department)))
EquivalentClasses(Student
 ObjectIntersectionOf(Person
  ObjectSomeValuesFrom(takesCourse Course)))
EquivalentClasses(Professor
 ObjectIntersectionOf(Person
  ObjectSomeValuesFrom(givesCourse Course)))
EquivalentObjectProperties(takesCourse
 ObjectInverseOf(givesCourse))
SubClassOf(ObjectSomeValuesFrom(headOf
 owl : Thing) Professor)
ObjectPropertyDomain(givesCourse Person)
```

2.1.3.2 Semantics

The semantics of OWL 2 constructs is given either using model theoretic OWL Direct Semantics or OWL RDF-based semantics [Hitzler *et al.* 2009].

OWL Direct Semantics In common with Description Logics, OWL 2 has a (First Order) model-theoretic semantics called the OWL 2 Direct Semantics. This semantics is basically equivalent to simply translating the ontology into a $\mathcal{SROIQ}(\mathbf{D})$ knowledge base as described in Table 2.1 and Table 2.2 and then applying the standard Description Logic semantics.

OWL RDF-based Semantics For ontologies that use the RDF syntax, an alternative semantic account can be given by extending the RDF model theory with new conditions that capture the meaning of the OWL 2 vocabulary as described in the OWL 2 RDF-Based semantics W3C document [Motik *et al.* 2009]. It extends the RDF semantics specification by additional semantic conditions for the OWL specific vocab-

ulary terms. This is due to the fact that transformation from an OWL ontology into an RDF graph and vice versa is possible[1]. This is to say, an OWL 2 ontology O can be transformed into an RDF graph $G = T(O)$ using a translation function T. Likewise, an RDF graph G that satisfies certain restrictions can be transformed into an OWL 2 DL ontology O_G.

In practice, the main difference between the Direct semantics and the RDF-Based semantics is that the latter can be applied to RDF graphs that do not respect the various restrictions on OWL 2 syntax [Motik *et al.* 2009]; indeed, the RDF-Based semantics can be applied to arbitrary RDF graphs. It is important to be aware, however, that additional meaning (beyond that derived from the RDF semantics [Hayes 2004]) is only given to those parts of the graph that correspond to OWL 2 constructions [Motik *et al.* 2009].

So far, we have presented two prominent languages of the semantic web, OWL and RDF. Now, we proceed with a discussion on how to query datasets created using these languages.

2.2 Querying

In the semantic web, querying RDF documents and OWL ontologies is done mainly using SPARQL. In this section, we present the foundations of SPARQL and its extension Path SPARQL (PSPARQL).

2.2.1 SPARQL

SPARQL [Prud'hommeaux & Seaborne 2008] is a W3C recommended query language for RDF. It is based on the notion of query patterns defined inductively from triple patterns. Next, we detail the syntax and semantics of SPARQL. We present the W3C standard syntax and an abstract syntax that is easily readable and can be translated into the μ-calculus (cf. Section 2.3.2).

2.2.1.1 W3C Syntax

SPARQL has an SQL-like syntax. In this section, we present the standard W3C recommended syntax. The anatomy of a SPARQL query consists of an optional prefix shortcut declaration, mandatory query result clause, optional dataset definition, mandatory query patterns (WHERE clause), and optional query modifiers. In the following, we briefly present these query constructs, for a detailed introduction, we refer the reader to [Prud'hommeaux & Seaborne 2008].

1. Namespace (or prefix) declaration: names in RDF and OWL are IRIs, these are often written as a shorthand using *prefix:localname*. Where *prefix:* is a prefix name that expands to an IRI, and *localname* is the remainder of the name. Its

[1] http://www.w3.org/TR/owl2-mapping-to-rdf/

intended purpose is to abbreviate IRI namespaces. In SPARQL, namespace declaration can be done using the PREFIX keyword as shown below:

PREFIX prefixname: <localname>

where prefixname : denotes the namespace name (the shorthand) and <localname> indicates the IRI (full name) of the resource.

Example 6. *Consider the following declarations respectively for the RDF and OWL vocabulary namespaces.*

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>

2. Query result clause: a SPARQL query has four query result constructs, namely, SELECT, ASK, CONSTRUCT and DESCRIBE. A SELECT query can optionally be specified to return unique results for each row. Query patterns often return duplicate bindings, and implementations must not eliminate duplicates unless explicitly instructed. In order to eliminate duplicate bindings, there are two select modifiers DISTINCT and REDUCED. The former guarantees no duplicated results whereas the latter may eliminate some, all, or no duplicates. The difference is that: DISTINCT can be expensive; REDUCED can do the straightforward de-duplication work (e.g. remove immediately repeated results) without having to remember every row. In many applications that is sufficient. A SPARQL SELECT statement can be defined using the following syntax:

   SELECT <var_list>

   where <var_list> indicates the list of variables to be projected.

3. Dataset definition: this is the place where the queried graphs for the query are specified. An RDF dataset is a collection of graphs. An RDF dataset comprises one graph, the *default graph*, which does not have a name, and zero or more *named graphs*, where each named graph is identified by a URI. A SPARQL query can match different parts of the query pattern against different graphs. In order to specify the list of queried graphs in SPARQL, one uses the following syntax:

   FROM <graph_uri>
   FROM NAMED <named_graph_uri>

   where <graph_list> indicates the IRI of the RDF default graph, similarly <named_graph_uri> denotes the IRI of the named graph.

4. Graph patterns: the graph patterns that are to be matched in the queried graph are placed in the where clause of a SPARQL query. Graph patterns are formed from triple patterns using joins (.), disjunctions (UNION), left outer join (OPTIONAL) or filter expressions (FILTER). *Triple patterns* are triples of an RDF graph where in place of IRIs, blank nodes or literals variables can appear in

subject, predicate or object positions. A set of triple patterns form *basic graph pattern* (BGP). Graph patterns that are connected with the UNION keyword to represent alternatives are known as *union graph patterns*. SPARQL allows optional results to be returned by using *optional graph patterns*. "When querying a collection of graphs, the GRAPH keyword is used to match patterns against named graphs. GRAPH can provide an URI to select one graph or use a variable which will range over the URIs of all the named graphs in the query's" dataset definition [Prud'hommeaux & Seaborne 2008]. Finally, graph patterns grouped together using curly braces, { and }, are called *group graph patterns*. They determine the scope of SPARQL constructs such as FILTER.

Example 7. *The following example shows the WHERE clause of a SPARQL query composed of basic, union, optional, and filter graph patterns.*

```
WHERE {
  P1 . P2 . {P3} UNION {P4 OPTIONAL {P5}} OPTIONAL {P6}
  FILTER (C)
}
```

where P1, ..., and P6 denote graph patterns and C denotes a filter expression (constraint).

5. Query modifiers: these are solution modifiers which are applied on the result set usually for ordering, slicing and rearranging. These include ORDER BY, HAVING, GROUP BY, LIMIT, OFFSET, and VALUES. ORDER BY allows to sort the extracted result according to some variables, either in descending or ascending order. The sort order can be fixed using the keywords DESC and ASC (where ASC is used as default sort order). The solution modifiers LIMIT and OFFSET can be used to fix the number of results that are returned and the first result mapping that should be output, respectively. They are particularly useful when combined with modifier ORDER BY. It should be noted that we do not consider these modifiers further in the rest of the thesis.

Next, we present an abstract syntax for SPARQL.

2.2.1.2 Abstract Syntax

In this thesis, we use an abstract syntax that can be easily translated into the μ-calculus [Pérez *et al.* 2009]. A tuple $t \in \text{UBV} \times \text{UV} \times \text{UBLV}$ in SPARQL, with V a set of variables disjoint from UBL, is called a triple pattern. Triple patterns grouped together using SPARQL operators ($\mathsf{AND}, \mathsf{UNION}, \mathsf{OPT}$[2]) form *query patterns* (or graph patterns).

[2] OPT is short for OPTIONAL.

Definition 1 (Query Pattern)**.** *A* query pattern q *is inductively defined as follows:*

$$q ::= t \mid q_1 \ \mathit{AND}\ q_2 \mid q_1 \ \mathit{UNION}\ q_2 \mid q_1 \ \mathit{OPT}\ q_2 \mid q_1 \ \mathit{FILTER}\ C$$

where C is a SPARQL constraint (a built-in condition [Pérez et al. 2009]). Constraints are introduced using the keyword FILTER. As atomic FILTER expressions, SPARQL allows unary predicates like BOUND, binary (in)equality predicates (= and !=), comparison operators like <, datatype conversion and string functions. Complex FILTER expressions can be built using !, || and &&, representing negation, disjunction, and conjunction, respectively.

SPARQL has four query constructs, viz. SELECT, ASK, CONSTRUCT, and DESCRIBE. We focus on SELECT queries which are the core of SPARQL queries.

Definition 2. *A* SPARQL SELECT query *is a query of the form $q\{\overrightarrow{w}\}$ where $\overrightarrow{w}$ is a tuple of variables that appear in q which are called* distinguished variables*, and q is a query pattern.*

Example 8 (SPARQL queries)**.** *Consider the following queries $q_1\{?x\}$ and $q_2\{?x\}$ on the graph of Example 1. q_1 selects all those who translated or wrote a poem whereas q_2 finds those who translated a poem or wrote anything else.*

```
SELECT ?x  WHERE  {
  { {?x ex:translated ?l} UNION {?x ex:wrote ?l} }
  ?l rdf:type ex:Poem .
}
```

```
SELECT ?x  WHERE {
  {?x ex:translated ?l . ?l rdf:type ex:Poem .} UNION
  {?x ex:wrote ?l }
}
```

q_1 finds all those authors who either translated or wrote a poem whereas q_2 selects those authors who translated a poem or wrote something.

The above example represents a *unary* SPARQL query – a query with one distinguished variable. Example 9 illustrates an *n-ary* SPARQL query – the number of distinguished variables in the query is n.

Example 9. *This query selects, author names', where they live in, and the population of the city they live in, for those who wrote a poem and live in the same city they were born in.*

```
SELECT ?n ?loc ?p  WHERE {
   ?x ex:wrote ?l .
   ?x ex: hasName ?n
```

```
    ?l rdf:type ex:Poem .
    ?x ex:livesIn ?loc .
    ?x ex:bornIn ?loc .
    ?loc ex:population ?p .
}
```

2.2.1.3 Semantics

The semantics of SPARQL queries is given by a partial mapping function ρ from V to UBL. The *domain* of ρ, $dom(\rho)$, is the subset of V on which ρ is defined. Two mappings ρ_1 and ρ_2 are said to be *compatible* if $\forall x \in dom(\rho_1) \cap dom(\rho_2)$, $\rho_1(x) = \rho_2(x)$. $\rho_1 \cup \rho_2$ is also a mapping (we use $\uplus$ when $\rho_1 \cap \rho_2 = \emptyset$). This allows for defining the join, union, and difference operations between two sets of mappings M_1, and M_2 as shown below:

$$\begin{aligned} M_1 \bowtie M_2 &= \{\rho_1 \cup \rho_2 \mid \rho_1 \in M_1, \rho_2 \in M_2 \text{ are compatible mappings } \} \\ M_1 \cup M_2 &= \{\rho \mid \rho \in M_1 \text{ or } \rho \in M_2\} \\ M_1 \setminus M_2 &= \{\rho \in M_1 \mid \forall \rho_2 \in M_2, \rho \text{ and } \rho_2 \text{ are not compatible } \} \end{aligned}$$

The *evaluation* of query patterns over an RDF graph G is inductively defined as follows:

$$[\![.]\!]_G : q \rightarrow 2^{V \times UBL}$$

$$[\![t]\!]_G = \{\rho |\ dom(\rho) = var(t) \text{ and } \rho(t) \in G\}$$

where $var(t)$ is the set of variables occurring in t.

$$\begin{aligned} [\![q_1 \text{ AND } q_2]\!]_G &= [\![q_1]\!]_G \bowtie [\![q_2]\!]_G \\ [\![q_1 \text{ UNION } q_2]\!]_G &= [\![q_1]\!]_G \cup [\![q_2]\!]_G \\ [\![q_1 \text{ OPT } q_2]\!]_G &= ([\![q_1]\!]_G \bowtie [\![q_2]\!]_G) \cup ([\![q_1]\!]_G \setminus [\![q_2]\!]_G) \\ [\![q_1 \text{ MINUS } q_2]\!]_G &= [\![q_1]\!]_G \setminus [\![q_2]\!]_G \\ [\![q\{\overrightarrow{w}\}]\!]_G &= \pi_{\overrightarrow{w}}([\![q]\!]_G) \end{aligned}$$

Where the projection operator $\pi_{\overrightarrow{w}}$ selects only those part of the mappings relevant to variables in $\overrightarrow{w}$. The semantics of FILTER expressions is defined as: given a mapping ρ and a SPARQL constraint C, we say that ρ satisfies C, denoted by $\rho(C) = \top$, if:

- $C = BOUND(x)$ with $x \in dom(\rho)$,
- $C = (x = c)$ with $x \in dom(\rho)$ and $\rho(x) = c$,
- $C = (x = y)$ with $x, y \in dom(\rho)$ and $\rho(x) = \rho(y)$,
- $C = (x! = c)$ with $x \in dom(\rho)$ and $\rho(x) \neq c$,
- $C = (x! = y)$ with $x, y \in dom(\rho)$ and $\rho(x) \neq \rho(y)$,
- $C = (x < c)$ with $x \in dom(\rho)$ and $\rho(x) < c$,
- $C = (x < y)$ with $x, y \in dom(\rho)$ and $\rho(x) < \rho(y)$,

- $C = (!C_1)$ with ρ does not satisfy C_1, in the following, we use $\rho(C) = \bot$ to denote this,
- $C = (C_1 \mid\mid C_2)$ with $\rho(C_1) = \top$ or $\rho(C_2) = \top$,
- $C = (C_1 \;\&\&\; C_2)$ with $\rho(C_1) = \top$ and $\rho(C_2) = \top$

It should be noted that there are semantic differences between the standard SPARQL semantics [Prud'hommeaux & Seaborne 2008] and the semantics the we use here (from [Pérez *et al.* 2009]), e.g. for FILTERs within OPTIONALs, or with respect to the 3 valued semantics of FILTER expressions i.e., when evaluating FILTER expressions the answers can be either true, false, or error. For more on the differences, we refer the reader to [Polleres 2012].

Definition 3 (Answers to a SPARQL query)**.** *Let $q\{\overrightarrow{w}\}$ be a SPARQL query, P its graph pattern, and G be an RDF graph, the set of* answers *to this query is given by:*

$$[\![q\{\overrightarrow{w}\}]\!]_G = \{\rho \mid \rho \in \pi_{\overrightarrow{w}}([\![P]\!]_G)\}$$

Example 10 (Answers to SPARQL queries)**.** *The answers to query $q_1\{?x\}$ and $q_2\{?x\}$ of Example 8 on graph G of Example 1 are respectively* $\{\langle\text{Poe}\rangle, \langle\text{Mallarme}\rangle\}$ *and* $\{\langle\text{Baudelaire}\rangle, \langle\text{Poe}\rangle, \langle\text{Mallarme}\rangle\}$*. Hence, $[\![q_1\{?x\}]\!]_G \subseteq [\![q_2\{?x\}]\!]_G$.*

SPARQL Query Evaluation under RDFS Entailment Regime

Definition 4 (SPARQL under RDFS entailment semantics)**.** *Given an RDF graph G and a basic graph pattern P, a partial mapping function ρ is a* solution for G and P under RDFS-entailment*, $\rho \in [\![P]\!]_G$, if:*

- *the domain of ρ is exactly the set of variable in P, i.e., $dom(\rho) = V(P)$,*
- *terms in the range of ρ occur in G,*
- *P', obtained from P by replacing blank nodes with either URIs, blank nodes, or RDF literals is such that: the RDF graph $sk(\rho(P'))$ is RDFS-entailed by $sk(G)$. The function $sk(.)$ replaces blank nodes with fresh URIs (URIs that are neither in the queried graph nor in the query).*

Since SPARQL's entailment regimes only change the evaluation of basic graph patterns, the evaluation of query patterns can be defined in the standard way [Prud'hommeaux & Seaborne 2008, Pérez *et al.* 2009]. Note that, the W3C working draft document[3] on the evaluation of SPARQL queries under the RDFS entailment regime considers the axiomatic triples. However, in this thesis, we do not consider the axiomatic triples, as can be noted from the definition above (which uses the RDFS semantics from [Muñoz *et al.* 2007]).

[3] http://www.w3.org/TR/sparql11-entailment/

2.2.2 PSPARQL

PSPARQL (short for Path SPARQL) extends SPARQL with regular expression patterns. PSPARQL overcomes the limitation of SPARQL1.0 which is the inability to express path queries. In fact, recently, W3C has taken the task of extending SPARQL with paths known as SPARQL 1.1 property paths. Before presenting the syntax and semantics of PSPARQL, let us briefly introduce the notion of regular expression patterns (cf. [Alkhateeb *et al.* 2009] for detailed discussion).

2.2.2.1 Regular Expressions

Regular expressions are patterns used to describe languages (i.e., sets of strings) from a given alphabet. Let $\Sigma = \{a_1, ..., a_n\}$ be an alphabet. A *string/word* is a finite sequence of symbols from the alphabet Σ. A word can be either empty ε or a sequence of alphabet symbols $a_1...a_n$. A *language* $\mathcal{L}$ is a set of words over Σ which is a subset of Σ^*, i.e, $\mathcal{L} \subseteq \Sigma^*$. If $A = a_1...a_n$ and $B = b_1...b_m$ are two words over some alphabet Σ, then $A.B$ is a word over the same alphabet defined as: $A \cdot B = a_1...a_n b_1...b_m$.

Definition 5 (Regular expression pattern)**.** *Given an alphabet* Σ *and a set of variables* V*, a* regular expression *can be constructed inductively as follows:*

$$e := uri \mid x \mid e_1 + e_2 \mid e_1 \cdot e_2 \mid e^+ \mid e^*$$

such that x denotes a variable, $e_1 + e_2$ denotes disjunction, $e_1 \cdot e_2$ denotes concatenation, e^+ denotes positive closure, and e^* denotes Kleene closure. Let U be a set of URIs and V a set of variables, a regular expression $\mathcal{R}(U, V)$ is a language over the alphabet $U \cup V$.

2.2.2.2 PSPARQL Syntax

The only difference between the syntax of SPARQL and PSPARQL is on triple patterns. Triple patterns in PSPARQL contain regular expressions in property positions instead of only URIs or variables as it is the case of SPARQL. Here, we refer to them as *path patterns*. Queries are formed based on the notion of query patterns defined inductively from path patterns: a tuple $t \in \text{UBV} \times \mathcal{R}(U, V) \times \text{UBLV}$, with V a set of variables disjoint from UBL, is called a path pattern. Path patterns grouped together using connectives (AND, UNION, OPT) form *query patterns*.

Definition 6 (PSPARQL query pattern)**.** *A* PSPARQL query pattern q *is inductively defined as follows :*

$$q = \; t \mid q_1 \; \textit{AND} \; q_2 \mid q_1 \; \textit{UNION} \; q_2 \mid q_1 \; \textit{OPT} \; q_2 \mid q_1 \; \textit{FILTER} \; C$$

A PSPARQL SELECT query can be defined in the same way as in Definition 2.

Example 11 (PSPARQL queries)**.** *The following queries are the rewritings of* q_1 *and* q_2 *of Example 8 using PSPARQL.*

```
 SELECT ?x
   WHERE {
       ?x    (:translated | :wrote) . rdf:type   :Poem .
   }

