Static Analysis of Semantic Web Queries with ShEx Schema Constraints

Abdullah Abbas

▶ **To cite this version:**

Abdullah Abbas. Static Analysis of Semantic Web Queries with ShEx Schema Constraints. Web. Université Grenoble - Alpes, 2017. English. ⟨NNT : ⟩. ⟨tel-01673074⟩

HAL Id: tel-01673074

https://inria.hal.science/tel-01673074v1

Submitted on 28 Dec 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

HAL Authorization

Communauté
UNIVERSITÉ Grenoble Alpes

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE GRENOBLE

Spécialité : **Informatique**

Arrêté ministériel : 25 mai 2016

Présentée par

Abdullah Abbas

Thèse dirigée par **Nabil Layaïda**
et codirigée par **Pierre Genevès** et **Cécile Roisin**

préparée au sein de **L'Institut National de Recherche en Informatique et en Automatique, Laboratoire d'Informatique de Grenoble**
dans **l'Ecole Doctorale Mathématiques, Sciences et Technologies de l'Information, Informatique**

Static Analysis of Semantic Web Queries with ShEx Schema Constraints

Thèse soutenue publiquement le **6 novembre 2017,**
devant le jury composé de :

Monsieur Jérôme Euzenat
Directeur de Recherche, Inria Grenoble Rhône-Alpes, Président
Monsieur Mohand-Saïd Hacid
Professeur, Université Claude Bernard Lyon 1 - LIRIS, Rapporteur
Monsieur Farouk Toumani
Professeur, Université Blaise Pascal - LIMOS, Rapporteur
Monsieur Antoine Zimmermann
Maître de conférences, EMSE - Institut Henri Fayol - ISCOD, Examinateur
Monsieur Pierre Genevès
Chercheur (HDR), CNRS, Co-Directeur de thèse
Madame Cécile Roisin
Professeure, Université Grenoble Alpes, Co-Directrice de thèse
Monsieur Nabil Layaïda
Directeur de Recherche, Inria Grenoble Rhône-Alpes, Invité

Abstract:

Data structured in the Resource Description Framework (RDF) are increasingly available in large volumes. This leads to a major need and research interest in novel methods for query analysis and compilation for making the most of RDF data extraction. SPARQL is the widely used and well supported standard query language for RDF data. In parallel to query language evolutions, schema languages for expressing constraints on RDF datasets also evolve. Shape Expressions (ShEx) are increasingly used to validate RDF data, and to communicate expected graph patterns. Schemas in general are important for static analysis tasks such as query optimisation and containment. Our purpose is to investigate the means and methodologies for SPARQL query static analysis and optimisation in the presence of ShEx schema constraints.

Our contribution is mainly divided into two parts. In the first part we consider the problem of SPARQL query containment in the presence of ShEx constraints. We propose a sound and complete procedure for the problem of containment with ShEx, considering several SPARQL fragments. Particularly our procedure considers OPTIONAL query patterns, that turns out to be an important feature to be studied with schemas. We provide complexity bounds for the containment problem with respect to the language fragments considered. We also propose alternative method for SPARQL query containment with ShEx by reduction into First Order Logic satisfiability, which allows for considering SPARQL fragment extension in comparison to the first method. This is the first work addressing SPARQL query containment in the presence of ShEx constraints.

In the second part of our contribution we propose an analysis method to optimise the evaluation of conjunctive SPARQL queries, on RDF graphs, by taking advantage of ShEx constraints. The optimisation is based on computing and assigning ranks to query triple patterns, dictating their order of execution. The presence of intermediate joins between the query triple patterns is the reason why ordering is important in increasing efficiency. We define a set of well-formed ShEx schemas, that possess interesting characteristics for SPARQL query optimisation. We then develop our optimisation method by exploiting information extracted from a ShEx schema. We finally report on evaluation results performed showing the advantages of applying our optimisation on the top of an existing state-of-the-art query evaluation system.