SELECT ?x
   WHERE {
       { ?x  (:translated  .  rdf:type)    :Poem }
          UNION
       { ?x :wrote  ?l .}
   }
```

2.2.2.3 PSPARQL Semantics

The semantics of PSPARQL queries can be defined in similar way as that of SPARQL. The only difference between SPARQL and PSPARQL is on the triple patterns. Here, we define the evaluation of PSPARQL triple patterns recursively as follows:

$$
\begin{aligned}
[\![<x, \varepsilon, y>]\!]_G &= \{\rho \mid \rho(x) = \rho(y)\} \\
[\![<x, z, y>]\!]_G &= \{\rho \mid \ <\rho(x), \rho(z), \rho(y)> \in G\} \\
[\![<x, e \dashv e', y>]\!]_G &= [\![<x, e, y>]\!]_G \cup [\![<x, e', y>]\!]_G \\
[\![<x, e.e', y>]\!]_G &= \exists n. [\![<x, e, n>]\!]_G \bowtie [\![<n, e', y>]\!]_G \\
[\![<x, e^+, y>]\!]_G &= \exists n_1, \ldots, n_k. \big([\![<x, e, y>]\!]_G \\
&\qquad \cup [\![<x, e, n_1>]\!]_G \bowtie [\![<n_1, e, y>]\!]_G \cup \ \cdots \\
&\qquad \cup [\![<x, e, n_1>]\!]_G \bowtie \cdots \bowtie [\![<n_k, e, y>]\!]_G\big) \\
&\qquad \text{such that } k \geq 0 \\
[\![<x, e^*, y>]\!]_G &= \{\rho \mid \rho(x) = \rho(y)\} \cup [\![<x, e^+, y>]\!]_G
\end{aligned}
$$

The evaluation of PSPARQL graph patterns over an RDF graph G is defined similarly to that of SPARQL graph patterns as shown in Section 2.2.1.3. The answers to a PSPARQL query are obtained in the same way as that of a SPARQL query which is shown in Definition 3.

Example 12 (Answers to PSPARQL queries)**.** *The answers to query* $q_1\{?x\}$ *and* $q_2\{?x\}$ *of Example 8 on graph* G *of Example 1 are respectively* $\{\langle\text{Poe}\rangle, \langle\text{Mallarme}\rangle\}$ *and* $\{\langle\text{Baudelaire}\rangle, \langle\text{Poe}\rangle, \langle\text{Mallarme}\rangle\}$*. Hence,* $[\![q_1\{?x\}]\!]_G \subseteq [\![q_2\{?x\}]\!]_G$*.*

Beyond this particular example, the goal of query containment is to determine whether this holds for any graph. In the following, we present the notion of query containment.

2.2.2.4 Query Containment

Evaluating a query under a set of schema axioms is a query whose answers are given with respect to graphs satisfying these axioms. Relying on this notion, we define query containment under schema axioms.

Definition 7 (Containment). *Given a set of (RDFS, $\mathcal{ALCH}$, or OWL) axioms $\mathcal{C}$ and two queries q_1 and q_2 with the same arity, q_1* is contained in *q_2 with respect to $\mathcal{C}$, denoted $q_1 \sqsubseteq_{\mathcal{C}} q_2$, iff $[\![q_1]\!]_G \subseteq [\![q_2]\!]_G$ for every graph G satisfying $\mathcal{C}$.*

The arity of a query is the number of variables which appear in the result clause of that query.

Example 13. *Given a schema $\mathcal{C} = \{(Student, sc, Person)\}$, and the following queries $q\{x\} = (x, type, Student)$ and $q'\{x\} = (x, type, Person)$. It can be seen that $\forall G.\ G \models \mathcal{C}, q \sqsubseteq_{\mathcal{C}} q'$ and $q' \not\sqsubseteq_{\mathcal{C}} q$.*

Definition 8 (Equivalence). *Two queries q_1 and q_2 with respect to $\mathcal{C}$ are equivalent, i.e., $q_1 \equiv_{\mathcal{C}} q_2$, iff $q_1 \sqsubseteq_{\mathcal{C}} q_2$ and $q_2 \sqsubseteq_{\mathcal{C}} q_1$.*

2.3 Modal Logics

In this section, we present the modal logic K_n and μ-calculus that we use to encode the containment problem of various SPARQL fragments.

2.3.1 K_n

A modal logic is a logic with modal operators (e.g., possibility and necessity) that extends predicate logic [Blackburn *et al.* 2007]. The syntax of K_n [Blackburn *et al.* 2007] is composed of a sets of *atomic propositions* $AP = \{b, q, r, ...\}$, a set of *modalities* $\text{Mod} = \{s, p, o, d, \bar{s}, \bar{p}, \bar{o}, \bar{d}\}$ for navigating in graphs. A *program* (modality) allows navigation in a transition system i.e.. going from one node (state) to another, a program's converse allows navigating backwards. To elaborate on the meaning of the modalities Mod, assume that there are nodes representing the subject, predicate and object of an RDF triple which are connected to a certain node (let us call it *triple node*, i.e., a node representing the mentioned triple). Now, in this small graph, one can navigate from the subject (resp. predicate, object) node to the triple node using the program s (resp. p, o) whereas their respective converse programs ($\bar{s}$, $\bar{p}$, $\bar{o}$) allow to move backwards i.e., from the triple node to subject, predicate, and object nodes. The program d and its converse $\bar{d}$ allow navigation between triple nodes. A modal logic formula, φ, can be defined inductively as follows:

$$\varphi \quad ::= \quad \top \mid \bot \mid q \mid \neg\varphi \mid \varphi \vee \psi \mid \varphi \wedge \psi \mid \langle a \rangle \varphi \mid [a]\varphi$$

where $q \in AP$ and $a \in \text{Mod}$ is either a transition program or its converse $\bar{a}$.

The semantics of K_n is given over a transition system, $\mathcal{M} = (S, R, L)$ where S is a non-empty set of nodes, $R : \text{Mod} \to 2^{S \times S}$ is the transition function, and $L : AP \to 2^S$ assigns a set of nodes to each atomic proposition where it holds. For converse modalities, R can be extended as $R(\bar{a}) = \{(s', s) \mid (s, s') \in R(a)\}$. A modal formula φ is *satisfied* in a state w of a transition system $\mathcal{M}$, denoted $\mathcal{M}, w \models \varphi$, according to the

following definition:

$$\begin{aligned}
\mathcal{M}, w \models \top &\text{ and } \mathcal{M}, w \not\models \bot \\
\mathcal{M}, w \models p &\text{ iff } w \in L(p) \\
\mathcal{M}, w \models \neg\varphi &\text{ iff } \mathcal{M}, w \not\models \varphi \\
\mathcal{M}, w \models \varphi \vee \psi &\text{ iff } \mathcal{M}, w \models \varphi \text{ or } \mathcal{M}, w \models \psi \\
\mathcal{M}, w \models \varphi \wedge \psi &\text{ iff } \mathcal{M}, w \models \varphi \text{ and } \mathcal{M}, w \models \psi \\
\mathcal{M}, w \models \langle a \rangle \varphi &\text{ iff there is some } v \in S \text{ such that} \\
&\quad (w, v) \in R(a) \text{ and } \mathcal{M}, v \models \varphi \\
\mathcal{M}, w \models [a]\varphi &\text{ iff for all } v \in S, \text{ if } (w, v) \in R(a), \text{ then } \mathcal{M}, v \models \varphi
\end{aligned}$$

A modal formula φ is *satisfiable* in K_n if there exists some $\mathcal{M} = (S, R, L)$ such that, for some $w \in S, \mathcal{M}, w \models \varphi$. In this case, we say that φ is satisfied in $\mathcal{M}$.

Example 14. *An example of a transition system is given in Figure 3.2.*

2.3.2 μ-calculus

The μ-calculus is a logic obtained by adding fixpoint operators to ordinary modal logic, or Hennessy-Milner logic [Kozen 1983]. The result is a very expressive logic, sufficient to subsume many other temporal logics such as CTL and CTL* [Blackburn *et al.* 2007]. The modal μ-calculus is easy to model-check, and so makes a good 'back-end' logic for tools. In this thesis, we mainly use the μ-calculus with nominals and converse modalities. In the following, we present its syntax and semantics.

The syntax of the μ-calculus is composed of countable sets of *atomic propositions* and *nominals* AP, a set of *variables* Var, a set of *programs and their respective converses* Prog for navigating in graphs. A μ-calculus formula, φ, can be defined inductively as follows:

$$\varphi \quad ::= \quad \top \mid \bot \mid q \mid X \mid \neg\varphi \mid \varphi \vee \psi \mid \varphi \wedge \psi \mid \langle a \rangle \varphi \mid [a]\varphi \mid \mu X \varphi \mid \nu X \varphi$$

where $q \in AP, X \in Var$ and $a \in Prog$ is a transition program or its converse $\bar{a}$. The greatest and least fixpoint operators (ν and μ) respectively introduce general and finite recursion in graphs [Kozen 1983]. A *sentence* is a formula with no free variable, i.e., each variable in the formula appears within the scope of μ or ν. Besides, we use the following syntactic sugars:

$$\begin{aligned}
\bot &= \neg\top \\
\varphi \vee \psi &= \neg(\neg\varphi \wedge \neg\psi) \\
[a]\varphi &= \neg\langle a \rangle \neg\varphi \\
\nu X.\varphi(X) &= \neg\mu X.\neg\varphi({}^{\neg X}/_{X}) \\
\varphi \Rightarrow \psi &= \neg\varphi \vee \psi \\
\varphi \Leftrightarrow \psi &= (\varphi \Rightarrow \psi) \wedge (\psi \Rightarrow \varphi)
\end{aligned}$$

The semantics of the μ-calculus is given over a transition system, $K = (S, R, L)$ where S is a non-empty set of nodes, $R : Prog \rightarrow 2^{S \times S}$ is the transition function, and $L : AP \rightarrow 2^S$ assigns a set of nodes to each atomic proposition or nominal where it holds, such that $L(p)$ is a *singleton* for each nominal p. For converse programs, R can be extended as $R(\bar{a}) = \{(s', s) \mid (s, s') \in R(a)\}$. The valuation function $V : \text{Var} \rightarrow 2^S$ maps each variable into a set of nodes. For a valuation V, variable X, and a set of nodes $S' \subseteq S$, $V[X/S']$ is the valuation that is obtained from V by assigning S' to X. The semantics of a formula in terms of a transition system K (a.k.a. Kripke structure) and a valuation function is represented by $[\![\varphi]\!]^K_V \subseteq S$. The semantics of basic μ-calculus formulae is defined as follows:

$$
\begin{aligned}
[\![\top]\!]^K_V &= S \\
[\![\bot]\!]^K_V &= \emptyset \\
[\![q]\!]^K_V &= L(q), q \in AP,\ L(q) \text{ is singleton if } q \text{ is a nominal} \\
[\![X]\!]^K_V &= V(X), X \in Var \\
[\![\neg\varphi]\!]^K_V &= S \backslash [\![\varphi]\!]^K_V \\
[\![\varphi \wedge \psi]\!]^K_V &= [\![\varphi]\!]^K_V \cap [\![\psi]\!]^K_V \\
[\![\varphi \vee \psi]\!]^K_V &= [\![\varphi]\!]^K_V \cup [\![\psi]\!]^K_V \\
[\![\langle a \rangle \varphi]\!]^K_V &= \{s \in S \mid \exists s' \in S.(s, s') \in R(a)\ \wedge s' \in [\![\varphi]\!]^K_V\} \\
[\![[a]\varphi]\!]^K_V &= \{s \in S \mid \forall s' \in S.(s, s') \in R(a) \Rightarrow s' \in [\![\varphi]\!]^K_V\} \\
[\![\mu X \varphi]\!]^K_V &= \bigcap \{S' \subseteq S \mid [\![\varphi]\!]^K_{V[X/S']} \subseteq S'\} \\
[\![\nu X \varphi]\!]^K_V &= \bigcup \{S' \subseteq S \mid S' \subseteq [\![\varphi]\!]^K_{V[X/S']}\}
\end{aligned}
$$

Note that the evaluation of sentences is independent of valuations and hence we define the following. For a sentence φ, a Kripke structure $K = (S, R, L)$, and $s \in S$, we denote $K, s \models \varphi$ if and only if $s \in [\![\varphi]\!]^K$, henceforth K is considered as a *model* of φ. In other words, K is considered as a model of ϕ if there exists an $s \in S$ such that $K, s \models \phi$. If a sentence has a model, then it is called *satisfiable*.

Another variety of the μ-calculus is the *μ-calculus with graded modalities*. Given a transition program or its converse a and a non-negative integer n, graded modalities generalize standard existential $\langle n, a \rangle$ and universal $[n, a]$ modalities [Kupferman *et al.* 2002]. For instance, $\langle n, a \rangle$ expresses that there exist at least n accessible states satisfying a certain formula and $[n, a] = \neg\langle n, a \rangle\neg$. The full μ-calculus, with graded modalities, converse modalities, and nominals, is undecidable [Bonatti *et al.* 2006] whereas its fragments are well-behaved as shown in Table 2.5, where $\mathcal{O}$ denotes nominals, $\mathcal{N}$ is for number restrictions (graded modalities), and $\mathcal{I}$ denotes converse modalities.

To study SPARQL query containment, only a specific subset of the μ-calculus presented above, namely the *alternation-free* modal μ-calculus with nominals and converse [Tanabe *et al.* 2008], is of interest. A μ-calculus formula φ is alternation-free if $\mu X.\varphi_1$ (respectively $\nu X.\varphi_1$) is a subformula of φ and $\nu Y.\varphi_2$ (respectively $\mu Y.\varphi_2$) is a subformula of φ_1 then X does not occur freely in φ_2. For instance, $\nu X.\mu Y.\langle s \rangle Y \wedge a) \vee \langle p \rangle X$

μ fragments	Complexity	Source
μ	ExpTime	[Kozen 1983]
$+\mathcal{N}$	ExpTime	[Kupferman *et al.* 2002]
$+\mathcal{O}+\mathcal{N}$	ExpTime	[Bonatti *et al.* 2006]
$+\mathcal{I}+\mathcal{N}$	ExpTime	[Bonatti *et al.* 2006]
$+\mathcal{O}+\mathcal{I}$	ExpTime	[Sattler & Vardi 2001]
$+\mathcal{O}+\mathcal{I}+\mathcal{N}$	Undecidable	[Bonatti & Peron 2004]

Table 2.5: Fragments of the full modal μ-calculus

is alternation-free but $\nu X.\mu Y.\langle s\rangle Y \wedge X) \vee a$ is not since X bound by ν appears freely in the scope of μY.

2.4 Related Work

Query optimization has been the subject of an important research effort for many types of query languages, with the common goal of speeding up query processing. The works found in [Stocker *et al.* 2008, Groppe *et al.* 2009, Schmidt *et al.* 2010] considered the problem of SPARQL query optimization. So, this thesis can be used to prove the correctness of query rewriting techniques. In the following we briefly review works that previously established closely related results for related query languages.

An early formalization of RDF(S) graphs has been presented in [Gutierrez *et al.* 2004], in which the complexity of query evaluation and containment is also studied. The authors investigate a datalog-style, rule-based query language for RDF(S) graphs. In particular, they establish the NP-completeness of query containment over simple RDF graphs, this result is also published in the RDF semantics document [Hayes 2004]. The query language is rather simple compared to SPARQL and no constraints were assumed for the problem. [Serfiotis *et al.* 2005] provides algorithms for the containment and minimization of RDF(S) query patterns utilizing concept and property hierarchies for the query language RQL (RDF Query Language). The NP-completeness is established for query containment concerning conjunctive and union of conjunctive queries. In line with this, a recent work in [Polleres 2007] shows how to translate SPARQL queries into non-recursive Datalog with negation. The paper does focus on query evaluation (not on query containment).

The work in [Groppe *et al.* 2009], investigated static analysis of SPARQL queries that are embedded in a Java program. It checks the correctness of the syntaxes of RDF data, SPARQL queries, and SPARUL update queries. Beyond this, their system called SWOBE (Semantic Web Objects Database Programming Language), detects if a query has a non-empty result set. Most recently, static analysis and optimization of OPTIONAL graph patterns is studied in [Letelier *et al.* 2012] where they proved the

Π_2^p-completeness of query subsumption and NP-completeness of query equivalence.

Besides works that focus on querying RDF graphs, in the following, we explore the relations and containment problem between SPARQL and query languages from other domains.

SPARQL vs. Relational Algebra It has been shown that SPARQL is equally expressive as relational algebra (RA) [Angles & Gutierrez 2008]. It is easy to see that relational algebra with SPJUD (Selection, Projection, Join, Union and Difference) [Abiteboul *et al.* 1995] operators is equivalent to that of SPARQL with SELECT, AND, UNION, OPTIONAL and FILTER as shown in Table 2.6. The algebraic operators that are defined in SPARQL resemble the algebraic operators defined in relational algebra; in particular, AND is mapped to the algebraic join, FILTER is mapped to the algebraic selection operator, UNION is mapped to the union operator, OPTIONAL is mapped to the left outer join (which allows for the optional padding of information), and SELECT is mapped to the projection operator. As opposed to the operators in relational algebra, which are defined on top of relations with fixed schema, the algebraic SPARQL operators are defined over so called mapping sets, obtained when evaluating triple patterns. In contrast to the fixed schema in relational algebra, the "schema" of mappings in SPARQL algebra is loose in the sense that such mappings may bind an arbitrary set of variables. This means that in the general case we cannot give guarantees about which variables are bound or unbound in mappings that are obtained during query evaluation.

Any relational language as powerful as relational algebra is called relationally complete. A relationally complete language can perform all basic, meaningful operations on relations. Since SQL is a superset of relational algebra, it is also relationally complete. There are some differences between the two query languages RA and SQL: Null values are usually excluded in the definition of relational algebra, except when operations like outer join are defined. Relational algebra treats relations as sets, i.e., duplicate tuples will never occur in the input/output relations of an RA operator. SQL relations are multisets (bags) and may contain duplicates. Duplicate elimination is explicit in SQL (SELECT DISTINCT). RA is also a yardstick for measuring the expressiveness of query languages. If a query language can express all possible RA queries, then it is said to be relationally complete. SQL is relationally complete. Vice versa, every SQL query (without null values, aggregation, and duplicates) can also be written in RA. Containment and equivalence of relational algebra are undecidable; this can be shown using the undecidability over finite structures of [Trakhtenbrot 1950].

Studies on the translation of SPARQL into relational algebra and SQL [Cyganiak 2005, Chebotko *et al.* 2006] indicate a close connection between SPARQL and relational algebra in terms of expressiveness. In [Polleres 2007], a translation of SPARQL queries into a datalog fragment (non-recursive datalog with negation) that is known to be equally expressive as relational algebra was presented. This translation makes the close connection between SPARQL and rule-based languages explicit and shows that RA is at least as expressive as SPARQL. Tackling the opposite direction, it was

Remark	RA	SPARQL
Selection (Restriction)	σ	FILTER
Projection	π	SELECT
Join (Inner Join)	$\bowtie$	AND
Left outer join	$⟕$	OPTIONAL
Union	$\cup$	UNION
Set Difference	$\setminus$	MINUS

Table 2.6: SPARQL vs. Relational algebra.

recently shown in [Angles & Gutierrez 2008] that SPARQL is relationally complete, by providing a translation of the above-mentioned datalog fragment into SPARQL. As argued in [Angles & Gutierrez 2008], the results from [Polleres 2007] and [Angles & Gutierrez 2008] taken together imply that SPARQL has the same expressive power as relational algebra. From early results on query containment in relational algebra and first-order logic, one can infer that containment in relational algebra is undecidable. Therefore, containment of SPARQL queries is also undecidable. Hence, in this thesis, we considered various fragments of SPARQL to study containment.

Databases Recently most work around the study of query containment has adopted Datalog to represent queries. A conjunctive query (CQ) in Datalog is simply a query where each predicate in the body of a rule references an extensional database relation [Abiteboul *et al.* 1995].

To decide containment for CQs there are two ways. The first method uses containment mapping. Consider a partial mapping λ from variables of a query q_1 to variables and constant of a query q_2 and extend μ to subgoals so that it maps the subgoals with same predicate name. Such a mapping is a *containment mapping* if it makes the subgoals in the body of q_1 a subset of the subgoals in the body of q_2, and the heads identical. q_2 *is contained in* q_1 (denoted $q_2 \sqsubseteq q_1$) if and only if there exists a containment mapping from q_1 to q_2. The second technique is based on canonicalization: to decide if $q_1 \sqsubseteq q_2$ one first freezes q_1 by replacing the variables of its body and head with constants. Then if q_2 includes the frozen head of q_1 in its answer set when applied over the canonical database consisting of just the frozen predicates of q_1, we may conclude that $q_1 \subseteq q_2$. When no predicate appears more than once in the body of the rule, a linear time algorithm exists to decide containment, otherwise the decision problem is NP-complete [Chandra & Merlin 1977]. This approach may be enriched to decide containment between conjunctive queries with inequalities ($CQ^{\neq}$) [Ullman 1997]. Containment between Datalog programs (that support recursion, but not negation) is undecidable [Shmueli 1993, Abiteboul *et al.* 1995]. Containment of a Datalog program within a conjunctive query is doubly exponential, while the converse question is easier [Chaudhuri & Vardi 1992].

Query Entailment is the decision problem associated with query answering. For CQs, query answering and containment are equivalent problems. In fact, query con-

tainment can be reduced to query answering [Calvanese *et al.* 1998]. In this regard, conjunctive query containment under the description logic $\mathcal{DLR}$ is studied in [Calvanese *et al.* 2008]. CQ query answering in the presence of simple ontologies (fragments of DL-Lite) has been studied [Calvanese *et al.* 2007, Lutz *et al.* 2009]. For expressive ontology languages, query entailment (and hence containment) in DLs ranging from $\mathcal{ALCI}$ to $\mathcal{SHIQ}$ is shown to be 2ExpTime in [Lutz 2008, Glimm *et al.* 2008, Ortiz *et al.* 2008a, Eiter *et al.* 2009]. See Table 2.7 for a partial summary of the studies on query answering.

In this study we do not deal with the same query language as the one dealt with in [Glimm *et al.* 2008]. In fact, the supported SPARQL fragment is strictly larger than the one studied in [Glimm *et al.* 2008]. Specifically, UCQs in [Glimm *et al.* 2008] are made of $C(x), R(x, y)$ for an atom C, a role R, and variables x and y, whereas we do also support queries capable of querying concept and role names at the same time, such as $q(x) = (x, y, z)$. Further, the purpose of reducing the problem to the μ-calculus is exactly about extending query containment to even more features (such as SPARQL 1.1 paths with recursion, entailment regimes, and negation). For instance, it is known that recursive paths can be easily supported in μ-calculus (using fixpoints). Beyond this, the novelty of the study is the reduction of the SPARQL containment problem to μ-calculus satisfiability, and the advantages of using such a logic: expressivity, good computational properties, extensibility. The main focus of the contribution is not the complexity bound by itself but rather a new approach with a broader logic, paving the way for future extensions as it was never done before.

DL	Axioms	Entailment
$\mathcal{ALC}$	$C_1 \sqsubseteq C_2$	ExpTime [Lutz 2008]
$\mathcal{ALCH}$	$C_1 \sqsubseteq C_2, R_1 \sqsubseteq R_2$	ExpTime [Ortiz *et al.* 2008b]
$\mathcal{ALCI}$	$C_1 \sqsubseteq C_2, R_1 \sqsubseteq R_2$	2ExpTime-hard [Lutz 2008]
$\mathcal{ALCHI}$	$C_1 \sqsubseteq C_2, R_1 \sqsubseteq R_2$	2ExpTime [Calvanese *et al.* 1998]
$\mathcal{SH}$	$C_1 \sqsubseteq C_2, R_1 \sqsubseteq R_2$	2ExpTime-hard [Eiter *et al.* 2009]
$\mathcal{SHIQ}$	$C_1 \sqsubseteq C_2, R_1 \sqsubseteq R_2$	2ExpTime-complete [Glimm *et al.* 2008]
$\mathcal{SHOQ}$	$C_1 \sqsubseteq C_2, R_1 \sqsubseteq R_2$	2ExpTime-complete [Glimm *et al.* 2008]

Table 2.7: The complexity of query entailment for the fragments of $\mathcal{SHOIQ}$.

Finally, with an implicit goal of minimizing query evaluation costs, in [Pichler *et al.* 2010] comprehensive complexity results were obtained for the problem of redundancy elimination on RDF graphs in the presence of rules (RDFS or OWL), constraints (tuple-generating dependencies) and with respect to SPARQL queries.

Semistructured data In line with CQs in databases and description logic worlds, we have regular path queries— languages that are used to query arbitrary length paths in graph databases — in semi structured data. Like CQs, they have been used and

studied widely. They are different from CQs in that, they allow recursion by using regular expression patterns. The problem of containment has been addressed for extensions of this language. In this regard, a prominent language used in semi-structured data is XPath. This language has been studied extensively over the last decade. These studies range from extending or reducing to static analysis. Static analysis of XPath queries has been studied in [Genevès *et al.* 2007], encompassing containment, equivalence, coverage, and satisfiability of XPath queries. In fact, this thesis is motivated by [Genevès *et al.* 2007] in that the approach to study these problems using a graph logic and provide a working implementation.

Other notable results come from the study of Regular Path Queries (RPQs). RPQs are extremely useful for expressing complex navigations in a graph. In particular, union and transitive closure are crucial when we do not have a complete knowledge of the structure of the knowledge base where this is the case for RDF graphs. Containment of (two-way) regular path queries (2RPQs) have been studied extensively [Calvanese *et al.* 2000,Calvanese *et al.* 2003,Barceló *et al.* 2010]. These languages are used to query graph databases and containment has been shown to be PSPACE-complete, this complexity bound jumps to EXPTIME-hard under the presence of functionality constraints. On the other hand, the containment of conjunctive 2RPQs is EXPSPACE-complete, this bound jumps to 2EXPTIME when considered under expressive description logic (DL) constraints [Calvanese *et al.* 2011]. However, it is exponential if the query on the right hand side has a tree structure (cf. for example, [Calvanese *et al.* 2008]). Further, paths are being included in the new version of SPARQL, thus this work can be used to test containment of path SPARQL queries under the RDFS entailment regime.

Containment under constraints Query containment has also been studied under different kinds of constraints. Results in this setting include, decidability of conjunctive query containment under functional and inclusion dependencies is studied in [Johnson & Klug 1984], also [Aho *et al.* 1979] proved decidability of this problem under functional and multi-valued dependencies. Further, decidability and undecidability results are proved in [Calvanese *et al.* 2008] for non-recursive datalog queries under expressive description logic constraints. Moreover, the undecidability is proved in [Calvanese & Rosati 2003] for recursive queries under inclusion dependencies.

The most closely related work is [Calvanese *et al.* 2008] in which query containment under description logic constraints is studied based on an encoding in propositional dynamic logic with converse (CPDL). They establish 2EXPTIME upper bound complexity for containment of queries consisting of union of conjunctive queries under $\mathcal{DLR}$ schema axioms. Our work is similar in spirit, in the sense that the μ-calculus is a logic that subsumes CPDL, and may open the way for extensions of the query languages and ontologies (for instance OWL-DL). Besides, the two languages are different since SPARQL allows for predicates to be used as subject or object of other triple patterns and can be in the scope of a variable. This is not directly allowed in $\mathcal{DLR}$ (union) of conjunctive queries. Our encoding of RDF graphs and SPARQL queries preserves this capability.

The evaluation of SPARQL queries under schema constraints is considered by W3C under the entailment regime principle. In this case, SPARQL queries are evaluated by taking into account the semantics of a schema language [Kollia *et al.* 2011]. It is possible to define query containment under such entailment regimes. We show how this can be done in this thesis.

Benchmarking Recently, static analysis and optimization of SPARQL queries has attracted widespread attention, notably [Chekol *et al.* 2011a, Letelier *et al.* 2012, Chekol *et al.* 2012a, Chekol *et al.* 2012b] for static analysis and [Stocker *et al.* 2008, Groppe *et al.* 2009, Schmidt *et al.* 2010, Letelier *et al.* 2012] for optimization. These studies have grounded the theoretical aspects of these fundamental problems. However, to the best of our knowledge, there is only one implementation from [Letelier *et al.* 2012] and it supports only conjunction and OPTIONAL queries with no projection (containment of basic and optional graph patterns).

On the other hand, in databases, containment of union conjunctive queries (UCQs) is well studied and has a well know NP-complete complexity. The importance of the study of this problem goes beyond the field of databases, it has its fair share from the description logic community. Many of the works, from description logics, concentrated on the problem of query answering as containment follows from it. These works, have sound theoretical proofs, algorithms, and mathematical explanations. However, they lack an implementation (or experimentation) of their approaches.

Finally, various SPARQL query evaluation performance benchmarks have been proposed [Bizer & Schultz 2008, Bizer & Schultz 2009, Schmidt *et al.* 2009], but no SPARQL query containment benchmark to our knowledge.

Model checking is a technique for automatically verifying correctness properties of finite-state systems. Specifications about the system are expressed as logic (in our case, μ-calculus) formulas, and efficient symbolic algorithms are used to traverse the model defined by the system and check if the specification holds or not. Extremely large state-spaces can often be traversed in minutes. From a complexity analysis point of view, model checking is less complex than satisfiability. For instance, for the alternation-free mu-calculus, the former has PTime complexity in terms of the size of the model and the formula, on the other hand, the latter is ExpTime-complete. Thus, there is a huge gap between the complexity of model checking and satisfiability test. Modal model checking is a computationally tractable task, but this is not the case for first-order logic. In fact, model checking first-order formulas is a PSPACE-complete task (see [Chandra & Merlin 1977]). A broad discussion can be found in [Blackburn *et al.* 2007]. In addition to using model checking as a query evaluation, it can also be used to check satisfaction of integrity constraints. In the case of RDF graphs, integrity constraints can be produced from RDF schema axioms. Thus, it is possible to determine if a given RDF graph satisfies a given set of schema constraints (or axioms). However, in this thesis, we do not study model checking (neither for query evaluation nor constraint satisfaction), we focus only on satisfiability (as the containment problem is reduced

into unsatisfiability test in the μ-calculus).

Chapter 3

Containment of SPARQL Queries Under Schema

Contents

In this chapter, we address SPARQL query containment under expressive description logic constraints. We apply an approach which has already been successfully applied for XPath [Genevès *et al.* 2007]. SPARQL is interpreted over graphs, hence we encode it in a graph logic, specifically the alternation-free fragment of the μ-calculus [Kozen 1983] with converse and nominals [Tanabe *et al.* 2008] interpreted over labeled transition systems. We show that this logic is powerful enough to deal with query containment for the fragment of SPARQL, made of basic and union graph patterns, in the presence of $\mathcal{ALCH}$ schema axioms. Furthermore, this logic admits exponential time decision procedures that can be implemented in practice [Tanabe *et al.* 2005, Genevès *et al.* 2007, Tanabe *et al.* 2008].

We show how to translate RDF graphs into transition systems and SPARQL queries and schema axioms into μ-calculus formulae. Therefore, query containment in SPARQL can be reduced to unsatisfiability in the μ-calculus. An additional benefit of using a μ-calculus encoding is to take advantage of fixpoints and modalities for encoding recursion. They allow to deal with natural extensions of SPARQL such as path queries [Alkhateeb *et al.* 2009] or queries modulo RDF Schema. For more on this, we refer the interested reader to Chapter 4 and 5.

3.1 RDF Graphs as Transition Systems

In this section, we show how to translate RDF graphs into labeled transition systems. First of all, translating RDF graphs into transition systems is necessary in order to

restrict the models of the μ-calculus formula obtained from the translation of queries. Additionally, if RDF graphs can be translated into transition systems, then model checking can be used to evaluate SPARQL queries. In fact, in this regard, there is already some progress as presented in the literature [Mateescu *et al.* 2009] where it is possible to extend and encode SPARQL queries in a logic and use model checking to evaluate the result of the query.

There are several ways of encoding RDF graphs as transition systems, for instance, consider the following:

- for each triple $(s, p, o) \in G$, s and o become nodes of the transition system and p is a transition program, there is an edge $\langle s, o \rangle$ where transition from node s to o and vice versa can be done using program p and its converse $\bar{p}$ respectively. While this approach is simple and intuitive, it does not work in the general case, i.e., in an RDF graph predicates or properties can also be nodes in an RDF graph as shown in Figure 3.1, thus, p cannot be a transition program.

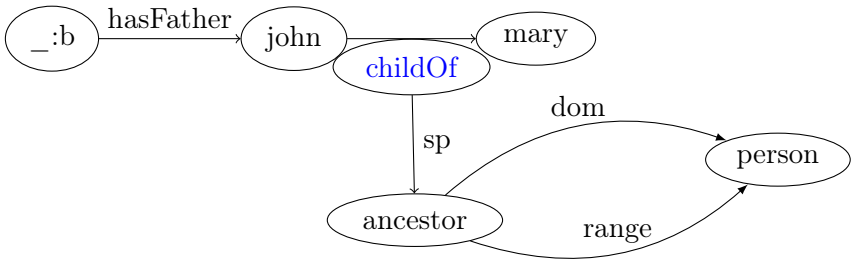


Figure 3.1: An RDF graph where a predicate appears as a node

- for each triple $(s, p, o) \in G$, s, p, and o become atomic propositions that are true in the states n_s, n_p, and n_o respectively of a transition system, there are are edges $\langle n_s, n_p \rangle$ and $\langle n_p, n_o \rangle$ that are accessible through transition programs 1 and 2 respectively. Similar to the above approach, this translation procedure does not work in the general case when encoding RDF schema graphs. For example, consider an RDF graph that contains the triple (subPropertyOf, subPropertyOf, subPropertyOf).

- the last approach considers encoding RDF graphs as bipartite graphs, i.e., for each $t = (s, p, o) \in G$ introduce two sets of nodes in the transition system: one set for each triple n_t and another set for each element of the triple n_s, n_p, and n_o where atomic propositions s, p and o are set to be true respectively. Additionally, there are edges $\langle n_s, n_t \rangle$, $\langle n_t, n_p \rangle$, and $\langle n_t, n_o \rangle$ in the transitions system that are accessible through programs s, p, o and their converses respectively. The idea of representing RDF triples as other types of graphs (for instance, hypergraphs) was first introduced in [Baget 2005], in fact, this translation coincides with the notion of reification of n-ary relations [Calvanese *et al.* 2008] that is one edge from the

triple node to subject, predicate, and object nodes of the triple in this case. This approach overcomes the limitations of the other two approaches. Thus, in the following, we discuss in detail how this technique works.

3.1.1 Encoding of RDF graphs

An RDF graph is encoded as a transition system in which nodes correspond to RDF entities and RDF triples. Edges relate entities to the triples they occur in. Different edges are used for distinguishing the functions (subject, object, predicate). Expressing predicates as nodes, instead of atomic programs, makes it possible to deal with full RDF expressiveness in which a predicate may also be the subject or object of a statement.

Definition 9 (Transition system associated to an RDF graph)**.** *Given an RDF graph, $G \subseteq UB \times U \times UBL$, the transition system associated to G, $\sigma(G) = (S, R, \mathsf{L})$ over $AP = \mathrm{U_G B_G L_G} \cup \{s', s''\}$, is such that:*

- $S = S' \cup S''$ *with* S' *and* S'' *the smallest sets such that* $\forall u \in U_G, \exists n^u \in S'$, $\forall b \in B_G, \exists n^b \in S'$, $\forall l \in L_G, \exists n^l \in S'$, *and* $\forall t \in G, \exists n^t \in S''$,
- $\forall t = (s, p, o) \in G$, $\langle n^s, n^t \rangle \in R(s)$, $\langle n^t, n^p \rangle \in R(p)$, *and* $\langle n^t, n^o \rangle \in R(o)$,
- $\mathsf{L} : AP \to 2^S$; $\forall u \in U_G, \mathsf{L}(u) = \{n^u\}$, $\forall b \in B_G, \mathsf{L}(b) = S'$, $\mathsf{L}(s') = S'$, $\forall l \in L_G, \mathsf{L}(l) = \{n^l\}$ *and* $\mathsf{L}(s'') = S''$,
- $\forall n^t, n^{t'} \in S''$, $\langle n^t, n^{t'} \rangle \in R(d)$.

The program d is introduced to render each triple accessible to the others and thus facilitate the encoding of queries. The function σ associates what we call a *restricted transition system* to any RDF graph. Formally, we say that a transition system K is a *restricted transition system* iff there exists an RDF graph G such that $K = \sigma(G)$.

A restricted transition system is thus a bipartite graph composed of two sets of nodes: S', those corresponding to RDF entities, and S'', those corresponding to RDF triples. For example, Figure 3.2 shows the restricted transition system associated with the graph of Example 1.

Given that the logic chosen to determine containment is μ-calculus with nominals (lacking functionality or number restrictions), one cannot impose that each triple node is connected to exactly one node for each of the three triple-components (subject, predicate, and object). However, we can impose a lighter restriction to achieve this by taking advantage of the technique introduced in [Genevès & Layaïda 2006]. Since it is not possible to ensure that there is only one successor, then we restrict all the successors to bear the same constraints. They thus become interchangeable (bisimulation). To do this, we introduce a rewriting function f such that all occurrences of $\langle a \rangle \varphi$ (existential formulas) are replaced by $\langle a \rangle \top \wedge [a] \varphi$. As such, f is inductively defined on the structure

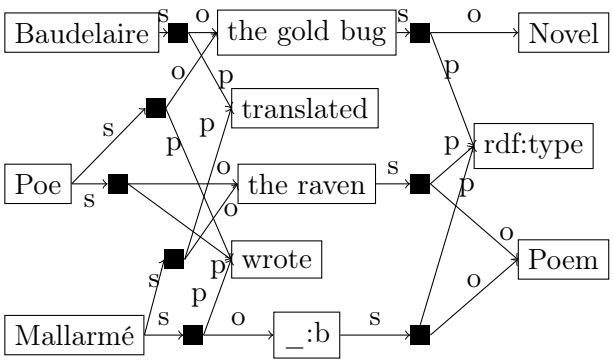


Figure 3.2: Transition system encoding the RDF graph of Example 1. Nodes in S'' are black anonymous nodes; nodes in S' are the other nodes (d-transitions are not displayed).

of a μ-calculus formula as follows:

$$
\begin{aligned}
f(\top) &= \top \\
f(q) &= q \quad q \in AP \cup Nom \\
f(X) &= X \quad X \in Var \\
f(\neg\varphi) &= \neg f(\varphi) \\
f(\varphi \wedge \psi) &= f(\varphi) \wedge f(\psi) \\
f(\varphi \vee \psi) &= f(\varphi) \vee f(\psi) \\
f(\langle a \rangle \varphi) &= \langle a \rangle \top \wedge [a] f(\varphi) \quad a \in \{\bar{s}, p, o\} \\
f(\langle a \rangle \varphi) &= \langle a \rangle f(\varphi) \quad a \in \{d, s, \bar{p}, \bar{o}\} \\
f([a]\varphi) &= [a] f(\varphi) \quad a \in \text{Prog} \\
f(\mu X.\varphi) &= \mu X.f(\varphi) \\
f(\nu X.\varphi) &= \nu X.f(\varphi)
\end{aligned}
$$

Thus, when checking for query containment, we assume that the formulas are rewritten using function f. Along with that, we also consider the following restrictions:

- The set of programs is fixed: $Prog = \{s, p, o, d, \bar{s}, \bar{p}, \bar{o}, \bar{d}\}$.
- A model must be a restricted transition system.

The last constraint can be expressed in the μ-calculus as follows:

Proposition 1 (RDF restriction on transition systems)**.** *Let φ be a formula that can be stated over a restricted transition system, φ is satisfied by some restricted transition system if and only if $f(\varphi) \wedge \varphi_r$ is satisfied by some transition system over $Prog$, i.e. $\exists K_r . [\![\varphi]\!]^{K_r} \neq \emptyset \iff \exists K . [\![f(\varphi) \wedge \varphi_r]\!]^{K} \neq \emptyset$, where:*

$$\varphi_r = \quad \nu X.\ \theta \wedge \kappa\ \wedge (\neg\langle d \rangle \top \vee \langle d \rangle X)$$

in which $\theta = \langle\bar{s}\rangle s' \wedge \langle p\rangle s' \wedge \langle o\rangle s' \wedge \neg\langle s\rangle\top \wedge \neg\langle\bar{p}\rangle\top \wedge \neg\langle\bar{o}\rangle\top$, *and*
$\kappa = [\bar{s}]\xi \wedge [p]\xi \wedge [o]\xi$ *with*

$$\xi = \neg\langle\bar{s}\rangle\top \wedge \neg\langle o\rangle\top \wedge \neg\langle p\rangle\top \wedge \neg\langle d\rangle\top \wedge \neg\langle\bar{d}\rangle\top \wedge \neg\langle s\rangle s' \wedge \neg\langle\bar{o}\rangle s' \wedge \neg\langle\bar{p}\rangle s'$$

The formula φ_r ensures that θ and κ hold in every node reachable by a d edge, i.e., in every s'' node. The formula θ forces each s'' node to have a subject, predicate and object. The formula $f(\varphi)$ enforces reification (makes sure that each s'' node is connected to one subject, one predicate, and one object node). The formula κ navigates from a s'' node to every reachable s' node, and forces the latter not to be directly connected to other subject, predicate or object nodes.

Proof. ($\Rightarrow$) Assume that $\exists K_r [\![f(\varphi)]\!]^{K_r} \neq \emptyset$, since φ_r is satisfied by any restricted transition system, one gets $[\![\varphi_r]\!]^{K_r} \neq \emptyset$. Hence it follows that, $\exists K_r [\![f(\varphi)]\!]^{K_r} \neq \emptyset$ and $[\![\varphi_r]\!]^{K_r} \neq \emptyset$ which imply $\exists K_r [\![f(\varphi)]\!]^{K_r} \wedge [\![\varphi_r]\!]^{K_r} \neq \emptyset$. From this, using the semantics of μ-calculus formula, one obtains $\exists K_r [\![f(\varphi) \wedge \varphi_r]\!]^{K_r} \neq \emptyset$. Since a restricted transition system is also a transition system, $K_r \subseteq K$, it follows that $\exists K.[\![f(\varphi) \wedge \varphi_r]\!]^{K} \neq \emptyset$.
($\Leftarrow$) Assume that $\exists K [\![f(\varphi) \wedge \varphi_r]\!]^{K} \neq \emptyset$. We construct a restricted transition system model $K_r = (S_r = S_r^{'} \cup S_r^{''}, R_r, L_r)$ and a function $g : K_r \to K$ from $K = (S, R, L)$. Add a node n_0' to S_r with $g(n_0') = n_0$ where $f(\varphi) \wedge \varphi_r$ is satisfied in K. Suppose we have constructed a node n_r of S_r. For $j \in \{s, p, o\}$, if there is $n \in S$ with $(g(n_r), n) \in R(j)$, then pick one such n and add a node n_r' to S_r with $g(n_r') = n$. In such construction, if there are concurrent $\bar{s}, p, o$ transitions from an $S_r^{''}$ node, we retain one transition for each modality. This is because, if such transitions are part of the model that satisfy $f(\varphi) \wedge \varphi_r$, then they will be under the influence of the constraints $f(.)$ and φ_r, and will bear these constraints. However, if they belong to K that does not satisfy the aforementioned formula, then cutting them will not affect the capacity of the model to be a model for the formula. Finally, for an atomic proposition p, $L_r(p) = \{n_r \in S_r \mid g(n_r) \in L(p)\}$. The RDF triple structure is maintained in K_r i.e. $\langle(s, s''), (s'', p), (s'', o)\rangle$ is valid throughout the graph. If there were node pairs outside of this structure, then φ_r will not be satisfied. Throughout the graph, θ, $f(.)$ and κ ensure that for each triple node $s'' \in S_r$, there exists an incoming subject edge, an outgoing property edge, and an outgoing object edge. Hence, $[\![\varphi_r]\!]^{K_r} \neq \emptyset$.
To verify that $[\![f(\varphi)]\!]^{K_r} \neq \emptyset$, it is enough to show that $[\![f(\varphi)]\!]^{K} \neq \emptyset \Rightarrow [\![f(\varphi)]\!]^{K_r} \neq \emptyset$ by induction on the structure of $f(\varphi)$. □

If a μ-calculus formula ψ appears under the scope of a least μ or greatest ν fixed point operator over all the programs $\{s, p, o, d, \bar{s}, \bar{p}, \bar{o}, \bar{d}\}$ as, $\mu X.\psi \vee \langle s\rangle X \vee \langle p\rangle X \vee \cdots$ or $\nu X.\psi \wedge \langle s\rangle X \wedge \langle p\rangle X \wedge \cdots$, then, for the sake of legibility, we denote the formulae by $lfp(X, \psi)$ and $gfp(X, \psi)$, respectively.

So far we have showed how RDF graphs can be translated into transition systems over which the μ-calculus formulas are translated. In the following, we propose methods that are used to encode schema axioms and queries as μ-calculus formulas. Thus, at a later point, we use these encodings to reduce containment test into the validity problem.

3.1.2 Encoding Axioms

In this section, we provide the encoding of $\mathcal{ALCH}$ axioms, which can be considered as a fragment of $\mathcal{SROIQ}$ without role inverse, role transitivity, role composition, nominals and qualified number restrictions. These encodings are used together with query encodings to determine if any two queries are contained in each other.

Definition 10 (μ-calculus encoding of an $\mathcal{ALCH}$ schema)**.** *Given a set of axioms $c_1, c_2, ..., c_n$ of a schema $\mathcal{C}$, the μ-calculus encoding of $\mathcal{C}$ is:*

$$\eta(\mathcal{C}) = \eta(c_1) \wedge \eta(c_2) \wedge ... \wedge \eta(c_n).$$

Where η translates each axiom into an equivalent formula using ω which recursively encodes concepts and roles:

- *Concept Inclusion*

$$\begin{aligned}
\eta(C_1 \sqsubseteq C_2) &= \mathit{gfp}\big(X, \omega(C_1) \Rightarrow \omega(C_2)\big) \\
\omega(\bot) &= \bot \\
\omega(A) &= A \\
\omega(\neg C) &= \neg\omega(C) \\
\omega(C_1 \sqcap C_2) &= \omega(C_1) \wedge \omega(C_2) \\
\omega(\exists R.C) &= \langle s\rangle\big(\langle p\rangle R \wedge \langle o\rangle(\langle s\rangle\langle o\rangle\omega(C))\big) \\
\omega(\forall R.C) &= [s]\big([p]R \Rightarrow [o]([s][o]\omega(C))\big) \\
\omega(\exists R^{-}.C) &= \langle \bar{o}\rangle\big(\langle p\rangle R \wedge \langle \bar{s}\rangle(\langle s\rangle\langle o\rangle\omega(C))\big) \\
\omega(\forall R^{-}.C) &= [\bar{o}]\big([p]R \Rightarrow [\bar{s}]([s][o]\omega(C))\big)
\end{aligned}$$

- *Role Inclusion*

$$\eta(R_1 \sqsubseteq R_2) = \mathit{gfp}\big(X, R_1 \Rightarrow R_2\big)$$

Next, we provide procedures to translate unions of conjunctive SPARQL queries, i.e., SPARQL queries that only use UNION and AND, into μ-calculus formulas.

3.2 SPARQL Query Containment

In this section, we encode queries as μ-calculus formulas. Then, we reduce query containment under schemas to μ-calculus unsatisfiability test and prove the correctness of this reduction.

3.2.1 Encoding Queries as μ-calculus Formulae

In this section, we discuss the encoding of the containment problem $q_1\{\overrightarrow{w}\} \sqsubseteq q_2\{\overrightarrow{w}\}$. For any query $q\{\overrightarrow{w}\}$, we call the variables in $\overrightarrow{w}$ *distinguished* or answer variables.

Furthermore, we denote the *non-distinguished* or existential variables in q by $ndvar(q)$, the URIs/constants by $uris(q)$, and the distinguished variables by $dvar(q)$. When encoding $q_1 \sqsubseteq q_2$, we call q_1 left-hand side query and q_2 right-hand side query. q_1 is a union of conjunctive SPARQL query whereas q_2 is a union of conjunctive SPARQL$_{cdfc}$ query (cf. Definition 13).

Queries are translated into μ-calculus formulas. The principle of the translation is that each triple pattern is associated with a sub-formula stating the existence of the triple somewhere in the graph. Hence, they are quantified by μ so as to put them out of the context of a state. In this translation, variables are replaced by nominals or some formula that are satisfied when they are at the corresponding position in such triple relations. A function called $\mathcal{A}$ is used to encode queries inductively on the structure of query patterns. AND and UNION are translated into boolean connectives $\wedge$ and $\vee$ respectively.

Encoding left-hand side query:

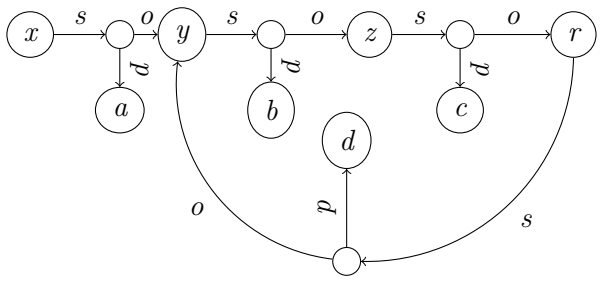
q_1 is frozen, that is, every term in q_1 becomes a nominal in μ-calculus. Here we introduce two sets of nominals, one set for denoting constants and the other for the distinguished variables. Further, function $\mathcal{A}$ is used to recursively compute a μ-calculus formula corresponding to q_1. The encoding of the SPARQL query is $\mathcal{A}(q)$ such that:

$$\begin{aligned}\mathcal{A}((x,y,z)) &= lfp\big(X, \langle\bar{s}\rangle x \wedge \langle p\rangle y \wedge \langle o\rangle z\big)\\ \mathcal{A}(q_1 \text{ AND } q_2) &= \mathcal{A}(q_1) \wedge \mathcal{A}(q_2)\\ \mathcal{A}(q_1 \text{ UNION } q_2) &= \mathcal{A}(q_1) \vee \mathcal{A}(q_2)\end{aligned}$$

In order to encode the right-hand side query, we need to define the notion of cyclic queries.

Definition 11 (Cyclic Query)**.** *A SPARQL query is referred to as cyclic if a transition graph induced from the query patterns is cyclic. The transition graph*[1] *is constructed in the same way as the transition system of Definition 9.*

Example 15. *Let q be the query $q\{x\} = (x,a,y),(y,b,z),(z,c,r),(r,d,y)$ where $ndvar(q) = \{y,z,r\}$ and $dvar(q) = \{x\}$. q is cyclic, as shown graphically,*



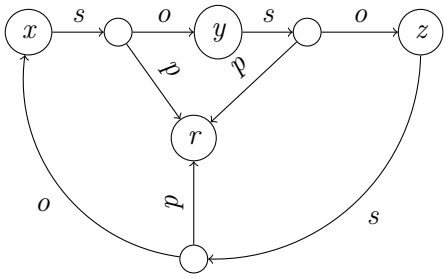
[1]The transition graph is similar to the tuple-graph used in [Calvanese *et al.* 2008] to detect the dependency among variables.

In this example, the cyclic component involves non-distinguished variables and constants $\{b, c, d\}$.

Example 16. *Consider the following cyclic query:*

$$q\{\} = (x, r, y)(y, r, z)(z, r, x)$$

a graph obtained from the graph patterns is shown below:



We can identify various features from this example:

- cyclicity*: the query contains a cycle,*
- distinguished variable-free*: the query does not refer to any distinguished variable,*
- constant-free*: the query does not refer to any constant.*

We refer to such cycles as constant and distinguished variable-free cycles (cdfc) *and denote such queries as* $SPARQL_{cdfc}$.

Definition 12. *A cdfc component of a query is a connected component of the query graph that:*

- *contains no constants,*
- *contains no distinguished variables,*
- *contains a cycle.*

We have carried out experiments on DBpedia query logs[2] in order to determine how many of the real world queries contain a cdfc component. Consequently, we found out that non of the queries (i.e., 0%) contain a cdfc component. Thus, this highly motivates the benefits of this thesis.

Definition 13. $SPARQL_{cdfc}$ *is the set of SPARQL queries which do not contain any cdfc component.*

[2]DBpedia 3.5.1 logs between 30/04/2010 and 20/07/2010 from ftp://download.openlinksw.com/support/dbpedia/

Encoding right-hand side query:

the encoding of the right-hand side query q' is different from that of the left due to the non-distinguished variables that appear in cycles in the query. The distinguished variables and constants are encoded as nominals whereas the non-distinguished variables $ndvar(q')$ are encoded as follows:

- If a non-distinguished variable x occurs only once in q', x is encoded as $\top$.

- If a non-distinguished variable appears multiple times in q', then we produce a set of mappings $m = \{m_1, \ldots, m_n\}$ such that each m_i contains formula assignments to the non-distinguished variables. m is produced as follows:

  - we denote the union of the set of distinguished variables and constants of q' by $\mathbf{X}$, i.e., $\mathbf{X} = uris(q') \cup dvar(q')$,

  - for any triple $t = (s, p, o)$, functions f_s, f_p, and f_o return the subject, predicate, and object of t respectively,

$$f_s((s,p,o)) = s$$
$$f_p((s,p,o)) = p$$
$$f_o((s,p,o)) = o$$

  - for each multiply occurring non-distinguished variable x_l, given that $\{x_1, \ldots, x_k\} \in ndvar(q')$, assign it one of the triple patterns $t_j \in q'$ where it appears in, i.e., x_l appears in the triple pattern t_j, from that we obtain m_i's as:

$$m_i = \bigcup_{l=1}^{k} \{x_l \mapsto \alpha(x_l, t_j) \mid x_l \in t_j\}$$

$$\alpha(x,t) = \begin{cases} \langle s \rangle \langle p \rangle f_p(t) & \text{if } x = f_s(t) \text{ and } f_p(t) \in \mathbf{X} \\ \langle s \rangle \langle o \rangle f_o(t) & \text{if } x = f_s(t) \text{ and } f_o(t) \in \mathbf{X} \\ \langle \bar{p} \rangle \langle \bar{s} \rangle f_s(t) & \text{if } x = f_p(t) \text{ and } f_s(t) \in \mathbf{X} \\ \langle \bar{o} \rangle \langle o \rangle f_o(t) & \text{if } x = f_p(t) \text{ and } f_o(t) \in \mathbf{X} \\ \langle \bar{o} \rangle \langle p \rangle f_p(t) & \text{if } x = f_o(t) \text{ and } f_p(t) \in \mathbf{X} \\ \langle \bar{o} \rangle \langle \bar{s} \rangle f_s(t) & \text{if } x = f_o(t) \text{ and } f_s(t) \in \mathbf{X} \end{cases}$$

  Note that there is an exponential number of m_i's in terms of the number of non-distinguished variables. More precisely, there are at most $O(n^k)$ mappings, where n is the number of triples where non-distinguished variables appear, and k is the number of non-distinguished variables.

- Finally, the function $\mathcal{A}$ uses m to encode the query inductively:

$$\begin{aligned}
\mathcal{A}(q, m) &= \bigvee_{i=1}^{|m|} \mathcal{A}(q, m_i) \\
\mathcal{A}((x, y, z), m) &= \mathit{lfp}\big(X, \langle \bar{s} \rangle d(m, x) \wedge \langle p \rangle d(m, y) \wedge \langle o \rangle d(m, z)\big) \\
\mathcal{A}(q_1 \text{ AND } q_2, m) &= \mathcal{A}(q_1, m) \wedge \mathcal{A}(q_2, m) \\
\mathcal{A}(q_1 \text{ UNION } q_2, m) &= \mathcal{A}(q_1, m) \vee \mathcal{A}(q_2, m) \\
d(m, x) &= \begin{cases} \varphi & \text{if } (x \mapsto \varphi) \in m \\ \top & \text{if } \mathit{unique}(x) \\ x & \text{otherwise} \end{cases}
\end{aligned}$$

The disjuncts in the encoding guarantee that possible set of substitutions m capture the intended semantics of a cyclic query that contains distinguished variables and constants in its cyclic component.

Example 17 (SPARQL query encoding)**.** *Consider the encoding of $q_1 \sqsubseteq q_2$ of Example 8. To encode q_1, we freeze the variables and constants and proceed with $\mathcal{A}$ such that $\mathcal{A}(q_1) =$*

$$\begin{aligned}
&\big(\mathit{lfp}\big(X, \langle \bar{s} \rangle x \wedge \langle p \rangle \mathit{translated} \wedge \langle o \rangle l\big) \\
&\quad \vee \mathit{lfp}\big(X, \langle \bar{s} \rangle x \wedge \langle p \rangle \mathit{wrote} \wedge \langle o \rangle l\big)\big) \wedge \\
&\mathit{lfp}\big(X, \langle \bar{s} \rangle l \wedge \langle p \rangle \mathit{type} \wedge \langle o \rangle \mathit{Poem}\big)
\end{aligned}$$

To encode q_2, one first computes $m = \{m_1, \ldots, m_n\}$. Given a multiply appearing non-distinguished variable $l \in \mathit{ndvar}(q_2)$, we produce m as follows:

$$\begin{aligned}
m_1 &= \{l \mapsto \alpha(l, (x, \mathit{translated}, l))\} = \{l \mapsto \langle \bar{o} \rangle \langle p \rangle \mathit{translated}\} \\
m_2 &= \{l \mapsto \alpha(l, (l, \mathit{type}, \mathit{Poem}))\} = \{l \mapsto \langle s \rangle \langle p \rangle \mathit{type}\} \\
m_3 &= \{l \mapsto \alpha(l, (x, \mathit{wrote}, l))\} = \{l \mapsto \langle \bar{o} \rangle \langle p \rangle \mathit{wrote}\}
\end{aligned}$$

Using $m = \{m_1, m_2, m_3\}$, the encoding of q_2 is produced inductively:

$$\begin{aligned}
\mathcal{A}(q_2, m) = \bigvee_{i=1}^{|m|} \mathcal{A}(q_2, m_i) &= \mathcal{A}(q_2, m_1) \vee \mathcal{A}(q_2, m_2) \vee \mathcal{A}(q_2, m_3) \\
&= \big(\mathit{lfp}\big(X, \langle\bar{s}\rangle x \wedge \langle p\rangle \mathit{translated} \wedge \langle o\rangle\langle\bar{o}\rangle\langle p\rangle \mathit{translated}\big) \\
&\quad \wedge \mathit{lfp}\big(X, \langle\bar{s}\rangle\langle\bar{o}\rangle\langle p\rangle \mathit{translated} \wedge \langle p\rangle \mathit{type} \wedge \langle o\rangle \mathit{Poem}\big) \\
&\quad \vee \mathit{lfp}\big(X, \langle\bar{s}\rangle x \wedge \langle p\rangle \mathit{wrote} \wedge \langle o\rangle\langle\bar{o}\rangle\langle p\rangle \mathit{translated}\big)\big) \vee \\
&\quad \big(\mathit{lfp}\big(X, \langle\bar{s}\rangle x \wedge \langle p\rangle \mathit{translated} \wedge \langle o\rangle\langle s\rangle\langle p\rangle \mathit{type}\big) \\
&\quad \wedge \mathit{lfp}\big(X, \langle\bar{s}\rangle\langle s\rangle\langle p\rangle \mathit{type} \wedge \langle p\rangle \mathit{type} \wedge \langle o\rangle \mathit{Poem}\big) \\
&\quad \vee \mathit{lfp}\big(X, \langle\bar{s}\rangle x \wedge \langle p\rangle \mathit{wrote} \wedge \langle o\rangle\langle s\rangle\langle p\rangle \mathit{type}\big)\big) \vee \\
&\quad \big(\mathit{lfp}\big(X, \langle\bar{s}\rangle x \wedge \langle p\rangle \mathit{translated} \wedge \langle o\rangle\langle\bar{o}\rangle\langle p\rangle \mathit{wrote}\big) \\
&\quad \wedge \mathit{lfp}\big(X, \langle\bar{s}\rangle\langle\bar{o}\rangle\langle p\rangle \mathit{wrote} \wedge \langle p\rangle \mathit{type} \wedge \langle o\rangle \mathit{Poem}\big) \\
&\quad \vee \mathit{lfp}\big(X, \langle\bar{s}\rangle x \wedge \langle p\rangle \mathit{wrote} \wedge \langle o\rangle\langle\bar{o}\rangle\langle p\rangle \mathit{wrote}\big)\big)
\end{aligned}$$

So far we offered various functions to produce formulas corresponding to the encodings of queries and schema axioms. Hence, the problem of containment under a schema can be reduced to formula unsatisfiability in the μ-calculus as:

$$q \sqsubseteq_{\mathcal{C}} q' \Leftrightarrow \eta(\mathcal{C}) \wedge \mathcal{A}(q) \wedge \neg\mathcal{A}(q', m) \wedge \varphi_r \text{ is unsatisfiable.}$$

For the sake of legibility in writing, we use $\Phi(\mathcal{C}, q, q')$ to denote $\eta(\mathcal{C}) \wedge \mathcal{A}(q) \wedge \neg\mathcal{A}(q', m) \wedge \varphi_r$.

3.2.2 Reducing Containment to Unsatisfiability

We prove the correctness of reducing query containment to unsatisfiability test.

Lemma 1. *Given a set of $\mathcal{ALCH}$ schema axioms $\mathcal{C}$, $\mathcal{C}$ has a model iff $\eta(\mathcal{C})$ is satisfiable.*

Proof. ($\Rightarrow$) assume that there exists a model $\mathcal{I} = (\Delta^{\mathcal{I}}, .^{\mathcal{I}})$ of $\mathcal{C}$ such that $\mathcal{I} \models \mathcal{C}$. We build a restricted transition system $K = (S, R, L)$ from $\mathcal{I}$ using the following:

- for each element of the domain $e \in \Delta^{\mathcal{I}}$, we create a node $n^e \in S'$,
- for each atomic concept A, if $a \in A^{\mathcal{I}}$, then $(n^a, t) \in R(s), (t, n^{type}) \in R(p)$, $(t, n^A) \in R(o)$, $L(type), = n^{type}, L(A) = n^A$ and $L(a) = n^a$ where $t \in S''$,
- for each atomic role R, if $(x, y) \in R^{\mathcal{I}}$, then $(n^x, t) \in R(s)$, $(t, n^R) \in R(p)$, and $(t, n^y) \in R(o)$ such that $n^x, n^y, n^R \in S'$, $t \in S''$, and $L(x) = n^x$, $L(R) = n^R$, $L(y) = n^y$,
- $S = S' \cup S''$

To show that $\eta(\mathcal{C})$ is satisfiable in K. We proceed inductively on the construction of the formula. Since the axioms, $\{c_1, \ldots, c_n\}$, are made of role or concept inclusions or transitivity, we consider the following cases:

– when $\eta(c_i) = gfp\big(X, \omega(C_1) \Rightarrow \omega(C_2)\big)$. Since $C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$, we get that $[\![\omega(C_1)]\!]^K \subseteq [\![\omega(C_2)]\!]^K$. And hence, $\omega(C_1) \Rightarrow \omega(C_2)$ is satisfiable in K. Besides, the general recursion ν guarantees that the constraint is satisfied in each state of the transition system. Therefore, $\eta(c_i)$ is satisfiable.

– when $\eta(c_i) = gfp\big(X, \omega(r_1) \Rightarrow \omega(r_2)\big)$. From $r_1^{\mathcal{I}} \subseteq r_2^{\mathcal{I}}$ we have that $\exists n^{r_1} \in L(r_1)$ implies $\exists n^{r_2} \in L(r_2)$ in K. Thus, $\exists s \in [\![\omega(r_1) \Rightarrow \omega(r_2)]\!]^K$. As K is a construction of $\mathcal{I}$, $\eta(c_i)$ is satisfiable in K.

Since K is a model of each $\eta(c_i)$, then $\eta(\mathcal{C})$ is satisfiable.
($\Leftarrow$) consider a transition system model K for $\eta(\mathcal{C})$. From K, we construct an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, .^{\mathcal{I}})$ and show that it is a model of $\mathcal{C}$.

- $\Delta^{\mathcal{I}} = S$, $A^{\mathcal{I}} = [\![A]\!]^K$ for each atomic concept A,
- $\top^{\mathcal{I}} = [\![\top]\!]^K$, for a top concept,
- $r^{\mathcal{I}} = \{(s, s') \mid \forall t \in [\![r]\!]^K \wedge t' \in S \wedge (s, t') \in R(s) \wedge (t', t) \in R(p) \wedge (t', s') \in R(o)\}$ for each atomic role r,

Consequently, formulas such as $gfp\big(X, \omega(r_1) \Rightarrow \omega(r_2)\big)$ and $gfp\big(X, \omega(C_1) \Rightarrow \omega(C_2)\big)$ are true in $\mathcal{I}$. The first formula expresses that there is no node in the transition system where $\omega(r_1)$ holds and $\omega(r_2)$ does not hold. This is equivalent to $\omega(r_1) \Rightarrow \omega(r_2)$ and $[\![r_1]\!]^K \subseteq [\![r_2]\!]^K$ since r_1 and r_2 are basic roles. Thus, we obtain $r_1^{\mathcal{I}} \subseteq r_2^{\mathcal{I}}$ and $\mathcal{I} \models r_1 \sqsubseteq r_2$.

On the other hand, for the latter formula from above, one can exploit its construction. Note however that, similar justifications as above can be worked out to arrive at $\mathcal{I} \models C_1 \sqsubseteq C_2$ if C_1 and C_2 are basic concepts. Nonetheless, if they are complex concepts, we proceed as below. Consider the case when $C_1 = A \sqcap B$ and $C_2 = \exists R.C$, $[\![\omega(C_1) \Rightarrow \omega(C_2)]\!]^K$

$$
\begin{aligned}
&\Leftrightarrow [\![\omega(A \sqcap B)]\!]^K \subseteq [\![\omega(\exists R.C)]\!]^K \\
&\Leftrightarrow [\![A \wedge B]\!]^K \subseteq [\![\langle s\rangle\big(\langle p\rangle R \wedge \langle o\rangle(\langle s\rangle\langle o\rangle C)\big)]\!]^K \\
&\Leftrightarrow [\![A]\!]^K \wedge [\![B]\!]^K \subseteq \{s \mid \exists s'.s \in [\![\langle s\rangle\langle p\rangle R]\!]^K \wedge s' \in [\![\langle s\rangle\langle o\rangle C]\!]^K\} \\
&\Leftrightarrow A^{\mathcal{I}} \cap B^{\mathcal{I}} \subseteq \{s \mid \exists s'.(s, s') \in R^{\mathcal{I}} \wedge s' \in C^{\mathcal{I}}\} \\
&\Leftrightarrow (A \sqcap B)^{\mathcal{I}} \subseteq (\exists R.C)^{\mathcal{I}} \\
&\Leftrightarrow \mathcal{I} \models C_1 \sqsubseteq C_2
\end{aligned}
$$

Accordingly, from $\mathcal{I} \models c_1 \wedge \cdots \wedge \mathcal{I} \models c_n$, it follows that $\mathcal{I} \models \mathcal{C}$. □

Theorem 1 ([Hayes 2004]). *Given a query $q\{\overrightarrow{w}\}$, there exists an RDF graph G such that $[\![q\{\overrightarrow{w}\}]\!]_G \neq \emptyset$.*

Proof. (*Sketch*) From any query it is possible to build an homomorphic graph by collecting all triples connected by AND and only those at the left of UNION (replacing variables by blanks). This graph is consistent as all RDF graphs. It is thus a graph satisfying the query. □

Lemma 2. *Let q be a SPARQL$_{cdfc}$ query, for every restricted transition system K whose associated RDF graph is G, we have that $[\![q]\!]_G \neq \emptyset$ iff $[\![\mathcal{A}(q,m) \wedge \varphi_r]\!]^K \neq \emptyset$.*

Proof. ($\Rightarrow$) Assume that $[\![q]\!]_G \neq \emptyset$ and consider that G is a canonical instance of q (cf. Theorem 1). Using G, we construct a restricted transition system $\sigma(G) = (S, R, L)$ in the same way as it is done in Definition 9. To prove that $\sigma(G)$ is a model of $\mathcal{A}(q,m)$, we consider two cases:

(i) when q is cyclic-free, and

(ii) when q contains a cyclic component among its non-distinguished variables which is not distinguished variable-free or constant-free

Firstly, (i) if q is cycle-free, then encoding the non-distinguished variables with $\top$ suffices to justify that $\sigma(G)$ is a model of its encoding.

Secondly, (ii) let us consider when q is cyclic, in this case, its encoding is $\bigvee_{i=1}^{|m|} \mathcal{A}(q, m_i)$. This disjunctive formula can encode multiply occurring non-distinguished variables using the distinguished variables and constants of q. Henceforth, creating a formula that is satisfiable in cyclic models.

It can be verified that $\sigma(G)$ is a model for one of the disjuncts $\mathcal{A}(q, m_i)$, this is because nominals encoding the constants and distinguished variables are true in $\sigma(G)$ as they exist already in G. In addition, the formulae encoding the non-distinguished variables are satisfied in $\sigma(G)$, since these variables are encoded in terms of the distinguished variables and constants of the query which are already shown to be true in $\sigma(G)$. Therefore, $\mathcal{A}(q,m)$ is satisfiable in $\sigma(G)$. To elaborate, if $(x, y, z) \in q$ and l is either x or y or z,

- for l either a distinguished variable or constant, the translation of l is satisfiable in $\sigma(G)$ since $[\![l]\!]^{\sigma(G)} \in L(l)$,
- for l a uniquely appearing non-distinguished variable, the translation of l is true in $\sigma(G)$ since its encoding $\top$ is true everywhere in the transition system,
- the translation l of a multiply occurring non-distinguished variable is true in $\sigma(G)$ since $\exists t \in S'.\, t \in L(c) \wedge t \in [\![c]\!]^{\sigma(G)}$, where c is a constant or distinguished variable in the triple (x, y, z).

Thus, since $\sigma(G)$ is a restricted transition system, we obtain that $[\![\mathcal{A}(q,m)\varphi_r]\!]_G \neq \emptyset$. ($\Leftarrow$) Assume that $[\![\mathcal{A}(q,m) \wedge \varphi_r]\!]^K \neq \emptyset$. From this we can derive that, $K = (S, R, L)$ is restricted transition system that satisfies $\mathcal{A}(q,m) \wedge \varphi_r$. We build an RDF graph G from K as follows:

- $\forall s_1, s_2, s_3 \in S' \wedge t \in S''.(s_1, t) \in R(s) \wedge (t, s_2) \in R(p) \wedge (t, s_3) \in R(o)$ and for each triple $t_i = (x_i, y_i, z_i) \in q$ if $s_1 \in L(x_i) \wedge s_2 \in L(y_i) \wedge s_3 \in L(z_i) \wedge z_i \in [\![\top]\!]^K$, then $(x_i, y_i, z_i) \in G$. This case holds if x_i, y_i and z_i are either distinguished variables or constants. Note here that if x_i or y_i or z_i appear in another triple

$t_j = (x_j, y_j, z_j) \in q$, then the equivalent item in t_j is replaced with the value of the corresponding entry in t_i.

- $\forall s_1, s_2, s_3 \in S' \wedge t \in S''.(s_1, t) \in R(s) \wedge (t, s_2) \in R(p) \wedge (t, s_3) \in R(o)$ and for each triple $t_i = (x_i, y_i, z_i) \in q$ if $s_1 \in L(x_i) \wedge s_2 \in L(y_i)$, then $(x_i, y_i, c_i) \in G$ where c_i is a fresh constant. This case holds if z_i is a non-distinguished variable. Similarly, the case when x_i or y_i or both are variables can be worked out.

- $\forall s_1, s_2, s_3 \in S' \wedge t \in S''.(s_1, t) \in R(s) \wedge (t, s_2) \in R(p) \wedge (t, s_3) \in R(o)$ and for each triple $t_i = (x_i, y_i, z_i) \in q$ and x_i is a non-distinguished variable that appears in a cycle and if $s_1 \in [\![\top]\!]^K \wedge s_2 \in L(y_i) \wedge s_3 \in L(z_i)$, then $(c_i, y_i, z_i) \in G$. Where c_i is a fresh constant and all such occurrences of the variable x_i in other triples are replaced by c_i's.

Since G is a technical construction obtained from a restricted transition system that associated with q, then it holds that $[\![q]\!]_G \neq \emptyset$. □

In the following, for the sake of legibility, we denote $\eta(\mathcal{C}) \wedge \mathcal{A}(q_1) \wedge \neg\mathcal{A}(q_2, m) \wedge \varphi_r$ by $\Phi(\mathcal{C}, q_1, q_2)$.

Theorem 2 (Soundness)**.** *Given a SPARQL query $q_1\{\overrightarrow{w}\}$, a SPARQL$_{cdfc}$ query $q_2\{\overrightarrow{w}\}$, and a set of $\mathcal{ALCH}$ axioms $\mathcal{C}$, if $\Phi(\mathcal{C}, q_1, q_2)$ is unsatisfiable, then $q_1\{\overrightarrow{w}\} \sqsubseteq_{\mathcal{C}} q_2\{\overrightarrow{w}\}$.*

Proof. We show the contrapositive. If $q_1 \not\sqsubseteq_{\mathcal{C}} q_2$, then $\Phi(\mathcal{C}, q_1, q_2)$ is satisfiable. One can verify that every model G of $\mathcal{C}$ in which there is at least one tuple satisfying q_1 but not q_2 can be turned into a transition system model for $\Phi(\mathcal{C}, q_1, q_2)$. To do so, consider a graph G that satisfies schema axioms $\mathcal{C}$. Assume also that there is a tuple $\overrightarrow{a} \in [\![q_1]\!]_G$ and $\overrightarrow{a} \notin [\![q_2]\!]_G$. Let us construct a transition system K from G. From Lemma 1, we obtain that $[\![\eta(\mathcal{C})]\!]^K \neq \emptyset$. Further, since K is a restricted transition system (cf. Definition 9), $[\![\varphi_r]\!]^K \neq \emptyset$. At this point, it remains to verify that $[\![\mathcal{A}(q_1)]\!]^K \neq \emptyset$ and $[\![\mathcal{A}(q_2, m)]\!]^K = \emptyset$.

Let us construct the formulas $\mathcal{A}(q_1)$ and $\mathcal{A}(q_2, m)$ by first skolemizing the distinguished variables using the answer tuple $\overrightarrow{a}$. Consequently, from Lemma 2 one obtains, $[\![\mathcal{A}(q_1)]\!]^K \neq \emptyset$. However, $[\![\mathcal{A}(q_2, m)]\!]^K = \emptyset$, this is because the nominals in the formula corresponding to the constants and non-distinguished variables are not satisfied in K. This implies that $[\![\neg\mathcal{A}(q_2, m)]\!]^K \neq \emptyset$. This is justified by the fact that if a formula φ is satisfiable in a restricted transition system, then $[\![\varphi]\!]^K = S$ thus $[\![\neg\varphi]\!]^K = \emptyset$. So far we have: $[\![\eta(\mathcal{C})]\!]^K \neq \emptyset$ and $[\![\varphi_r]\!]^K \neq \emptyset$ and $[\![\mathcal{A}(q_1)]\!]^K \neq \emptyset$ and $[\![\neg\mathcal{A}(q_2, m)]\!]^K \neq \emptyset$. Without loss of generality, $[\![\Phi(\mathcal{C}, q_1, q_2)]\!]^K \neq \emptyset$. Therefore, $\Phi(\mathcal{C}, q_1, q_2)$ is satisfiable. □

Theorem 3 (Completeness)**.** *Given a SPARQL query $q_1\{\overrightarrow{w}\}$, a SPARQL$_{cdfc}$ query $q_2\{\overrightarrow{w}\}$, and a set of $\mathcal{ALCH}$ axioms $\mathcal{C}$, if $\Phi(\mathcal{C}, q_1, q_2)$ is satisfiable, then $q_1\{\overrightarrow{w}\} \not\sqsubseteq_{\mathcal{C}} q_2\{\overrightarrow{w}\}$.*

Proof. $\Phi(\mathcal{C}, q, q')$ is satisfiable $\Rightarrow \exists K.[\![\Phi(\mathcal{C}, q, q')]\!]^K \neq \emptyset$. Consequently, K is a restricted transition system due to $[\![\varphi_r]\!]^K \neq \emptyset$ (cf. Proposition 1). Using $K = (S' \cup S'', R, L)$ we construct a model $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ of $\mathcal{C}$ such that $q \not\sqsubseteq q'$ holds:

- $\Delta^{\mathcal{I}} = S'$, $A^{\mathcal{I}} = [\![A]\!]^K$ for each atomic concept A,
- $\top^{\mathcal{I}} = [\![\top]\!]^K$, for a top concept,
- $r^{\mathcal{I}} = \{(s, s') \mid \forall t \in [\![r]\!]^K \wedge t' \in S'' \wedge (s, t') \in R(s) \wedge (t', t) \in R(p) \wedge (t', s') \in R(o)\}$ for each atomic role r,
- for each constant c in q and q', $c^{\mathcal{I}} = [\![c]\!]^K$,
- for each distinguished and non-distinguished variable v in q, $v^{\mathcal{I}} = [\![v]\!]^K$, and
- for each distinguished variable v in q', $v^{\mathcal{I}} = [\![v]\!]^K$.

One can utilize Lemma 1 to verify that indeed $\mathcal{I}$ is a model of $\mathcal{C}$. Thus, it remains to show that $[\![q]\!]_{\mathcal{I}} \not\subseteq [\![q']\!]_{\mathcal{I}}$. From our assumption, one anticipates the following:

$$\begin{aligned}[\![\mathcal{A}(q) \wedge \neg\mathcal{A}(q')]\!]^K \neq \emptyset &\Rightarrow [\![\mathcal{A}(q)]\!]^K \neq \emptyset \text{ and } [\![\neg\mathcal{A}(q', m)]\!]^K \neq \emptyset \\ &\Rightarrow [\![\mathcal{A}(q)]\!]^K \neq \emptyset \text{ and } [\![\mathcal{A}(q', m)]\!]^K = \emptyset\end{aligned}$$

Note here that, if a formula φ is satisfiable in a restricted transition system K_r, then $[\![\varphi]\!]^{K_r} = S$. We use a function f to construct an RDF graph G from the interpretation $\mathcal{I}$. f uses assertions in $\mathcal{I}$ to form triples:

$$\begin{aligned}f(a \in A^{\mathcal{I}}) &= (a, \texttt{type}, A) \in G \\ f((a, b) \in r^{\mathcal{I}}) &= (a, r, b) \in G \\ f((a, b) \in (r^-)^{\mathcal{I}}) &= (b, r, a) \in G \\ f((x, y, z)) &= (x, y, z) \in G,\ \forall (x, y, z) \in q\end{aligned}$$

As a consequence, $[\![q]\!]_G \neq \emptyset$ and $[\![q']\!]_G = \emptyset$ because G contains all those triples that satisfy q and not q'. Therefore, we get $[\![q]\!]_G \not\subseteq [\![q']\!]_G$. Fundamentally, there are two issues to be addressed (i) when q' is not cyclic and (ii) when q' contains a cycle. (i) if there are no cycles in q', then replacing non-distinguished variables with $\top$ suffices (cf. the proof of Lemma 2). On the other hand, (ii) can be dealt with nominals, i.e., since cycles can be expressed by a formula in a μ-calculus extended with nominals and inverse, cyclic queries can be encoded by such a formula. Hence, the constraints expressed by $\neg\mathcal{A}(q', m)$ are satisfied in a transition system containing cycles □

3.2.3 Complexity

In the following, we establish the complexity of the containment problem under schema axioms. The schema axioms can be formed using the fragments of $\mathcal{ALCH}$. The expressiveness of the schema language is limited as such due to the expressive power of the logic used for the encoding: μ-calculus with nominals and converse becomes undecidable when extended with graded modalities [Bonatti *et al.* 2006].

Proposition 2 (Query satisfiability). *Given an $\mathcal{ALCH}$ schema $\mathcal{C}$ and a query q, the complexity of satisfiability of q with respect to $\mathcal{C}$ is $2^{\mathcal{O}(|\mathcal{C}|+|q|)}$.*

Proposition 3. *SPARQL query containment under the fragments of $\mathcal{ALCH}$ schema axioms can be determined in a time of $2^{\mathcal{O}(n^2 log\ n)}$ where $n = \mathcal{O}(|\eta(\mathcal{C})|+|\mathcal{A}(q_1)|+|\mathcal{A}(q_2)|)$ is the size of the formula, and $\eta(\mathcal{C})$, $\mathcal{A}(q_1)$ and $\mathcal{A}(q_2)$ denote the encodings of schema axioms $\mathcal{C}$, and queries q_1 and q_2.*

Note that due to duplication in the encoding of q_2, the size of $|\mathcal{A}(q_2)|$ is exponential in terms of the non-distinguished variables that appear in cycles in the query. Hence, we obtain a 2ExpTime upper bound for containment. As pointed out in [Calvanese *et al.* 2008], the problem is solvable in ExpTime if there is no cycle on the right hand side query. This complexity is a lower bound due to the complexity of satisfiability in μ-calculus which is $2^{\mathcal{O}(n^2 log\ n)}$ [Sattler & Vardi 2001, Tanabe *et al.* 2008].

3.3 Conclusion

We have introduced a mapping from RDF graphs into transition systems and the encodings of queries and schema axioms in the μ-calculus. We proved that this encoding is correct and can be used for checking query containment. We have provided implementable algorithms, as a consequence, this work opens a way to use available implementations of μ-calculus satisfiability solvers from [Genevès *et al.* 2007] and [Tanabe *et al.* 2008] as already done in Chapter 7 of this thesis. Beyond this, we have established a double exponential upper bound for containment test under $\mathcal{ALCH}$ axioms. The presented encoding is sound (in the sense that whenever the algorithm detects that q_1 is contained in q_2, the containment holds), however it is not complete (i.e., it may happen that the algorithm fails to detect an existing containment relationship) if q_2 is not a SPARQL$_{cdfc}$ query (Definition 13). Interestingly, the cases where the algorithm might be incomplete can be detected, since those cases correspond to the queries of type SPARQL$_{cdfc}$, hence the user can be warned about such a potential risk. These cases have already been dealt with in Chapter 6.

Here, we would like to emphasize that, in addition to the complexity bound we provide, no implementation has been reported in previous works.

Chapter 4

Containment of Path SPARQL Queries

Contents

Regular path queries (RPQs) are useful for expressing complex navigations in a graph. In particular, union and transitive closure are crucial when one does not have a complete knowledge of the structure of the knowledge base. SPARQL 1.0 lacks recursion mechanism and supports a simple form of RPQs however its extensions such as PSPARQL [Alkhateeb *et al.* 2009] and its successor SPARQL1.1 support this feature. PSPARQL (Path SPARQL) allows querying of arbitrary length paths by using regular expression patterns.

Conjunctive RPQs have been studied in [Florescu *et al.* 1998] where an EXPSPACE algorithm for query containment is proved. Containment of (two-way) regular path queries (2RPQs) and their extensions have also been studied [Calvanese *et al.* 2000, Calvanese *et al.* 2003, Barceló *et al.* 2010]. Most recently, containment of RPQs under description logic constraints have been studied in [Calvanese *et al.* 2011], and it has been shown that this problem is 2ExpTime-complete.

Most notably and closely related results on query containment come from the study of regular path queries (RPQs) [Calvanese *et al.* 2000]. The difference between [Calvanese *et al.* 2000] and our work lies in the features supported by the languages. While RPQs in [Calvanese *et al.* 2000] support backward navigation and conjunction, PSPARQL supports variables in paths and union. Since the logic adopted for this study allows backward navigation, inverted paths can easily be supported. Further, using our approach, we can deal with containment of PSPARQL queries under expressive description logic axioms.

In this chapter, we show how to translate PSPARQL queries into μ-calculus formulas and then reduce containment into unsatisfiability test. We prove that the reduction is

sound and complete when the right-hand side query is of type PSPARQL$_{cdfc}$ (refer to Definition 14), and establish complexity bounds for the problem. Towards the end of this chapter, we propose how to decide containment in the presence of $\mathcal{ALCH}$ axioms, using their encoding from Chapter 3.

4.1 PSPARQL Containment

In this section, we show how to reduce the containment of PSPARQL queries into μ-calculus formula unsatisfiability test. In doing so, first, we introduce a function that translates the queries inductively into formulas in the aforementioned logic. Second, the encodings (a.k.a formulas) are tested for satisfiability in order to determine containment.

4.1.1 Encoding PSPARQL Queries

In this section, we show how to encode queries as μ-calculus formulas. Then, in Chapter 5, we use these encodings to test query containment under the RDFS and OWL entailment regimes. Before discussing the encoding procedure, we briefly assess the issue of blank nodes. Blank nodes are existential variables that denote the existence of unnamed resources. Their definition matches the definition of non-distinguished variables in a query. Thus, blank nodes in the queries can be considered as non-distinguished variables. As a result, every occurrence of a blank node in the query is replaced by a fresh variable.

The encoding of PSPARQL queries is similar to that of SPARQL queries, the only difference lies on the encoding of regular expression patterns that appear in the predicate positions of the triple patterns. As in Chapter 3, we give separate encodings of the queries on the left and right-hand sides of $\sqsubseteq$. We consider the left-hand side to be a PSPARQL query whereas the right-hand side query must be a PSPARQL$_{cdfc}$ query. This is because we do not address PSPARQL queries that contain constant and distinguished variable free cyclic component as discussed in 13.

Encoding left-hand side query: the encoding of the left-hand side query is similar to that of its counterpart for SPARQL. The only difference lies on the encoding of regular expression patterns as shown below:

$$\begin{aligned}
\mathcal{A}\big((x, e, z)\big) &= \mathit{lfp}\big(X, \langle \bar{s} \rangle x \wedge \mathcal{R}(e, z)\big) \\
\mathcal{A}\big((x, e^*, z)\big) &= (x \Leftrightarrow z) \vee \mathit{lfp}\big(X, \langle \bar{s} \rangle x \wedge \mathcal{R}(e^+, z)\big) \\
\mathcal{A}(q_1 \text{ AND } q_2) &= \mathcal{A}(q_1) \wedge \mathcal{A}(q_2) \\
\mathcal{A}(q_1 \text{ UNION } q_2) &= \mathcal{A}(q_1) \vee \mathcal{A}(q_2)
\end{aligned}$$

Regular expression patterns that appear in the query are encoded using the function $\mathcal{R}$. This function takes two arguments (the predicate which is a regular expression

pattern and the object of a triple).

$$\begin{aligned}
\mathcal{R}(uri, y) &= \langle p\rangle uri \ \wedge \langle o\rangle y \\
\mathcal{R}(x, y) &= \langle p\rangle x \wedge \langle o\rangle y \\
\mathcal{R}(e \dashv e', y) &= (\mathcal{R}(e, y) \vee \mathcal{R}(e', y)) \\
\mathcal{R}(e \cdot e', y) &= \mathcal{R}(e, \langle s\rangle \mathcal{R}(e', y)) \\
\mathcal{R}(e^{+}, y) &= \mu X.\mathcal{R}(e, y) \vee \mathcal{R}(e, \langle s\rangle X) \\
\mathcal{R}(e^{*}, y) &= \mathcal{R}(e^{+}, y) \vee \langle \bar{s}\rangle y
\end{aligned}$$

Example 18 (Encoding Kleene star *)**.** *Consider the following query:*

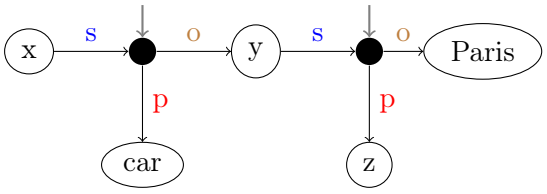
$$\begin{aligned}
q\{x, y\} &= (x, z, y)(z, sp^{*}, ancestor) \\
\mathcal{A}(q) &= lfp(X, \langle \bar{s}\rangle x \wedge \langle p\rangle z \wedge \langle o\rangle y) \ \wedge (z \Leftrightarrow ancestor \\
&\quad \vee lfp\big(X, \mu Y.\mathcal{R}(sp, ancestor) \vee \mathcal{R}(sp, \langle s\rangle Y\big) \\
&= lfp(X, \langle \bar{s}\rangle x \wedge \langle p\rangle z \wedge \langle o\rangle y) \ \wedge (z \Leftrightarrow ancestor \\
&\quad \vee lfp\big(X, \mu Y.\langle p\rangle sp \wedge \langle o\rangle ancestor \vee \langle p\rangle sp \wedge \langle s\rangle\langle o\rangle Y\big))
\end{aligned}$$

In order to encode the right-hand side query, we use the notion of cyclic queries discussed in Chapter 3. The only difference in the transitions systems, obtained from SPARQL and PSPARQL queries to determine cyclicness, is that: for SPARQL queries, nodes of the transition system are either variables or constants (URIs) whereas in the case of PSPARQL, nodes can also be regular expression patterns. As for SPARQL queries (Definition 13), we identify a fragment of union of conjunctive PSPARQL queries that contain a cyclic component among its non-distinugished variables which constant and distinguished variable-free. We refer to these kinds of queries as PSPARQL$_{cdfc}$.

Definition 14. *A PSPARQL query that contains a cyclic component which is constant and distinguished variable-free is called PSPARQL$_{cdfc}$.*

Encoding right-hand side query: the distinguished variables and constants are encoded as nominals whereas the non-distinguished variables are encoded as:

- the non-distinugished variables appearing only once are encoded as $\top$.
- before we show how multiply appearing non-distinguished variables are encoded, we present a motivating example. Consider the SPARQL query $q\{x\} = (x, car, y)$ AND $(y, z, Paris)$, which has one distinguished variable x and two non-distinguished variables y and z. As shown in Chapter 3, z can be encoded as $\top$ whereas to encode y, we produce its encodings using the nominals corresponding to the encodings of the distinguished variables and constants in q. We obtain two possible subformulas for y, $\{y \rightarrow \langle \bar{o}\rangle\langle p\rangle car, y \rightarrow \langle s\rangle\langle o\rangle Paris\}$, using these formulas we can construct the encoding for q. A restricted transition system that corresponds to the query q is shown below:



Now, think of the same way of encoding multiply appearing non-distinguished variables in PSPARQL queries. For instance, assume that we are encoding the PSPARQL query $q\{x\} = (x, car \cdot (train \mid bus)^+, y)$ AND (y, z, r). In this query, x is a distinguished variable and $y, z,$ and r are non-distinguished variables. z and r can be encoded as $\top$ whereas y cannot be encoded as $\top$. We cannot simply introduce fresh nominals n_1 and n_2 for the respective triples in the query, due to the fact that the triple pattern $(x, car \cdot (train \mid bus)^+, y)$ is composed of finitely many triples. Therefore, in order to address this, instead of introducing a fresh nominal, we can create a formula from the other terms (constants and distinguished variables) of the triple pattern. If the predicate of the triple pattern is a variable, we can encode it with respect to the constants and distinguished variables of the query. Thus, the encoding of y is: $\{y \to (\langle\bar{o}\rangle train \vee \langle\bar{o}\rangle bus)\}$. Thus, using all these encodings, the formula associated to q can be constructed inductively. In general, if a non-distinguished variable appears multiple times, then one performs the subsequent steps. Note that the mathematical notations used here have been introduced in Section 3.2.1.

1. If a non-distinguished variable appears multiple times in q', then we produce a set of mappings $m = \{m_1, \ldots, m_n\}$ such that each m_i contains formula assignments to the non-distinguished variables of the query.

2. for each multiply occurring non-distinguished variable x_l, given that $\{x_1, \ldots, x_k\} \in ndvar(q')$, assign it one of the triple patterns $t_j \in q'$ where it appears in, i.e., x_l appears in the triple pattern t_j, from that we obtain m_i's as:

$$m_i = \bigcup_{l=1}^{k} \{x_l \mapsto \alpha(x_l, t_j) \mid x_l \in t_j\}$$

$$\alpha(x,t) = \begin{cases} \varphi(s', f_p(t)) & \text{if } x = f_s(t) \text{ and } f_p(t) \in \mathbf{X} \\ \langle s\rangle\langle o\rangle f_o(t) & \text{if } x = f_s(t), f_o(t) \in \mathbf{X} \text{ and } f_p(t) \notin \mathbf{X} \\ \langle\bar{p}\rangle\langle\bar{s}\rangle f_s(t) & \text{if } x = f_p(t) \text{ and } f_s(t) \in \mathbf{X} \\ \langle\bar{o}\rangle\langle o\rangle f_o(t) & \text{if } x = f_p(t) \text{ and } f_o(t) \in \mathbf{X} \\ \varphi(o', f_p(t)) & \text{if } x = f_o(t) \text{ and } f_p(t) \in \mathbf{X} \\ \langle\bar{o}\rangle\langle\bar{s}\rangle f_s(t) & \text{if } x = f_o(t), f_s(t) \in \mathbf{X} \text{ and } f_p(t) \notin \mathbf{X} \end{cases}$$

where in the function φ, s' and o' denote subject, and object, of the triple

pattern t_j. φ is defined as:

$$\begin{aligned}
\varphi(s, a) &= \langle s\rangle\langle p\rangle a & \varphi(o, a) &= \langle \bar{o}\rangle\langle p\rangle a \\
\varphi(s, a \cdot b) &= \varphi(s, a) & \varphi(o, a.b) &= \varphi(o, b) \\
\varphi(s, a \shortmid b) &= \big(\varphi(s, a) \vee \varphi(s, b)\big) & \varphi(o, a \shortmid b) &= \big(\varphi(o, a) \vee \varphi(o, b)\big) \\
\varphi(s, a^+) &= \varphi(s, a) & \varphi(o, a^+) &= \varphi(o, a) \\
\varphi(s, a^*) &= \varphi(s, a) & \varphi(o, a^*) &= \varphi(o, a)
\end{aligned}$$

- finally the function $\mathcal{A}$ works inductively on the query structure using m to generate the formula. As for the left-hand side query, $\mathcal{R}$ is used to produce the encodings of regular expressions. $\mathcal{A}$ is the same as before except that it uses the new m and $\mathcal{R}$ functions (cf. Section 3.2.1).

Example 19 (Encoding queries)**.** *Consider the encoding of $q \sqsubseteq q'$, where*

$$\begin{aligned}
q\{x, z\} &= (x, (c \shortmid d) \cdot (a \shortmid b), z) \\
q'\{x, z\} &= (x, c \shortmid d, y)\ \mathit{AND}\ (y, a \shortmid b, z)
\end{aligned}$$

- *The encoding of q is obtained by freezing the query and recursively constructing the formula using $\mathcal{A}$.*

$$\begin{aligned}
\mathcal{A}(q) &= \mathit{lfp}\big(X, \langle \bar{s}\rangle x \wedge \mathcal{R}\big((c \shortmid d) \cdot (a \shortmid b), z\big)\big) \\
&= \mathit{lfp}\big(X, \langle \bar{s}\rangle x \wedge (\langle p\rangle c \vee \langle p\rangle d) \wedge \langle o\rangle\langle s\rangle((\langle p\rangle a \vee \langle p\rangle b) \wedge \langle o\rangle z)\big)
\end{aligned}$$

- *The encoding of q' is as follows:*
  - *the constants and distinguished variables are encoded as nominals,*
  - *$y \in var(q')$ is encoded as $\varphi(o, (c \shortmid d))$, since y is an object of the triple $(x, (c \shortmid d), y)$. Hence, $m_1 = \{y \mapsto (\langle \bar{o}\rangle\langle p\rangle c \vee \langle \bar{o}\rangle\langle p\rangle d)\}$. On the other hand, y can also be encoded as $\varphi(s, (a \shortmid b))$, since y is a subject of the triple $(y, a \shortmid b, z)$. Thus, we get $m_2 = \{y \mapsto (\langle s\rangle\langle p\rangle a \vee \langle s\rangle\langle p\rangle b)\}$.*
  - *finally, we use $\mathcal{A}$ to encode q' recursively:*

$$\begin{aligned}
\mathcal{A}(q', m) &= \bigvee_{i=1}^{|m|} \mathcal{A}(q', m_i) = \mathcal{A}(q', m_1) \vee \mathcal{A}(q', m_2) \\
&= \big(\mathit{lfp}\big(X, \langle \bar{s}\rangle x \wedge (\langle p\rangle c \vee \langle p\rangle d) \wedge \langle o\rangle(\langle \bar{o}\rangle\langle p\rangle c \vee \langle \bar{o}\rangle\langle p\rangle d)\big) \\
&\quad \wedge \mathit{lfp}\big(Y, \langle \bar{s}\rangle(\langle \bar{o}\rangle\langle p\rangle c \vee \langle \bar{o}\rangle\langle p\rangle d) \wedge (\langle p\rangle a \vee \langle p\rangle b) \wedge \langle o\rangle z\big)\big) \\
&\quad \vee \\
&\quad \big(\mathit{lfp}\big(X, \langle \bar{s}\rangle x \wedge (\langle p\rangle c \vee \langle p\rangle d) \wedge \langle o\rangle(\langle s\rangle\langle p\rangle a \vee \langle s\rangle\langle p\rangle b)\big) \\
&\quad \wedge \mathit{lfp}\big(Y, \langle \bar{s}\rangle(\langle s\rangle\langle p\rangle a \vee \langle s\rangle\langle p\rangle b) \wedge (\langle p\rangle a \vee \langle p\rangle b) \wedge \langle o\rangle z\big)\big)
\end{aligned}$$

An interesting point to note is that the encodings of two equivalent SPARQL and PSPARQL queries differ syntactically but are semantically equivalent. For instance,

the following SPARQL q_s and PSPARQL q_p queries:

$$q_s\{x,y\} = (x,c,y) \text{ UNION } (x,d,y)$$
$$q_p\{x,y\} = (x,c \mid d,y)$$

produce the encodings:

$$\mathcal{A}(q_s) = lfp\big(X, \langle\bar{s}\rangle x \wedge \langle p\rangle c \wedge \langle o\rangle y\big) \vee \quad lfp\big(Y, \langle\bar{s}\rangle x \wedge \langle p\rangle d \wedge \langle o\rangle y\big)$$
$$\mathcal{A}(q_p) = lfp\big(Z, \langle\bar{s}\rangle x \wedge (\langle p\rangle c \vee \langle p\rangle d) \wedge \langle o\rangle y\big)$$

As can be seen, the two formulas $\mathcal{A}(q_s)$ and $\mathcal{A}(q_p)$ are syntactically different but semantically equivalent.

4.1.2 Reducing Containment to Unsatisfiability

We prove the correctness of the encoding procedure as shown in Lemma 2. Once correct encodings of queries have been produced, the next step requires reducing containment into the validity problem in the μ-calculus. Intuitively, $q \sqsubseteq q'$ is reduced to the validity test of $\mathcal{A}(q) \wedge \neg\mathcal{A}(q', m) \wedge \varphi_r$. In doing so, we prove the soundness and completeness of the reduction. To give an overview of the overall procedure, we start of with an example.

Example 20 (Containment test)**.** *We show the containment of the following queries: select all descendants and ancestors (q) whose names are "john" and (q′) who share the same name.*

$q\{x,z\} = (x, name, \text{“}john\text{”})$ AND $(x, ancestor^*, z)$ AND $(z, name, \text{“}john\text{”})$

$q'\{x,z\} = (x, name, y)$ AND $(x, ancestor^*, z)$ AND $(z, name, y)$

We proceed by first obtaining their encodings. Consider the encoding of $q \sqsubseteq q'$, we encode triple patterns using θ and $m = \{y \mapsto \langle\bar{o}\rangle name\}$.

$$\begin{aligned}
\mathcal{A}(q) = lfp\big(X, \theta(x, name, \text{“}john\text{”})\big) \wedge \\
lfp\big(X, \theta(x, ancestor^*, z)\big) \wedge \\
lfp\big(X, \theta(z, name, \text{“}john\text{”})\big) \\
\neg\mathcal{A}(q', m) = gfp\big(X, \neg\theta(x, name, \langle\bar{o}\rangle name)\big) \vee \\
gfp\big(X, \neg\theta(x, ancestor^*, z)\big) \vee \\
gfp\big(X, \neg\theta(z, name, \langle\bar{o}\rangle name)\big)
\end{aligned}$$

The formula $\mathcal{A}(q) \wedge \neg\mathcal{A}(q', m)$ is unsatisfiable because $\mathcal{A}(q)$ requires its model to satisfy the encoding of each triple pattern somewhere in the transition system. On the contrary, the formula $\neg\mathcal{A}(q', m)$ requests this model to satisfy the negation of the encoding of the triples in the entire transition system. Hence, this leads to a contradiction and no such model exists for the formula. Therefore, $q \sqsubseteq q'$. On the other hand, it can be verified similarly that $q' \not\sqsubseteq q$.

Lemma 3. *Given a PSPARQL query $q\{\overrightarrow{w}\}$, there exists an RDF graph G such that $[\![q\{\overrightarrow{w}\}]\!]_G \neq \emptyset$.*

Proof. (*Sketch*) From any query it is possible to build an homomorphic graph by collecting all triples connected by AND and only those at the left of UNION (replacing variables by blanks). For the triples that contain regular expression patterns instead of variables, we can easily use a function g to rewrite the query as:

$$
\begin{aligned}
g((x,y,z)) &= (x,y,z)\\
g((x,e,z)) &= (x,e,z)\\
g((x,e_1 \cdot e_2,z)) &= g((x,e_1,y)) \text{ AND } g((y,e_2,z)) \text{ where } y \text{ is a fresh variable}\\
g((x,e_1 \mid e_2,z)) &= g((x,e_1,y)) \text{ UNION } g((y,e_2,z)) \text{ where } y \text{ is a fresh variable}\\
g((x,e^+,z)) &= g((x,e,y_1)) \text{ AND } g((y_1,e,y_2)) \text{ AND } \cdots \text{ AND } g((y_i,e,z))\\
&\quad \text{such that each } y_i \text{ is a fresh variable where } 1 \leq i \leq n\\
&\quad \text{and } n \geq 1. \text{ If } n = 1, \text{ then } y_i = z\\
g((x,e^*,z)) &= g((x,\varepsilon,z)) \text{ UNION } g((x,e^+,z))
\end{aligned}
$$

The graph obtained in such way is consistent as all RDF graphs are [Hayes 2004], thus, this graph satisfies the query q. □

Lemma 4. *Let q be a PSPARQL$_{\text{cdfc}}$ query, for every restricted transition system K whose associated RDF graph is G, we have that $[\![q]\!]_G \neq \emptyset$ iff $[\![\mathcal{A}(q,m) \wedge \varphi_r]\!]^K \neq \emptyset$.*

Proof. ($\Rightarrow$) $[\![q]\!]_G \neq \emptyset$ implies that G is at least a canonical instance of q and it produced using a function f as shown below:

- if $(x,y,z) \in q$, then $f((x,y,z)) = (x,y,z) \in G$,
- if $(x,e,z) \in q$, then $f((x,e,z)) = (x,e,z) \in G$,
- if $(x, e \cdot e', z) \in q$, then $f((x,e,y)) \in G$ and $f((y,e',z)) \in G$,
- if $(x, e \mid e', z) \in q$, then $f((x,e,z)) \in G$ or $f((x,e',z)) \in G$,
- if $(x,e^+,z) \in q$, then $f((x,e,y_1)) \in G$ and ... and $f((y_n,e,z)) \in G$,
- if $(x,e^*,z) \in q$, then $f((x,e^+,z)) \in G$

Since G is an instance of q, G is a model of q (cf. Theorem 3). Now, we construct a transition system $\sigma(G) = (S,R,L)$ in the same way as is done in Definition 9. To prove that $\sigma(G)$ is a model of $\mathcal{A}(q,m)$, we consider two cases:

(i) when q contains a cyclic component among its non-distinguished variables which is connected to a distinguished variable or a constant, and

(ii) when q is cycle-free

First, (i) consider when q is cyclic, in this case, its encoding is $\bigvee_{i=1}^{|m|} \mathcal{A}(q, m_i)$. Using the nominals that correspond to the encodings of the distinguished variables and constants of q, one can successfully create a formula that can encode multiply occurring non-distinguished variables. Henceforth, creating a formula that is satisfiable in cyclic models. Further, the disjuncts in the encoding guarantee that possible set of substitutions in m capture the intended semantics of a cyclic query. One can verify that $\sigma(G)$ is a model of the disjuncts $\mathcal{A}(q, m_i)$, this is because nominals encoding the constants and distinguished variables are true in $\sigma(G)$ as they exist in G. Furhter, since the formulas corresponding to the encoding of the non-distinguished variables are obtained using the constants or distinguished variables, they are also true in $\sigma(G)$. Therefore, $\mathcal{A}(q, m)$ is satisfiable in $\sigma(G)$. To elaborate, if l is either x or y or z of the triple $(x, y, z) \in q$,

- for l either a distinguished variable or constant, l is satisfiable in $\sigma(G)$ since $[\![l]\!]^{\sigma(G)} \in L(l)$,
- for l a uniquely appearing non-distinguished variable, l is true in $\sigma(G)$ since $\top$ is true everywhere in the transition system,
- a multiply occurring non-distinguished variable l is true in $\sigma(G)$ since $\exists t \in S'.t \in L(c) \wedge t \in [\![c]\!]^{\sigma(G)}$, where c is a nominal encoding a constant or distinguished variable of the triple (x, y, z).
- for $l = e$ a regular expression, its encoding $\mathcal{R}(e, z)$ is satisfiable in $\sigma(G)$ if e is satisfiable in $\sigma(G)$.

Thus, since $\sigma(G)$ is a restricted transition system, we obtain that $[\![\mathcal{A}(q, m)\varphi_r]\!]_G \neq \emptyset$.
If (ii) q is cycle-free, then the encoding the non-distinguished variables with $\top$ suffices to justify that $\sigma(G)$ is a model of its encoding.
($\Leftarrow$) Assume that $[\![\mathcal{A}(q, m) \wedge \varphi_r]\!]^K \neq \emptyset$. We now create an RDF graph G from K as follows:

- if $\forall s_1, s_2, s_3 \in S' \wedge t \in S''.(s_1, t) \in R(s) \wedge (t, s_2) \in R(p) \wedge (t, s_3) \in R(o)$ and for each triple $t_i = (x_i, y_i, z_i) \in q$ if $s_1 \in L(x_i) \wedge s_2 \in L(y_i) \wedge s_3 \in L(z_i)$, then $(x_i, y_i, z_i) \in G$. This case holds if x_i, y_i and z_i are either distinguished variables or constants. Note here that if x_i or y_i or z_i appear in another triple $t_j = (x_j, y_j, z_j) \in q$, then the equivalent item in t_j is replaced with the value of the corresponding entry in t_i.
- if $\forall s_1, s_2, s_3 \in S' \wedge t \in S''.(s_1, t) \in R(s) \wedge (t, s_2) \in R(p) \wedge (t, s_3) \in R(o)$ and for each triple $t_i = (x_i, y_i, z_i) \in q$ if $s_1 \in L(x_i) \wedge s_2 \in L(y_i)$, then $(x_i, y_i, c_i) \in G$ where c_i is a fresh constant. This case holds if z_i is a non-distinguished variable. Similarly, the case when x_i or y_i or both are variables can be worked out.
- if $\forall s_1, s_2, s_3 \in S' \wedge t \in S''.(s_1, t) \in R(s) \wedge (t, s_2) \in R(p) \wedge (t, s_3) \in R(o)$ and for each triple $t_i = (x_i, e_i, z_i) \in q$ if $s_1 \in [\![\langle s \rangle \langle p \rangle e_i]\!]^K \in L(e_i) \wedge s_2 \in L(e_i) \wedge s_3 \in [\![\langle \bar{o} \rangle \langle p \rangle e_i]\!]^K$, then $(c_i, e_i, d_i) \in G$ where c_i and d_i are fresh constants. This case holds if x_i and

y_i are multiply occurring non-distinguished variables. Similarly, all the other cases can be worked out.

Since G is a technical construction obtained from a restricted transition system associated to q, it holds that $[\![q]\!]_G \neq \emptyset$. □

In the following, for the sake of legibility, we denote $A(q) \wedge \neg A(q', m) \wedge \varphi_r$ by $\Phi(q, q')$.

Theorem 4 (Soundness and Completeness)**.** *Given a PSPARQL query q and a PSPARQL$_{\text{cdfc}}$ q', $\Phi(q, q')$ is unsatisfiable if and only if $q \sqsubseteq q'$.*

Proof. ($\Rightarrow$) we prove the contrapositive, $q \not\sqsubseteq q' \Rightarrow \Phi(q, q')$ is satisfiable. Assume there exists a graph G such that there exists a tuple $\overrightarrow{a} \in [\![q]\!]_G$ and $\overrightarrow{a} \notin [\![q']\!]_G$. We construct a restricted transition system K from G. From Proposition 1, we get that $[\![\varphi_r]\!]^K \neq \emptyset$. We use a tuple of URIs $\overrightarrow{a}$ to instantiate the distinguished variables in q and q'. Using the encodings of the instantiated queries and from Lemma 4, it can be inferred that $[\![\mathcal{A}(q)]\!]^K \neq \emptyset$ and $[\![\mathcal{A}(q', m)]\!]^K = \emptyset$. The later is not satisfiable in K because the nominals corresponding to the constants $\overrightarrow{a}$ are not satisfied. Consequently, $[\![\neg\mathcal{A}(q', m)]\!]^K \neq \emptyset$ and $\mathcal{A}(q) \wedge \neg\mathcal{A}(q', m)$ is satisfiable. Therefore, we arrive at $\Phi(q, q')$ is satisfiable.

($\Leftarrow$) we show that if $\Phi(q, q')$ is satisfiable, then $q \not\sqsubseteq q'$. Consider a restricted transition system model K for $\Phi(q, q')$. We construct an RDF graph G from K and we need to verify that $[\![q]\!]_G \not\subseteq [\![q']\!]_G$. To do so, we start from the assumption, $[\![\mathcal{A}(q) \wedge \neg\mathcal{A}(q', m)]\!]^K \neq \emptyset$. Subsequently, $[\![\mathcal{A}(q)]\!] \neq \emptyset$ and $[\![\mathcal{A}(q', m)]\!]^K = \emptyset$ because G contains all those triples that satisfy q and not q'. Besides, if q' contains a cycle, the constraints expressed by $\neg\mathcal{A}(q', m)$ are satisfied due to the ability, in a μ-calculus extended with nominals and converse, to express a formula that is satisfied in cyclic models. Therefore, $q \not\sqsubseteq q'$. □

4.1.2.1 Complexity

Due to duplication in the encoding of the right-hand side query q', the size of $|\mathcal{A}(q', m)|$ is exponential in terms of the non-distinguished variables that appear in cycles in the query. Therefore, we obtain a 2ExpTime upper bound for containment. The problem is solvable in ExpTime if there is no cycle in the right-hand side query. In this case, this complexity is a lower bound due to the complexity of satisfiability in the μ-calculus.

Proposition 4. *PSPARQL query containment can be solved in a time of $2^{\mathcal{O}(n)}$, where n is an exponential size encoding of the containment problem.*

Proof. This proposition is a consequence of Theorem 4. The size of the encoding of the containment problem is exponential due to cycles in the right-hand side query. In the μ-calculus, the satisfiability test of a formula can be performed in exponential time. Consequently, containment of PSPARQL queries can be performed in a double exponential amount of time. □

4.1.3 Inverted Path SPARQL Queries

Having chosen a fragment of the μ-calculus that is equipped with converse henceforth enabling backward traversal in a transition system, it is natural to study the containment of inverted path queries (a.k.a. two-way regular path queries). The fragment of PSPARQL queries studied in Section 4.1.1 resemble as that of the well-studied UCRPQs (Union of conjunctive regular path queries). In order to allow backward navigation in a graph, we add inverted regular path expressions e^- to UCRPQS resulting in a query language UC2RPQs (union of conjunctive two-way RPQs). Note that, the inverse operator $^-$ handles only atomic expressions. To start with, consider the following example.

Example 21. *Consider the query q "find pairs of students who take courses together."*

```
SELECT ?x ?y
WHERE {
        ?x :takesCourse. :takesCourse- ?y .
}
```

4.1.3.1 Encoding Inverted PSPARQL Queries

Adding inverse to PSPARQL changes only triple patterns, thus it is sufficient to modify the encoding of triple patterns as the other query constructs are formed inductively. The encoding of triple patterns that contain regular expression patterns with inverses is detailed in Definition 15. The rest of the translation is similar to the encoding of PSPARQL queries (cf. Section 4.1.1).

Definition 15. *PSPARQL triple patterns with inverse can be encoded into μ-calculus as follows, using $\mathcal{A}$ and $\mathcal{R}$ that are extended from Section 4.1.1 to accommodate the inverse operator $^-$.*

$$
\begin{aligned}
\mathcal{A}((x, e^-, y)) &= \mathcal{A}((y, e, x)) \\
\mathcal{A}((x, e_1^- \mid e_2, y)) &= \mathcal{A}((x, e_1^-, y)) \vee \mathcal{A}((x, e_2, y)) \\
\mathcal{A}((x, e_1 \mid e_2^-, y)) &= \mathcal{A}((x, e_1, y) \vee \mathcal{A}((x, e_2^-, y)) \\
\mathcal{A}((x, e_1^- \mid e_2^-, y)) &= \mathcal{A}((x, e_1^-, y)) \vee \mathcal{A}((x, e_2^-, y)) \\
\mathcal{A}((x, e_1^-.e_2, y)) &= \langle o \rangle x \wedge \mathcal{R}(e_1^- \cdot e_2, y) \\
\mathcal{A}((x, e_1.e_2^-, y)) &= \langle \bar{s} \rangle x \wedge \mathcal{R}(e_1, \langle \bar{o} \rangle \mathcal{R}(e_2^-, y)) \\
\mathcal{A}((x, e_1^-.e_2^-, y)) &= \langle o \rangle y \wedge \mathcal{R}(e_2^-, \langle \bar{o} \rangle \mathcal{R}(e_1^-, y)) \\
\mathcal{A}((x, (e^-)^+, y)) &= \langle o \rangle x \wedge \mathcal{R}((e^-)^+, y) \\
\mathcal{A}((x, (e^-)^*, y)) &= \langle o \rangle x \wedge \mathcal{R}((e^-)^*, y)
\end{aligned}
$$

where $\mathcal{R}$ is extended to encode inverted path expressions inductively as:

$$\begin{aligned}
\mathcal{R}(u^-, y) &= \langle p\rangle u \wedge \langle \bar{s}\rangle y \\
\mathcal{R}((e^-)^+, y) &= \mu X.\mathcal{R}(e^-, y) \vee \mathcal{R}(e^-, \langle \bar{o}\rangle X) \\
\mathcal{R}((e^-)^*, y) &= \mathcal{R}((e^-)^+, y) \vee \langle \bar{s}\rangle y \\
\mathcal{R}(e_1^- \cdot e_2, y) &= \mathcal{R}(e_1^-, \langle s\rangle \mathcal{R}(e_2, y)) \\
\mathcal{R}(e_1 \cdot e_2^-, y) &= \mathcal{R}(e_1, \langle \bar{o}\rangle \mathcal{R}(e_2^-, y)) \\
\mathcal{R}(e_1^- \cdot e_2^-, y) &= \mathcal{R}(e_1^-, \langle \bar{o}\rangle \mathcal{R}(e_2^-, y))
\end{aligned}$$

Example 22. *The following formula is an encoding of the query in Example 21.*

$$\begin{aligned}
\mathcal{A}(q) &= \mathcal{A}\big((x, takesCourse \cdot takesCourse^-, y)\big) \\
&= lfp\big(X, \langle \bar{s}\rangle x \wedge \mathcal{R}(takesCourse, \langle \bar{o}\rangle \mathcal{R}(takesCourse^-, y))\big) \\
&= lfp\big(X.\langle \bar{s}\rangle x \wedge \langle p\rangle takesCourse \wedge \langle o\rangle\langle \bar{o}\rangle \mathcal{R}(takesCourse^-, y)\big) \\
\mathcal{A}(q) &= lfp\big(X, \langle \bar{s}\rangle x \wedge \langle p\rangle takesCourse \wedge \langle o\rangle\langle \bar{o}\rangle(\langle p\rangle takesCourse \wedge \langle \bar{s}\rangle y)\big)
\end{aligned}$$

Reducing Containment to Unsatisfiability To reduce containment to the validity problem in the μ-calculus, first we need to verify that our encoding procedure produces a correct encoding of a PSPARQL query. As such, we need to verify that both the left-hand side $\mathcal{A}(q)$ and right-hand side $\mathcal{A}(q, m)$ encodings are correct. We show only for the right-hand side query, the result for the left immediately follows from it. To do this, we rely on the following lemma.

Lemma 5. *Let q be a PSPARQL query, for every restricted transition system K whose associated RDF graph is G, we have that $[\![q]\!]_G \neq \emptyset$ iff $[\![\mathcal{A}(q, m) \wedge \varphi_r]\!]^K \neq \emptyset$.*

Proof. Here, it suffices to show that every PSPARQL query with inverse can be transformed into one without inverse with the same interpretation. To do so, we use a

function f that syntactically removes the inverses from a PSPARQL query:

$$
\begin{aligned}
f\big((x,y,z)\big) &= (x,y,z)\\
f\big((x,uri,z)\big) &= (x,uri,z)\\
f\big(x,e^-,y)\big) &= (y,e,x)\\
f\big((x,e_1^- \cdot e_2,y)\big) &= f\big((z,e_1,x)\big) \text{ AND } f\big((z,e_2,y)\big)\\
&\qquad \text{where } z \text{ is a fresh variable}\\
f\big((x,e_1 \cdot e_2^-,y)\big) &= f\big((x,e_1,z)\big) \text{ AND } f\big((y,e_2,z)\big)\\
f\big((x,e_1^- \cdot e_2^-,y)\big) &= f\big((z,e_1,x)\big) \text{ AND } f\big((y,e_2,z)\big)\\
f\big((x,e_1^- \mid e_2,y)\big) &= f\big((y,e_1,x)\big) \text{ UNION } f\big((x,e_2,y)\big)\\
f\big((x,e_1 \mid e_2^-,y)\big) &= f\big((x,e_1,y)\big) \text{ UNION } f\big((y,e_2,x)\big)\\
f\big((x,e_1^- \mid e_2^-,y)\big) &= f\big((y,e_1,x)\big) \text{ UNION } f\big((y,e_2,x)\big)\\
f\big(x,(e^-)^+,y)\big) &= f\big(z,e,x)\big) \text{ AND } f\big((y,e^+,z)\big)\\
f\big(x,(e^-)^*,y)\big) &= (x,\varepsilon,y) \text{ UNION } f\big((x,(e^-)^+,y)\big)\\
f(q_1 \text{ AND } q_2) &= q_1 \text{ AND } q_2\\
f(q_1 \text{ UNION } q_2) &= q_1 \text{ UNION } q_2
\end{aligned}
$$

Thus, using f, we can transform a PSPARQL query q with inverse into one $f(q)$ without inverse. Our transformation preserves the semantics of PSPARQL queries with inverse as already shown in [Alkhateeb 2008]. As a consequence, we can conclude that the above lemma holds from the proof of Lemma 2. □

Theorem 5 (Soundness and Completeness)**.** *Given a PSPARQL query q with inverse and a PSPARQL$_{\text{cdfc}}$ q' with inverse, $\mathcal{A}(q) \wedge \neg\mathcal{A}(q',m) \wedge \varphi_r$ is unsatisfiable if and only if $q \sqsubseteq q'$.*

Proof. Using the transformation function f, from Lemma 5, we can rephrase the above theorem as: given two PSPARQL queries q and q' with inverse, it holds that $\mathcal{A}(f(q)) \wedge \neg\mathcal{A}(f(q'),m) \wedge \varphi_r$ is unsatisfiable if and only if $f(q) \sqsubseteq f(q')$. Since this argument has already been proved in Theorem 4, we obtain that the above theorem also holds. □

To demonstrate how the containment test is carried out, we provide the following example taken from [Calvanese *et al.* 2000]. The queries ask for all pairs of individuals $(?x, ?y)$ that are in the transitive closure of the sibling (including stepsibling) relation such that there is some individual $?z$ that $?x$ and $?y$ have two descendants who are respectively the father and mother of $?z$.

Example 23. *Given PSPARQL queries q and q' on a graph modelling a family tree, we produce the encoding to determine $q' \sqsubseteq q$ as follows:*