Résumé:

La disponibilité de gros volumes de données structurées selon le modèle Resource Description Framework (RDF) est en constante augmentation. Cette situation implique un intérêt scientifique et un besoin important de rechercher de nouvelles méthodes d'analyse et de compilation de requêtes pour tirer le meilleur parti de l'extraction de données RDF. SPARQL est le plus utilisé et le mieux supporté des langages de requêtes sur des données RDF. En parallèle des langages de requêtes, les langages de définition de schéma d'expression de contraintes sur des jeux de données RDF ont également évolués. Les Shape Expressions (ShEx) sont de plus en plus utilisées pour valider des données RDF et pour indiquer les motifs de graphes attendus. Les schémas sont importants pour les tâches d'analyse statique telles que l'optimisation ou l'injection de requêtes. Notre intention est d'examiner les moyens et méthodologies d'analyse statique et d'optimisation de requêtes associés à des contraintes de schéma.

Notre contribution se divise en deux grandes parties. Dans la première, nous considérons le problème de l'injection de requêtes SPARQL en présence de contraintes ShEx. Nous proposons une procédure rigoureuse et complète pour le problème de l'injection de requêtes avec ShEx, en prenant en charge plusieurs fragments de SPARQL. Plus particulièrement, notre procédure gère les patterns de requêtes OPTIONAL, qui s'avèrent former un important fonctionnalité à étudier avec les schémas. Nous fournissons ensuite les limites de complexité de notre problème en considération des fragments gérés. Nous proposons également une méthode alternative pour l'injection de requêtes SPARQL avec ShEx. Celle-ci réduit le problème à une satisfiabilité de Logique de Premier Ordre, qui permet de considérer une extension du fragment SPARQL traité par la première méthode. Il s'agit de la première étude traitant l'injection de requêtes SPARQL en présence de contraintes ShEx.

Dans la seconde partie de nos contributions, nous proposons une méthode d'analyse pour optimiser l'évaluation de requêtes SPARQL groupées, sur des graphes RDF, en tirant avantage des contraintes ShEx. Notre optimisation s'appuie sur le calcul et l'assignation de rangs aux triple patterns d'une requête, permettant de déterminer leur ordre d'exécution. La présence de jointures intermédiaires entre ces patterns est la raison pour laquelle l'ordonnancement est important pour gagner en efficicacité. Nous définissons un ensemble de schémas ShEx bien-formulés, qui possède d'intéressantes caractéristiques pour l'optimisation de requêtes SPARQL. Nous développons ensuite notre méthode d'optimisation par l'exploitation d'informations extraites d'un schéma ShEx. Enfin, nous rendons compte des résultats des évaluations effectuées, montrant les avantages de l'application de notre optimisation face à l'état de l'art des systèmes d'évaluation de requêtes.

Acknowledgments

First of all, I would like to thank **l'Université Grenoble Alpes** and **l'Ecole Doctorale Mathématiques, Sciences et Technologies de l'Information, Informatique** that gave me the opportunity to be enrolled as a PhD student, and financed my stay in France for 3 years.

I want to express my gratitude to my supervisors Dr. Nabil Layaïda, Dr. Pierre Genevès, and Prof. Cécile Roisin for their support, constructive remarks, and continuous sharing of knowledge and expertise along this work. I would not forget to thank Prof. Jérôme Euzenat who was already available to help, and his supervision during my internship was a great launch and motivation for my research career.

I thank Inria Grenoble Rhône-Alpes for providing a very causal and comfortable working atmosphere.

I thank my colleagues that I have met during the 3 years, and of course I thank my family and friends for their great support all the time.

Finally, the warmest acknowledgment is for the French community which accepted my presence, and helped me at work, and outside work.

Contents

List of Figures xi

1 Introduction 1
1.1 Motivation and Objectives . . . 2
1.2 Summary of Contributions . . . 4
1.3 Thesis Outline . . . 4
1.3.1 Part I - State-of-the-Art . . . 4
1.3.2 Part II - Contribution I . . . 5
1.3.3 Part III - Contribution II . . . 5

I State-of-the-Art 7

2 Resource Description Framework (RDF) 9
2.1 Introduction . . . 10
2.2 RDF 1.1 Formal Definition . . . 11
2.2.1 Abstract Syntax . . . 11
2.3 Interpretation of RDF . . . 13
2.4 Datatypes for Literals . . . 14
2.5 RDF Serialization . . . 15
2.5.1 N-Triples, N-Quads, Turtle and TriG . . . 15
2.5.2 RDF/XML . . . 16
2.5.3 JSON-LD and RDFa . . . 17