```
q =  SELECT ?x  ?y
     WHERE {
        ?x (:father- . :father | :mother- . :mother)+ ?y .
        ?x (:father | :mother)* . :father ?z .
        ?x (:father | :mother)* . :mother ?z .
     }

q' =  SELECT ?x ?y
      WHERE {
         ?x (:father- . :father . :mother- . :mother) ?y .
         ?x :father . :mother- ?y
      }
```

It is possible to verify that $q' \sqsubseteq q$. To do so, we produce the encoding of these queries,

$$\begin{aligned}\mathcal{A}(q') &= lfp\big(X, \langle o\rangle x \wedge \mathcal{R}(father^- \cdot father \cdot mother^- \cdot mother, y)\big) \\ &\quad \wedge \underbrace{lfp\big(Y, \langle \bar{s}\rangle x \wedge \mathcal{R}(father \cdot mother^-, y)\big)}_{\psi} \\ &= lfp\big(X, \langle o\rangle x \wedge \mathcal{R}(father^-, \langle s\rangle \mathcal{R}(father \cdot mother^- \cdot mother, y))\big) \wedge \psi \\ &= lfp\big(X, \langle o\rangle x \wedge \langle p\rangle father \wedge \langle \bar{s}\rangle\langle s\rangle \mathcal{R}(father, \langle s\rangle \mathcal{R}(mother^- \cdot mother, y))\big) \wedge \psi \\ &= lfp\big(X, \langle o\rangle x \wedge \langle p\rangle father \wedge \langle \bar{s}\rangle\langle s\rangle \mathcal{R}\big(father, \langle s\rangle \mathcal{R}(mother^- \cdot mother, y))\big) \wedge \psi\end{aligned}$$

$$\begin{aligned}\mathcal{A}(q, m) = lfp\big(X, \langle \bar{s}\rangle x \wedge \mathcal{R}((father^- \cdot father \mid mother^- \cdot mother)^+, y)\big) \\ \wedge lfp\big(Y, \langle \bar{s}\rangle x \wedge \mathcal{R}((father \mid mother)^* \cdot father, \langle \bar{o}\rangle\langle p\rangle father)\big) \\ \wedge lfp\big(Z, \langle \bar{s}\rangle x \wedge \mathcal{R}((father \mid mother)^* \cdot mother, \langle \bar{o}\rangle\langle p\rangle mother)\big)\end{aligned}$$

Note that the encoding of the right-hand side query q, $\mathcal{A}(q, m)$ is equivalent to $\mathcal{A}(q)$ because there are no non-distinguished variables in q. Thus, we can use the same encoding procedure for q' and q. If we feed the encoding $\mathcal{A}(q') \wedge \neg\mathcal{A}(q, m) \wedge \varphi_r$ to a μ-calculus satisfiability solver, it will respond "no" leading to the result $q' \sqsubseteq q$.

Complexity The complexity of encoding path queries with inverse as formulas is exponential (due to the encoding of the right-hand side query $\mathcal{A}(q', m)$). Thus, adding the inverse operator to determine containment of path SPARQL queries has no extra complexity.

Proposition 5. *Given two PSPARQL queries q and q' with inverse, determining the containment of q in q' can be performed in a double exponential amount of time.*

4.2 Containment of PSPARQL Queries under Schema

Most of the results on query containment concern conjunctive queries and their extensions. In [Chandra & Merlin 1977], NP-completeness has been established for conjunc-

tive queries. In [Klug 1988, van der Meyden 1992], Π_2^p-completeness of containment of conjunctive queries with inequalities was proved and in [Sagiv & Yannakakis 1980] the case of queries with the union and difference operators was studied. Other studies consider containment in the presence of various types of constraints [Aho *et al.* 1979, Levy & Sagiv 1993, Levy & Rousset 1996, Florescu *et al.* 1998]. In relation, we investigate the containment of path SPARQL queries in the presence of description logic schema axioms.

In this section, we study the containment problem for conjunctive regular path queries with inverse (CRPQIs) in the presence of ontological axioms. We refer to this class of SPARQL queries as *PSPARQL queries with inverse*. The expressive power of the regular expression patterns is limited in that negation and number restrictions are not allowed as containment becomes undecidable [Barceló *et al.* 2010]. Query containment is undecidable if we do not limit the expressive power of the query and schema languages. In fact, in knowledge representation formalisms suitable query languages have been designed for retaining decidability [Glimm *et al.* 2008, Lutz 2008, Calvanese *et al.* 2008, Eiter *et al.* 2009, Calvanese *et al.* 2011, Bienvenu 2012]. Thus, we consider the containment, of PSPARQL queries with inverse, under the presence of $\mathcal{ALCH}$ schema axioms.

To motivate the study of containment in the presence of ontologies, we start with the following examples.

Example 24. *Consider the following queries that query all cities which are reachable from Paris using a sequence of transportation means.*

```
q =  SELECT ?y
     WHERE {
        :train rdfs:subPropertyOf :Transport .
        :Paris :train+ ?y .
    }

q' =  SELECT ?y
      WHERE {
         :Paris :transport+ ?y .
      }
```

With SPARQL simple entailment semantics, we have that $q \not\sqsubseteq q'$. *However, if we consider the RDFS semantics (a.k.a. RDFS entailment regime), then we get* $q \sqsubseteq q'$.

Example 25. *Let us modify the queries in Example 24 and reconsider the containment problem among them:*

```
q =  SELECT ?y
     WHERE {
         :Paris (:transport . :transport) ?y .
    }
```

```
q′ =  SELECT ?y
      WHERE {
         :Paris (:train . :plane) ?y .
         :train rdfs:subPropertyOf :transport .
         :plane rdfs:subPropertyOf :transport .
      }
```

Under the RDFS semantics, it can be seen that $q' \sqsubseteq q$ whereas $q \not\sqsubseteq q'$. However, containment fails in both directions under simple entailment semantics of PSPARQL.

These examples motivate why it is necessary to address containment and equivalence problems in the presence of schema axioms. In fact, query answering for expressive description logic languages has been an important research direction in the knowledge representation domain. In line with this, the W3C SPARQL working group has started an important effort in order to extend SPARQL's basic graph pattern to other entailment regimes (RDF, OWL, RIF). This is indispensable because answering queries under entailment regimes allows to include more answers by performing reasoning as this cannot be done in SPARQL's simple entailment semantics.

For this study we consider $\mathcal{ALCH}$ an extension of the description logic $\mathcal{ALC}$ with role hierarchy. It encompasses the lightweight ontology language RDFS. For a separate study of containment under RDFS semantics (or OWL entailment regimes), we refer the reader to Chapter 5.

Encoding queries and schema axioms: the encoding of PSPARQL queries with inverse has been presented in the previous section and likewise $\mathcal{ALCH}$ axioms can be translated into μ-calculus inductively on the structure of the schema constructs as already done in Chapter 3.

4.2.1 Reducing Containment to Unsatisfiability

PSPARQL queries can be translated into μ-calculus formulas using function $\mathcal{A}$ and likewise $\mathcal{ALCH}$ ontology axioms become formulas with function η (cf. Chapter 3 for details). Thus, given two queries q and q', and schema axioms $\mathcal{S}$, the problem of testing $q \sqsubseteq_{\mathcal{S}} q'$ can be reduced to checking the satisfiability of the formula $\eta(\mathcal{S}) \wedge \mathcal{A}(q) \wedge \neg\mathcal{A}(q', m) \wedge \varphi_r$. In the following, for legibility, we denote this formula by $\Phi(\mathcal{S}, q, q')$.

Theorem 6 (Soundness and Completeness)**.** *Given a PSPARQL query q, a PSPARQL$_{\text{cdfc}}$ query q', and a set of $\mathcal{ALCH}$ schema axioms $\mathcal{S}$, $q \sqsubseteq_{\mathcal{S}} q'$ if and only if $\Phi(\mathcal{S}, q, q')$ is unsatisfiable.*

Proof. The proof follows from the proof of Theorem 4 and Theorem 2. □

This theorem leads to the following proposition. The complexity of determining PSPARQL containment under the presence of schema axioms has a double exponential upper bound. This bound is a result of ExpTime satisfiability of a μ-calculus formula and an exponential size encoding.

Proposition 6 (Complexity). *Given a PSPARQL query q, a PSPARQL$_{\text{cdfc}}$ query q', and $\mathcal{ALCH}$ ontology axioms $\mathcal{S}$, the problem of checking $q \sqsubseteq_{\mathcal{S}} q'$ can be determined in a time of $2^{\mathcal{O}(n)}$. n is the size of the encoding which is exponential.*

The double exponential upper bound reduces to just single exponentiation when the right-hand side query has a tree structure.

4.3 Conclusion

In this chapter, we have addressed containment of SPARQL queries with paths. We took a similar approach to [Genevès *et al.* 2007] that established the optimal complexity for XPath query containment. The problem of PSPARQL query containment has been reduced to satisfiability testing in the μ-calculus. For that purpose, we encoded PSPARQL queries as formulas. We proved that the reduction is sound and complete when the right-hand side query is of type PSPARQL$_{cdfc}$ and that determining containment requires a double exponential amount of time.

Paths are included in the new version of SPARQL 1.1 which is currently under standardization by W3C hence our results are a step towards query containment for SPARQL 1.1. Further, this work is also relevant for determining containment of SPARQL queries under RDFS entailment regime as discussed in Chapter 5. Similarly, the encoding procedures can be extended to richer types of queries, e.g., query modulo OWL ontologies.

Chapter 5

Containment under Entailment Regimes

Contents

Presently, SPARQL is being extended with different entailment regimes and regular path expressions. The semantics of SPARQL relies on the definition of basic graph pattern matching that is built on top of RDF simple entailment [Hayes 2004]. However, it may be desirable to use SPARQL to query triples entailed from subclass, subproperty, range, domain, and other relations which can be represented using RDF schema. The SPARQL specification defines the results of queries based on simple RDF entailment. The specification also presents a general parametrized definition of graph pattern matching that can be expanded to other entailments beyond simple entailment. Query answering under the RDFS entailment regime can be achieved via: (1) materialization (computing the deductive closure of the queried graph), (2) rewriting the queries using the schema, and (3) hybrid (combining materialization and query rewriting) [Glimm 2011]. We use a technique based on the approaches (1) and (2) to study the problem of SPARQL query containment under the RDFS and OWL entailment regimes.

Entailment regimes take into account the semantics of schema information embedded in the queries, so this is different from the one in Chapter 3. We study SPARQL query containment under the RDFS (respectively OWL Direct Semantics) entailment regime defined as: for any graph (respectively knowledge base) that satisfies RDFS (respectively OWL) axioms, checking whether the results of one query are included in the results of another query. To do so, we translate SPARQL queries and a fragment of RDFS and OWL schema axioms into μ-calculus formulae. We refer to the fragment

of the RDFS (resp. OWL) entailment regime as *smallRDFS* (resp. *OWL-$\mathcal{ALCH}$*) entailment regime. Then, we show how query containment in SPARQL under smallRDFS entailment can be reduced to unsatisfiability in the μ-calculus. Beyond, smallRDFS entailment, we show the same procedure can be extended for the OWL-$\mathcal{ALCH}$ Direct Semantics entailment regime. This is the same encoding as using the $\mathcal{ALCH}$ language considered in Chapter 3. But, we provide three different approaches of this problem based on the techniques of Chapter 3 and 4.

You may have RDFS (or OWL) statements:

1. in the query,
2. in the queried graph, or
3. in a separate schema.

You may decide to interpret them by using simple RDF entailment, RDFS entailment, or OWL entailment. In principle, there are 6 possibilities but as far as query containment is concerned, 2 cannot be considered because containment is independent from the queried graph, this can be simulated by option number 3, so we have the following containment problems under entailment regimes: containment under simple RDF entailment $\sqsubseteq^{rdf}_{\emptyset}$ (a.k.a. containment of SPARQL queries), containment under schema $\sqsubseteq^{rdf}_{\mathcal{S}}$, containment of RDFS queries (SPARQL queries that contain RDF schema information) $\sqsubseteq^{\text{rdfs}}_{\emptyset}$, and containment of RDFS queries under schema ($\sqsubseteq^{\text{rdfs}}_{\mathcal{S}}$). We write $\sqsubseteq^{rdf}_{\mathcal{S}}$ as just $\sqsubseteq_{\mathcal{S}}$.

Before diving into the presentation of query containment under simpleRDFS and OWL-$\mathcal{ALCH}$ entailment regimes, we would like to point out one important difference between containment under schema axioms and containment under entailment regimes. In doing so, we consider the example below.

Example 26. *Difference between containment test under RDFS schema axioms versus RDFS entailment.*

$$\mathcal{S} = \{\}$$

$$\frac{q_1\{x,y\} = (x, parentOf, y) \quad q_2\{x,y\} = (x, motherOf, y) \text{ AND } (motherOf, \texttt{sp}, parentOf)}{q_1 \not\sqsubseteq^{\text{rdfs}}_{\mathcal{S}} q_2,\ q_2 \sqsubseteq^{\text{rdfs}}_{\mathcal{S}} q_1 \qquad q_1 \not\sqsubseteq_{\mathcal{S}} q_2,\ q_2 \not\sqsubseteq_{\mathcal{S}} q_1}$$

$$\mathcal{S} = \{(motherOf, \texttt{sp}, parentOf)\}$$

$$\frac{q\{x,y\} = (x, parentOf, y) \quad q'\{x,y\} = (x, motherOf, y)}{q \not\sqsubseteq^{\text{rdfs}}_{\mathcal{S}} q',\ q' \sqsubseteq^{\text{rdfs}}_{\mathcal{S}} q \qquad q \not\sqsubseteq_{\mathcal{S}} q',\ q' \sqsubseteq_{\mathcal{S}} q}$$

From this example, one can see that if schema information is included in the queries for which containment is to be determined, containment under entailment regimes is

used. However, if explicitly schema axioms are given to test containment, then either containment under schema or containment under entailment regimes can be used.

The RDFS entailment regime extends the basic graph patterns of a SPARQL1.0 query, for instance, $(x, sc, transport)$ can be extended to

$$(x, sc^{+}, transport).$$

In this regard, we take another example to once again outline the differences.

Example 27. *Let us reconsider the containment between q and q' of Example 25. Clearly, containment under simple entailment does not hold, i.e., $q \not\sqsubseteq_s q'$ and $q' \not\sqsubseteq_s q$. On the other hand, when considering containment under RDFS entailment, we have $q' \sqsubseteq_{rdfs} q$ and $q \not\sqsubseteq_{rdfs} q'$. Since, no schema axioms are provided, containment fails in both directions for containment under schema axioms, i.e., $q \not\sqsubseteq_{\mathcal{S}} q'$ and $q' \not\sqsubseteq_{\mathcal{S}} q$.*

Example 28. *Consider querying types of transport means:*

$$q\{x\} = (x, type, y) \text{ AND } (y, subClassOf, transport)$$
$$q'\{x\} = (x, type, transport)$$

Under the RDFS entailment regime, $q \sqsubseteq_{rdfs} q'$ but $q' \not\sqsubseteq_{rdfs} q$.

Relying on these important differences between containment under entailment regimes and under schema axioms, we continue addressing the problem. For the containment under simpleRDFS entailment regime, we propose three different approaches to determine containment. But the encoding of queries is the same as that of the one given in Chapter 3 except when queries contain schema axioms. In that case, those schema axioms are encoded similarly as those of constraining schema axioms. Finally, we study the containment problem under the OWL-$\mathcal{ALCH}$ entailment regime for SPARQL-OWL queries [Glimm *et al.* 2008, Kollia *et al.* 2011]. Importantly, under both simpleRDFS and OWL-$\mathcal{ALCH}$ entailment regime semantics, there are no non-distinguished variables since answers are computed from the basic graph patterns of the query, projection is considered as a post-processing step. Projection is performed, after the evaluation of basic graph patterns, i.e., after all the variables are bound to values in the ontology. Thus, when dealing with containment, the encoding of the right-hand side query can be obtained in the same way as that of the left-hand side query (cf. Section 3).

In this thesis, we do not follow the W3C definitions of the SPARQL entailment regimes[1]. Instead, we identify a small fragment of SPARQL, AND and UNION graph patterns, and schema languages as discussed above. Our approach excludes the RDF(S) *axiomatic triples* and non-standard use of vocabulary terms [Hogan *et al.* 2009], as shown in the following definition:

Definition 16 (Non-standard vocabulary usage [Hogan *et al.* 2009])**.** *An RDF triple t has non-standard vocabulary usage if the following condition holds:*

[1] http://www.w3.org/TR/sparql11-entailment/

- *a property in Voc* = {dom, range, sc, sp, ObjectAllValuesFrom, ObjectSomeValuesFrom, EquivalentClasses, EquivalentObjectProperties, ObjectUnionOf, ObjectIntersectionOf} *appears in a position different from the predicate position.*

Besides, the so called schema queries such as $(x, \text{subClassOf}, y)$ or $(z, \text{subPropertyOf}, r)$ or $(x, \text{inverseOf}, y)$ are also excluded as shown in Definition 17. A more detailed discussion on the issue of schema queries can be found [Krötzsch 2012].

Definition 17 (Schema-free queries)**.** *A SPARQL query is schema-free if it does not contain schema vocabulary (a property in Voc of Definition 16) in the property position of its BGP.*

5.1 Containment under simpleRDFS Entailment Regime

In the following, we propose three approaches to determine query containment under the simpleRDFS entailment regime: encoding the RDFS semantics, query rewriting, and encoding the schema approaches. Interestingly enough, it has been show that the standard set of entailment rules for RDFS is incomplete and that this can be corrected by allowing blank nodes in predicate position [Ter Horst 2005]. Nonetheless, for our purposes, we consider the subset of RDFS inference rules presented below. Note that this subset makes up a core fragment of RDFS, [Muñoz *et al.* 2007] showed that this fragment is minimal and well-behaved as well as its semantics is equivalent to that of the full RDFS. This fragment has the following deductive RDFS inference rules:

- Subclass (`sc`)

$$\frac{(a, \texttt{sc}, b)\ (b, \texttt{sc}, c)}{(a, \texttt{sc}, c)} \qquad \frac{(a, \texttt{sc}, b)\ (x, \texttt{type}, a)}{(x, \texttt{type}, b)} \tag{5.1}$$

- Subproperty (`sp`)

$$\frac{(a, \texttt{sp}, b)\ (b, \texttt{sp}, c)}{(a, \texttt{sp}, c)} \qquad \frac{(a, \texttt{sp}, b)\ (x, a, y)}{(x, b, y)} \tag{5.2}$$

- Typing (`dom`, `range`)

$$\frac{(a, \texttt{dom}, b)\ (x, a, y)}{(x, \texttt{type}, b)} \qquad \frac{(a, \texttt{range}, b)\ (x, a, y)}{(y, \texttt{type}, b)} \tag{5.3}$$

5.1.1 Encoding the RDFS Semantics

When queries are evaluated under the RDFS entailment regime, the queried graph is materialized or saturated using RDFS inference rules (or simply rules) and the schema. Henceforth, implicit or inferred triples are considered when computing the result of the query. Since no specific graphs are considered when dealing with containment, we encode schema and rules. In addition, blank nodes that appear in the schema graph are

skolemized, i.e., replaced by fresh constants that do not appear neither in the queries nor schema.

Definition 18. *The encoding of an RDF schema graph $S = \{t_1, \cdots, t_n\}$ is produced by encoding each schema triple $t_i = (x, y, z) \in S$ such that:*

$$\Phi_S = \bigwedge_{i=1 \wedge t_i \in S}^{n} lfp\big(X, (\langle \bar{s} \rangle x \wedge \langle p \rangle y \wedge \langle o \rangle z)\big)$$

where x, y, and z are atomic propositions corresponding to triple elements.

Definition 19 (Encoding inference rules)**.** *The μ-calculus encoding of RDFS inference rules of (5.1)–(5.3) is the disjunction of formulas (1) to (6) such that:*

$$\begin{aligned}
&(1) \quad gfp\big(X, \mu Y.\langle p \rangle \mathtt{sc} \wedge \langle o \rangle \top \vee \langle p \rangle \mathtt{sc} \wedge \langle s \rangle Y\big)\\
&(2) \quad gfp\big(X, \theta(x, \mathtt{type}, \theta(a, \mathtt{sc}, b)) \Rightarrow \theta(x, \mathtt{type}, b)\big)\\
&(3) \quad gfp\big(X, \mu Y.\langle p \rangle \mathtt{sp} \wedge \langle o \rangle \top \vee \langle p \rangle \mathtt{sp} \wedge \langle s \rangle Y\big)\\
&(4) \quad gfp\big(X, \theta(x, \theta(a, \mathtt{sp}, b), y) \Rightarrow \theta(x, b, y)\big)\\
&(5) \quad gfp\big(X, \theta(x, \theta(a, \mathtt{dom}, b), y) \Rightarrow \theta(x, \mathtt{type}, b)\big)\\
&(6) \quad gfp\big(X, \theta'(x, \theta(a, \mathtt{range}, b), y) \Rightarrow \theta(y, \mathtt{type}, b)\big)\\
\theta(x, y, z) = x \wedge \langle s \rangle(\langle p \rangle y \wedge \langle o \rangle z) \qquad &\theta'(x, y, z) = \ z \wedge \langle \bar{o} \rangle\big(\langle p \rangle(y \wedge \langle \bar{s} \rangle x)
\end{aligned}$$

We denote this formula by Φ_R.

Transitive RDFS properties, `sc` and `sp` are encoded as transitive closure: $\mathtt{sc}^+$ and $\mathtt{sp}^+$, respectively. This is because, with the μ-calculus, it is not possible to enforce transitivity of a property in a transition system. This is a consequence of the tree-model property of the μ-calculus, and the fact that transitivity does not hold in tree-shaped models [Sattler & Vardi 2001].

So far, we have produced the encodings of queries $\mathcal{A}(q)$, RDFS inference rules Φ_R, and RDFS graph Φ_S. Using these encodings, in the following, we reduce query containment to unsatisfiability in the μ-calculus and prove the correctness of this reduction.

Theorem 7 (Soundness and Completeness)**.** *Given SPARQL queries q and q' and a schema S, $q \sqsubseteq^S_{\text{rdfs}} q'$ is unsatisfiable if and only if $\Phi_R \wedge \Phi_S \wedge A(q) \wedge \neg A(q') \wedge \varphi_r$.*

Proof. The proof of this theorem directly follows from the proof of Theorem 2 and Theorem 3. □

5.1.2 Query Rewriting

SPARQL query containment under simpleRDFS entailment regime can be determined by rewriting queries using the RDFS inference rules (shown in equation (5.1)–(5.3)) and then reducing the encoding of the rewriting to unsatisfiability test. This rewriting, first introduced in [Alkhateeb *et al.* 2009], is done using PSPARQL as explained in the following definition.

Definition 20 (SPARQL to PSPARQL)**.** *Given a SPARQL query q, a rewriting function τ produces its PSPARQL equivalent as follows:*

$$\begin{aligned}
\tau((s, sc, o)) &= (s, sc^+, o) \\
\tau((s, sp, o)) &= (s, sp^+, o) \\
\tau((s, p, o)) &= (s, x, o) \text{ AND } (x, sp^*, p) \text{ such that } p \notin \{sc, sp, type, dom, range\} \\
\tau((s, type, o)) &= (s, type.sc^*, o) \text{ UNION } (s, x, y) \text{ AND } (x, sp^*.domain.sc^*, o) \\
&\quad \text{UNION } (y, x, s) \text{ AND } (x, sp^*.range.sc^*, o) \\
\tau((s, x, o)) &= (s, x, o) \text{ when } x \text{ is a variable} \\
\tau(q_1 \text{ AND } q_2) &= \tau(q_1) \text{ AND } \tau(q_2) \quad \tau(q_1 \text{ UNION } q_2) = \tau(q_1) \text{ UNION } \tau(q_2)
\end{aligned}$$

Example 29. *Consider the rewritings of the following queries: q and q' respectively.*

```
SELECT ?s
WHERE {
   ?s ?x ?d .
   ?x rdfs:subpropertyOf rdf:type .
   ?d rdfs:subclassOf ?c .
}
```

```
SELECT ?s
WHERE {
   ?s rdf:type ?c .
}
```

The rewritings of the above queries, $\tau(q)$ and $\tau(q')$, is shown below:

```
SELECT ?s
WHERE {
   ?s  ?x  ?d .
   ?x  rdfs:subpropertyOf+ rdf:type .
   ?d rdfs:subclassOf+  ?c .
}
```

```
SELECT ?s
WHERE {
   { ?s   rdf:type . rdfs:subclassOf*   ?c . }
      UNION
   { ?s  ?r  ?y .
     ?r  rdfs:subpropertyOf* . rdfs:domain . rdfs:subclassOf*  ?c . }
     UNION
   { ?y ?r ?s .
     ?r  rdfs:subpropertyOf* . rdfs:range . rdfs:subclassOf*  ?c . }

}
```

Note that this example also shows a non-standard use of the RDF(S) vocabulary [Hogan et al. 2009]. That is, the property $\mathrm{rdf:type}$ *appears in the object position of the triple* $(?x, \mathrm{rdfs:subclassOf}, \mathrm{rdf:type})$.

Proposition 7. *Given SPARQL queries* q *and* q'*, and a rewriting function* τ*,* $q \sqsubseteq^{\mathrm{rdfs}}_{\mathcal{S}} q' \Leftrightarrow \tau(q) \sqsubseteq^{\mathrm{rdf}}_{\mathcal{S}} \tau(q')$.

Proof. Assume that $q \sqsubseteq^{\mathrm{rdfs}}_{\mathcal{S}} q'$, we prove that $\tau(q) \sqsubseteq^{\mathrm{rdf}}_{\mathcal{S}} \tau(q')$. To prove this, we proceed inductively on the query structure.
(Base case) q and q' are made of triple patterns:

- if q and q' do not contain the vocabularies $\mathrm{type}, \mathrm{sc}, \mathrm{sp}, \mathrm{dom}$, and range, then $\tau(q) = q$ and $\tau(q') = q'$. Consequently, the proposition holds.

- if q and q' contain the vocabularies sc and sp in predicate positions of triple patterns, then they can be translated into transitive closures: sc^+ and sp^+, respectively. Evidently, the proposition holds since sc (resp. sp) is contained in sc^+(resp. sp^+). These vocabulary terms cannot appear as subject and objects of triple patterns because a non-standard use of a vocabulary is prohibited.

- if q and q' contain triple patterns that have properties in predicate positions, i.e., consider the triple pattern (s, p, o), then the rewriting is $\tau((s, p, o)) = (s, x, o)$ AND (x, sp^*, p). In this case also, it holds that $q \sqsubseteq^{\mathrm{rdfs}}_{\mathcal{S}} q' \Leftrightarrow \tau(q) \sqsubseteq^{\mathrm{rdf}}_{\mathcal{S}} \tau(q')$. Here, p cannot be *sc*, *sp*, or *type*, because the rewriting of those terms is done separately. Instead, if p is a variable then, the rewriting is the same as the original one. Hence, the claim still holds.

- if q and q' contain the vocabulary type, then their rewritings contain all possible ways of inferring type relations between an instance (the subject) and a class (the object) of a triple. Consider the triple pattern $(x, type, y)$, its rewriting is $\tau((x, type, y)) = (x, type.sc^*, y)$. Obviously the rewriting contains the original one. Importantly, the rewriting extends the result set and does not affect the containment relation. Thus, $\tau(q) \sqsubseteq^{\mathrm{rdf}}_{\mathcal{S}} \tau(q')$.

(Inductive case) q and q' are made of graph patterns: since the rewriting of graph patterns is obtained from the rewritings of triple patterns by induction, and using the *base case*, we have that the rewriting does not affect the containment relationship. Thus, it follows that $q \sqsubseteq^{\mathrm{rdfs}}_{\mathcal{S}} q' \Leftrightarrow \tau(q) \sqsubseteq^{\mathrm{rdf}}_{\mathcal{S}} \tau(q')$. □

Theorem 8 (Soundness and Completeness)**.** *Given an RDF schema* $\mathcal{S}$*, SPARQL queries* q *and* q'*, and a rewriting function* τ*,* $\tau(q) \sqsubseteq_{\mathcal{S}} \tau(q') \Leftrightarrow \Phi_S \wedge \mathcal{A}(\tau(q)) \wedge \neg\mathcal{A}(\tau(q')) \wedge \varphi_r$ *is unsatisfiable.*

Proof. The proof of this theorem follows from that of Proposition 7 and Theorem 4. □

5.1.3 Encoding the Schema

In this third approach, in order to determine query containment under the simpleRDFS entailment regime, we encode the schema triples (axioms) as formulae. As a consequence, the models are constrained to satisfy the encoding of the axioms. We consider *subclass*, *subproperty*, *domain*, and *range*, and *transitivity closure* (Tr(sc) or Tr(sp)) schema axioms.

Definition 21. *Given a set of axioms $s_1, s_2, ..., s_n$ of a schema $\mathcal{S}$, the μ-calculus encoding of $\mathcal{S}$ is: $\eta(\mathcal{S}) = \eta(s_1) \wedge \eta(s_2) \wedge ... \wedge \eta(s_n)$.*
We use a function η to translate each s_i into an equivalent μ-calculus formula:

$$\begin{aligned}
\eta((C_1, \texttt{sc}, C_2)) &= \mathit{lfp}\big(X, C_1 \Rightarrow C_2\big)\\
\eta((R_1, \texttt{sp}, R_2)) &= \mathit{lfp}\big(X, R_1 \Rightarrow R_2\big)\\
\eta((R, \texttt{dom}, C)) &= \mathit{lfp}\big(X, \langle s\rangle(\langle p\rangle R \Rightarrow \langle p\rangle\texttt{type} \wedge \langle o\rangle C)\big)\\
\eta((R, \texttt{range}, C)) &= \mathit{lfp}\big(X, \langle \bar{o}\rangle\langle p\rangle R \Rightarrow \langle s\rangle(\langle p\rangle\texttt{type} \wedge \langle o\rangle C)\big)\\
\eta(Tr(sc)) &= \mathit{gfp}\big(X, \mu Y.\langle p\rangle sc \wedge \langle o\rangle\top \vee \langle p\rangle sc \wedge \langle s\rangle Y\big)\\
\eta(Tr(sp)) &= \mathit{gfp}\big(X, \mu Y.\langle p\rangle sp \wedge \langle o\rangle\top \vee \langle p\rangle sp \wedge \langle s\rangle Y\big)
\end{aligned}$$

Note here that transitive RDFS properties sc *and* sp *are encoded as transitive closures:* $\texttt{sc}^+$ *and* $\texttt{sp}^+$*, respectively.*

Lemma 6. *Given a set of RDF schema axioms $\mathcal{C} = \{c_1, \cdots, c_n\}$, $\mathcal{C}$ has a model iff $\eta(\mathcal{C})$ is satisfiable.*

Proof. The proof of this lemma is immediate from the proof of Lemma 1. □

Theorem 9 (Soundness and Completeness)**.** *Given queries q, q', and a set of RDF schema axioms $\mathcal{S}$, $q \sqsubseteq^{\mathcal{S}}_{\text{rdfs}} q'$ if and only if $\eta(\mathcal{S}) \wedge \mathcal{A}(q) \wedge \neg\mathcal{A}(q') \wedge \varphi_r$ is unsatisfiable.*

Proof. The proof of this theorem follows from that of Theorem 2 and Theorem 3. □

5.2 Containment under OWL-$\mathcal{ALCH}$ Entailment Regime

As SPARQL is being extended to become a query language for ontologies (RDFS and OWL), it is important to study the problem of containment under a knowledge base schema. SPARQL's basic graph pattern is designed in such a way that it can be extended to query different ontology languages. Since 2009, the W3C SPARQL working group is extending SPARQL for RDFS, OWL, and RIF ontology and rule languages respectively. SPARQL is based on simple graph matching (i.e., based on simple entailment) and is not able to include implicitly stored answers that require reasoning. Thus, in order to be able to query ontologies other than based on simple entailment. It is necessary to extend the entailment regime for the respective ontology language. This allows to generate more results that are inferred using the semantics of the axioms in the ontology.

For this work, we consider a fragment of OWL 2 which is based on the description logic $\mathcal{ALCH}$. OWL is a W3C recommended knowledge representation language in the semantic web. OWL 2 has two different semantics OWL 2 direct semantics (based the description logic semantics) and OWL RDF-based semantics.

5.2.1 SPARQL Query Containment under the OWL-$\mathcal{ALCH}$ Direct semantics Entailment Regime

Checking the containment of SPARQL queries under the direct semantics (OWL DS) entailment can be reduced to encoding OWL ontology axioms and queries as μ-calculus formulas and checking the unsatisfiability of the encoding (or formula). Encoding OWL axioms (built from the DL underlying OWL, $\mathcal{SROIQ}(D)$) into μ-calculus with nominals and inverse leads to undecidability. Thus, we need to consider a fragment of OWL that alleviates this problem. Removing the features such as role inverse ($\mathcal{I}$), role composition ($\mathcal{R}$), data type restrictions ($\mathbf{D}$), (qualified) cardinality restrictions ($\mathcal{Q}$) from $\mathcal{SROIQ}(D)$ reduces the complexity from N2ExpTime to ExpTime. The OWL fragment based on this logic can be encoded into μ-calculus. A DL-based syntax and semantics of this language is already presented in Chapter 2.

5.2.1.1 OWL-$\mathcal{ALCH}$

Here we present the syntax of OWL-$\mathcal{ALCH}$ concepts, roles and axioms. OWL 2 has various syntactic notations, we use the user friendly functional style syntax[2]. The OWL functional style syntax is constructed as shown in Table 5.1 and 5.2.

Descriptions(C)	
A	A
owl:Thing	$\top$
owl:Bottom	$\bot$
ObjectUnionOf(C_1 ... C_n)	$C_1 \sqcup \cdots \sqcup C_n$
ObjectIntersectionOf(C_1 ... C_n)	$C_1 \sqcap \cdots \sqcap C_n$
ObjectComplementOf(C)	$\neg C$
ObjectSomeValuesFrom(R C)	$\exists R.C$
ObjectAllValuesFrom(R C)	$\forall R.C$
Descriptions(R)	
owl:topObjectProperty	$\top_2$
owl:bottomObjectProperty	$\bot_2$
P	P
Individuals (o)	
o	o
_:*anonymous*	_:*anonymous*

Table 5.1: OWL-$\mathcal{ALCH}$ syntax: concept, role and individual constructs.

[2] http://www.w3.org/TR/owl-primer/

OWL-$\mathcal{ALCH}$ axioms: as the name implies OWL-$\mathcal{ALCH}$ axioms lack role inverse, role composition, number restrictions, and description logic datatypes. This choice allows to retain satisfiability of containment under the μ-calculus fragment chosen for this study.

OWL syntax	DL syntax
SubClassOf(C_1 C_2)	$C_1 \sqsubseteq C_2$
EquivalentClasses(C_1 ... C_n)	$C_1 \equiv \cdots \equiv C_n$
DisjointClasses(C_1 ... C_n)	$C_i \sqcap C_j \sqsubseteq \bot, i \neq j$
SubObjectPropertyOf(R_1 R_2)	$R_1 \sqsubseteq R_2$
EquivalentObjectProperties(R_1 ... R_n)	$R_1 \equiv \cdots \equiv R_n$
ObjectPropertyDomain(R C)	$\exists R.\top \sqsubseteq C$
ObjectPropertyRange(R C)	$\top \sqsubseteq \forall R.C$
ClassAssertion(C o)	$o : C$
ObjectPropertyAssertion(R o_1 o_2)	$(o_1, o_n) : R$

Table 5.2: OWL-$\mathcal{ALCH}$ axioms.

5.2.2 SPARQL-OWL

SPARQL-OWL is proposed as a query language for OWL ontologies based on the OWL Direct Semantics entailment regime [Glimm *et al.* 2008]. In OWL direct semantics, the evaluation of a SPARQL query is done in such a way that both the query and the queried graph are translated into OWL DL ontologies where the translations are OWL structural objects with variables (SPARQL-OWL query) and OWL ontology respectively. If there exists a mapping where the SPARQL-OWL query is entailed by the ontology, then those mappings form an answer to the query. We refer the reader to [Glimm 2011] for a more detailed discussion.

SPARQL-OWL has no officially standardized syntax at the moment. We adopt the suggestions from [Glimm *et al.* 2008] and [Sirin & Parsia 2007] (for SPARQL-DL) which is based on the functional style syntax of OWL constructs. The syntax of SPARQL-OWL ontology graph patterns (OGPs) is formed from OWL axiom templates where variables can appear in the templates instead of just IRIs as given in the definition below.

Definition 22 (Syntax). *A SPARQL-OWL query is constructed inductively using*

OWL constructs as follows:

OGP ←`ClassAssertion(C o)` | `SubClassOf(C1 C2)` |
`ObjectPropertyAssertion(R o1 o2)` |
`EquivalentClasses(C1 C2)` |
`DisjointClasses(C1 C2)` |
`SubObjectPropertyOf(R1 R2)` |
`EquivalentObjectProperties(R1 R2)` |
BGP_1 *AND* BGP_2 | BGP_1 *UNION* BGP_2
`C` ::= `iri` | `var` `R` ::= `iri` | `var` `o` ::= `iri` | `var`

Example 30. *Select all departments along with their chairs.*