3 SPARQL and Queries 19
3.1 Introduction . . . 20
3.2 Abstract Syntax . . . 20
3.3 SPARQL Query . . . 21
3.4 Well-Designed OPT Patterns. . . . 22
3.5 RPQs, NREs, GXPath, and Property Path Patterns . . . 23
3.6 SPARQL Semantics . . . 23
3.7 Preparing Required Fragments for the Rest of this Thesis . . . 25
3.7.1 Allowed SPARQL Filters . . . 25

4 Schema and Ontology Languages for RDF 27
4.1 Introduction . . . 28
4.2 Abstract Syntax . . . 30
4.3 Semantics . . . 31
4.4 Other Schema and Ontology Languages for RDF . . . 31
4.4.1 SHACL . . . 32
4.4.2 Ontology Languages . . . 33
4.5 Conclusion . . . 36

5 First Order Logic (FOL) 39
5.1 Introduction . . . 40
5.2 FOL Survey . . . 40
5.3 FOL^2 . . . 40
5.4 FOL Theorem Provers Implementations and Serialisations . . . 42
5.5 Conclusion . . . 42

6 Query Containment and Query Satisfiability 45
6.1 Introduction . . . 46
6.2 Containment of the SPARQL OPT Fragment . . . 47
6.3 Containment of SPARQL with RDFS Entailment . . . 48
6.4 Conclusion . . . 49

7 Query Optimization 51
7.1 Introduction . . . 52
7.2 Data Indexing and Query Planning (H_2RDF+) . . . 53
7.3 Typed Data . . . 55
7.4 Distributed RDF Store and Triple Pattern Ordering (SPARQLGX) . . . 56

II Contribution I - SPARQL Query Containment with ShEx Constraints 59

8 Reduction into ShEx Validation and SPARQL Containment 61
8.1 Introduction . . . 62
8.2 Containment Procedure Overview . . . 63
8.3 Query Transformation . . . 64
8.3.1 BGP Transformation . . . 64
8.3.2 AND-OPT Transformation . . . 65
8.3.3 AND-OPT-(UNION) Transformation . . . 67
8.4 Query Containment with ShEx . . . 67
8.5 Complexity . . . 69
8.5.1 SPARQL AND-(OPT)-(UNION) Fragments . . . 69
8.6 Implementation and Experimentation . . . 71
8.7 Conclusion . . . 72

9 Reduction into FOL Satisfiability 75
9.1 Introduction . . . 76
9.2 ShEx Document Encoding . . . 77
9.3 Query Encodings . . . 78
9.3.1 Predicates as Variables . . . 79
9.3.2 “Containing Query” Variables . . . 80
9.4 Implementation and Experimentation . . . 80
9.5 Extensions . . . 82
9.5.1 Property Path Patterns . . . 82
9.5.2 Filters . . . 83
9.5.3 Minus . . . 84
9.6 Conclusion . . . 85

III Contribution II - Optimising SPARQL Evaluation with ShEx 87

10 Selectivity Estimation for SPARQL Triple Patterns 89
10.1 Introduction . . . 90
10.2 Definitions . . . 91
10.2.1 Abstract Syntax of the Considered ShEx Fragment . . . 91
10.2.2 Preliminary Definitions . . . 92
10.3 Well-Formed Data-Schema Pairs . . . 92
10.3.1 Cardinality Constraints . . . 94
10.3.2 Shape Distinction . . . 94
10.3.3 Data Nodes Isolation . . . 95
10.4 Shape Relation Graph . . . 95
10.5 Ranking . . . 96
10.5.1 Hierarchical Relations between ShEx Shapes . . . 97
10.5.2 Predicate Distributions Among ShEx Shapes . . . 99
10.5.3 SPARQL Query Triple Rankings . . . 100
10.6 Evaluation . . . 100
10.6.1 Experiment 1: With Web Index Schema . . . 101
10.6.2 Experiment 2: With LDBC SNB Schema and gMark Queries . . . 103
10.7 Conclusion . . . 106

11 Conclusion and Perspectives 109
11.1 Summary . . . 110
11.2 Perspectives . . . 111
11.2.1 SPARQL Containment with the mixture of ShEx and OWL . . . 111
11.2.2 RDF Distributed Data ShEx Validation . . . 112
11.2.3 SPARQL Optimization by Transformations with ShEx . . . 112

Bibliography 115

A Appendix: Chapter 8 and 9 Experiments 121
A.1 Chapter 8 and 9, Experiments: Queries & Schemas . . . 122