```
SELECT ?x ?y WHERE {
   ClassAssertion( Chair ?x ) .
   ObjectPropertyAssertion (headOf ?x ?y )
}
```

Now, we provide a compact summary of the semantics of SPARQL-OWL query evaluation under OWL-$\mathcal{ALCH}$ DS entailment regime. A detailed discussion with insightful examples can be found in [Glimm & Krötzsch 2010, Glimm 2011, Kollia *et al.* 2011]. The following definition is taken from [Glimm 2011].

Definition 23 (SPARQL-OWL query evaluation under OWL DS entailment regime)**.** *A partial mapping function ρ is a solution for a OGP P and G under OWL Direct Semantics entailment if:*

1. *G can be mapped into an OWL DL ontology $O(G)$*
2. *P can be mapped into an extended OWL DL axioms $O(P)$*
3. *Domain of ρ is exactly the set of variables in P*
4. *Terms in the range of ρ occur in $O(G)$ or Voc(OWL): OWL vocabulary terms*
5. *If $O(P')$ obtained from $O(P)$ by replacing anonymous individuals (blank nodes) with either IRIs or blank nodes is such that:*
   $O(G) \cup O(P')$ is an OWL DL ontology and $sk(O(G)) \models_{OWLDS} sk(\rho(O(P')))$

The function $sk(.)$ replaces blank nodes with fresh IRIs (IRIs that are neither in the queried graph nor in the query). $\models_{OWLDS}$ denotes the OWL DS entailment relation.

Definition 24 (Containment of SPARQL-OWL queries)**.** *Given two SPARQL-OWL queries q and q', and a set of OWL-$\mathcal{ALCH}$ TBox axioms $\mathcal{S}$, q is contained in q' under the OWL-$\mathcal{ALCH}$ entailment regime, denoted as $q \sqsubseteq^{\text{owl}}_{\mathcal{S}} q'$, if and only if for any RDF graph G that satisfies the axioms $\mathcal{S}$, $[\![q]\!]_G \subseteq [\![q']\!]_G$.*

5.2.2.1 Encoding SPARQL-OWL Queries

The encoding of SPARQL-OWL queries is slightly different from that of SPARQL queries due to the basic graph patterns. In SPARQL-OWL, basic graph patterns can contain disjunctions and conjunctions of concepts, for instance consider the following query and its respective translation into a SPARQL query (by introducing blank nodes):

```
SELECT ?x WHERE {
  ClassAssertion (ObjectUnionOf
     ( AdminStaff  Professor )  ?x )
}
```

```
SELECT ?x WHERE {
 { ?x a AdminStaff .}
 UNION
 {?x a Professor .}
}
```

Here, we discuss the encoding of basic graph patterns as the encoding of the other query constructs and containment problem is similar to the one in [Chekol *et al.* 2012b] and Chapter 3. Concept and role constructors are encoded as in a similar way as that of $\mathcal{ALCH}$ concepts and roles whereas axioms that appear in a BGP are encoded inductively as shown in Definition 25.

Definition 25 (Encoding SPARQL-OWL queries)**.** *The encoding of SPARQL-OWL*

query q is $\mathcal{A}(q)$ *such that:*

$$\begin{aligned}
\mathcal{A}(OGP) &= lfp\big(X, \lambda(BGP)\big) \\
\mathcal{A}(q_1 \textit{ AND } q_2) &= \mathcal{A}(q_1) \wedge \mathcal{A}(q_2) \\
\mathcal{A}(q_1 \textit{ UNION } q_2) &= \mathcal{A}(q_1) \vee \mathcal{A}(q_2)
\end{aligned}$$

$$\begin{aligned}
\lambda\big(\texttt{ClassAssertion(C o)}\big) &= \langle \bar{s} \rangle \lambda(o) \wedge \langle p \rangle type \wedge \langle o \rangle \lambda(C) \\
\lambda\big(\texttt{ObjectPropertyAssertion(R } o_1 \texttt{ } o_2\texttt{)}\big) &= \lambda(o_1) \wedge \langle p \rangle P \wedge \langle o \rangle \lambda(o_2) \\
\lambda(A) &= A \\
\lambda(IRI) &= IRI \\
\lambda(var) &= \begin{cases} var \; if \; var \in distinguished\ variable \\ \top \; if \; var \in non\text{-}distinguished\ variable \end{cases} \\
\lambda(\texttt{ObjectUnionOf(}C_1 \texttt{ } C_2\texttt{)}) &= \lambda(C_1) \vee \lambda(C_2) \\
\lambda(\texttt{ObjectIntersecectionOf(}C_1 \texttt{ } C_2\texttt{)}) &= \lambda(C_1) \wedge \lambda(C_2) \\
\lambda(\texttt{ObjectComplementOf(C)}) &= \neg\lambda(C) \\
\lambda(\texttt{SubClassOf(}C_1 \texttt{ } C_2\texttt{)}) &= \langle \bar{s} \rangle \lambda(C_1) \wedge \langle p \rangle subClassOf \wedge \langle o \rangle \lambda(C_2) \\
\lambda(\texttt{EquivalentClasses(}C_1 \ldots C_n\texttt{)}) &= \big(\langle \bar{s} \rangle \lambda(C_1) \wedge \langle p \rangle subClassOf \wedge \langle o \rangle \lambda(C_2)\big) \wedge \\
&\quad \big(\langle \bar{s} \rangle \lambda(C_2) \wedge \langle p \rangle subClassOf \wedge \langle o \rangle \lambda(C_1)\big) \\
\lambda(\texttt{ObjectSomeValuesFrom(R C)}) &= \big(\langle s \rangle (\langle p \rangle a \wedge \langle o \rangle owl : Restriction) \wedge \\
&\quad \langle s \rangle (\langle p \rangle owl : onProperty \wedge \langle o \rangle \lambda(R)) \wedge \\
&\quad \langle s \rangle (\langle p \rangle owl : someValuesFrom \wedge \langle o \rangle \lambda(C))\big) \\
\lambda(\texttt{ObjectAllValuesFrom(R C)}) &= \big(\langle s \rangle (\langle p \rangle a \wedge \langle o \rangle owl : Restriction) \wedge \\
&\quad \langle s \rangle (\langle p \rangle owl : onProperty \wedge \langle o \rangle \lambda(R)) \wedge \\
&\quad \langle s \rangle (\langle p \rangle owl : allValuesFrom \wedge \langle o \rangle \lambda(C))\big) \\
\lambda(\texttt{SubObjectPropertyOf(}R_1 \texttt{ } R_2\texttt{)}) &= \langle \bar{s} \rangle \lambda(R_1) \wedge \langle p \rangle subPropertyOf \wedge \langle o \rangle \lambda(R_2) \\
\lambda(\texttt{ObjectPropertyDomain(R C)}) &= \langle \bar{s} \rangle \lambda(R) \wedge \langle p \rangle domain \wedge \langle o \rangle \lambda(C) \\
\lambda(\texttt{ObjectPropertyRange(R C)}) &= \langle \bar{s} \rangle \lambda(R) \wedge \langle p \rangle range \wedge \langle o \rangle \lambda(C)
\end{aligned}$$

Example 31 (Encoding of a query)**.** *Consider the inductive encoding of the following SPARQL-OWL query q:*

```
SELECT ?x
WHERE {
   SubClassOf(:GraduateStudent
         ObjectSomeValuesFrom(:takesCourse ?x))
}
```

$$
\begin{aligned}
\mathcal{A}(q) &= lfp\big(X, \lambda(BGP)\big) \\
\lambda(BGP) &= \langle \bar{s} \rangle \lambda(GraduateStudent) \wedge \langle p \rangle subClassOf \wedge \\
&\quad \langle o \rangle \lambda(ObjectSomeValuesFrom(takesCourse\ ?x)) \\
&= \langle \bar{s} \rangle \lambda(GraduateStudent) \wedge \langle p \rangle subClassOf \wedge \\
&\quad \langle o \rangle \big(\langle s \rangle (\langle p \rangle a \wedge \langle o \rangle owl:Restriction) \wedge \\
&\quad\quad \langle s \rangle (\langle p \rangle owl:onProperty \wedge \langle o \rangle \lambda(takesCourse)) \wedge \\
&\quad\quad \langle s \rangle (\langle p \rangle owl:someValuesFrom \wedge \langle o \rangle \lambda(?x)) \big) \\
&= \langle \bar{s} \rangle \lambda(GraduateStudent) \wedge \langle p \rangle subClassOf \wedge \\
&\quad \langle o \rangle \big(\langle s \rangle (\langle p \rangle a \wedge \langle o \rangle owl:Restriction) \wedge \\
&\quad\quad \langle s \rangle (\langle p \rangle owl:onProperty \wedge \langle o \rangle takesCourse) \wedge \\
&\quad\quad \langle s \rangle (\langle p \rangle owl:someValuesFrom \wedge \langle o \rangle \top) \big)
\end{aligned}
$$

Containment of SPARQL-OWL queries is reduced to the validity problem in μ-calculus. When encoding the left-hand side and the right-hand side queries using function $\mathcal{A}$ in Definition 25, all the IRIs (concept and role names, OWL and RDFS vocabularies) and variables become nominals in the μ-calculus.

5.2.2.2 Encoding OWL-$\mathcal{ALCH}$ TBox Axioms

The encoding of OWL-$\mathcal{ALCH}$ axioms can be produced by working inductively on the construction of the axiom.

Definition 26 (Encoding OWL-$\mathcal{ALCH}$ axioms)**.** *The μ-calculus encoding of TBox*

axioms is formulated using the function η as shown below:

$$
\begin{aligned}
\eta(\top) &= \top \\
\eta(A) &= A \\
\eta(P) &= P \\
\eta(\texttt{ObjectUnionOf}(\texttt{C}_1 \cdots \texttt{C}_n)) &= \eta(C_1) \vee \cdots \vee \eta(C_n) \\
\eta(\texttt{ObjectIntersectionOf}(\texttt{C}_1 \cdots \texttt{C}_n)) &= \eta(C_1) \wedge \cdots \wedge \eta(C_n) \\
\eta(\texttt{ObjectComplementOf}(\texttt{C})) &= \neg\eta(C) \\
\eta(\texttt{ObjectSomeValuesFrom}(\texttt{R C})) &= \langle s\rangle\big(\langle p\rangle\eta(R) \wedge \langle o\rangle(\langle s\rangle\langle o\rangle\eta(C))\big) \\
\eta(\texttt{ObjectAllValuesFrom}(\texttt{R C})) &= [s]\big([p]r \Rightarrow [o]([s][o]\eta(C))\big) \\
\eta(\texttt{SubClassOf}(\texttt{C}_1\ \texttt{C}_2)) &= \mathit{lfp}\big(X, \eta(C_1) \Rightarrow \eta(C_2)\big) \\
\eta(\texttt{EquivalentClasses}(\texttt{C}_1 \ldots \texttt{C}_n)) &= \mathit{lfp}\big(X, \eta(C_1) \Leftrightarrow \cdots \Leftrightarrow \eta(C_n)\big) \\
\eta(\texttt{DisjointClasses}(\texttt{C}_1 \cdots \texttt{C}_n)) &= \mathit{lfp}\big(X, (\eta(C_1) \wedge \cdots \wedge \eta(C_n)) \Rightarrow \bot\big) \\
\eta(\texttt{SubObjectPropertyOf}(\texttt{R}_1\ \texttt{R}_2)) &= \mathit{gfp}\big(X, \eta(R_1) \Rightarrow \eta(R_2)\big) \\
\eta(\texttt{EquivalentObjectProperties}(\texttt{R}_1\ \ldots \texttt{R}_n)) &= \mathit{gfp}\big(X, \eta(R_1) \Leftrightarrow \cdots \Leftrightarrow \eta(R_n)\big) \\
\eta(\texttt{ObjectPropertyDomain}(\texttt{R C})) &= \eta(\texttt{SubClassOf}(\\
&\qquad \texttt{ObjectSomeValuesFrom}(\texttt{R}\ \top)\ \texttt{C})) \\
\eta(\texttt{ObjectPropertyRange}(\texttt{R C})) &= \eta(\texttt{SubClassOf}(\top \\
&\qquad \texttt{ObjectAllValuesFrom}(\texttt{R C})))
\end{aligned}
$$

Using the above definition, we introduce the following: given an OWL-$\mathcal{ALCH}$ TBox schema $\mathcal{S} = \{s_1, \ldots, s_n\}$, its encoding can be produced by using:

$$\eta(\mathcal{S}) = \eta(s_1) \wedge \cdots \wedge \eta(s_n)$$

So far, we have managed to define procedures to translate SPARQL-OWL queries and OWL-$\mathcal{ALCH}$ axioms into μ-calculus formulas. Now, it remains to reduce SPARQL-OWL containment under OWL-$\mathcal{ALCH}$ direct semantics entailment regime into unsatisfiability test. Given two SPARQL-OWL queries q and q', and a set of OWL-$\mathcal{ALCH}$ axioms $\mathcal{S}$, $q \sqsubseteq_{\mathcal{S}} q'$ if and only if $\mathcal{A}(q) \wedge \neg\mathcal{A}(q') \wedge \varphi_r \wedge \eta(S)$. Before proving this argument, Theorem 10, we demonstrate the reduction process with an example below.

Example 32 (Containment test)**.** *Consider the following queries:*

- *q: Select any two students that take the same course.*

```
SELECT ?x ?y
WHERE {
  ClassAssertion(Student ?x) .
  ClassAssertion(Student ?y) .
  ObjectPropertyAssertion(takesCourse ?x ?z) .
  ObjectPropertyAssertion(takesCourse ?y ?z) .
}
```

- q': *Select any two students that take the same graduate course.*

```
SELECT ?x ?y
WHERE {
  ClassAssertion(Student ?x) .
  ClassAssertion(Student ?y) .
  ObjectPropertyAssertion(takesCourse ?x ?z) .
  ObjectPropertyAssertion(takesCourse ?y ?z) .
  ClassAssertion(GraduateCourse ?z) .
}
```

It is obvious that $q' \sqsubseteq q$ *and* $q \not\sqsubseteq q'$ *since the set of any two students taking the same graduate courses is included in the set of any two students taking the same courses. In fact, this argument also holds if the containment test is performed under the OWL DS entailment regime. This can be proved by translating the queries into formulas and checking the existence of a model for this formula i.e.,* $\mathcal{A}(q) \wedge \neg\mathcal{A}(q')$.

$$
\begin{aligned}
\mathcal{A}(q) = (&lfp\big(X, \langle\bar{s}\rangle x \wedge \langle p\rangle a \wedge \langle o\rangle Student\big) \\
&\wedge lfp\big(X, \langle\bar{s}\rangle y \wedge \langle p\rangle a \wedge \langle o\rangle Student\big) \\
&\wedge lfp\big(X, \langle\bar{s}\rangle x \wedge \langle p\rangle takesCourse \wedge \langle o\rangle z\big) \\
&\wedge lfp\big(X, \langle\bar{s}\rangle y \wedge \langle p\rangle takesCourse \wedge \langle o\rangle z\big)) \\
&\bigwedge \\
\neg\mathcal{A}(q') = (&gfp\big(X, [\bar{s}]\neg x \vee [p]\neg a \vee [o]\neg Student\big) \\
&\vee gfp\big(X, [\bar{s}]\neg y \vee [p]\neg a \vee [o]\neg Student\big) \\
&\vee gfp\big(X, [\bar{s}]\neg x \vee [p]\neg takesCourse \vee [o]\neg z\big) \\
&\vee gfp\big(X, [\bar{s}]\neg y \vee [p]\neg takesCourse \vee [o]\neg z\big) \\
&\vee gfp\big(X, [\bar{s}]\neg z \vee [p]\neg a \vee [o]\neg GraduateCourse\big))
\end{aligned}
$$

$\mathcal{A}(q) \wedge \neg\mathcal{A}(q')$ *is satisfiable due to the disjunct*

$$gfp\big(X, [\bar{s}]\neg z \vee [p]\neg a \vee [o]\neg GraduateCourse\big).$$

That is to say, if a tableau procedure is used to determine the satisfiability of this formula the branch containing that disjunct will remain open [Tanabe et al. 2008]. In line with this, a similar argument can be sketched in order to check the unsatisfiability of $\mathcal{A}(q') \wedge \neg\mathcal{A}(q)$.

Reducing containment of SPARQL-OWL queries under OWL-$\mathcal{ALCH}$ DS entailment regime into unsatisfiability test in the μ-calculus is proved in the following theorem.

Theorem 10 (Reducing containment to unsatisfiability)**.** *Given two SPARQL-OWL queries* q *and* q'*, and a set of OWL-*$\mathcal{ALCH}$ *TBox axioms* $\mathcal{S}$*,* $q \sqsubseteq^{\text{owl}}_{\mathcal{S}} q'$ *if and only if* $\mathcal{A}(q) \wedge \neg\mathcal{A}(q') \wedge \eta(\mathcal{S}) \wedge \varphi_r$ *is unsatisfiable.*

Proof. The proof follows directly from the proof of SPARQL query containment under $\mathcal{ALCH}$ axioms (cf. Chapter 3). To achieve this, SPARQL-OWL queries are translated into SPARQL queries, basically this is as simple as translating function-style syntax BGP's into Turtle syntax BGP's using blank nodes (as shown in [Glimm 2011]). Equivalently, (function-style syntax) OWL-$\mathcal{ALCH}$ axioms become just $\mathcal{ALCH}$ axioms. Here, the nominals in the axioms or individual nominals become nominals in the μ-calculus Thus, the respective translations can be encoded as μ-calculus formulas and reduced to unsatisfiability test (cf. the proof of Theorem 2). □

Complexity Reducing the containment problem $q \sqsubseteq q'$ to a μ-calculus formula $\mathcal{A}(q) \wedge \neg\mathcal{A}(q')$ requires a polynomial amount of time. Testing whether the formula is unsatisfiable can be done in ExpTime.

Under OWL DS entailment semantics there are no non-distinguished variables since answers are computed from the basic graph patterns of the query. Projection is performed, after the evaluation of basic graph patterns, i.e., after all the variables are bound to values in the ontology. Note that the ExpTime complexity jumps to 2ExpTime if non-distinguished variables are introduced and treated as existentially quantified variables.

Proposition 8 (Complexity)**.** *The complexity of determining the containment between two SPARQL-OWL queries under OWL-$\mathcal{ALCH}$ DS entailment regime $q \sqsubseteq_{\mathcal{S}} q'$ is 2^n. Where $n = \mathcal{O}(|\mathcal{A}(q)| + |\mathcal{A}(q')| + |\eta(\mathcal{S})|)$.*

Proof. This result follows from containment of SPARQL queries under $\mathcal{ALCH}$ schema axioms (cf. Section 3). Since there are no non-distinguished variables in the queries, we use the same encoding function $\mathcal{A}$ for both the left and right-hand side queries. Consequently, we obtain that the size of the encoding of the right-hand side query is not exponential, i.e., it is linear in the size of the query. Thus, the size of the overall encoding is linear, i.e., $n = \mathcal{O}(|\mathcal{A}(q)| + |\mathcal{A}(q')| + |\eta(\mathcal{S})|)$. In μ-calculus, testing the satisfiability of a formula requires exponential amount of time. Additionally, knowledge base satisfiability test in $\mathcal{ALCH}$ is ExpTime. As a consequence, containment under OWL-$\mathcal{ALCH}$ DS entailment regime can be done in a time of 2^n. □

5.3 Conclusion

In this chapter, we have presented procedures to produce the encoding of queries, inference rules and schema as formulas. Henceforth, query containment under simpleRDFS entailment is reduced to formula satisfiability test in the μ-calculus. We introduced three approaches to achieve this, namely (1) encoding the RDFS semantics, (2) query rewriting, and (3) encoding the schema. Unlike (1) and (2), the third approach can be extended for a more expressive schema language as done in Section 6.3.1. The power of the logic and our encoding allows one to take advantage of more expressive schema languages. For instance, a good candidate could be the description logic $\mathcal{SROIQ}$ [Horrocks *et al.* 2006] underlying OWL 2. In that direction, we have produced an ExpTime

procedure for determining containment of SPARQL-OWL queries under OWL-$\mathcal{ALCH}$ DS entailment regime. It is very important to note that, this bound rises to 2ExpTime if non-distinguished variables appear in the right-hand side of the query, i.e., if non-distinguished variables are treated as having existential semantics in the evaluation of basic graph patterns.

Chapter 6

Containment of Tree-structured SPARQL Queries

Contents

In general, a query is referred to as cyclic if the graph structure induced from the query is cyclic. It has been long noted that cyclic queries contribute largely in the complexity of containment and equivalence problems. Most notably, Chandra and Merlin 1977 [Chandra & Merlin 1977] proved that containment and equivalence of relational conjunctive queries is NP-complete. However, this complexity reduces to PTime if the queries are acyclic [Bernstein & Chiu 1981, Yannakakis 1981]. Further, the study in [Calvanese *et al.* 2008] demonstrated that containment of $\mathcal{DLR}$ (description logics with n-ary relations) conjunctive queries is double exponential but this complexity bound reduces to exponential if the query on the right-hand side of the containment has a tree structure. In other words, cycles among the non-distinguished variables contribute largely in containment and equivalence complexity. Meanwhile, we observed that most of the real world queries currently used are acyclic. Hence, studying acyclic query containment is relevant.

In this chapter, we address the problem of query containment for tree-structured SPARQL queries. The objective of this chapter is that when the queries have tree-structure, the complexity of the problem can reduce to PSPACE in the absence of constraint axioms. We show that this is possible, by reducing containment to unsatisfiability in the modal logic K_n. We first encode queries as K_n formulas. Then, we reduce query containment to unsatisfiability in the logic. And finally, we prove the correctness of this reduction and provide experimental results.

6.1 Tree-structured SPARQL Queries

In order to create a compliance benchmark (see Chapter 7) and assess the current state-of-the-art, for this study, we have identified a class of SPARQL queries called

tree-structured queries. These are queries that have a tree structure when seen as a graph. In the following, we introduce this notion.

In relational databases, Bernstein and Chiu [Bernstein & Chiu 1981] classified queries into two types: *tree* (is an undirected graph in which any two vertices are connected by exactly one simple path) and *cyclic* queries. An algorithm to decide whether a query is cyclic or not was presented in their paper. This algorithm is based on the idea of representing queries as hypergraphs. In fact, queries have often been considered as hypergraphs (see [Chekuri & Rajaraman 1997, Bernstein & Chiu 1981] for instance). It is possible to represent queries as hypergraphs where the nodes of the hypergraph are the variables and constants in the query. There is one hyperedge corresponding to each query subgoal that includes the variables and constants occurring in that subgoal. These studies proved that (Boolean) conjunctive query evaluation, while NP-hard in general, is polynomial in case of acyclic queries, i.e., queries whose associated hypergraphs are acyclic [Yannakakis 1981]. This distinction of queries as tree and cyclic has its advantages: for example, the evaluation of acyclic (Boolean) conjunctive queries is highly parallelizable [Gottlob *et al.* 2001].

Borrowing this notion from databases, we propose to view SPARQL queries as graphs. More specifically, a SPARQL query is represented as a bipartite graph, with two kinds of nodes: triple nodes and term nodes (are URIs, blank nodes, and literals). Using this graph, one is able to determine whether a query is cyclic or not, a formal explanation is given in Definition 11.

To the best of our knowledge, no experimental work has been conducted to verify how many of real world queries are acyclic or cyclic. To answer this question, we analyzed the DBpedia query log[1] and obtained 378,530 real world queries, out of which 15.8% were syntactically incorrect. Using the remaining 74.2%, we tested the cyclicness and acyclicness of queries and found out that: more than 87% of these queries are acyclic. This suggests that acyclic queries make up a major part of the real world queries, thus, their separate study is justified. Precise results on the analysis of the structure of queries (tree, directed acyclic graph (DAG), and cyclic) are presented in Table 6.1.

Query graph		Query type	# of queries	perc.
Acyclic	Tree	UCQs	100001	31%
		Other	50144	16%
	DAG	UCQs	50355	16%
		Other	95722	30%
Cyclic			22591	7%
Incorrect syntax			59717	

Table 6.1: Query characteristics in DBPedia logs.

[1] DBpedia 3.5.1 logs between 30/04/2010 and 20/07/2010 from ftp://download.openlinksw.com/support/dbpedia/

Beyond the cyclic and acyclic tests, we checked how many of the queries have projection, i.e., not all variables in the graph pattern are distinguished, or not. We found out that 63% of the queries have projection and 37% of the queries have no projection. Further, all of the cyclic queries have projection and out of the acyclic ones, 65% of the queries have projection and the rest have no projection. Concerning OPT , 88964 queries use it, i.e., 23.5%. In addition, only 40448 (10.6%) of which are conjunctive queries. These results and the state of tools motivate the design principles behind the benchmark structure in Chapter 7.

6.2 Encoding Queries as K_n Formulae

Triples in queries can be taken as paths in the transition system, hence triple patterns of queries can be expressed by a formula which represents this path. For instance, a query which contains the triple pattern $q\{x\} = (x, a, b)$ can be expressed as $x \wedge \langle s \rangle (\langle p \rangle a \wedge \langle o \rangle b)$ which states that x is satisfied if there exists a path in a transition system where starting from this node labelled with x, nodes with labels a and b can be reached by programs s, p and s, o respectively. In fact, variables are replaced by atomic propositions that may be satisfied in a state of the transition system. In the example, the variable x becomes an atomic proposition (AP).

In this way, the encoding of triple patterns is done by using the function $\mathcal{P}'$, given in Definition 28. $\mathcal{P}$ produces a formula by computing a sequence of transitions (or modalities) starting from a given distinguished variable (which is treated as a point of evaluation or focus, for instance in $\mathcal{M}, w_f \models \varphi$, we call w_f a focus). As an argument it takes a focus, a URI u or variable w_f for which the transitions are to be computed and the query itself q.

On the other hand, the function $\mathcal{A}$ in Definition 27 encodes queries inductively on the structure of query patterns. AND and UNION are replaced by boolean connectives $\wedge$ and $\vee$ respectively.

Definition 27. *The encoding of a SPARQL query $q\{\overrightarrow{w}\}$ is $\mathcal{A}(q, q, w_f)$ such that:*

$$\begin{aligned} \mathcal{A}(q, t, w_f) &= \mathcal{P}(q, t, w_f) \text{ where } w_f \in \overrightarrow{w} \text{ is a distinguished variable.} \\ \mathcal{A}(q, q_1 \text{ AND } q_2, w_f) &= \mathcal{A}(q, q_1, w_f) \wedge \mathcal{A}(q, q_2, w_f) \\ \mathcal{A}(q, q_1 \text{ UNION } q_2, w_f) &= \mathcal{A}(q, q_1, w_f) \vee \mathcal{A}(q, q_2, w_f) \end{aligned}$$

If the focus w_f does not appear in a triple pattern, then we can use a function, say $\mathcal{P}$, to recursively construct the formula. In turn, this function can use $\mathcal{P}'$ shown in Definition 28.

Definition 28. *The function $\mathcal{P}'$ computes a formula from a given triple pattern $t =$*

(x, y, z) and focus w_f as follows:

$$\begin{aligned}
\mathcal{P}' : t \times w_f &\rightarrow \varphi \in \mathsf{K}_n \\
\mathcal{P}'((x,y,z),x) &= x \wedge \langle s \rangle (\langle p \rangle y \wedge \langle o \rangle z) \\
\mathcal{P}'((x,y,z),y) &= y \wedge \langle \bar{p} \rangle (\langle \bar{s} \rangle x \wedge \langle o \rangle z) \\
\mathcal{P}'((x,y,z),z) &= z \wedge \langle \bar{o} \rangle (\langle \bar{s} \rangle x \wedge \langle p \rangle z)
\end{aligned}$$

Example 33. *Consider the encoding of $q\{x\} = (x, a, b)$ using $\mathcal{P}$.*

$$\begin{aligned}
\mathcal{P}(q,t,x) &= \mathcal{P}((x,a,b),(x,a,b),x) \\
&= \mathcal{P}'((x,a,b),x) \\
&= x \wedge \langle s \rangle (\langle p \rangle a \wedge \langle o \rangle b)
\end{aligned}$$

In the encoding function $\mathcal{A}$, q appears multiple times, this duplication allows to facilitate the encoding process. It does not contribute to the duplication of formulas in the encoding output. Hence, this keeps the translation linear.

Example 34 (Query encoding)**.** *The recursive encoding of query q_1 of Example 8, $\mathcal{A}(q_1, q_1, x)$, is:*

$$\begin{aligned}
\mathcal{A}(q_1,q_1,x) &= \mathcal{A}(q, ((x, translated, l) \ \textsc{union} \ (x, wrote, l)) \ \textsc{and} \ (l, type, poem), x) \\
&= (\mathcal{A}(q,(x,translated,l),x) \vee \mathcal{A}(q,(x,wrote,l),x)) \\
&\quad \wedge \mathcal{A}(q,(l,type,poem),x) \\
&= (\mathcal{P}(q,(x,translated,l),x) \vee \mathcal{P}(q,(x,wrote,l),x)) \\
&\quad \wedge \mathcal{P}(q,(l,type,poem),x) \\
&= (x \wedge \langle s \rangle (\langle p \rangle translated \wedge \langle o \rangle l) \quad \vee \quad x \wedge \langle s \rangle (\langle p \rangle wrote \wedge \langle o \rangle l)) \quad \wedge \\
&\quad \langle s \rangle \langle o \rangle (l \wedge \langle s \rangle (\langle p \rangle type \wedge \langle o \rangle poem)) \\
&= (x \wedge \langle s \rangle (\langle p \rangle translated \wedge \langle o \rangle l) \quad \vee \quad x \wedge \langle s \rangle (\langle p \rangle wrote \wedge \langle o \rangle l)) \quad \wedge \\
&\quad \langle s \rangle \langle o \rangle (l \wedge \langle s \rangle (\langle p \rangle type \wedge \langle o \rangle poem))
\end{aligned}$$

Alternative approach for encoding tree-structured queries Alternatively, another approach for encoding tree-structured queries is by using preorder traversal of a tree. First, we construct a tree out of a given query, then we traverse the tree using a focus (a distinguished variable). The principle of the encoding is that a tree is constructed from the basic graph patterns (BGPs) of the query using a function called $constTree(.)$. The construction of the tree is shown in Definiton 29. Once the tree is obtained a preorder traversal starting from a distinguished variable (which is the focus and root of the tree) is applied to construct the formula, we use a function called $dft(T, w)$ for this task where T is the tree construction of a BGP and w is a distinguished variable or root of the tree. This procedure is best described by the following example.

The distinguished variables and the URIs are encoding as atomic propositions

whereas the non-distinguished variables become $\top$.

Example 35. *Consider the following query composed of a BGP.*

```
SELECT ?x
WHERE {
  ?x a :Student .
  ?x :takesCourse ?y .
  ?y a :GraduateCourse .
  ?x :born ?z .
  ?z :locatedIn ?r .
  ?r a :Country .
}
```

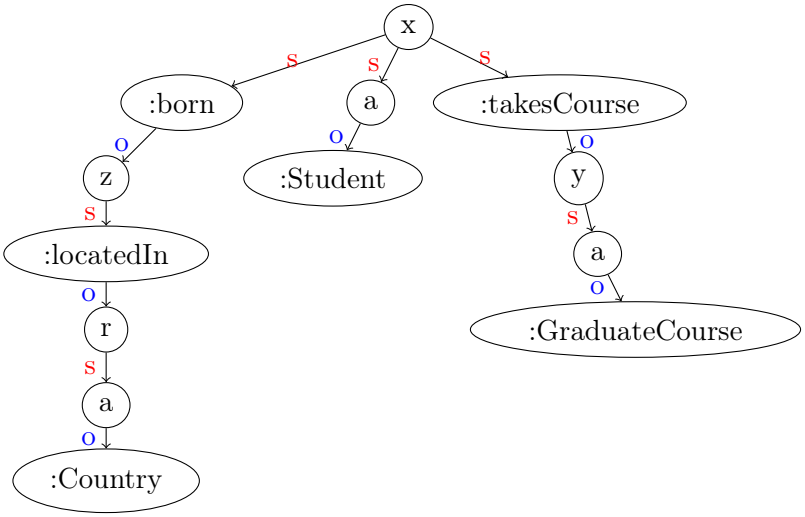


Figure 6.1: A tree obtained from the BGP of the query

A preorder traversal of the tree results in:

$$x \wedge \langle s\rangle(born \wedge \langle o\rangle(z \wedge \langle s\rangle(locatedIn \wedge \langle o\rangle(r \wedge \langle s\rangle(a \wedge \langle o\rangle Country))))\wedge$$
$$\langle s\rangle(a \wedge \langle o\rangle Student)\wedge$$
$$\langle s\rangle(takesCourse \wedge \langle o\rangle(y \wedge \langle s\rangle(a \wedge \langle o\rangle GraduateCourse)))$$

In order to encode UNION graph patterns, we transform queries into union normal form [Pérez *et al.* 2009] and then apply the procedure inductively.

Definition 29 (Constructing a tree from a BGP)**.** *For each triple* $(x, y, z) \in BGP$*, a graph* $T = (N, E)$ *can be obtained as follows:*

- $x, y, z \in N$,
- $(x, y), (y, z) \in E$,

– $l_E((x,y)) = s$ *and* $l_E((y,z)) = o$ *are labellings of edges.*

We refer to this procedure as $constTree(BGP)$.

Definition 30 (Translation Procedure)**.** *Tree-structured union normal form SPARQL queries can be translated into* K_n *formulas inductively as follows:*

$$\begin{aligned} \mathcal{A}(t, \overrightarrow{w}) &= dft(constTree(BGP), \overrightarrow{w}) \\ \mathcal{A}(BGP, \overrightarrow{w}) &= dft(constTree(BGP), \overrightarrow{w}) \\ \mathcal{A}(P_1 \ \textsc{union} \ P_2, \vec{w}) &= \mathcal{A}(P_1, \overrightarrow{w}) \vee \mathcal{A}(P_2, \overrightarrow{w}) \end{aligned}$$

Where $dft(T, w)$ is a recursive preorder traversal function.

So far, we have provided two alternative ways to encode tree-structured queries. In both methods, the translation is linear. Next we provide theorems together with their respective proofs in order to verify the correctness of the encoding. Theorem 12 ensures the soundness and completeness of the encoding. Note that the proof is done for unary queries but it can be extended for n-ary queries. For the sake of legibility in the proofs, we denote the query encoding function $\mathcal{A}(q, q, w)$ by $\mathcal{A}(q, w)$.

Theorem 11. *For any graph* G *and query* $q\{w\}$,

$$(\{w \to u\} \uplus \rho') \in [\![q\{w\}]\!]_G \Leftrightarrow \sigma(G), n_u \models \mathcal{A}(q, w)$$

Proof. The proof is by induction on the structure of the query.
(Base case): ($\Rightarrow$) $q\{x\} = (x, y, z)$. If $(\{w \to u\} \uplus \rho') = \rho \in [\![q]\!]_G$, this implies $(\rho(x), \rho(y), \rho(z)) \in G$. Hence $\sigma(G)$ contains:

- $t \in S'', n_{\rho(x)}, n_{\rho(y)}, n_{\rho(z)} \in S'$
- $(n_{\rho(x)}, t) \in R(s), (t, n_{\rho(y)}) \in R(p), (t, n_{\rho(z)}) \in R(o)$, and
- $n_{\rho(x)} \in L(\rho(x)), n_{\rho(y)} \in L(\rho(y)), n_{\rho(z)} \in L(\rho(z))$.

The formula $x \wedge \langle s \rangle(\langle p \rangle y \wedge \langle o \rangle z)$ evaluates to $\emptyset$ if there is no triple matching (x, y, z) in G, otherwise it evaluates to the set of all n_x states. $L(n_x)$ is the set of states which satisfy $\mathcal{A}((x, y, z), x)$ in $\sigma(G)$. Consequently,
$\sigma(G), n_x \models x \wedge \langle s \rangle(\langle p \rangle y \wedge \langle o \rangle z)$.
$\sigma(G), n_x \models \mathcal{A}((x, y, z), x)$

($\Leftarrow$) $\sigma(G), n_x \models \mathcal{A}((x, y, z), x)$ entails that there is a state $t \in S''$ and $n_{\rho(x)}$, $n_{\rho(y)}$, $n_{\rho(z)} \in S'$, such that $\langle n_{\rho(x)}, t \rangle \in R(s)$, $\langle t, n_{\rho(y)} \rangle \in R(p)$, and $\langle t, n_{\rho(z)} \rangle \in R(o)$ and $n_{\rho(x)} \in L(\rho(x))$, $n_{\rho(y)} \in L(\rho(y))$ and $n_{\rho(z)} \in L(\rho(z))$. Since the transition system $\sigma(G)$ is the encoding of an RDF graph G, this means that $(\rho(x), \rho(y), \rho(z)) \in G$. Subsequently, $\rho \in [\![(x, y, z)]\!]_G$ i.e., $\rho = \{w \to u\} \uplus \rho') \in [\![(x, y, z)]\!]_G$ for $w = x$. This concludes the proof for the base case.

(Inductive case) (for AND, the transcription for UNION is immediate):
$(\{w \to u\} \uplus \rho) \in [\![(q_1 \text{ AND } q_2)]\!]_G$

$$
\begin{aligned}
&\Leftrightarrow (\{w \to u\} \uplus \rho) \in [\![q_1\{w\}]\!]_G \uplus [\![q_2\{w\}]\!]_G \\
&\Leftrightarrow (\{w \to u\} \uplus \rho_1) \in [\![q_1\{w\}]\!]_G \text{ and } (\{w \to u\} \uplus \rho_2) \in [\![q_2\{w\}]\!]_G \\
&\quad \text{with } \rho_1 \text{ and } \rho_2 \text{ and } \{w \to u\} \text{ being compatible}^*. \\
&\Leftrightarrow \sigma(G), n_u \models \mathcal{A}(q_1, w) \text{ and } \sigma(G), n_u \models \mathcal{A}(q_2, w) \quad \text{by induction hypothesis} \\
&\Leftrightarrow \sigma(G), n_u \models \mathcal{A}(q_1, w) \wedge \mathcal{A}(q_2, w) \\
&\Leftrightarrow \sigma(G), n_u \models \mathcal{A}(q_1 \text{ AND } q_2, w)
\end{aligned}
$$

* In one direction, they are compatible because they come from ρ; in the other direction because they are extracted from the assignment satisfying the initial formula. The argument here only works if w appears on both sides of the connective AND. If it does not, then $\{w \to u\}$ is not joint to ρ_2, but the next statement is then trivially true. □

As a consequence of this theorem, we have the following corollary for unary queries by simply applying the projection operation.

Corollary 1. *For any graph G and query $q\{w\}$,*

$$\rho \in [\![q\{w\}]\!]_G \iff \sigma(G), n_{\rho(w)} \models \mathcal{A}(q, w)$$

6.3 Reducing Containment to Unsatisfiability

In this section, we address the problem of query containment, $q_1\{w\} \sqsubseteq q_2\{w\}$, by reducing it to the problem of unsatisfiability in the logic. For a proper translation of the queries we use a variable renaming function ϕ_q^w which renames all variables in q, but the distinguished variables in w, to obtain independent variables.

Theorem 12. *Given unary SPARQL queries $q_1\{w\}$ and $q_2\{w\}$, $q_1\{w\} \sqsubseteq q_2\{w\}$ iff $\mathcal{A}(q_1, w) \wedge \neg\phi_{q_1}^w(\mathcal{A}(q_2, w)) \wedge \varphi_r$ is unsatisfiable.*

Proof. $q_1\{w\} \sqsubseteq q_2\{w\}$

$$
\begin{aligned}
&\Leftrightarrow \forall G.[\![q_1\{w\}]\!]_G \subseteq [\![q_2\{w\}]\!]_G \\
&\Leftrightarrow \forall G.(\sigma(G), n_u \models \mathcal{A}(q_1, w) \Rightarrow \sigma(G), n_u \models \phi_{q_1}^w(\mathcal{A}(q_2, w))) \\
&\quad \text{by Corollary 11 and transparent renaming} \\
&\Leftrightarrow \forall G.[\![\mathcal{A}(q_1, w)]\!]^{\sigma(G)} \neq \emptyset \Rightarrow [\![\phi_{q_1}^w(\mathcal{A}(q_2, w))]\!]^{\sigma(G)} \neq \emptyset \\
&\Leftrightarrow \forall G.[\![\mathcal{A}(q_1, w) \Rightarrow \phi_{q_1}^w(\mathcal{A}(q_2, w))]\!]^{\sigma(G)} \neq \emptyset \\
&\Leftrightarrow \forall G.[\![(\mathcal{A}(q_1, w)) \wedge \neg\phi_{q_1}^w(\mathcal{A}(q_2, w))]\!]^{\sigma(G)} = \emptyset \\
&\Leftrightarrow \forall M.[\![(\mathcal{A}(q_1, w)) \wedge \neg\phi_{q_1}^w(\mathcal{A}(q_2, w))]\!]^{\mathcal{M}} = \emptyset \\
&\Leftrightarrow \mathcal{A}(q_1, w)) \wedge \neg\phi_{q_1}^w(\mathcal{A}(q_2, w)) \text{ unsatisfiable} \\
&\Leftrightarrow \forall M.\ M \not\models \mathcal{A}(q_1, w)) \wedge \neg\phi_{q_1}^w(\mathcal{A}(q_2, w))
\end{aligned}
$$

□

All the non-distinguished variables (whether unique or multiply occurring) in the right-hand side of the query are encoded as $\top$, thus this hinders from the exponential blow-up in the size of the formula.

Theorem 12 is expressed independently from queries being unary (indeed the existence of a solution is independent from the projection). In fact, this holds for n-ary queries, but our proof relies on Corollary 1 unary query encoding.

Example 36 (SPARQL query containment)**.** *Consider the encodings of the queries in Example 8:*

$$\begin{aligned}\varphi_1 = &(\mathcal{A}((x, translated, l)) \vee \mathcal{A}((x, wrote, l))) \\ &\wedge \mathcal{A}((l, type, Poem)) \\ \varphi_2 = &(\mathcal{A}((x, translated, l)) \wedge \\ &\mathcal{A}((l, type, Poem))) \vee \mathcal{A}((x, wrote, l))\end{aligned}$$

Checking $\varphi_1 \wedge \neg\varphi_2 \wedge \varphi_r$ in a satisfiability solver provides "no", hence $q_1\{x\} \sqsubseteq q_2\{x\}$. However, the solver provides "yes" for $\varphi_2 \wedge \neg\varphi_1 \wedge \varphi_r$, leading to $q_2\{x\} \not\sqsubseteq q_1\{x\}$. Indeed, the transition system of Figure 3.2 is a model of this formula and thus, the RDF graph of Example 1 is a counter-example to the containment.

6.3.1 Complexity

Our translation of the query containment problem does not involve duplication of logical formulas of variable size. Therefore, the translation produces a logical formula of linear-size in terms of the size of the queries. Thus, we reduced the problem of query containment to unsatisfiability in modal logic K_n [Blackburn *et al.* 2007] which is PSPACE-Complete.

Proposition 9. *SPARQL query containment can be solved in a polynomial amount of space.*

Note that this complexity result is only an upper bound. In [Pérez *et al.* 2009], it has been proved that the complexity of evaluating of SPARQL queries is PSPACE-Complete.

The contribution of this approach is its extensibility to other SPARQL graph patterns such as OPTIONAL and MINUS with or without admitting additional (complexity) cost. Consider, for instance, extend the proposed approach to the containment test between MINUS graph patterns (see Chapter 8). It suffices to extend the encoding function $\mathcal{A}$ in Definition 27. For example, encoding of a MINUS query q can be done as:

$$\mathcal{A}(q, q_1 \text{ MINUS } q_2, w_f) = \mathcal{A}(q, q_1, w_f) \wedge \neg\mathcal{A}(q, q_2, w_f)$$

Likewise, the same extension can be done to test containment of well-formed OPTIONAL graph patterns as shown in Example 37. For more on this, we refer the interested reader to Chapter 8 and [Letelier *et al.* 2012].

Example 37 (Containment)**.** *Determine the containment of the following queries.*

q_1

```
1 SELECT *
2 WHERE {
3   ?x :a ?y
4   OPTIONAL { ?x :b ?z }
5   OPTIONAL { ?x :c ?r }
6 }
```

q_2

```
SELECT *
WHERE {
  ?x :a ?y
  OPTIONAL {
     ?x :b ?z
     OPTIONAL { ?x :c ?r }
  }
}
```

From the semantics of the optional operator, it can be inferred that $q_1 \not\sqsubseteq q_2$ *and* $q_2 \sqsubseteq q_1$*. To prove this, one proceeds by encoding the queries as formulas and reduce the encoding to satisfiability test. For the sake of readability, we denote the triple patterns in the query (lines 3, 4 and 5) as P1, P2 and P3. Encoding of* q_1 *is* $\mathcal{A}(P1 \text{ OPT } P2 \text{ OPT } P3)$*:*

$$
\begin{aligned}
&= \mathcal{A}\big((P1 \text{ OPT } P2) \text{ OPT } P3\big) \\
&= \mathcal{A}\big((P1 \text{ OPT } P2) \text{ AND } P3\big) \text{ UNION } \mathcal{A}\big(P1 \text{ OPT } P2\big) \\
&= \big(\mathcal{A}\big(P1 \text{ AND } P2\big) \text{ UNION } \mathcal{A}(P1)\big) \text{ AND } \mathcal{A}(P3) \text{ UNION } \mathcal{A}\big(P1 \text{ AND } P2\big) \text{ UNION } \mathcal{A}(P1) \\
&= \big(\mathcal{A}(P1) \text{ AND } \mathcal{A}(P2) \text{ AND } \mathcal{A}(P3)\big) \text{ UNION } \big(\mathcal{A}(P1) \text{ AND } \mathcal{A}(P3)\big) \\
&\quad \text{UNION } \big(\mathcal{A}(P1) \text{ AND } \mathcal{A}(P2)\big) \text{ UNION } \mathcal{A}(P1) \\
&= \big(x \wedge \langle s\rangle(a \wedge \langle o\rangle y) \wedge \langle s\rangle(b \wedge \langle o\rangle z) \wedge \langle s\rangle(c \wedge \langle o\rangle r)\big) \\
&\quad \vee \big(x \wedge \langle s\rangle(a \wedge \langle o\rangle y) \wedge \langle s\rangle(c \wedge \langle o\rangle r)\big) \\
&\quad \vee \big(x \wedge \langle s\rangle(a \wedge \langle o\rangle y) \wedge \langle s\rangle(b \wedge \langle o\rangle z)\big) \vee \big(x \wedge \langle s\rangle(a \wedge \langle o\rangle y)\big)
\end{aligned}
$$

Encoding of q_2 *is* $\mathcal{A}(P1 \text{ OPT } (P2 \text{ OPT } P3))$*:*

$$
\begin{aligned}
&= \mathcal{A}(P1 \text{ AND } (P2 \text{ OPT } P3)) \text{ UNION } \mathcal{A}(P1) \\
&= \mathcal{A}(P1) \text{ AND } \mathcal{A}(P2 \text{ OPT } P3) \text{ UNION } \mathcal{A}(P1) \\
&= \mathcal{A}(P1) \text{ AND } (\mathcal{A}(P2 \text{ AND } P3) \text{ UNION } \mathcal{A}(P2)) \text{ UNION } \mathcal{A}(P1) \\
&= \mathcal{A}(P1) \text{ AND } (\mathcal{A}(P2) \text{ AND } \mathcal{A}(P3) \text{ UNION } \mathcal{A}(P2)) \text{ UNION } \mathcal{A}(P1) \\
&= \mathcal{A}(P1) \text{ AND } \mathcal{A}(P2) \text{ AND } \mathcal{A}(P3) \text{ UNION } \mathcal{A}(P1) \text{ AND } \mathcal{A}(P2) \text{ UNION } \mathcal{A}(P1) \\
&= \big(x \wedge \langle s\rangle(a \wedge \langle o\rangle y) \wedge \langle s\rangle(b \wedge \langle o\rangle z) \wedge \langle s\rangle(c \wedge \langle o\rangle r)\big) \\
&\quad \vee \big(x \wedge \langle s\rangle(a \wedge \langle o\rangle y) \wedge \langle s\rangle(b \wedge \langle o\rangle z)\big) \vee \big(x \wedge \langle s\rangle(a \wedge \langle o\rangle y)\big)
\end{aligned}
$$

Reducing $q_1 \sqsubseteq q_2$ *to the satisfiability test of* $\mathcal{A}(q_1) \wedge \neg\mathcal{A}(q_2)$

$$
\begin{aligned}
= (&\big(x \wedge \langle s\rangle(a \wedge \langle o\rangle y) \wedge \langle s\rangle(b \wedge \langle o\rangle z) \wedge \langle s\rangle(c \wedge \langle o\rangle r)\big) \\
&\vee \big(x \wedge \langle s\rangle(a \wedge \langle o\rangle y) \wedge \langle s\rangle(c \wedge \langle o\rangle r)\big) \\
&\vee \big(x \wedge \langle s\rangle(a \wedge \langle o\rangle y) \wedge \langle s\rangle(b \wedge \langle o\rangle z)\big) \vee \big(x \wedge \langle s\rangle(a \wedge \langle o\rangle y)\big)\big) \\
&\qquad \bigwedge \\
&\big(\big(\neg x \vee [s](\neg a \vee [o]\neg y) \vee [s](\neg b \vee [o]\neg z) \vee [s](\neg c \vee [o]\neg r)\big)\big) \\
&\wedge \big(\neg x \vee [s](\neg a \vee [o]\neg y) \vee [s](\neg b \vee [o]\neg z)\big) \wedge \big(\neg x \vee [s](\neg a \vee [o]\neg y)\big)\big)
\end{aligned}
$$

This formula is satisfiable due to the subformula $\big(x \wedge \langle s\rangle(a \wedge \langle o\rangle y) \wedge \langle s\rangle(c \wedge \langle o\rangle r)\big)$ *being satisfiable. Hence,* $q_1 \not\sqsubseteq q_2$*. Now, let us look at the other direction to determine whether* $q_2 \sqsubseteq q_1$*, reducing this* $\mathcal{A}(q_2) \wedge \neg\mathcal{A}(q_1)$

$$
\begin{aligned}
= (&\big(\big(x \wedge \langle s\rangle(a \wedge \langle o\rangle y) \wedge \langle s\rangle(b \wedge \langle o\rangle z) \wedge \langle s\rangle(c \wedge \langle o\rangle r)\big) && (6.1)\\
&\vee \big(x \wedge \langle s\rangle(a \wedge \langle o\rangle y) \wedge \langle s\rangle(b \wedge \langle o\rangle z)\big) \vee \big(x \wedge \langle s\rangle(a \wedge \langle o\rangle y)\big)\big) && (6.2)\\
&\qquad \bigwedge && (6.3)\\
&\big(\big(\neg x \vee [s](\neg a \vee [o]\neg y) \vee [s](\neg b \vee [o]\neg z) \vee [s](\neg c \vee [o]\neg r)\big)\big) && (6.4)\\
&\wedge \big(\neg x \vee [s](\neg a \vee [o]\neg y) \vee [s](\neg c \vee [o]\neg r)\big) && (6.5)\\
&\wedge \big(\neg x \vee [s](\neg a \vee [o]\neg y) \vee [s](\neg b \vee [o]\neg z)\big) \wedge \big(\neg x \vee [s](\neg a \vee [o]\neg y)\big)\big) && (6.6)
\end{aligned}
$$

This formula is unsatisfiable: each of the disjuncts (1) – (3) are conjuncted with their negated equivalent (4) – (6) resulting in $\varphi \wedge \neg\varphi$ *which is always false. Therefore, this result implies that* $q_2 \sqsubseteq q_1$*.*

6.4 Experimentation

In order to experiment with the proposed approach, the satisfiability solvers from [Tanabe *et al.* 2005] and [Genevès *et al.* 2007] are used to test containment and equivalence among different queries. We call the solver from [Tanabe *et al.* 2005] AFMU (alternation-free mu-calculus) and the one from [Genevès *et al.* 2007] TreeSolver. A set of queries, shown in Appendix A Section A.1.1, are tested for their containment and equivalence with running times depicted in Table 6.4. In fact, the running time is dependent on the processor speed and the size of the queries. To encode queries as $\mathcal{K}_n$ formulas, we used the Jena SPARQL API [2].

All the experiments were conducted on a MacBook Pro, Intel Core i7, 2GHz processor speed, and 4GB memory under V10.6.8. The Java programs were executed with JRE v1.6.0_22. Table 6.4 shows the running times, in milliseconds, of containment tests. Tests 3, 15 and 16 do not terminate for the AFMU satisfiability solver, they are displayed as blanks (-) in Table 6.4.

A comparison of the performance of the solvers, TreeSolver and AFMU, is shown

[2] http://jena.sourceforge.net/ARQ/

		TreeSolver		AFMU	
Test No.	$q \sqsubseteq q'$?	Yes/No	time	Yes/No	time
1	Q1a $\sqsubseteq$ Q1b	Yes	12	Yes	39
2	Q1b $\sqsubseteq$ Q1a	No	11	No	38
3	Q2a $\sqsubseteq$ Q2b	Yes	208	-	-
4	Q2b $\sqsubseteq$ Q2a	Yes	198	Yes	2328
5	Q3a $\sqsubseteq$ Q3b	Yes	18	Yes	34
6	Q3b $\sqsubseteq$ Q3a	No	15	No	33
7	Q4c $\sqsubseteq$ Q4b	Yes	204	Yes	1746
8	Q4b $\sqsubseteq$ Q4c	No	182	No	993
9	Q5a $\sqsubseteq$ Q5b	No	10	No	28
10	Q5b $\sqsubseteq$ Q5a	Yes	10	Yes	28
11	Q6a $\sqsubseteq$ Q6b	No	28	No	45
12	Q6b $\sqsubseteq$ Q6a	No	28	No	44
13	Q6a $\sqsubseteq$ Q6c	Yes	21	Yes	33
14	Q6c $\sqsubseteq$ Q6a	No	15	No	32
15	Q7a $\sqsubseteq$ Q7b	No	503	-	-
16	Q7b $\sqsubseteq$ Q7a	No	475	-	-
17	Q8a $\sqsubseteq$ Q8b	Yes	228	Yes	391
18	Q8b $\sqsubseteq$ Q8a	No	177	No	322
19	Q9a $\sqsubseteq$ Q9b	No	83	No	1937
20	Q9b $\sqsubseteq$ Q9a	No	79	No	1941
21	Q9c $\sqsubseteq$ Q9b	No	83	No	1937
22	Q9b $\sqsubseteq$ Q9c	Yes	98	Yes	7767

Table 6.2: Containment of tree-structured SPARQL queries

in Figure 6.2. As can be seen, TreeSolver outperforms AFMU in all of the test cases. A detailed discussion, regarding these solvers, can be found in the next chapter.

6.5 Conclusion

In this chapter, we have first studied the profile of real-world queries with respect to theoretical characterization and found that (1) a large part of these queries are acyclic, and (2) those parts that either contain projections (effective SELECT) or not, are significant. We have studied the demographics of SPARQL queries on a large example, from this and state-of-the-art of query containment solvers, we have designed a benchmark suite in order to test the performance of the containment solvers, as detailed in Chapter 7. Beyond this, we have also shown that the double exponential

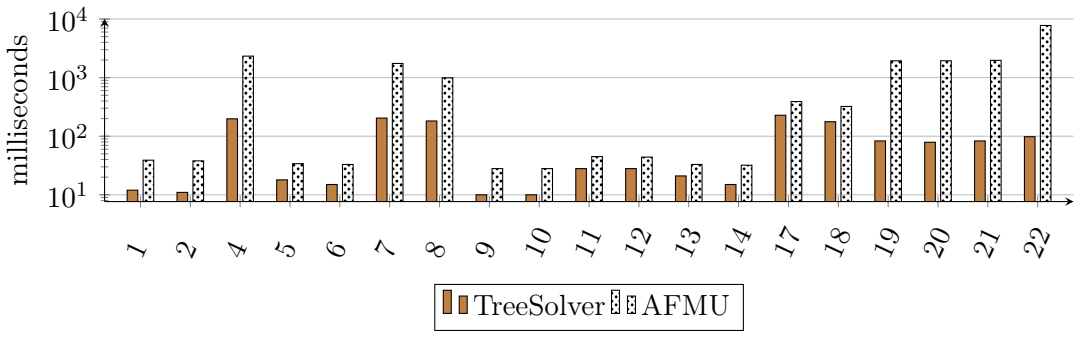


Figure 6.2: A comparison of TreeSolver and AFMU (logarithmic scale)

upper bound for containment of queries that is proved in Chapters 3 and 5 can reduce to just PSPACE. In fact, it is also shown, in Chapter 3, that this bound reduces to just ExpTime when the query on the right hand side has a tree structure. Meanwhile, here, we have proved that this problem can even reduce to PSPACE. This is due to that fact that containment of union of conjunctive queries is known to be NP-complete. This can also be shown for SPARQL, by translating AND-UNION SPARQL queries into the conjunctive queries of relational algebra. Hence, the principal advantage of this work is that, our methods can be extended for other graph patterns beyond basic and union graph patterns, for instance, MINUS graph patterns. In addition, we proved that this encoding is correct and can be used for checking query containment. We also have characterized the complexity of containment test. Beyond all, experiments were carried out to justify our approach. Two satisfiability solvers were used for this task [Tanabe *et al.* 2005] and [Genevès *et al.* 2007].

Chapter 7

Containment Benchmark

Contents

The problem of SPARQL query containment has recently attracted a lot of attention due to its fundamental purpose in query optimization and information integration. New approaches to this problem, have been put forth, that can be implemented in practice. However, these approaches suffer from various limitations: coverage (size and type of queries), response time (how long it takes to determine containment), and the technique applied to encode the problem. In order to experimentally assess implementation limitations, we designed a benchmark suite offering different experimental settings depending on the type of queries, projection and reasoning (RDFS). We have applied this benchmark to three available systems using different techniques highlighting the strengths and weaknesses of such systems.

The overall purpose of this chapter is designing a benchmark and evaluating the performance of the current state-of-the-art using this benchmark. Using the encoding procedure in Chapter 3 and exponential time satisfiability solvers from [Tanabe *et al.* 2005] and [Genevès *et al.* 2007], we conduct an experiment to test the containment and equivalence of tree-structured SPARQL queries. We compare the performance of these two satisfiability solvers, [Tanabe *et al.* 2005] and [Genevès *et al.* 2007]. Moreover, we also evaluate the performance of the subsumption and equivalence solver from [Letelier *et al.* 2012]. However, as it is not an exponential time solver like the

other two ([Tanabe *et al.* 2005] and [Genevès *et al.* 2007]), we judge its performance on its own.

7.1 Benchmark

In this section, we present the query containment setup, structure of the benchmark and the benchmark description. We first present the design of containment benchmark suites. Each test suite is made of elementary test cases asking for the containment of one query into another. We then introduce the principles and software used for evaluating containment solver. The benchmark and software is available on-line at `http://sparql-qc-bench.inrialpes.fr/`.

7.1.1 Query Containment Setup

A test case for containment comprises two queries q and q', and an optional schema $\mathcal{S}$. The query containment solver (QC solver) produces yes or no answers, i.e., yes if q is contained in q' and no otherwise. A general workflow diagram designed for this purpose is illustrated in Figure 7.1.

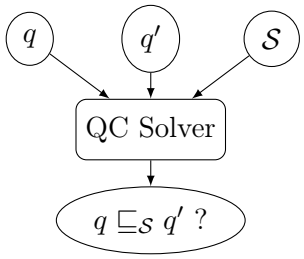


Figure 7.1: General workflow of query containment tests.

7.1.2 Structure of the Benchmark

The four key requirements of a benchmark laid out by the benchmark handbook [Gray 1992] are: understandability i.e., the queries and hence axioms should be understandable, scalability, portability i.e., being able to run on different platforms, and finally relevance i.e., testing typical operations such as joins, disjunctions, and typing restrictions. Thus, we designed the benchmark following these principles.

There are three qualitative dimensions along which tests can be designed: the type of graph pattern connectors (AND, UNION , MINUS , Projection, OPT , FILTER etc.), the type of ontology: (no schema, RDFS, SHI, OWL, etc.) and the query structure (tree, DAG, cyclic). In addition to these dimensions, there are quantitative measures that can be used, like:

– the number of triple patterns,

– the number of variables,

– the number of join (triple patterns involving more than one variable),

– the size of the ontology.

We designed test suites of homogeneous qualitative dimensions selected with respect to the capacity of the current state-of-the-art solvers. The benchmark contains three test suites:

– **C**onjunctive **Q**ueries with **No Proj**ection (CQNoProj)

– **U**nion of **C**onjunctive **Q**ueries with **Proj**ection (UCQProj)

– **U**nion of **C**onjunctive **Q**ueries under **RDFS** reasoning (UCQrdfs)

We did not provide tests of cyclic queries since only one solver is currently able to deal with them. However, this would be a natural addition to these tests.

Each test suite contains tests of different quantitative measures. Most of them are used for conformance testing, i.e., testing that solvers return the correct answer. But we also identify some stress tests trying to evaluate solvers at or beyond their limits. Conformance tests are small size queries while stress tests are larger in size, e.g., queries containing more than 15 joins. We discuss these test suites below.

For our compliance benchmark, queries are chosen according to the following criteria: projection (or no projection), operator nesting, number of connectives (joins and disjunctions), and requiring RDFS reasoning. Our queries also vary in general characteristics like selectivity, query size, and different types of joins. One thing that should be noted is that, the benchmark criteria are selected in line with the capacity of the current state-of-the-art. The benchmark contains three test suites:

7.1.2.1 CQNoProj

This test suite is designed for containment of basic graph patterns. It contains conjunctive queries with no projection. We have identified 20 different test cases (nop1–nop20), each one testing containment between two queries. All the cases in this setting are shown in Table 7.1, along with the number of connectives and variables in the queries. Q7a and Q7b have been presented in Example 38.

Example 38. *Consider the following queries that retrieve students' information from a university dataset.*

Select all students' information.

Q7a

```
SELECT ?x ?y
WHERE {
   ?x a :Student . ?x :name ?y .
   ?x :nickName ?z . ?x :telephone ?t .
   ?x :ssn ?ssn .  ?x :age ?a .
   ?x :sex ?sex . ?x :emailAddress ?e .
   ?x :memberOf ?d . ?x :takesCourse ?c .
}
```

Select master students' information.

Q7b

```
SELECT ?x ?y
WHERE {
   ?x a :Student . ?x :name ?y .
   ?x :nickName ?z . ?x :telephone ?t .
   ?x :ssn ?ssn .  ?x :age ?a .
   ?x :sex ?sex .  ?x :emailAddress ?e .
   ?x :memberOf ?d . ?x :takesCourse ?c .
   ?x :masterDegreeFrom ?master .
}
```

Test case	Problem	AND	Vars	Join
nop1	Q1a $\sqsubseteq$ Q1b	1	1	0
nop2	Q1b $\sqsubseteq$ Q1a	0		0
nop3	Q2a $\sqsubseteq$ Q2b	5	3	3
nop4	Q2b $\sqsubseteq$ Q2a	5		3
nop5	Q3a $\sqsubseteq$ Q3b	2	2	2
nop6	Q3b $\sqsubseteq$ Q3a	1		1
nop7	Q4c $\sqsubseteq$ Q4b	5	3	2
nop8	Q4b $\sqsubseteq$ Q4c	3		2
nop9	Q6a $\sqsubseteq$ Q6b	2	3	1
nop10	Q6b $\sqsubseteq$ Q6a	2		1

Test case	Problem	AND	Vars	Join
nop11	Q6a $\sqsubseteq$ Q6c	2	3	1
nop12	Q6c $\sqsubseteq$ Q6a	0		1
nop13	Q6b $\sqsubseteq$ Q6c	2	3	1
nop14	Q6c $\sqsubseteq$ Q6b	0		1
nop15	Q7a $\sqsubseteq$ Q7b	9	10	9
nop16	Q7b $\sqsubseteq$ Q7a	10		9
nop17	Q8a $\sqsubseteq$ Q8b	3	4	3
nop18	Q8b $\sqsubseteq$ Q8a	2		3
nop19	Q9a $\sqsubseteq$ Q9b	4	3	2
nop20	Q9b $\sqsubseteq$ Q9a	4		2

Table 7.1: The CQNoProj testsuite. In the AND column, figures correspond to the number of AND in the left-hand side query of the test. Vars is the number of variables in each queries and Join the number of triples in which occurs at least three variables.

The more difficult tests used for stress testing are nop3, nop4, nop15, and nop16. The two former ones have a larger number of conjunction (and of join), while the two latter ones have an even larger number of conjunctions and variables.

7.1.2.2 UCQProj

This test suite is made of 28 test cases, each comprising two acyclic union of conjunctive queries with projection. In fact, 14 tests contain projection only, 6 tests contains union only and 2 tests contains both (see Table 7.2). Keeping these tests together allows for comparing these supposed complex tests to less complex ones. The test cases differ in the number of distinguished variables (*Dvars*) and connectives (conjunction or union). Particular stress tests are p3, p4 (without union nor projection), p15, p16, p23, and p24.

7.1.2.3 UCQrdfs

In query containment under RDFS reasoning, there are 28 test cases (Table 7.3). In comparison to the test cases in UCQProj and CQNoProj setting, the query sizes are small. Each test case is composed of two acyclic UCQs and a schema. There are 4 different small schemas, C1–C4 whose characteristics, with respect of in the type and number of axioms used, are presented in Table 7.3). The most difficult tests are supposed to be rdfs23, rdfs24, rdfs25, rdfs26, rdfs27 and rdfs28 with both projection and union.

7.1.3 Benchmarking software architecture

For testing containment solvers we designed an experimental setup which comprises several software components. This setup is illustrated in Figure 7.2. It simply considers

Test case	Problem	AND	UNION	Dvars	Vars	Join
p1	Q11a $\sqsubseteq$ Q11b	1	0	1	1	0
p2	Q11b $\sqsubseteq$ Q11a	0	0		1	0
p3	Q12a $\sqsubseteq$ Q12b	5	0	3	3	3
p4	Q12b $\sqsubseteq$ Q12a	5	0		3	3
p5	Q13a $\sqsubseteq$ Q13b	2	0	2	2	2
p6	Q13b $\sqsubseteq$ Q13a	1	0		2	1
p7	Q14c $\sqsubseteq$ Q14b	3	0	1	3	2
p8	Q14b $\sqsubseteq$ Q14c	5	0		3	2*
p9	Q15a $\sqsubseteq$ Q15b	0	0	2	3	1
p10	Q15b $\sqsubseteq$ Q15a	0	0		2	1
p11	Q16a $\sqsubseteq$ Q16b	2	0	1	3	1
p12	Q16b $\sqsubseteq$ Q16a	2	0		3	1
p13	Q16a $\sqsubseteq$ Q16c	2	0	1	3	1
p14	Q16c $\sqsubseteq$ Q16a	0	0		3	1
p15	Q17a $\sqsubseteq$ Q17b	9	0	10	10	9
p16	Q17b $\sqsubseteq$ Q17a	10	0		11	10
p17	Q18a $\sqsubseteq$ Q18b	3	0	4	4	3
p18	Q18b $\sqsubseteq$ Q18a	2	0		4	3
p19	Q19a $\sqsubseteq$ Q19b	4	0	2	3	2
p20	Q19b $\sqsubseteq$ Q19a	4	0		3	2
p21	Q19c $\sqsubseteq$ Q19b	4	0	2	4	3
p22	Q19b $\sqsubseteq$ Q19c	4	0		3	2
p23	Q20a $\sqsubseteq$ Q20b	2	7	10	10	9
p24	Q20b $\sqsubseteq$ Q20a	8	1		10	9
p25	Q21a $\sqsubseteq$ Q21b	6	2	2	4-6	5
p26	Q21b $\sqsubseteq$ Q21a	8	0		6	5
p27	Q22a $\sqsubseteq$ Q22b	3	1	2	2	2
p28	Q22b $\sqsubseteq$ Q22a	3	1		2	2

Table 7.2: The UCQProj test suite.

a containment checker as a software module taking as input two SPARQL queries (q and q'), eventually an RDF Schema ($\mathcal{S}$), and returns true or false depending if q' is entailed by q (under the constraints of $\mathcal{S}$).

This has been provided as a Java interface using Jena to express queries and RDF Schema. It simply takes test instances and provide them to the interface, timing the ex-

Schema	Axiom types
C1	subclass (2)
C2	domain (1) and range (1)
C3	subproperty (2), subclass (1) and domain (1)
C4	subclass (1)

Test	Ontology	Problem	AND	UNION	Dvars	Vars	Join
rdfs1	C1	Q39a $\sqsubseteq$ Q39c	0	0	1	1	0
rdfs2		Q39c $\sqsubseteq$ Q39a	0	1		1	0
rdfs3	C1	Q39a $\sqsubseteq$ Q39b	0	0	1	1	0
rdfs4		Q39b $\sqsubseteq$ Q39a	0	0		1	0
rdfs5	C1	Q39b $\sqsubseteq$ Q39c	0	0	1	1	0
rdfs6		Q39c $\sqsubseteq$ Q39b	0	1		1	0
rdfs7	C1	Q39d $\sqsubseteq$ Q39e	4	0	1	3	2
rdfs8		Q39e $\sqsubseteq$ Q39d	4	0		3	2
rdfs9	C2	Q40b $\sqsubseteq$ Q40d	0	0	1	2	1
rdfs10		Q40d $\sqsubseteq$ Q40b	0	0		1	0
rdfs11	C2	Q40e $\sqsubseteq$ Q40b	1	0	1	2	2
rdfs12		Q40b $\sqsubseteq$ Q40e	0	0		1	0
rdfs13	C3	Q41b $\sqsubseteq$ Q41c	0	0	1	2	1
rdfs14		Q41c $\sqsubseteq$ Q41b	0	0		2	1
rdfs15	C3	Q41b $\sqsubseteq$ Q41d	0	0	1	2	1
rdfs16		Q41d $\sqsubseteq$ Q41b	0	0		2	1
rdfs17	C3	Q41c $\sqsubseteq$ Q41d	0	0	1	2	1
rdfs18		Q41d $\sqsubseteq$ Q41c	0	0		2	1
rdfs19	C3	Q41b $\sqsubseteq$ Q41a	0	0	1	2	1
rdfs20		Q41a $\sqsubseteq$ Q41b	0	0		1	0
rdfs21	C3	Q41e $\sqsubseteq$ Q41a	0	1	1	2	2
rdfs22		Q41a $\sqsubseteq$ Q41e	0	0		1	0
rdfs23	C4	Q43a $\sqsubseteq$ Q43b	3	1	2	2	2
rdfs24		Q43b $\sqsubseteq$ Q43a	3	1		2	2
rdfs25	C4	Q43a $\sqsubseteq$ Q43c	3	1	2	2	2
rdfs26		Q43c $\sqsubseteq$ Q43a	3	1		2	2
rdfs27	C4	Q43b $\sqsubseteq$ Q43c	3	3	2	2	2
rdfs28		Q43c $\sqsubseteq$ Q43b	3	1		2	2

Table 7.3: The UCQrdfs test suite.

ecution of the containment test within the dashed box of Figure 7.2, i.e., after query and schema parsing. We have developed three wrappers implementing this interface for the

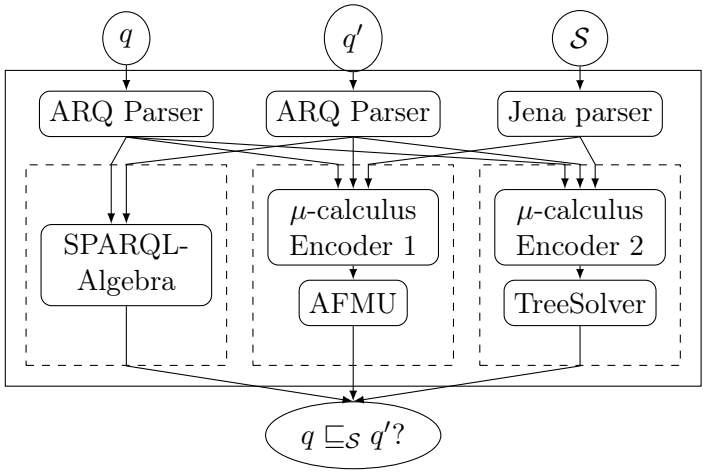


Figure 7.2: Experimental setup for query containment test. The tester (plain rectangle) parses queries and schemas and pass them to a solver wrapper (dashed rectangle).

respective systems: AFMU, TreeSolver and SPARQL-Algebra. AFMU and TreeSolver first require to encode the queries and schema into μ-calculus formulas. This advantages SPARQL-Algebra because it works directly on ARQ representation, whereas the two other solvers have first the ARQ representation translated into a μ-calculus formula which is then parsed and transformed in each solver's internal representation.

Other systems could be wrapped in the same interface and tested in the same conditions. This platform may also be used for providing non regression tests for containment solvers.

7.2 Query Containment Solvers

We briefly present three state-of-the-art query containment solvers used in the experiments. Our goal is to characterize their capabilities in order to design appropriate benchmarks. In order to do so, we also analyze actual queries used on the semantic web.

Out of the three systems, SPARQL-Algebra is self contained whereas two are μ-calculus satisfiability solvers that need an intermediate query translation into formulas to determine containment.

7.2.1 SPARQL-Algebra

SPARQL-Algebra is an implementation of SPARQL query subsumption and equivalence based on the theoretical results in [Letelier *et al.* 2012]. This implementation supports AND and `OPT` queries with no projection. An on-line version of the solver is available at `http://db.ing.puc.cl/sparql-algebra/`.

7.2.2 AFMU

AFMU (Alternation Free two-way μ-calculus) [Tanabe *et al.* 2005], is a satisfiability solver for the alternation-free fragment of the μ-calculus [Kozen 1983]. It is a prototype implementation which determines the satisfiability of a μ-calculus formula by producing a yes-or-no answer.

To turn it into a query containment solver, it is necessary to turn the problem into a μ-calculus satisfiability problem.

In previous work [Chekol *et al.* 2012a, Chekol *et al.* 2012b], we developed techniques for encoding queries into the μ-calculus ($\mathcal{A}$) in order to determine the containment of SPARQL queries. Of the three approaches introduced in [Chekol *et al.* 2012a] to deal with RDFS, we have chosen the encoding of the schema (η) into the μ-calculus. We use these encoding schemes for deciding $q \sqsubseteq_{\mathcal{S}} q'$: it is necessary to check if the encoding of its negation, $\eta(\mathcal{S}) \wedge \mathcal{A}(q) \wedge \neg\mathcal{A}(q')$, is satisfiable. If this is the case, then containment does not hold, otherwise, it is established.

7.2.3 TreeSolver

The XML tree logic solver *TreeSolver*[1] performs static analysis of XPath queries which comprises containment, equivalence and satisfiability. To perform these tasks, the solver translates XPath queries into μ-calculus formulas and then it tests the unsatisfiability of the formula. Unlike AFMU, the unsatisfiability test is performed in a time of $2^{\mathcal{O}(n)}$ whereas it is $2^{\mathcal{O}(n \log n)}$ for AFMU, where n is the size of the formula.

Using TreeSolver follows the same procedure as using AFMU with a slightly different encoding. Indeed, because TreeSolver is restricted to tree-shaped models, we use a specific encoding of query formulas.

7.2.3.1 Features supported by solvers

AA summary of the features supported by these query containment solvers is presented in Table 7.4.

System	projection	UCQ	optional	blanks	cycles	RDFS
SPARQL-Algebra			√		√	
AFMU	√	√		√		√
TreeSolver	√	√		√		√

Table 7.4: Comparison of features supported by current systems.

Part of the query structures can be transformed into concept expressions in description logics and submitted to satisfiability (or subsumption) tests as well. So, in principle, query containment solvers based on description logic reasoners could be designed. However, we do not know any such solver.

[1] http://wam.inrialpes.fr/websolver/

7.3 Experimentation

We evaluated the three identified query containment solvers with the three test suites. Rather than a definitive assessment of these solvers, our goal is to give first insights into the state-of-the art and highlight deficiencies of engines based on the benchmark outcome. None of these systems is sharply optimized as testified by numerous printout. However, their behavior is sufficient for highlighting test difficulty.

We run experiments on a Debian Linux virtual machine configured with four processors and 20GB of RAM running under a Dell PowerEdge T610 with 2*Intel Xeon Quad Core 2.26GHz E5607 processors and 32GB of RAM, under Linux ProxMox 2 (Debian).

The solvers were not genuinely reentrant. Hence, each test case has been run in a separate process after that the first case of each suite has been run as a warm up. TreeSolver was not able to support this warm-up on tests p9 and p10 (this penalized it).

All solvers are Java programs. The Java virtual machines were run with maximum heap size of 2024M and a timeout at 15s (15000ms). The μ-calculus solvers take advantage of a native BDD library. Using the native implementation doubles the speed of these solvers, however, it also brings large initialization time (in spite of warm-up set up) and memory problems.

Reported figures are the average of 5 runs (we run the tests 7 times and ruled out each time the best and worse performance).

7.3.1 CQNoProj Results

On the conjunctive queries without projection, the SPARQL Algebra implementation is 10 times faster than the μ-calculus implementations (Figure 7.3). This comes as no surprise, since the latter are exponential time solvers whereas the former is a polynomial time solver.

TreeSolver and AFMU time out whenever containment is determined between queries that contain more than 10 joins (for instance, test cases nop15 and nop16). The fact that SPARQL-Algebra does not suffer from these sets, shows that the encoding of the μ-calculus solvers can be improved for such practical cases.

SPARQL-Algebra responded incorrectly, in test case nop7 (cf. Table 7.1), when blank nodes are used in the queries. It is not expected to deal with blank nodes. The other solvers are able to take them into account.

7.3.2 UCQProj Results

On the UCQProj test suite (see Section 7.1.2.2), we compared the two systems able to deal with UNION : TreeSolver and AFMU. As can be seen in Figure 7.4, AFMU is always slightly faster than TreeSolver in all of the benchmark test cases (beside p9, p10 for initialization reasons). This is surprising given the difference in complexity of both solvers.

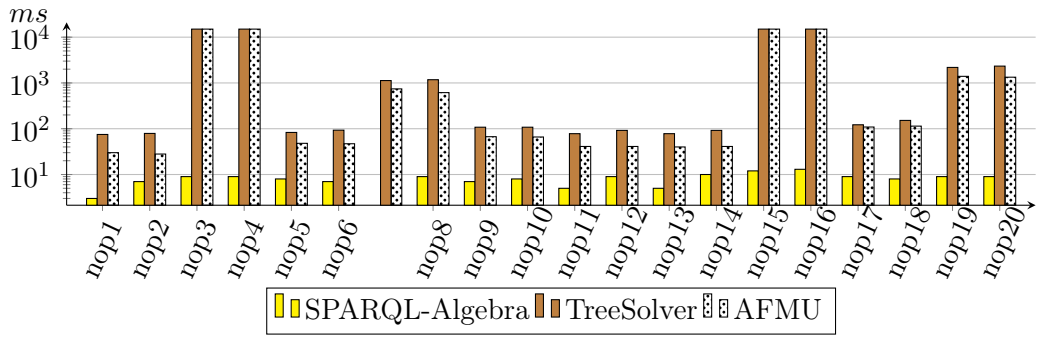


Figure 7.3: Results for the CQNoProj test suite (logarithmic scale).

In the stress queries (p3, p4, p15, p16, p23, p24, p25, p26) both systems time out. The necessary run time tends to be far longer as it often ends up in filling the available heap. For some of these tests (p15-16), this could also be improved by adopting a better encoding of triples.

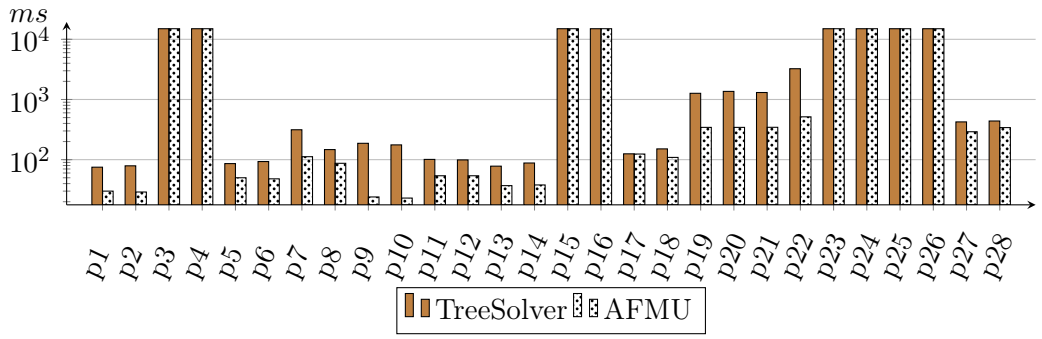


Figure 7.4: Results for the UCQProj test suite (logarithmic scale).

We run the same tests with the Java BDD implementation for both solvers (Figure 7.5). This shows slower run times for both systems but a nearly constant one for TreeSolver which does only time out on two tests (p24 and p26). This indicates that the solver's behavior is dominated by BDD initialization in spite of the warm-up.

7.3.3 UCQrdfs Results

The results for containment of acyclic UCQs under RDFS (cf. Section 7.1.2.3) are better. Figure 7.6 shows that both solvers answer containment queries within a few hundreds of milliseconds. Time outs are observed for tests 7 and 8, which contains both a (simple) subsumption test and several simple joins. In this test set, TreeSolver is in general slightly better than AFMU. As expected, queries with larger size or projection and union are more difficult. AFMU returns an incorrect answer for rdfs9 which seems to be a bug in the solver.

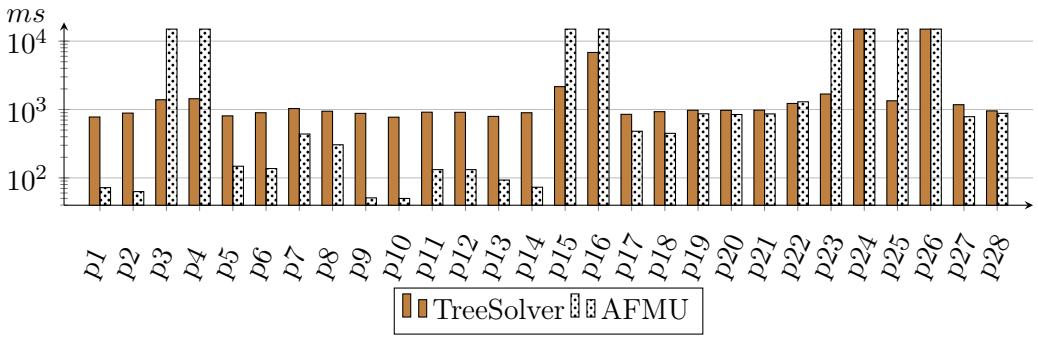


Figure 7.5: Results for the UCQProj test suite with Java BDD implementation (logarithmic scale).

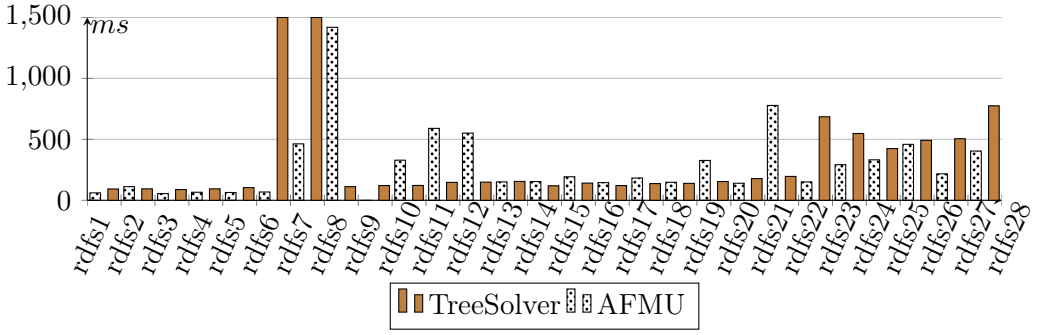


Figure 7.6: Results for the UCQrdfs test suite.

7.3.4 Discussion

In summary, all the solvers under all of the experimental settings responded positively i.e., they all determined containment correctly under their stated application limits (we tested this independently). However, from these experiments, a lot remains to be done in order to alleviate the shortcomings of the current systems.

SPARQL-Algebra is faster on its domain of application. The advantages of this solver compared to the others are that it supports subsumption of `OPT` query patterns and also cyclic CQs. However, blank nodes are not supported.

AFMU is able to determine containment of acyclic UCQs under ontological axioms. For queries of reasonable size, the solver determined their containment correctly. The problem is that when queries have a larger size, e.g., more than 8 joins, the solver saturate memory. This is shown for test cases nop15 and nop16 (Figure 7.3) as well as for test cases. However, the implementation of this solver is not optimal: the authors have documented improvements p15 and p16 (Figure 7.4). Moreover, determining containment of general UCQs (beyond the acyclic ones) will require extending the solver.

TreeSolver has similar limitations as AFMU: no support for cyclic queries and difficulty with queries of large size, such as nop16 and p16. This is surprising with respect

to its worse case complexity. However, as shown in Figure 7.5, the solver has potential to scale if it can find more accurate ways to initialize memory.

In addition, it is a good thing that determining the type of queries to compare (cyclic, disjunctive, with blank nodes, with projections, etc.) is easy. Hence, it is possible to build a system assembling these solvers and providing the best possible performances for each cases.

7.4 Conclusion

In this chapter, we have designed a containment benchmark made of three query containment test suites testing conjunctive queries without projection, union of conjunctive queries with projection and with RDFS reasoning. We have applied these to existing containment solvers (SPARQL-Algebra, AFMU and TreeSolver). Obviously, current μ-calculus solvers are not optimized for conjunctive queries without projection which are better dealt with by SPARQL-Algebra. Hence, so far the best strategy is to use one solver when queries are conjunctive and do not have projection and another when they have projection, union and axioms. Fortunately, this situation is easy to diagnose.

This chapter contributed to the study of SPARQL query containment in several ways:

- from the state-of-the-art of query containment solvers, we have designed a benchmark suite and methodology for this problem;
- finally, after evaluating current solutions on existing solvers, we can report the following lessons:
  - All tested solutions perform correctly with respect to their declared applicability limits (which are easily testable);
  - SPARQL query containment can be practically evaluated, in spite of its complexity, in a reasonable time with respect to network communication costs,
  - the current state-of-the-art is at its early stage and requires improvement and new ways to determine containment and equivalence of queries, in order to become a useful tool for query optimizers.

These benchmark suites are well-suited for pinpointing the theoretical shortcomings of containment solvers.

We plan to improve and extend this benchmark, in particular by adding other test suites designed for containment of cyclic queries and queries under expressive description logic axioms such as OWL2.

Chapter 8

Extensions

Contents

In this chapter, as in for AND-UNION graph patterns, we consider set semantics of OPTIONAL graph patterns in order to study containment. We propose to fill the gap left out by [Letelier *et al.* 2012], i.e., we deal with queries with projections $q\{\overrightarrow{x}\}$. For static analysis, the set semantics ensures the correct encoding of OPTIONAL patterns as μ-calculus formulas.

We also address the containment problem for SPARQL queries that are made from MINUS graph patterns. MINUS graph patterns are well known in the database community, often referred to as conjunctive queries with negation $CQ^{\neg}$.

Finally, we present a brief discussion on the containment of SPARQL queries under the bag (multiset) semantics.

8.1 Containment of MINUS Graph Patterns

In databases, positive conjunctive queries are considered as the basic database queries [Chandra & Merlin 1977]. Conjunctive queries with negation extend this class with negation on subgoals. In SPARQL 1.1, MINUS and NOT-EXISTS are used to formulate negated queries. When only positive conjunctive queries are considered, query containment checking is NP-complete [Chandra & Merlin 1977]. It can be verified by checking if there is a homomorphism from the containing query to the contained query. When atomic negation is considered, the problem becomes much more complex: it is Π_2^P-complete [Farré *et al.* 2006] ($\Pi_2^p = coNP^{NP}$, that is, Π_2^p is what can be computed in $coNP$ with access to an NP oracle). A general approach for checking $q \sqsubseteq q'$, first presented in [Levy & Sagiv 1993] for conjunctive queries with inequalities and adapted in [Ullman 1997] for conjunctive queries with negation, consists of considering all ways of completing q with positive information. Intuitively, this amounts to generating representations of all database instances that satisfy q. On the other hand, in [Leclere & Mugnier 2006, Mohamed *et al.* 2011], negative information is also explicitly added. Such queries obtained from q by adding missing information, either positively or negatively, are called completions of q (and total completions if no more information can

be added). Then $q \sqsubseteq q'$ if and only if there is a homomorphism from q' to each total completion of q. However, the number of (total) completions of q is exponential in the size of q. Several proposals have aimed at reducing the number and the size of completions ([Wei & Lausen 2002, Leclere & Mugnier 2006, Mohamed *et al.* 2011]). Coming back to SPARQL, in this section, we introduce a different approach than the ones used in databases.

SPARQL 1.1 (the upcoming version) allows the creation of negated queries using MINUS and NOT-EXISTS keywords. The MINUS keyword and the ability to use the NOT operator with EXISTS both provide a cleaner alternative to the FILTER(!bound(?var)) trick used in SPARQL 1.0 to make missing values part of the retrieval criteria.

Example 39. *Query all students who are not attending a Java course as follows:*

q

```
PREFIX  :  <http://www.example.com/>
SELECT ?x WHERE  {
  ?x a :student
  MINUS  { ?x  :takes  "java" }
}
```

also, we can ask for all female students who are not attending a Java course.

q_p

```
PREFIX  :  <http://www.example.com/>
SELECT ?x  WHERE  {
   ?x a :student . ?x a :female
     MINUS  { ?x  :takes  "java"  }
}
```

SQL inspired operators NOT EXISTS and MINUS, representing two ways of thinking about negation, one based on testing whether a pattern exists in the data, given the bindings already determined by the query pattern, and one based on removing matches based on the evaluation of two patterns. In some cases they can produce different answers [Harris & Seaborne 2012]. The (NOT) EXISTS graph pattern filter operator returns true or false depending on whether the pattern matches the current query solution. The MINUS graph pattern statement removes existing matches from the current query solution, which boils down to a standard set subtraction. To perform this operation, both graph patterns have to be evaluated since the MINUS operator works only on those matches which were already found. Note that this behaviour differs from that of the NOT EXIST operator.

Before jumping into the problem, let us briefly sketch the fragment of SPARQL-MINUS graph patterns considered for this study.

Syntax In order to retain a decidable fragment, nestings in the right-hand side of the MINUS operator are not allowed. Thus, the syntax of SPARQL MINUS graph patterns considered in this study has the following form:

$$\mathrm{PP}_1 \text{ AND } \cdots \text{ AND } \mathrm{PP}_n \text{ MINUS } \{\mathrm{NP}_1\} \text{ MINUS } \cdots \text{ MINUS } \{\mathrm{NP}_m\}$$

where PP denotes positive graph patterns and NP denotes graph patterns that appear to the right of MINUS, and $n \geq 1, m \geq 0$. If $m = 0$, then the query becomes a positive conjunctive query.

Semantics The semantics of this fragment remains similar to that of the original, i.e., set difference between the mappings of the positive patterns PP and negative patterns NP.

Example 40. *For the queries in Example 39, it can be verified that $q_p \sqsubseteq q$ and $q \not\sqsubseteq q_p$. As already discussed in the previous chapters, one approach to determine containment queries is: reducing containment to formula unsatisfiability. Thus, queries are encoded as formulas as explained below.*

Similar to that of the encoding of positive graph patterns, see Chapter 3 for instance, the translation of the containment problem concerning MINUS graph patterns requires careful handling of the non-distinguished variables for the query on the right-hand side. Therefore, to encode $q \sqsubseteq q'$, we proceed as follows:

Encoding left-hand side query (q): a MINUS graph pattern can be encoded inductively into a μ-calculus formula as:

$$\begin{aligned}
\mathcal{A}((x,y,z)) &= \mathit{lfp}\big(X, \langle \bar{s} \rangle x \wedge \langle p \rangle y \wedge \langle o \rangle z\big) \\
\mathcal{A}(q_1 \ \text{AND}\ q_2) &= \mathcal{A}(q_1) \wedge \mathcal{A}(q_2) \\
\mathcal{A}(q_1 \ \text{MINUS}\ q_2) &= \mathcal{A}(q_1) \wedge \neg \mathcal{A}(q_2)
\end{aligned}$$

Encoding right-hand side query (q'): translation can be done in the same way as for the positive graph patterns discussed in Chapter 3. Here, we repeat a sketch of that translation procedure:

$$\begin{aligned}
\mathcal{A}(q', m) &= \bigvee_{i=1}^{|m|} \mathcal{A}(q, m_i) \\
\mathcal{A}((x,y,z), m) &= \mathit{lfp}\big(X, \langle \bar{s} \rangle d(m,x) \wedge \langle p \rangle d(m,y) \wedge \langle o \rangle d(m,z)\big) \\
\mathcal{A}(q_1 \ \text{AND}\ q_2, m) &= \mathcal{A}(q_1, m) \wedge \mathcal{A}(q_2, m) \\
\mathcal{A}(q_1 \ \text{MINUS}\ q_2, m) &= \mathcal{A}(q_1, m) \wedge \neg \mathcal{A}(q_2, m) \\
d(m,x) &= \begin{cases} \varphi & \text{if } (x \mapsto \varphi) \in m \\ \top & \text{if } \mathit{unique}(x) \\ x & \text{otherwise} \end{cases}
\end{aligned}$$

Example 41. *Consider the encoding of $q \sqsubseteq q'$ where:*

$$\begin{aligned}
q(x,y) &= (x,a,y) \\
q'(x,y) &= (x,a,y) \ \textit{MINUS}\ (x,b,y)
\end{aligned}$$

The encoding of q is $\mathcal{A}(q) = \mu X.\langle\bar{s}\rangle x \wedge \langle p\rangle a \wedge \langle o\rangle z \vee lfp(X)$. q' can be encoded like that of q because there are no non-distinugished variables in q', thus

$$\begin{aligned}\mathcal{A}(q') &= \mathcal{A}\big((x,a,y)\ \textsc{minus}\ (x,b,y)\big)\\ &= \mathcal{A}\big((x,a,y)\big) \wedge \neg\mathcal{A}\big((x,b,y)\big)\\ &= lfp\big(X, \langle\bar{s}\rangle x \wedge \langle p\rangle a \wedge \langle o\rangle z\big) \wedge \neg lfp\big(Y, \langle\bar{s}\rangle x \wedge \langle p\rangle b \wedge \langle o\rangle z\big)\end{aligned}$$

In the following, for the sake of legibility, we use $\Phi(q, q')$ to denote $\mathcal{A}(q) \wedge \neg\mathcal{A}(q', m) \wedge \varphi_r$.

Theorem 13. *Given SPARQL MINUS queries q and q',*

$$q \sqsubseteq q' \Leftrightarrow \Phi(q, q')$$

is unsatisfiable.

Proof. **Soundness of the encoding** $\Phi(q, q')$ is unsatisfiable implies that $q \sqsubseteq q'$. Consider the contrapositive, $q \not\sqsubseteq q'$ implies $\Phi(q, q')$ is satisfiable. Assume that, $q \not\sqsubseteq q'$. Thus, it follows that there exists a certain RDF graph where a tuple $\overrightarrow{a_i} \in ans(q)$ and $\overrightarrow{a_i} \notin ans(q')$, where $ans(q)$ denotes the answers to q. Now, let us construct a transition system K, using a graph obtained from q by instantiating it using $\overrightarrow{a_i}$. Since K is a construction of q, it follows that $K \models \mathcal{A}(q)$. Consider a sub-transition system/ sub graph of $K = (W, R, L)$ i.e., $K' \subseteq K$. We construct $K' = (W', R', L')$ as:

- for any $a_j \in \overrightarrow{a_i}$ if $\exists\overrightarrow{w} \in W$ where $L(a_j) = \overrightarrow{w}$, then add $\overrightarrow{w}$ to W',
- $\forall n \in W'$ and $\forall n' \in W'$, if $(n, n') \in R(s)$ or $(n, n') \in R(p)$ or $(n, n') \in R(o)$, then $(n, n') \in R'(s)$ or $(n, n') \in R'(p)$ or $(n, n') \in R'(o)$. Note that n' denotes the triple nodes in W,
- $L'(a_j) = L(a_j)$.

To prove that $K' \models \mathcal{A}(q)$, it suffices to see that K' is produced as a canonicalization of q using $\overrightarrow{a_i}$. Thus, the encoding of q is satisfiable in K'. On the other hand, $K' \not\models \mathcal{A}(q')$, since $\overrightarrow{a_i} \notin ans(q')$. Hence, $K' \models \neg\mathcal{A}(q')$. It remains to verify that $K' \models \mathcal{A}(q) \wedge \neg\mathcal{A}(q')$, but it obvious from the semantics of a μ-calculus formula. Also, since K' is a restricted transition system, we have that $K' \models \varphi_r$. Overall, summing up the pieces, we arrive at $K' \models \mathcal{A}(q) \wedge \neg\mathcal{A}(q') \wedge \varphi_r$. Therefore, $\Phi(q, q')$ is satisfiable.

Completeness of the encoding $\Phi(q, q')$ satisfiable implies that $q \not\sqsubseteq q'$. Let us assume that there exists a restricted transition system K such that, $K \models \mathcal{A}(q) \wedge \neg\mathcal{A}(q', m) \wedge \varphi_r$. We construct an RDF graph G from $K = (W, R, L)$ such that q is non-empty when evaluating on G. It is possible because $K \models \varphi_r$.

For each $w_1, w_2, w_3, t \in W$ and for atomic propositions a, b, c, if $L(a) = w_1$, $L(b) = w_2$, $L(c) = w_3$ and $(w_1, t) \in R(s) \wedge (t, w_2) \in R(p) \wedge (t, w_3) \in R(o)$, then $(a, b, c) \in G$. This process is repeated for all states and atomic propositions of the transition system.

After obtaining an RDF graph G from K, it remains to show that $[\![q]\!]_G \not\subseteq [\![q']\!]_G$. From our assumption, we get the following:

$$[\![\mathcal{A}(q) \wedge \neg\mathcal{A}(q')]\!]^K \neq \emptyset \Rightarrow [\![\mathcal{A}(q)]\!]^K \neq \emptyset \text{ and } [\![\neg\mathcal{A}(q', m)]\!]^K \neq \emptyset$$
$$\Rightarrow [\![\mathcal{A}(q)]\!]^K \neq \emptyset \text{ and } [\![\mathcal{A}(q', m)]\!]^K = \emptyset$$

If a formula φ is satisfiable in a restricted transition system K_r, then $[\![\varphi]\!]^{K_r} = S$. As a consequence, $[\![q]\!]_G \neq \emptyset$ and $[\![q']\!]_G = \emptyset$ because G contains all those triples that satisfy q and not q'. Therefore, we get $[\![q]\!]_G \not\subseteq [\![q']\!]_G$. If q' is cyclic, then its non-distinguished variables are encoded using nominals as cycles can be expressed by a formula in a μ-calculus extended with nominals and inverse. Thus, the constraints expressed by $\neg\mathcal{A}(q', m)$ are satisfied in a transition system containing cycles. On the other hand, if q' has a tree-structure, the problem becomes easier as it suffices to replaces its non-distinguished variables with $\top$. Thus, this concludes the completeness part and the overall proof. □

From Theorem 13, we obtain the following proposition on the complexity of containment test.

Proposition 10 (Complexity)**.** *Given SPARQL MINUS queries q and q', checking $q \sqsubseteq q'$ can be performed in a double exponential amount of time.*

8.2 On the Containment of OPTIONAL Graph Patterns

Recently, optimization and static analysis of OPTIONAL graph patterns have been studied in [Letelier *et al.* 2012]. In their study, they have revealed that the containment test can be done in Π_2^P-complete amount of time for well-designed OPTIONAL graph patterns. They have also provided implementations to support their theoretical results. While it is obvious that the study has important implications, it fails to consider projections of SELECT queries, i.e, containment and equivalence are studied for AND-OPTIONAL graph patterns.

If an OPTIONAL pattern fails to match for a particular solution, any variables in that pattern remain unbound (no value) for that solution. We represent the "no value" in this discussion by null. Let us proceed with an example:

Example 42. *Consider the following queries:*

$$q_1\{y, z\} = (x, name, y) \text{ AND } (x, email, z)$$
$$q_2\{y, z\} = (x, name, y) \text{ OPT } (x, email, z)$$

Compare the evaluation of q_1 and q_2 over the graph below,

$$G = \{ \ (p_1, name, n_1), (p_1, email, e_1), (p_2, name, n_2),$$
$$(p_3, name, n_3), (p_3, email, e_3), (p_4, name, n_4),$$
$$(p_5, email, e_5) \ \}$$

$$[\![q_1]\!]_G = \{(n_1, e_1), (n_3, e_3)\}$$
$$[\![q_2]\!]_G = \{(n_1, e_1), (n_2, null), (n_3, e_3), (n_4, null)\}$$

Clearly, $[\![q_1]\!]_G \subseteq [\![q_2]\!]_G$ but $[\![q_2]\!]_G \not\subseteq [\![q_1]\!]_G$. This holds also for the general case i.e., for any RDF graph, because in the evaluation of q_1, y is bound only when z is bound due to conjunction. On the other hand, in the evaluation of q_2, y is bound irrespective of z i.e., the value of y is returned whether z is bound to a specific value or null. Thus, $q_1 \sqsubseteq q_2$ but $q_2 \not\sqsubseteq q_1$.

It has been shown that the combined complexity of SPARQL query evaluation raises from PTime-membership for the conjunctive fragment to PSPACE-completeness when OPTIONAL is considered [Pérez *et al.* 2009, Schmidt *et al.* 2010]. In [Pérez *et al.* 2009], the class of well-designed SPARQL graph patterns was introduced as a fundamental fragment of OPTIONAL queries with good behavior for query evaluation. In particular, it was shown that the complexity of the evaluation problem for the well-designed fragment is coNP-complete. For this study, we consider this specific fragment.

Definition 31 (Well-designed graph patterns)**.** *A graph pattern is well-designed iff for every* OPT *in the pattern*

$$(\ldots\ldots(P_1 \ \text{OPT} \ P_2)\ldots\ldots)$$

if a variable occurs inside P_2 and anywhere outside the OPT*, then the variable must also occur inside P_1 [Letelier et al. 2012].*

Example 43. (x, a, b) OPT $\big((y, c, d)$ AND $(x, e, z)\big)$ *in this example, the variable x occurs inside and outside the* OPT *operator.*

The semantics of the OPT operator leads to a different definition to query containment. In databases, containment of two queries is defined if they have the same number of distinguished variables.

Definition 32 (Optional Queries Containment)**.**

$$q_1\{\overrightarrow{x}\} \sqsubseteq q_2\{\overrightarrow{x}\} \Leftrightarrow \forall G. \bigwedge_{i=1}^{n} (\pi_{x_i}([\![q_1]\!]_G) \subseteq \pi_{x_i}([\![q_2]\!]_G))$$

where $x_i \in \overrightarrow{x}$ and $n = |\overrightarrow{x}|$

Encoding OPTIONAL Graph Patterns Relying on the set semantics of OPTIONAL graph pattern evaluation, we propose to transform a pattern of the form $(P_1$ OPT $P_2)$ into $((P_1$ AND $P_2)$ UNION $P_1)$. The transformed query can be easily encoded into a μ-calculus formula. To encode $q\{\overrightarrow{x}\} \sqsubseteq q'\{\overrightarrow{x}\}$, we proceed as follows:

Encoding q: when translating q, the IRIs and variables are translated into nominals in μ-calculus. The OPT operator is encoded into a conjunction and a disjunction as

shown below.

$$\begin{aligned}\mathcal{A}((x,y,z)) &= lfp\big(X, \langle\bar{s}\rangle x \wedge \langle p\rangle y \wedge \langle o\rangle z\big)\\ \mathcal{A}(q_1 \ \text{AND}\ q_2) &= \mathcal{A}(q_1) \wedge \mathcal{A}(q_2)\\ \mathcal{A}(q_1 \ \text{OPT}\ q_2) &= \mathcal{A}(q_1 \ \text{AND}\ q_2) \vee \mathcal{A}(q_1)\end{aligned}$$

Encoding q': IRIs and non-distinguished variables are encoded into nominals and the non-distinguished variables are translated in the same way as AND-UNION queries in Chapter 3.

The encoding of OPTIONAL patterns is not linear, it grows exponentially with respect to the size of the OPTIONAL patterns. In addition, the encoding of the right-hand side query q' is also exponential due to the non-distinguished variables that appear in cycles in the query.

Reducing OPTIONAL pattern containment into unsatisfiability test

Conjecture 1. *Given two well-formed OPTIONAL pattern queries q and q', $q \sqsubseteq q' \Leftrightarrow \mathcal{A}(q) \wedge \neg\mathcal{A}(q', m) \wedge \varphi_r$.*

8.3 On the containment of SPARQL queries under bag semantics

A *bag* or *multiset* is a collection of objects each of which occurs one or more times in the collection. Two kinds of semantics are given to SPARQL: set and bag semantics. Bag semantics is the W3C standard semantics given to SPARQL. In databases, for SQL queries three kinds of semantics have been proposed: *bag semantics*, i.e. duplicate tuples are allowed in both the database and the results of queries, *bag-set* semantics, i.e. duplicates are allowed in the result of the queries but not in the database, and finally *set semantics*, that is, duplicates are not allowed in the result of the queries [Afrati *et al.* 2010]. For conjunctive queries, the containment problem under both bag and bag-set semantics is proved to be Π_2^P-hard [Chaudhuri & Vardi 1993]. On the other hand, if both the left and right-hand side queries do not contain projections, the problem can be solved in $n^2 log(n)$ amount of time [Afrati *et al.* 2010]. As already discussed, under set semantics the complexity of containment test is NP-complete [Chandra & Merlin 1977]. Note that, containment of union of conjunctive queries under bag semantics is undecidable [Ioannidis & Ramakrishnan 1995]. In addition, the containment of conjunctive queries with inequalities under bag semantics is also undecidable [Jayram *et al.* 2006].

On the other side, in this thesis we have addressed the containment problem for various fragments of SPARQL under set semantics. However, for the conjunctive fragments of SPARQL, the results from the study of conjunctive queries under bag-set semantics in databases carry over. In the following, we denote by $\sqsubseteq_s$ the containment

problem under set semantics and $\sqsubseteq_b$ under bag semantics. The difference between set and bag semantics for containment in SPARQL is shown by the following example.

Example 44. *Find people and their names:*

q
```
SELECT ?x ?y
WHERE {
  ?x :name ?y .
}
```

q_1
```
SELECT ?x ?y
WHERE {
  ?x :name ?y .
  ?x :name ?z .
}
```

Under set semantics q and q_1 are equivalent however under bag semantics they are not. This is due to the redundant join in q_1 that affects output tuples multiplicities. On the other hand, it is easy to see that $q \sqsubseteq_b q_1$ under bag semantics but $q_1 \not\sqsubseteq_b q$.

Example 45. *Consider the containment problem between the following queries under both set and bag semantics:*

q_1
```
SELECT ?age
WHERE {
  ?x a :Student .
  ?x :age ?age .
}
```

q_2
```
SELECT ?age WHERE {
  ?x a :Student .
  ?x a :Employee .
  ?x :age ?age .
  ?x :job ?title .
}
```

Under set semantics, $q_2 \sqsubseteq_s q_1$ since there exists a homomorphism from q_1 to q_2 whereas under bag semantics $q_2 \not\sqsubseteq_b q_1$ this is due to a student may have multiple jobs. Conversely, under both set and bag semantics, $q_1 \not\sqsubseteq q_2$.

From these examples, it is easy to see that the bag semantics affects the containment and equivalence problems. Nevertheless, the satisfiability problem (whether a query has a solution or not) remains unaffected. If $q \sqsubseteq_b q'$, then it always holds that $q \sqsubseteq_s q'$, however, the converse is not always true as already shown in the above examples. Even the contrapositive, if $q \not\sqsubseteq_s q'$, then $q \not\sqsubseteq_b q'$ does not necessarily hold if the queries contain MINUS graph patterns. Another important point to consider is when queries (or one of them) contain no projections. In this regard, for conjunctive queries in databases, it has been proved that this problem is decidable in polynomial time [Afrati *et al.* 2010]. As such, the results of evaluating such queries on any set-valued database are always sets; hence to decide bag-set containment of such queries it suffices to decide containment under set semantics. Consequently, we can draw the following: as SPARQL queries are evaluated over set valued databases (RDF graphs), containment of conjunctive SPARQL queries without projections is decidable in polynomial time. Importantly, we noted that the bag semantics of SPARQL is similar to that of the bag-set semantics of conjunctive queries in databases, this is due to RDF data (or relations) are sets and not multisets. For more on the transformation of SPARQL into relationa algebra and RDF datasets into database relations, we refer the reader to [Cyganiak 2005].

8.4 Conclusion

In this chapter, we have proposed an approach to encode SPARQL conjunctive queries with negation (known as MINUS queries) in the μ-calculus to study containment test. We have proved that this encoding is sound and complete and runs in a double exponential amount of time in the worst-case. This bound lowers to just exponential when the query on the right-hand side has a tree-structure. In fact, containment of tree-structured MINUS queries is PSPACE as already shown in Chapter 6. Moreover, we have extended the encoding procedure for union of conjunctive fragments of SPARQL to well-designed OPTIONAL graph patterns. Finally, we have discussed the issues related to containment test under bag semantics.

Chapter 9

Conclusion

Contents

We now summarize the main contributions of this work, and propose further research directions.

9.1 Summary

Containment is a well-studied problem for relational database query languages. It started with a pioneering paper from Chandra and Merlin [Chandra & Merlin 1977] and has been addressed for different query languages since. Thus, SPARQL is no exception given the advantages of containment thence this study is carried out. We summarize the results achieved in the development of this work.

Having well-behaved computational and model theoretic properties and implementations that have been put to practice, μ-calculus has been chosen for the task of static analysis of SPARQL queries. μ-calculus formulas are interpreted over transition systems. SPARQL queries are evaluated over RDF graphs, these graphs can be transformed to other types of graphs: hypergrahs, bipartite graphs, transition systems and others. Thus, given a graph logic, RDF graphs can be translated into transition systems and SPARQL queries into μ-calculus formulas, thus the formulas can be interpreted over transition systems. Chapter 3 presents a technique for translating RDF graphs into transition systems. In fact, the transition systems are bipartite graphs obtained by introducing three nodes for each triple elements (subject, predicate and object nodes) and a fourth one (called triple node) for each triple in the graph. The subject, predicate and object nodes are connected to a triple node, the edges are labelled with transition programs s, p and o respectively.

Containment of queries can be reduced to satisfiability test by encoding the set inclusion as implication and the queries as formulas. In Chapter 3, algorithms are proposed for translating queries and $\mathcal{ALCH}$ axioms into μ-calculus formulas. The principle of the translation is based on reification where each triple is represented by a node, connected to the subject, predicate and object elements of the query, for instance, the query $(x, name, y)$ is encoded into $\langle \bar{s} \rangle x \wedge \langle p \rangle name \wedge \langle o \rangle y$. After producing the encoding of a triple pattern, the least fixpoint operator μ is used to propagate the

encoding to all nodes of the transition system. That is, μ encodes a reflexive transitive closure over all programs. The encoding of the right-hand side query is different from that of the left due to the non-distinguished variables that appear in cycles in the query. In fact, this is the reason for high complexity bounds in containment problems in general. Finally, the soundness and completeness of the reduction is proved and a double exponential upper bound complexity is established for the problem. This complexity bound is proven to be complete for the schema language $\mathcal{ALCH}$.

The next result comes from the study of the containment of path SPARQL queries (a fragment of SPARQL 1.1 property paths[1]). In this direction, in Chapter 4, we have addressed three related problems: (i) the containment of PSPARQL queries, (ii) the containment of PSPARQL queries with inverse (also known as C2RPQs in semi-structured data), and finally (iii) the containment of PSPARQL queries under the presence of expressive description logic axioms. For these problems, we propose novel encoding techniques to reduce containment to unsatisfiability test. For all these problems, containment can be tested in a double exponential amount of time.

Being a W3C recommended language and growing fame in the semantic web, SPARQL is currently being extended as a query language for ontologies (a.k.a. SPARQL entailment regimes) [Glimm & Krötzsch 2010]. In this direction, we contribute towards the study of the containment problem under entailment regimes. In Chapter 5, we addressed this issue for two entailment regimes: containment under the simpleRDFS entailment regime and containment under the OWL-$\mathcal{ALCH}$ Direct Semantics entailment regime. For the later problem, we used the query language, SPARQL-OWL, originally proposed in [Glimm 2011]. Moreover, we have also used a fragment of OWL 2 to retain decidability of the problem, as encoding the full OWL 2 is not possible with the μ-calculus without graded modalities. The fragment of OWL we selected is called OWL-$\mathcal{ALCH}$, i.e, the part based on the description logic $\mathcal{ALCH}$. Deciding the containment of SPARQL-OWL queries under the OWL-$\mathcal{ALCH}$ Direct semantics requires exponential amount of time. This is because in the entailment regimes non-distinguished variables cannot appear by design, i.e., all the variables in the basic graph pattern are treated as distinguished, projection is a post-processing step and not part of entailment, the containment problem is ExpTime. However, this bound rises to 2ExpTime if the non-distinguished variables are treated in the usual way (as existential variables) a priori to performing reasoning.

When path query containment is considered under ontology axioms, it is necessary to chose an expressive logic (such as the μ-calculus) upto the expressivity of the queries and schema language. But when studying containment without schema axioms (or when the schema language is weak), one can use a less expressive logic than the μ-calculus to perform containment test. In Chapter 6, we studied the containment of tree-structured SPARQL queries by encoding them in the modal logic K_n. The complexity of satisfiability test in K_n is PSPACE. It is less expressive than μ-calculus. The encoding methods have been experimented using benchmark queries. Beforehand, we carried out experiments in order to asses how many of real world queries are of tree-structure.

[1] http://www.w3.org/TR/sparql11-property-paths/

From the DBpedia querylog, 87% of the queries are found to be tree-structured. Thus, this result on its own is enough to motivate the purpose of Chapter 6.

In order to complement the theoretical results that have been established in this thesis, we have carried out experiments. In this regard, in Chapter 7, we proposed a compliance benchmark for containment, equivalence and satisfiability of semantic web queries. The benchmark is used to test the current-state-of-the-art tools. A comparison of these tools based on running times is discussed. Furthermore, as already discussed, the benchmark is designed to assess the containment solvers that are available presently, with room for extension.

Finally, in Chapter 8, we propose to extend our approaches to study the containment problem for negation and optional patterns.

Overall, we summarize the main results achieved in this thesis in Table 9.1. The results, shown in the complexity column, that are marked with * are novel to this work whereas the other results come from the study of relational algebra in databases (and OPTIONAL graph pattern containment is studied in [Letelier *et al.* 2012]).

9.2 Perspectives

It is beyond the scope of this thesis to address the containment problem for every possible fragment of SPARQL (resp. PSPARQL) and schema language (for instance, $\mathcal{SH}$ family). This work lies the theoretical and experimental foundations, provides simply extendible and implementable methods, for instance, the containment benchmark is designed with respect to the current-state-of-the-art, so it needs to evolve side-by-side with the containment solvers.

Table 9.1 features some of the open problems, those labeled with a dash - symbol. The schema languages we considered come from the fragments of $\mathcal{SROIQ}$, this logic underlies the foundations of OWL 2. A detailed discussion on the complexity of the fragments of SPARQL can be found in [Pérez *et al.* 2009]. Another interesting theoretical setting is presented in Table 9.2 where fragments of the μ-calculus, SPARQL graph patterns, and various types of description logics are shown. A line of research problems can be seen here by considering different combinations of SPARQL graph patterns and schema languages (DL) in an expressive μ-calculus fragment to study static analysis.

	Graph pattern	Schema Language	Complexity of Containment
SPARQL	AND		NP [Chandra & Merlin 1977]
	AND-UNION		NP [Chandra & Merlin 1977]
	OPT		Π_2^P [Letelier *et al.* 2012]
	AND-OPT		Π_2^P [Letelier *et al.* 2012]
	AND-UNION-OPT		undecidable
	MINUS		2ExpTime*
SPARQL	AND	$\mathcal{ALCH}$	2ExpTime*
	AND-UNION	$\mathcal{ALCH}$	2ExpTime*
	AND-UNION	simpleRDFS entailment	ExpTime*
	AND-UNION	OWL-$\mathcal{ALCH}$ entailment	ExpTime*
	OPT	-	-
	AND-OPT	-	-
	MINUS	-	-
PSPARQL	AND		2ExpTime*
	AND-UNION		2ExpTime*
	OPT		-
	AND-OPT		-
	MINUS		-
PSPARQL	AND	$\mathcal{ALCH}$	2ExpTime*
	AND-UNION	$\mathcal{ALCH}$	2ExpTime*
	OPT	-	-
	AND-OPT	-	-
	MINUS	-	-

Table 9.1: Summary of results on the containment of SPARQL queries.

DL	Complexity
$\mathcal{ALC}$	ExpTime
$\mathcal{ALCI}$	ExpTime
$\mathcal{ALCHI}$	ExpTime
$\mathcal{ALCHIF}$	ExpTime
$\mathcal{SHIF}$ (OWL-Lite)	ExpTime
$\mathcal{SHIN}$	ExpTime
$\mathcal{SHOIN}$ (OWL-DL)	NExpTime
$\mathcal{SHIQ}$	ExpTime
$\mathcal{SHOIQ}$	NExpTime
$\mathcal{SROIQ}$	N2ExpTime
OWL profiles	

SPARQL fragment	Complexity
AND	PTime
AND-UNION	NP
AND-OPT	coNP
AND-UNION-OPT	PSPACE
μ **extensions**	**Complexity**
μ	ExpTime
$+ \mathcal{O} + \mathcal{I} + \mathcal{N}$	Undecidable
$+ \mathcal{O} + \mathcal{I}$	ExpTime
$+\mathcal{I} + \mathcal{N}$	ExpTime
$+ \mathcal{O} + \mathcal{N}$	ExpTime
$+ \mathcal{N}$	ExpTime

Table 9.2: A tradeoff among μ-calculus fragments, DL schema languages and SPARQL fragments where $\mathcal{O}$ = nominals, $\mathcal{N}$ = number restrictions (graded modalities), and $\mathcal{I}$ = inverse (backward modalities).

Bibliography

[Abiteboul *et al.* 1995] S. Abiteboul, R. Hull and V. Vianu. Foundations of databases, volume 8. Addison-Wesley, 1995. (Cited on pages 4, 35 and 36.)

[Afrati *et al.* 2010] F.N. Afrati, M. Damigos and M. Gergatsoulis. *Query containment under bag and bag-set semantics.* Information Processing Letters, vol. 110, no. 10, pages 360–369, 2010. (Cited on pages 121 and 122.)

[Aho *et al.* 1979] A. V. Aho, Y. Sagiv and J. D. Ullman. *Equivalences Among Relational Expressions.* SIAM J. Comput., vol. 8, no. 2, pages 218–246, 1979. (Cited on pages 38 and 70.)

[Alkhateeb *et al.* 2009] F. Alkhateeb, J.F. Baget and J. Euzenat. *Extending SPARQL with regular expression patterns (for querying RDF).* J. Web Semantics, vol. 7, no. 2, pages 57–73, 2009. (Cited on pages 29, 41, 57 and 77.)

[Alkhateeb 2008] F. Alkhateeb. *Querying RDF (S) with regular expressions.* PhD thesis, Université Joseph Fourier, 2008. thesis. (Cited on page 68.)

[Angles & Gutierrez 2008] R. Angles and C. Gutierrez. *The Expressive Power of SPARQL.* The Semantic Web-ISWC 2008, pages 114–129, 2008. (Cited on pages 4, 35 and 36.)

[Baader & Nutt 2003] F. Baader and W. Nutt. *The description logic handbook.* pages 43–95. Cambridge University Press, New York, NY, USA, 2003. (Cited on page 14.)

[Baader *et al.* 2007] F. Baader, D. Calvanese, D. McGuinness, D. Nardi and P. F. Patel-Schneider, editeurs. The description logic handbook: Theory, implementation, and applications. Cambridge University Press, 2007. ISBN 9780511717383. (Cited on pages 14, 15 and 17.)

[Baget 2005] J.F. Baget. *RDF Entailment as a Graph Homomorphism.* The Semantic Web–ISWC 2005, pages 82–96, 2005. (Cited on page 42.)

[Barceló *et al.* 2010] P. Barceló, C. Hurtado, L. Libkin and P. Wood. *Expressive languages for path queries over graph-structured data.* In PODS'10, pages 3–14. ACM, 2010. (Cited on pages 4, 38, 57 and 70.)

[Berners-Lee & Hendler 2001] T. Berners-Lee and J. Hendler. *Scientific publishing on the semantic web.* Nature, vol. 410, pages 1023–1024, 2001. (Cited on page 9.)

[Bernstein & Chiu 1981] P.A. Bernstein and D.M.W. Chiu. *Using semi-joins to solve relational queries.* Journal of the ACM (JACM), vol. 28, no. 1, pages 25–40, 1981. (Cited on pages 91 and 92.)

[Bienvenu 2012] M. Bienvenu. *On the Complexity of Consistent Query Answering in the Presence of Simple Ontologies*. In Twenty-Sixth AAAI Conference on Artificial Intelligence, 2012. (Cited on pages 3 and 70.)

[Bizer & Schultz 2008] C. Bizer and A. Schultz. *Benchmarking the performance of storage systems that expose SPARQL endpoints*. In Proc. 4 th International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS), 2008. (Cited on page 39.)

[Bizer & Schultz 2009] C. Bizer and A. Schultz. *The Berlin SPARQL benchmark*. International Journal on Semantic Web and Information Systems (IJSWIS), vol. 5, no. 2, pages 1–24, 2009. (Cited on page 39.)

[Bizer *et al.* 2009] C. Bizer, T. Heath and T. Berners-Lee. *Linked data-the story so far*. International Journal on Semantic Web and Information Systems (IJSWIS), vol. 5, no. 3, pages 1–22, 2009. (Cited on page 9.)

[Blackburn *et al.* 2007] P. Blackburn, J. van Benthem and F. Wolter. Handbook of Modal Logic. Elsevier, 2007. (Cited on pages 5, 31, 32, 39 and 98.)

[Bonatti & Peron 2004] P.A. Bonatti and A. Peron. *On the undecidability of logics with converse, nominals, recursion and counting*. Artificial Intelligence, vol. 158, no. 1, pages 75–96, 2004. (Cited on page 34.)

[Bonatti *et al.* 2006] P. A. Bonatti, C. Lutz, A. Murano and M. Y. Vardi. *The Complexity of Enriched μ-calculi*. Automata, Languages and Programming, pages 540–551, 2006. (Cited on pages 33, 34 and 55.)

[Calvanese & Rosati 2003] D. Calvanese and R. Rosati. *Answering Recursive Queries under Keys and Foreign Keys is Undecidable*. In Proc. of the 10th Int. Workshop on Knowledge Representation meets Databases (KRDB 2003), volume 79, pages 3–14, 2003. (Cited on page 38.)

[Calvanese *et al.* 1998] D. Calvanese, G. De Giacomo and M. Lenzerini. *On the decidability of query containment under constraints*. In Proceedings of PODS, pages 149–158. ACM, 1998. (Cited on page 37.)

[Calvanese *et al.* 2000] D. Calvanese, G. De Giacomo, M. Lenzerini and M. Y. Vardi. *Containment of Conjunctive Regular Path Queries with Inverse*. In Proc. of the 7th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2000), pages 176–185, 2000. (Cited on pages 3, 4, 38, 57, 68 and 150.)

[Calvanese *et al.* 2003] D. Calvanese, G. De Giacomo, M. Lenzerini and M. Y. Vardi. *Reasoning on Regular Path Queries*. SIGMOD Record, vol. 32, no. 4, pages 83–92, 2003. (Cited on pages 38 and 57.)

[Calvanese *et al.* 2007] 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, vol. 39, no. 3, pages 385–429, 2007. (Cited on page 37.)

[Calvanese *et al.* 2008] D. Calvanese, G. De Giacomo and M. Lenzerini. *Conjunctive Query Containment and Answering under Description Logics Constraints*. ACM Trans. on Computational Logic, vol. 9, no. 3, pages 22.1–22.31, 2008. (Cited on pages 3, 37, 38, 42, 47, 56, 70, 91 and 150.)

[Calvanese *et al.* 2011] D. Calvanese, M. Ortiz and M. Simkus. *Containment of Regular Path Queries under Description Logic Constraints*. In Proc. of the 22nd Int. Joint Conf. on Artificial Intelligence (IJCAI 2011), 2011. (Cited on pages 38, 57 and 70.)

[Chandra & Merlin 1977] 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, pages 77–90. ACM, 1977. (Cited on pages 3, 36, 39, 69, 91, 115, 121, 125, 128, 149 and 153.)

[Chaudhuri & Vardi 1992] S. Chaudhuri and M.Y. Vardi. *On the equivalence of recursive and nonrecursive datalog programs*. In Proceedings of the eleventh ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems, pages 55–66. ACM, 1992. (Cited on page 36.)

[Chaudhuri & Vardi 1993] S. Chaudhuri and M.Y. Vardi. *Optimization of real conjunctive queries*. In Proceedings of the twelfth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems, pages 59–70. ACM, 1993. (Cited on page 121.)

[Chebotko *et al.* 2006] A. Chebotko, S. Lu, H.M. Jamil and F. Fotouhi. *Semantics preserving SPARQL-to-SQL query translation for optional graph patterns*. Rapport technique, Technical Report TR-DB-052006-CLJF, 2006. (Cited on page 35.)

[Chekol *et al.* 2011a] M. W. Chekol, J. Euzenat, P. Genevès and N. Layaïda. *PSPARQL Query Containment*. In DBPL'11, August 2011. (Cited on pages 6 and 39.)

[Chekol *et al.* 2011b] M. W. Chekol, J. Euzenat, P. Genevès and N. Layaïda. *PSPARQL Query Containment*. Research report 7641, June 2011. `http://hal.inria.fr/inria-00598819/PDF/RR-7641.pdf`. (Cited on page 6.)

[Chekol *et al.* 2012a] M. W. Chekol, J. Euzenat, P. Genevès and N. Layaïda. *SPARQL Query Containment under RDFS Entailment Regime*. In IJCAR'12, pages 134–148. Springer, 2012. (Cited on pages 6, 39 and 110.)

[Chekol *et al.* 2012b] M. W. Chekol, J. Euzenat, P. Genevès and N. Layaïda. *SPARQL Query Containment under SHI Axioms*. In AAAI'12, volume 1, pages 10–16, 2012. (Cited on pages 6, 39, 84 and 110.)

[Chekuri & Rajaraman 1997] C. Chekuri and A. Rajaraman. *Conjunctive query containment revisited*. Database Theory–ICDT'97, pages 56–70, 1997. (Cited on page 92.)

[Cyganiak 2005] R. Cyganiak. *A relational algebra for SPARQL*. Digital Media Systems Laboratory HP Laboratories Bristol. HPL-2005-170, 2005. (Cited on pages 35 and 122.)

[Eiter *et al.* 2009] T. Eiter, C. Lutz, M. Ortiz and M. Šimkus. *Query answering in description logics with transitive roles*. In Proc. of IJCAI, pages 759–764, 2009. (Cited on pages 37 and 70.)

[Farré *et al.* 2006] C. Farré, W. Nutt, E. Teniente and T. Urpí. *Containment of conjunctive queries over databases with null values*. Database Theory–ICDT 2007, pages 389–403, 2006. (Cited on page 115.)

[Fernández *et al.* 2010] J. Fernández, M. Martínez-Prieto and C. Gutierrez. *Compact representation of large RDF data sets for publishing and exchange*. The Semantic Web–ISWC 2010, pages 193–208, 2010. (Cited on page 2.)

[Florescu *et al.* 1998] D. Florescu, A. Levy and D. Suciu. *Query containment for conjunctive queries with regular expressions*. PODS '09, pages 139–148. ACM, 1998. (Cited on pages 4, 57 and 70.)

[Genevès & Layaïda 2006] Pierre Genevès and Nabil Layaïda. *A system for the static analysis of XPath*. ACM Trans. Inf. Syst., vol. 24, no. 4, pages 475–502, 2006. (Cited on page 43.)

[Genevès *et al.* 2007] P. Genevès, N. Layaïda and A. Schmitt. *Efficient Static Analysis of XML Paths and Types*. In PLDI '07, pages 342–351, New York, NY, USA, 2007. ACM. (Cited on pages 3, 4, 38, 41, 56, 72, 100, 102, 103, 104 and 150.)

[Gerber *et al.* 2008] A. Gerber, A. Van der Merwe and A. Barnard. *A functional semantic web architecture*. The Semantic Web: Research and Applications, pages 273–287, 2008. (Cited on pages 10 and 11.)

[Glimm & Krötzsch 2010] B. Glimm and M. Krötzsch. *SPARQL Beyond Subgraph Matching*. The Semantic Web–ISWC 2010, pages 241–256, 2010. (Cited on pages 83 and 126.)

[Glimm *et al.* 2008] B. Glimm, I. Horrocks, C. Lutz and U. Sattler. *Conjunctive query answering for the description logic SHIQ*. J Artif Intell Res, vol. 31, pages 157–204, 2008. (Cited on pages 37, 70, 75 and 82.)

[Glimm 2011] B. Glimm. *Using SPARQL with RDFS and OWL entailment*. Reasoning Web. Semantic Technologies for the Web of Data, pages 137–201, 2011. (Cited on pages 73, 82, 83, 89 and 126.)

[Gottlob *et al.* 2001] G. Gottlob, N. Leone and F. Scarcello. *The complexity of acyclic conjunctive queries.* Journal of the ACM (JACM), vol. 48, no. 3, pages 431–498, 2001. (Cited on page 92.)

[Grau *et al.* 2008] B.C. Grau, I. Horrocks, B. Motik, B. Parsia, P. Patel-Schneider and U. Sattler. *OWL 2: The next step for OWL.* Web Semantics: Science, Services and Agents on the World Wide Web, vol. 6, no. 4, pages 309–322, 2008. (Cited on page 18.)

[Gray 1992] J. Gray. Benchmark handbook: for database and transaction processing systems. Morgan Kaufmann Publishers Inc., 1992. (Cited on page 104.)

[Groppe *et al.* 2009] J. Groppe, S. Groppe and J. Kolbaum. *Optimization of SPARQL by using coreSPARQL.* In ICEIS (1), pages 107–112, 2009. (Cited on pages 34 and 39.)

[Gutierrez *et al.* 2004] C. Gutierrez, C. Hurtado and A. O. Mendelzon. *Foundations of Semantic Web Databases.* PODS '04, pages 95–106, New York, NY, USA, 2004. (Cited on pages 12 and 34.)

[Harris & Seaborne 2012] S. Harris and A. Seaborne. *SPARQL 1.1 Query Language.* W3C Working Draft, 2012. (Cited on page 116.)

[Hayes 2004] P. Hayes. *RDF Semantics.* W3C Recommendation, 2004. (Cited on pages 1, 11, 12, 14, 23, 34, 52, 63 and 73.)

[Heath & Bizer 2011] T. Heath and C. Bizer. *Linked data: Evolving the web into a global data space.* Synthesis Lectures on the Semantic Web: Theory and Technology, vol. 1, no. 1, pages 1–136, 2011. (Cited on page 9.)

[Hitzler *et al.* 2009] P. Hitzler, M. Krötzsch, B. Parsia, P. F. Patel-Schneider and S. Rudolph. *OWL 2 Web Ontology Language Primer.* W3C Rec., 2009. (Cited on page 22.)

[Hogan *et al.* 2009] A. Hogan, A. Harth and A. Polleres. *Scalable authoritative OWL reasoning for the web.* International Journal on Semantic Web and Information Systems (IJSWIS), vol. 5, no. 2, pages 49–90, 2009. (Cited on pages 75 and 79.)

[Horrocks & Patel-Schneider 2003] I. Horrocks and P. F. Patel-Schneider. *Reducing OWL entailment to description logic satisfiability.* The Semantic Web-ISWC 2003, pages 17–29, 2003. (Cited on page 18.)

[Horrocks & Patel-Schneider 2010] I. Horrocks and P.F. Patel-Schneider. *Knowledge Representation and Reasoning on the Semantic Web: OWL*, 2010. http://www.cs.ox.ac.uk/ian.horrocks/Publications/download/2010/HoPa10a.pdf. (Cited on pages 17, 18, 20 and 21.)

[Horrocks *et al.* 2006] I. Horrocks, O. Kutz and U. Sattler. *The even more irresistible SROIQ.* In Proc. of KR 2006, pages 57–67, 2006. (Cited on pages 15 and 89.)

[Ioannidis & Ramakrishnan 1995] Y.E. Ioannidis and R. Ramakrishnan. *Containment of conjunctive queries: Beyond relations as sets.* ACM Transactions on Database Systems (TODS), vol. 20, no. 3, pages 288–324, 1995. (Cited on pages 4 and 121.)

[Jayram *et al.* 2006] TS Jayram, P.G. Kolaitis and E. Vee. *The containment problem for real conjunctive queries with inequalities.* In PODS, pages 80–89. ACM, 2006. (Cited on page 121.)

[Johnson & Klug 1984] D. S. Johnson and A. C. Klug. *Testing Containment of Conjunctive Queries under Functional and Inclusion Dependencies.* Journal of Computer and System Sciences, vol. 28, no. 1, pages 167–189, 1984. (Cited on page 38.)

[Kazakov 2008] Y. Kazakov. *SRIQ and SROIQ are Harder than SHOIQ.* In Description Logics, 2008. (Cited on pages 15 and 18.)

[Klug 1988] A. Klug. *On conjunctive queries containing inequalities.* Journal of the ACM (JACM), vol. 35, no. 1, pages 146–160, 1988. (Cited on page 70.)

[Kollia *et al.* 2011] I. Kollia, B. Glimm and I. Horrocks. *SPARQL Query Answering over OWL Ontologies.* In Proc. 8th ESWC, Heraklion (GR), volume 6643 of *LNCS*, pages 382–396, 2011. (Cited on pages 39, 75 and 83.)

[Kozen 1983] D. Kozen. *Results on the propositional μ-calculus.* Theor. Comp. Sci., vol. 27, pages 333–354, 1983. (Cited on pages 32, 34, 41 and 110.)

[Krötzsch 2012] M. Krötzsch. *The Not-So-Easy Task of Computing Class Subsumptions in OWL RL.* In The Semantic Web - ISWC 2012 - 11th International Semantic Web Conference, Boston, MA, USA, November 11-15, 2012, Proceedings, Part I, volume 7649 of *Lecture Notes in Computer Science*, pages 279–294. Springer, 2012. (Cited on page 76.)

[Kupferman *et al.* 2002] O. Kupferman, U. Sattler and M. Vardi. *The complexity of the graded μ-calculus.* Automated Deduction—CADE-18, pages 173–206, 2002. (Cited on pages 33 and 34.)

[Leclere & Mugnier 2006] M. Leclere and M.L. Mugnier. *Some algorithmic improvements for the containment problem of conjunctive queries with negation.* Database Theory–ICDT 2007, pages 404–418, 2006. (Cited on pages 115 and 116.)

[Letelier *et al.* 2012] A. Letelier, J. Pérez, R. Pichler and S. Skritek. *Static analysis and optimization of semantic web queries.* In PODS'12, pages 89–100. ACM, 2012. (Cited on pages 4, 34, 39, 99, 103, 109, 115, 119, 120, 127, 128, 152 and 153.)

[Levandoski & Mokbel 2009] J.J. Levandoski and M.F. Mokbel. *RDF data-centric storage*. In Web Services, 2009. ICWS 2009. IEEE International Conference on, pages 911–918. IEEE, 2009. (Cited on page 2.)

[Levy & Rousset 1996] A. Levy and M.C. Rousset. *Verification of Knowledge Bases based on Containment Checking*. Artificial Intelligence, vol. 101, pages 101–1, 1996. (Cited on page 70.)

[Levy & Sagiv 1993] A.Y. Levy and Y. Sagiv. *Queries independent of updates*. In proc. VLDB, pages 171–171. Citeseer, 1993. (Cited on pages 70 and 115.)

[Lutz *et al.* 2009] C. Lutz, D. Toman and F. Wolter. *Conjunctive Query Answering in the Description Logic EL Using a Relational Database System*. In IJCAI, pages 2070–2075, 2009. (Cited on page 37.)

[Lutz 2008] C. Lutz. *The complexity of conjunctive query answering in expressive description logics*. Automated Reasoning, pages 179–193, 2008. (Cited on pages 37 and 70.)

[Mateescu *et al.* 2009] R. Mateescu, S. Meriot and S. Rampacek. Extending SPARQL with Temporal Logic. INRIA Research Report, RR-7056, 2009. (Cited on page 42.)

[Mohamed *et al.* 2011] K. Mohamed, M. Leclère and M.L. Mugnier. *A theoretical and experimental comparison of algorithms for the containment of conjunctive queries with negation*. In Database and Expert Systems Applications, pages 466–480. Springer, 2011. (Cited on pages 115 and 116.)

[Motik *et al.* 2009] B. Motik, P. F. Patel-Schneider and B. Parsia. *OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax*. W3C Rec., 2009. (Cited on pages 18, 22 and 23.)

[Muñoz *et al.* 2007] S. Muñoz, J. Pérez and C. Gutierrez. *Minimal Deductive Systems for RDF*. volume 4519 of *LNCS*, pages 53–67, 2007. (Cited on pages 12, 13, 28 and 76.)

[Ortiz *et al.* 2008a] M. Ortiz, D. Calvanese and T. Eiter. *Data Complexity of Query Answering in Expressive Description Logics via Tableaux*. Journal of Automated Reasoning, vol. 41, no. 1, pages 61–98, 2008. (Cited on page 37.)

[Ortiz *et al.* 2008b] M. Ortiz, M. Šimkus and T. Eiter. *Worst-case optimal conjunctive query answering for an expressive description logic without inverses*. In Proc. of AAAI, volume 8, 2008. (Cited on page 37.)

[Pérez *et al.* 2009] J. Pérez, M. Arenas and C. Gutierrez. *Semantics and complexity of SPARQL*. ACM Transactions on Database Systems (TODS), vol. 34, no. 3, page 16, 2009. (Cited on pages 25, 26, 28, 95, 98, 120 and 127.)

[Pichler *et al.* 2010] R. Pichler, A. Polleres, S. Skritek and S. Woltran. *Redundancy elimination on rdf graphs in the presence of rules, constraints, and queries.* Web Reasoning and Rule Systems, pages 133–148, 2010. (Cited on page 37.)

[Polleres 2007] A. Polleres. *From SPARQL to rules (and back).* In WWW '07, pages 787–796, 2007. (Cited on pages 4, 34, 35 and 36.)

[Polleres 2012] A. Polleres. *How (well) do Datalog, SPARQL and RIF interplay?* Datalog in Academia and Industry, pages 27–30, 2012. (Cited on page 28.)

[Prud'hommeaux & Seaborne 2008] E. Prud'hommeaux and A. Seaborne. *SPARQL Query Language for RDF.* W3C Rec., 2008. (Cited on pages 23, 25 and 28.)

[Sagiv & Yannakakis 1980] Y. Sagiv and M. Yannakakis. *Equivalences among relational expressions with the union and difference operators.* Journal of the ACM (JACM), vol. 27, no. 4, pages 633–655, 1980. (Cited on page 70.)

[Sattler & Vardi 2001] U. Sattler and M. Y. Vardi. *The Hybrid μ-Calculus.* In IJCAR, pages 76–91, 2001. (Cited on pages 34, 56 and 77.)

[Schmidt *et al.* 2009] M. Schmidt, T. Hornung, G. Lausen and C. Pinkel. *SP^2Bench: A SPARQL Performance Benchmark.* In ICDE'09, pages 222–233. Ieee, 2009. (Cited on page 39.)

[Schmidt *et al.* 2010] M. Schmidt, M. Meier and G. Lausen. *Foundations of SPARQL Query Optimization.* In ICDT '10, pages 4–33, New York, NY, USA, 2010. ACM. (Cited on pages 34, 39 and 120.)

[Serfiotis *et al.* 2005] G. Serfiotis, I. Koffina, V. Christophides and V. Tannen. *Containment and Minimization of RDF/S Query Patterns.* In The Semantic Web - ISWC 2005, volume 3729 of *LNCS*, pages 607–623, 2005. (Cited on page 34.)

[Shmueli 1993] O. Shmueli. *Equivalence of datalog queries is undecidable.* The Journal of Logic Programming, vol. 15, no. 3, pages 231–241, 1993. (Cited on page 36.)

[Sirin & Parsia 2007] E. Sirin and B. Parsia. *Sparql-dl: Sparql query for owl-dl.* In 3rd OWL Experiences and Directions Workshop (OWLED-2007), volume 4, 2007. (Cited on page 82.)

[Stocker *et al.* 2008] M. Stocker, A. Seaborne, A. Bernstein, C. Kiefer and D. Reynolds. *SPARQL Basic Graph Pattern Optimization Using Selectivity Estimation.* In Proceeding of the 17th international conference on World Wide Web, WWW '08, pages 595–604, New York, NY, USA, 2008. ACM. (Cited on pages 34 and 39.)

[Tanabe *et al.* 2005] Y. Tanabe, K. Takahashi, M. Yamamoto, A. Tozawa and M. Hagiya. *A Decision Procedure for the Alternation-Free Two-Way Modal μ-calculus.* In TABLEAUX, pages 277–291, 2005. (Cited on pages 41, 100, 102, 103, 104 and 110.)

[Tanabe *et al.* 2008] Y. Tanabe, K. Takahashi and M. Hagiya. *A Decision Procedure for Alternation-Free Modal μ-calculi.* In Advances in Modal Logic, pages 341–362, 2008. (Cited on pages 33, 41, 56 and 88.)

[Ter Horst 2005] 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, vol. 3, no. 2-3, pages 79–115, 2005. (Cited on pages 13, 14 and 76.)

[Tobies 2001] S. Tobies. *Complexity results and practical algorithms for logics in knowledge representation.* PhD thesis, RWTH Aachen, 2001. thesis. (Cited on page 18.)

[Trakhtenbrot 1950] B.A. Trakhtenbrot. *Impossibility of an algorithm for the decision problem in finite classes.* Doklady Akademii Nauk SSSR, vol. 70, pages 569–572, 1950. (Cited on page 35.)

[Ullman 1997] J. Ullman. *Information integration using logical views.* Database Theory – ICDT'97, pages 19–40, 1997. (Cited on pages 36 and 115.)

[van der Meyden 1992] R. van der Meyden. *The Complexity of Querying Indefinite Data about Linearly Ordered Domains.* In PODS, pages 331–345, 1992. (Cited on page 70.)

[Wei & Lausen 2002] F. Wei and G. Lausen. *Containment of conjunctive queries with safe negation.* Database Theory–ICDT 2003, pages 346–360, 2002. (Cited on page 116.)

[Yannakakis 1981] M. Yannakakis. *Algorithms for acyclic database schemes.* In VLDB'81, volume 7, pages 82–94, 1981. (Cited on pages 91 and 92.)

Appendix A

Benchmark

In the following, we present the benchmark queries. The benchmark is designed for tree-structured and cyclic SPARQL queries. The tree-structured queries in each category of the test suites: CQNoProj, UCQProj, and UCQrdfs are displayed in Section A.1. A single test suite for cyclic SPARQL queries containing some corner cases is shown in A.2. Alternatively, the benchmark is also available online at `http://sparql-qc-bench.inrialpes.fr`. The benchmark queries use the namespace prefixes shown below:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX : <http://www.example.org/>
```

A.1 Tree-structured SPARQL Queries

A.1.1 CQNoProj Test Suite

Q1a

```
SELECT * WHERE {
  ?x :takesCourse "Course10" .
  ?x :takesCourse "Course20" .
}
```

Q1b

```
SELECT * WHERE {
  ?x :takesCourse "Course10" .
}
```

Q2a

```
SELECT * WHERE {
      ?x a :Student .
      ?x :registeredAt ?y .
      ?y a :University .
      ?x :placeOfBirth ?z .
      ?z a :City .
      ?y :locatedAt ?z .
}
```

Q2b

```
SELECT * WHERE {
      ?x a :Student .
      ?x :registeredAt ?y .
      ?x :placeOfBirth ?z .
      ?y a :University .
      ?y :locatedAt ?z .
      ?z a :City .
}
```

Q3a

```
SELECT * WHERE {
      ?x a :Professor .
      ?x :graduatedFrom ?y .
      ?x :memeberOf ?y
}
```

Q3b

```
SELECT * WHERE {
      ?x a :Professor .
      ?x :memeberOf ?y
}
```

Q4a

```
SELECT * WHERE {
        ?x :takesCourse ?c1 .
        ?c1 :shortName "Cs200" .
        ?x :takesCourse ?c2 .
        ?c2 :shortName "Cs301" .
        ?x :takesCourse ?c3 .
        ?x :shortName "Cs401" .
}
```

Q4b

```
SELECT * WHERE {
        ?x :takesCourse ?c1 .
        ?c1 :shortName "Cs200" .
        ?x :takesCourse ?c2 .
        ?c2 :shortName "Cs301" .
}
```

Q4c

```
SELECT * WHERE {
        ?x :takesCourse ?c1 .
        ?c1 :shortName "Cs200" .
        ?x :takesCourse ?c2 .
        ?c2 :shortName "Cs301" .
        ?x :takesCourse _:b .
        _:b :shortName "Cs401" .
}
```

Q5a

```
SELECT * WHERE {
        ?x  ?z ?y
}
```

Q5b

```
SELECT * WHERE {
        ?x :name ?y .
}
```

Q6a

```
SELECT * WHERE {
        ?x a :Department .
        ?y a :University .
        ?x ?z ?y .
}
```

Q6b

```
SELECT * WHERE {
        ?x a :Faculty .
        ?y a :University .
        ?x ?z ?y .
}
```

Q6c

```
SELECT * WHERE {
        ?x ?z ?y .
}
```

Q7a

```
SELECT * WHERE
{
  ?x a :Student .
  ?x :name ?y .
  ?x :nickName ?z .
  ?x :telephone ?tel .
  ?x :ssn ?ssn .
  ?x :age ?age .
  ?x :sex ?sex .
  ?x :emailAddress ?email .
  ?x :memberOf ?dept .
  ?x :takesCourse ?course .
}
```

Q7b

```
SELECT * WHERE
{
  ?x a :Student .
  ?x :name ?y .
  ?x :nickName ?z .
  ?x :telephone ?tel .
  ?x :ssn ?ssn .
  ?x :age ?age .
  ?x :sex ?sex .
  ?x :emailAddress ?email .
  ?x :memberOf ?dept .
  ?x :takesCourse ?course .
  ?x :masterDegreeFrom :OklahomaUniversity .
}
```

Q8a

```
SELECT * WHERE {
        ?x :subOrganizationOf ?y .
        ?y :subOrganizationOf ?z .
        ?z :subOrganizationOf ?r .
        ?r :subOrganizationOf :Unibz .
}
```

Q8b

```
SELECT * WHERE {
        ?x :subOrganizationOf ?y .
        ?y :subOrganizationOf ?z .
        ?z :subOrganizationOf ?r .
}
```

Q9a

```
SELECT * WHERE {
        ?x a :GraduateStudent .
        ?y a :Department .
        ?x :memberOf ?y .
        ?y :subOrganizationOf :University0 .
        ?x :email ?z .
}
```

Q9b

```
SELECT * WHERE {
        ?x a :GraduateStudent .
        ?y a :Department .
        ?x :memberOf ?y .
        ?y :subOrganizationOf :University1 .
        ?x :email ?z .
}
```

A.1.2 UCQProj Test Suite

Q1a

```
SELECT ?x WHERE {
   ?x :takesCourse "Course10" .
   ?x :takesCourse "Course20" .
}
```

Q1b

```
SELECT ?x WHERE {
   ?x :takesCourse "Course10" .
}
```

Q2a

```
SELECT ?x ?y ?z WHERE {
        ?x a :Student .
        ?x :registeredAt ?y .
        ?y a :University .
        ?x :placeOfBirth ?z .
        ?z a :City .
        ?y :locatedAt ?z .
}
```

Q2b

```
SELECT ?x ?y ?z WHERE {
        ?x a :Student .
        ?x :registeredAt ?y .
        ?x :placeOfBirth ?z .
        ?y a :University .
        ?y :locatedAt ?z .
        ?z a :City .
}
```

Q3a

```
SELECT ?x ?y WHERE {
        ?x a :Professor .
        ?x :graduatedFrom ?y .
        ?x :memeberOf ?y
}
```

Q3b

```
SELECT ?x ?y WHERE {
        ?x a :Professor .
        ?x :memeberOf ?y
}
```

Q4a

```
SELECT ?x  WHERE {
        ?x :takesCourse ?c1 .
        ?c1 :shortName "Cs200" .
        ?x :takesCourse ?c2 .
        ?c2 :shortName "Cs301" .
        ?x :takesCourse ?c3 .
        ?x :shortName "Cs401" .
}
```

Q4b

```
SELECT ?x WHERE {
        ?x :takesCourse ?c1 .
        ?c1 :shortName "Cs200" .
        ?x :takesCourse ?c2 .
        ?c2 :shortName "Cs301" .
}
```

Q4c

```
SELECT ?x   WHERE {
        ?x :takesCourse ?c1 .
        ?c1 :shortName "Cs200" .
        ?x :takesCourse ?c2 .
        ?c2 :shortName "Cs301" .
        ?x :takesCourse _:b .
        _:b :shortName "Cs401" .
}
```

Q5a

```
SELECT ?x ?y  WHERE {
        ?x ?z ?y
}
```

Q5b

```
SELECT ?x ?y  WHERE {
        ?x :name ?y .
}
```

Q6a

```
SELECT ?z WHERE {
        ?x a :Department .
        ?y a :University .
        ?x ?z ?y .
}
```

Q6b

```
SELECT  ?z WHERE {
        ?x a :Faculty .
        ?y a :University .
        ?x ?z ?y .
}
```

Q6c

```
SELECT  ?z WHERE {
        ?x ?z ?y .
}
```

Q7a

```
SELECT ?x ?y ?z ?t ?s ?ag ?sx ?email
 ?dept ?course WHERE
{
  ?x a :Student .
  ?x :name ?y .
  ?x :nickName ?z .
  ?x :telephone ?t .
  ?x :ssn ?s .
  ?x :age ?ag .
  ?x :sex ?sx .
  ?x :emailAddress ?email .
  ?x :memberOf ?dept .
  ?x :takesCourse ?course .
}
```

Q7b

```
SELECT ?x ?y ?z ?t ?s ?ag ?sx ?email
 ?dept ?course  WHERE
{
  ?x a :Student .
  ?x :name ?y .
  ?x :nickName ?z .
  ?x :telephone ?t .
  ?x :ssn ?s .
  ?x :age ?ag .
  ?x :sex ?sx .
  ?x :emailAddress ?email .
  ?x :memberOf ?dept .
  ?x :takesCourse ?course .
  ?x :masterDegreeFrom ?master .
}
```

Q7c

```
SELECT ?x ?y ?z ?t ?s ?ag ?sx ?email
 ?dept ?course WHERE
{
  ?x a :Student .
  ?x :name ?y .
  ?x :nickName ?z .
  ?x :telephone ?t .
  ?x :sex ?sx .
  ?x :emailAddress ?email .
  ?x :memberOf ?dept .
  ?x :takesCourse ?course .
}
```

Q8a

```
SELECT ?x ?y ?z ?r  WHERE {
        ?x :subOrganizationOf ?y .
        ?y :subOrganizationOf ?z .
        ?z :subOrganizationOf ?r .
        ?r :subOrganizationOf :Unibz .
}
```

Q8b

```
SELECT ?x ?y ?z ?r WHERE {
        ?x :subOrganizationOf ?y .
        ?y :subOrganizationOf ?z .
        ?z :subOrganizationOf ?r .
}
```

Q9a

```
SELECT ?x ?z WHERE {
        ?x a :GraduateStudent .
        ?y a :Department .
        ?x :memberOf ?y .
        ?y :subOrganizationOf :UniversityO .
        ?x :email ?z .
}
```

Q9b

```
SELECT ?x  ?z WHERE {
        ?x a :GraduateStudent .
        ?y a :Department .
        ?x :memberOf ?y .
        ?y :subOrganizationOf :University1 .
        ?x :email ?z .
}
```

Q9c

```
SELECT ?x  ?z WHERE {
        ?x a :GraduateStudent .
        ?y a :Department .
        ?x :memberOf ?y .
        ?y :subOrganizationOf ?u .
        ?x :email ?z .
}
```

Q10a

```
SELECT * WHERE
{
  ?x a :Student .
  {?x :name ?y }
    UNION
  {?x :nickName ?z }
    UNION
  {?x :telephone ?tel }
    UNION
  {?x :ssn ?ssn }
    UNION
  {?x :age ?age }
    UNION
  {?x :sex ?sex }
    UNION
  {?x :emailAddress ?email }
    UNION
  {?x :memberOf ?dept }
  ?x :takesCourse ?course .
}
```

Q10b

```
SELECT * WHERE {
  ?x a :Student .
  {?x :name ?y }
    UNION
  {
    ?x :nickName ?z .
    ?x :telephone ?tel .
    ?x :ssn ?ssn .
    ?x :age ?age .
    ?x :sex ?sex .
    ?x :emailAddress ?email .
    ?x :memberOf ?dept .
    ?x :masterDegreeFrom ?master
  }
  ?x :takesCourse ?course .
}
```

q11a

```
SELECT ?name ?email
WHERE
{
  ?x a :Student .
  ?x :name ?name .
  ?x :email ?email .
  {
    ?x :takesCourse ?c .
    ?c :shortName "Course10" .
  }
  UNION
  { ?x :takesCourse ?c .
    ?c :shortName "Course20" .
  }
  UNION
  { ?x :takesCourse ?c .
    ?c :shortName "Course30" .
  }
}
```

Q11b

```
SELECT ?name ?email
WHERE {
  ?x a :Student .
  ?x :name ?name .
  ?x :email ?email .
  ?x :takesCourse ?c1 .
  ?c1 :shortName "Course10" .
  ?x :takesCourse ?c2 .
  ?c2 :shortName "Course20" .
  ?x :takesCourse ?c .
  ?c :shortName "Course30" .
}
```

Q12a

```
SELECT ?x ?y WHERE
{
  {
    { ?x a :UndergradStudent .
      ?x :takesCourse ?y }
      UNION
    { ?x a :GraduateStudent .
      ?x :takesCourse ?y
    }
  }
  ?y a :CsCourse .
}
```

Q12b

```
SELECT ?x ?y WHERE
{
  { ?x a :UndergradStudent .
    ?x :takesCourse ?y .
    ?y a :CsCourse
  }
    UNION
  {
     ?x a :GraduateStudent .
     ?x :takesCourse ?y
  }
}
```

A.1.3 UCQrdfs Test Suite

C1

```
{  (:GraduateStudent,rdfs:subClassOf,:Student),
    (:UndergradStudent,rdfs:subClassOf,:Student)}
```

C2

```
{ (:headOf,rdfs:domain,:Professor),
   (:headOf,rdfs:range,:Department)}
```

C3

```
{
  (:maleHeadOf,rdfs:subPropertyOf,:headOf),
  (:femaleHeadOf,rdfs:subPropertyOf,:headOf ),
  (:FullProfessor,rdfs:subClassOf,:Professor),
  (:headOf,rdfs:domain,:FullProfessor)
}
```

C4

```
{ (:CsCourse,rdfs:subClassOf,:Course) }
```

Q10a

```
SELECT ?x ?y WHERE
{
  ?x a :Professor .
  ?x :worksFor ?y .
  ?y a :Department .
}
```

Q10b

```
SELECT ?x  WHERE
{
  ?x :headOf ?y .
}
```

Q10c

```
SELECT ?x ?y WHERE
{
  ?x a :Chair .
  ?x :worksFor ?y .
}
```

Q10d

```
SELECT ?x WHERE
{
  ?x a :Professor .
}
```

Q10e

```
SELECT ?x   WHERE
{
  ?x :headOf ?y .
  ?x :worksFor ?y .
}
```

Q11a

```
SELECT ?x WHERE {
  ?x a :Professor .
}
```

Q11b

```
SELECT ?x WHERE
{
  ?x :maleHeadOf ?y .
}
```

Q11c

```
SELECT ?x WHERE {
  ?x :femaleHeadOf ?y .
}
```

Q11d

```
SELECT ?x WHERE {
  ?x :headOf ?y.
}
```

Q11e

```
SELECT ?x WHERE {
       { ?x :maleHeadOf ?y .}
             UNION
       { ?x :femaleHeadOf ?y .}
}
```

Q13a

```
SELECT ?x ?y WHERE
{
  {
    { ?x a :UndergradStudent .
      ?x :takesCourse ?y }
      UNION
    { ?x a :GraduateStudent .
      ?x :takesCourse ?y
    }
  }
  ?y a :CsCourse .
}
```

Q13b
```
SELECT ?x ?y WHERE
{
  { ?x a :UndergradStudent .
    ?x :takesCourse ?y .
    ?y a :CsCourse
  }
    UNION
  {
     ?x a :GraduateStudent .
     ?x :takesCourse ?y
  }
}
```

Q13c
```
SELECT ?x ?y WHERE
{
  {
    {
      ?x a :UndergradStudent .
      ?x :takesCourse ?y
    }
     UNION
    {
      ?x a :GraduateStudent .
      ?x :takesCourse ?y
    }
   }
   ?y a :Course .
}
```

Q9a
```
SELECT ?x WHERE {
  ?x a :Student .
}
```

Q9b
```
SELECT ?x WHERE {
  ?x a :GraduateStudent .
}
```

Q9c
```
SELECT ?x WHERE {
  {?x a :GraduateStudent . }
    UNION
  {?x a :UndergradStudent . }
}
```

Q9d
```
SELECT ?x WHERE {
  ?x a :Student .
  ?x :takesCourse ?y .
  ?x :telephone ?tel .
  ?x :sex "male" .
  ?y :courseName "Course10" .
}
```

Q9e
```
SELECT ?x WHERE {
  ?x a :GraduateStudent .
  ?x :takesCourse ?y .
  ?x :telephone ?tel .
  ?x :sex "male" .
  ?y :courseName "Course10" .
}
```

A.2 Cyclic Queries

Q1a
```
SELECT * WHERE {
   ?x ?r ?y .
   ?y ?r ?z .
   ?z ?r ?x .
   ?x :p :a .
   ?z :p :a .
   ?y :p :a .
}
```

Q1b
```
SELECT * WHERE {
   ?x ?r ?y .
   ?y ?r ?z .
   ?z ?r ?x .
   ?x :p :a .
}
```

Q2a
```
SELECT * WHERE {
   ?x ?r ?x .
}
```

Q2b

```
SELECT * WHERE {
  ?x ?y ?z .
  ?z ?r ?x .
}
```

Q2c

```
SELECT * WHERE {
  ?x ?r ?y .
  ?y ?r ?x .
}
```

Q3a

```
SELECT *  WHERE {
  ?x ?r ?y .
  ?x ?r ?x .
}
```

Q3b

```
SELECT * WHERE {
  ?x ?r ?y .
}
```

Q4a

```
SELECT * WHERE {
  ?x :r ?y .
  ?y :r ?z .
  ?z :r ?x .
  :a :b :c .
}
```

Q4b

```
SELECT * WHERE {
  ?x :r ?y .
  ?y :r ?z .
  ?z :r ?x .
  ?x :b :c .
}
```

Q5a

```
SELECT * WHERE {
  ?x :blue ?y .
  ?y :red ?z .
  ?z :red ?r .
  ?r :red ?y .
}
```

Q5b

```
SELECT * WHERE {
  ?x :blue ?y .
  ?y :red ?z .
  ?z :red ?y .
  ?z :red ?z .
}
```

Q5c

```
SELECT * WHERE {
  ?x :blue ?y .
  ?y :red ?z .
  ?z :red ?y .
}
```

Q6a

```
SELECT * WHERE {
  ?x :r ?y .
  ?y :r ?z .
  ?z :r ?x .
  ?x :r ?w .
}
```

Q6b

```
SELECT * WHERE {
  ?x :r ?y .
  ?y :r ?z .
  ?z :r ?x .
}
```

Q7a

```
SELECT * WHERE {
  ?x ?r ?y .
  ?y ?r ?z .
  ?z ?r ?x .
  ?x ?r ?w .
}
```

Q7b

```
SELECT * WHERE {
  ?x ?r ?y .
  ?y ?r ?z .
  ?z ?r ?x .
}
```

Appendix B

Résumé étendu

B.1 Motivation et objectifs

Le Web sémantique est une extension du World Wide Web (WWW) permettant de créer et de partager du contenu compréhensible par les machines. Plus d'une décennie de recherches a permis d'élaborer différents langages cadres pour le web sémantique comme langages définis et recommandés par le World Wide Web Consortium (W3C) : « Resource Description Framework » (RDF) et « Ontology Web Language » (OWL). RDF permet d'exprimer des informations structurées sur le Web sous forme de graphes. Un document RDF est un ensemble de triplets $(sujet, predicat, objet)$ qui peut être représenté par un graphe orienté étiqueté (d'où le nom de graphe RDF). L'interrogation des données du web sémantique, exprimées en RDF, se fait principalement avec le langage de requêtes SPARQL. Il a suscité différentes recherche, en particulier pour l'extension du langage et pour l'optimisation de requêtes. L'interrogation de graphes RDF avec SPARQL est réalisée par l'appariement de motifs de graphes, i.e., motifs de triplet reliés au moyen de jointures exprimées en utilisant plusieurs occurrences de la même variable. Comme les requêtes dans le web sémantique sont évaluées sur de gigantesques graphes RDF, des optimisations sont nécessaires pour repérer les requêtes minimales permettant d'alléger le coût de calcul lié à l'évaluation de la requête. L'inclusion de requête joue un rôle essentiel dans l'optimisation. Une requête est incluse dans une autre si, pour tout graphe RDF, le résultat de la première requête est inclus dans le résultat de la seconde requête. L'inclusion de requête a été un point central de discussion pour de nombreuses applications des bases des données et de connaissances, comme les entrepôts de données, l'intégration de données et l'optimisation de requêtes. Compte tenu de ces avantages, il est intéressant d'étudier l'inclusion, l'équivalence et la satisfiabilité de requêtes SPARQL. Ces problèmes sont collectivement appelés analyse statique de requêtes SPARQL.

Comme les problèmes d'équivalence et de satisfiabilité peuvent être réduits à l'inclusion de requêtes, nous nous concentrons principalement sur le problème de l'inclusion. Dans les bases de données relationnelles, l'inclusion d'union de requêtes conjonctives a été étudiée à l'aide de l'appariement d'inclusion également connue sous le nom d'homomorphisme de graphe et bases de données canoniques. On sait que, pour les requêtes conjonctives, l'évaluation et l'inclusion de requêtes sont des problèmes équivalents parce qu'il a été montré que l'inclusion peut être réduite à l'évaluation de requêtes [Chandra & Merlin 1977]. Malheureusement, l'application de ces techniques sur un langage de requête muni d'ontologies et d'expressions régulières n'est pas complètement possible. C'est pourquoi la théorie des automates est souvent utilisée pour

résoudre les problèmes d'inclusion dans le cas de langages de requêtes de données semi-structurées (appelées aussi requêtes de chemin régulier) [Calvanese *et al.* 2000]. Ce problème de l'inclusion a également été traité par des approches basées sur la réduction au test de satisfiabilité. Cela consiste à exprimer les requêtes dans une logique particulière prenant en charge les fonctionnalités du langage de requêtes considéré, puis à réduire le problème global à la satisfiabilité. Il existe un certain nombre de travaux basés sur cette technique [Calvanese *et al.* 2000, Genevès *et al.* 2007, Calvanese *et al.* 2008]. Cette approche a également inspiré cette thèse.

Plus précisément, [Genevès *et al.* 2007] a développé une logique d'arbre à partir d'un fragment du μ-calcul et l'a appliqué pour encoder les requêtes XPath et effectuer des tâches d'analyse statique. L'étude a été étendue à plusieurs autres langages à savoir XQuery, CSS et JSON. Cette thèse étudie l'inclusion de requêtes SPARQL selon une logique expressive appelée le μ-calcul. Compte tenu de la définition de SPARQL comme une projection d'un graphe dans un autre il peut être encodé dans une logique l'interprétation dont est sur un graphe. Une des logiques qui a ces caractéristiques est le μ-calcul. C'est une logique modale avec une expressivité forte et une complexité de calcul intéressante. Elle permet d'utiliser des opérateurs de point fixe pour effectuer la navigation dans un graphe d'une manière globale. Par ailleurs, les procédures de décision dans cette logique, qui sont théoriquement de complexité exponentielle, peuvent être mises en œuvre efficacement dans la pratique. On réduit donc le problème d'inclusion de requêtes SPARQL en un problème de la validité en μ-calcul. Pour ce faire, les graphes RDF sont codés comme des systèmes de transitions, qui conservent les caractéristiques de graphes RDF, et les requêtes et les axiomes du schéma sont codés comme des formules de μ-calcul. Ainsi, l'inclusion de requête peut être réduite en un test de validité dans cette logique. Cela nous permet de régler le problème de l'analyse statique des requêtes SPARQL, et notamment la satisfiabilité, l'inclusion et l'équivalence de requêtes.

Dans la section suivante, nous présentons les principales contributions de cette thèse.

B.1.1 Résumé des contributions

Après avoir présenté le contexte global et les idées de base derrière l'inclusion et l'équivalence des requêtes du web sémantique, nous allons maintenant résumer le travail présenté dans cette thèse et mettre en évidence les contributions importantes. Comme cela devrait être clair à partir de la discussion précédente, nous mettrons l'accent sur l'analyse statique de l'union de requêtes SPARQL (et PSPARQL) conjonctives. Les contributions de cette thèse sont au nombre six:

- Nous fournissons les procédures de codage pour déterminer l'inclusion de l'union de requêtes SPARQL conjonctives (respectivement Path SPARQL ou PSPARQL).

- Nous proposons une technique pour déterminer l'inclusion de l'union de requêtes SPARQL conjonctives par rappart à un schéma de la logique de description

$\mathcal{ALCH}$. Pour ce faire, nous encodons les requêtes et les axiomes de schéma sous forme de formules du μ-calcul, réduisant l'inclusion au test d'unsatisfiabilité.

- La troisième contribution de cette thèse est l'inclusion de requête PSPARQL. Pour résoudre ce problème, nous encodons les requêtes PSPARQL en formula du μ-calcul, puis nous réduisons le test d'inclusion au problème de la validité dans le μ-calcul. La complexité de la détermination d'inclusion des requêtes PSPARQL (également sous les axiomes de $\mathcal{ALCH}$) est en double exponentielle. Cette complexité est une borne supérieure pour le problème.

- Nous fournissons trois approches différentes pour vérifier l'inclusion de requête SPARQL sous le régime d'implication de RDFS ('RDFS entailment regime'). Nous prouvons que ce problème peut être résolu en temps exponentiel. En outre, nous appuyons nos résultats théoriques avec une mise en œuvre. Nous étudions également le problème d'inclusion sous le régime d'implication de la semantique directe de OWL. Pour ce faire, nous identifions des fragments décidable de languages d'ontologie et de la requête.

- La cinquième contribution: (i) montre expérimentalement que 87% des requêtes SPARQL du monde réel sont arborescente, (ii) réduit le problème d'inclusion arborescente à la logique modale K_n, et enfin (iii) compare expérimentalement les performances des solveurs d'inclusion à l'aide de la méthode proposée.

- Nous présentons un premier banc de test (benchmark) pour analyser statiquement les requêtes du web sémantique. Nous avons mis au point des suites de tests (avec et sans les axiomes de l'ontologie) qui testent et comparent les performances et l'exactitude des solveurs d'inclusion. En outre, le benchmark est testée sur les outils actuellement disponibles montrant que l'approche prise et praticable

B.2 Organisation de la thèse

Chapitre 2: Preliminaires Dans ce chapitre, nous présentons les préliminaires qui sont utilisés dans le développement de cette thèse. Nous présentons les bases du web sémantique, suivies d'un bref aperçu des logiques modales, et enfin nous terminons ce chapitre par une vaste revue des travaux connexes.

Chapitre 3: Inclusion de Requête SPARQL sous Schéma Ce chapitre propose une procédure pour traduire les graphes RDF en systèmes de transition. Pour ce faire, les graphes RDF deviennent des graphes bipartis avec deux ensembles de nœuds: les nœuds triples et les nœuds sujet, prédicat, objet où la navigation peut être fait en utilisant des programmes de transition. La tâche suivante pour déterminer l'inclusion nécessite le codage des axiomes et des requêtes sous forme de formules du μ-calcul, puis à réduire le test d'inclusion au test de validité de la logique. Nous avons prouvé que cette réduction est correcte et complete et nécessite une quantité de temps double exponentielle.

Chapitre 4: Inclusion de Requête SPARQL à Chemin Dans ce chapitre, nous prolongeons les procédures du Chapitre 3 pour étudier le problème d'inclusion de requêtes PSPARQL. Nous divisons cette tâche en trois: (i) nous étudions le problème pour les requêtes PSPARQL (équivalents aux RPQ conjonctives), (ii) ensuite nous ajoutons l'inverse aux chemins d'accès de PSPARQL (équivalent aux RPQ (Regular Path Query) conjonctives bidirectionnelles) et étudions l'inclusion pour ce fragment, enfin (iii) nous nous attaquons au problème sous les axiomes de schéma logique description.

Chapitre 5:Inclusion de Requête SPARQL sous Régime d'Implication Trois approches pour déterminer l'inclusion des requêtes SPARQL sous régime d'implication RDFS sont discutées dans ce chapitre en empruntant certaines procédures d'encodage des Chapitres 3 et 4. De plus, nous montrons comment ces approches peuvent être étendue pour le régime d'implication OWL sémantique directe.

Chapitre 6: Inclusion de Requête SPARQL Arborescente Pour les requêtes arborescentes, qui constituent une grande partie des requêtes SPARQL du monde réel, nous considérons une logique moins expressive que le μ-calcul pour étudier l'analyse statique comme on le verra dans ce chapitre. Nous fournissons des résultats expérimentaux pour les essais d'inclusion avec un certain nombre des requêtes de référence (benchmark).

Chapitre 7: Un Banc de Test pour L'inclusion, L'équivalence et la Satisfiabilité de Requêtes Web Sémantique Dans ce chapitre, nous présentons un banc d'essais de tests d'inclusion pour les requêtes SPARQL. Nous proposons plusieurs suites de tests qui peuvent être utilisés pour tester les solveurs d'inclusion. En conséquence, nous avons effectué des expériences pour tester et comparer des solveurs d'inclusion de l'état de l'art.

Chapitre 8: Extensions Dans ce chapitre, nous présentons brièvement l'étude d'inclusion de négations requêtes SPARQL et la sémantique de multi- ensembles.

Chapitre 9: Conclusion La contribution de cette thèse comprend l'identification de divers fragments de requêtes du web sémantique et logiques de description de schéma dont l'inclusion est décidable. En outre, il fournit des procédures éprouvées théoriquement et expérimentalement pour vérifier l'inclusion de ces fragments décidable. Au-delà, la contribution comprend aussi un banc de test pour les solveurs d'inclusion. Cette référence est utilisée pour tester et comparer les solveurs d'inclusion de l'état de l'art. Dans l'ensemble, nous résumons les principaux résultats obtenus dans cette thèse dans la Table B.1. Les résultats, présentés dans la colonne de complexité, qui sont marqués d'une * sont nouveaux à ce travail, tandis que les autres résultats proviennent de l'étude de l'algèbre relationnelle dans les bases de données et l'inclusion de modèle graphique OPTIONAL étudiée dans [Letelier *et al.* 2012].

	Motif de graphes	Langage de schéma	Complexité d'inclusion
SPARQL	AND	-	NP [Chandra & Merlin 1977]
	AND-UNION		NP [Chandra & Merlin 1977]
	OPT		Π_2^P [Letelier *et al.* 2012]
	AND-OPT		Π_2^P [Letelier *et al.* 2012]
	AND-UNION-OPT		undecidable
	MINUS		2ExpTime*
SPARQL	AND	$\mathcal{ALCH}$	2ExpTime*
	AND-UNION	$\mathcal{ALCH}$	2ExpTime*
	AND-UNION	RDFS entailment	ExpTime*
	AND-UNION	OWL entailment	ExpTime*
	OPT	-	-
	AND-OPT	-	-
	MINUS	-	-
PSPARQL	AND	-	2ExpTime*
	AND-UNION		2ExpTime*
	OPT		-
	AND-OPT		-
	MINUS		-
PSPARQL	AND	$\mathcal{ALCH}$	2ExpTime*
	AND-UNION	$\mathcal{ALCH}$	2ExpTime*
	OPT	-	-
	AND-OPT	-	-
	MINUS	-	-

Table B.1: Sommaire des résultats sur l'inclusion des requêtes SPARQL et PSPARQL.

Perspectives Résoudre le problème d'inclusion pour chaque fragment possible de SPARQL (respectivement PSPARQL) et langage de schéma ($\mathcal{SHIQ}$, DL-Lite, $\mathcal{SHOI}$, $\mathcal{SHOIN}(D)$, $\mathcal{SROI}$) est un prolongement possible de cette thèse. Ce travail pose les fondements théoriques et expérimentales, fournit les méthodes simplement extensibles et applicables, par exemple, la référence en matière d'inclusion est conçu en ce qui concerne l'état actuel de l'art, donc il a besoin d'évoluer avec les solveurs d'inclusion.

Dans la Table B.1 sont montrés certains problèmes ouverts, ceux marquées avec un symbole -.