B Appendix: Chapter 10 Experiments 125
B.1 Chapter 10, Experiment 1: Shape Relation Graph . . . 126
B.2 Chapter 10, Experiment 1: Queries . . . 127
B.3 Chapter 10, Experiment 2: Schema . . . 128
B.4 Chapter 10, Experiment 2: Shape Relation Graph . . . 129
B.5 Chapter 10, Experiment 2: Queries . . . 130

Appendices 121

List of Figures

1.1 Semantic Web Layer Cake . . . 3

2.1 RDF graph example . . . 10
2.2 The LOD cloud diagram . . . 12
2.3 RDF graph example . . . 16

3.1 SPARQL query matching . . . 21
3.2 Pattern tree example . . . 23

4.1 ShEx validation examples . . . 30
4.2 A graphical representation of the RDF graph for the example where green dashed lines indicate RDF-entailed triples and red dashed lines indicate triples that are also RDFS-entailed. . . . 36

6.1 An example benefit for query containment . . . 46

7.1 SPARQL query evaluation complexity summary . . . 53
7.2 Grouped intermediate results [Papailiou et al., 2013] . . . 54
7.3 Join on grouped intermediate results [Papailiou et al., 2013] . . . 55
7.4 Query response times with WatDiv1k [Graux et al., 2016b] . . . 57

8.1 Containment Procedure Diagram . . . 63
8.2 Pattern tree example (well-designed OPT pattern in normal form) . . . 66

9.1 Containment Procedure Diagram by Encoding to FOL . . . 76

10.1 Comparing ranking-optimized query evaluation with other systems (WebIndex data) . . . 103
10.2 Comparing ranking-optimized query evaluation with other systems (SNB data 5M nodes) . . . 105
10.3 Comparing ranking-optimized query evaluation with other systems (SNB data 30M nodes) . . . 105
10.4 Comparing ranking-optimized query evaluation with other systems (SNB data 50M nodes) . . . 106
10.5 Comparing ranking-optimized query evaluation with other systems (SNB data 100M nodes) . . . 106

A.1 Queries for SPARQL containment experimentation . . . 122
A.2 Schemas for SPARQL containment experimentation . . . 123

B.1 Shape relation graph (WebIndex) . . . 126
B.2 Hand-crafted experiment queries . . . 127
B.3 Schema (LDBC SNB) . . . 128
B.4 Shape relation graph (LDBC SNB) . . . 129

1

Introduction

Contents

1.1 Motivation and Objectives **2**
1.2 Summary of Contributions **4**
1.3 Thesis Outline **4**
1.3.1 Part I - State-of-the-Art 4
1.3.2 Part II - Contribution I 5
1.3.3 Part III - Contribution II 5

1.1 Motivation and Objectives

The World Wide Web Consortium (W3C) [Berners-Lee and Jaffe, 1994] is an international organization that works to develop Web standards. W3C's mission is to lead the Web to its full potential by developing protocols and guidelines that ensure the long-term growth of the Web.

The term "Semantic Web" refers to W3C's vision of the Web of linked data. It was coined by Tim Berners-Lee, the inventor of the World Wide Web and director of W3C. It is a Web of data where the Resource Description Framework (RDF) provides its essential mean of data representation and publishing. RDF, with other semantic web technologies, aims at describing the Web in a meaningful way, by standardizing their representations, structuring them, establishing relations and links between them, providing inference rules within them, and providing easy ways to access them. The Semantic Web can be seen as an extension to the classic Web of documents, not an alternative for it.

The essential goal of the Semantic Web is to enable automation of the data on the web by defining standards and to develop systems that can support trusted interactions over the network.

A pre-RDF ground work started in 1997. RDF was adopted as a W3C recommendation in 1999. In 2013, more than four million Web domains contained Semantic Web markup according to keynote by Ramanathan V. Guha at ISWC 2013 (The 12th International Semantic Web Conference) [Guha, 2013].

If the Semantic Web is viewed as a global database, then it is easy to understand why one would need a query language for that data. SPARQL is the query language for the Semantic Web.

The architecture of the Semantic Web can be viewed as a series of layers, where each layer serves some purpose in the definition of the Semantic Web. This idea is known as the Semantic Web Layer Cake, and is attributed to Tim Berners-Lee. A representation of the Semantic Web Layer Cake is shown in Figure 1.1. Each layer is less general than the layers below it.

The W3C provides standards within the stack of the Semantic Web Layer Cake like RDF (a data interchange), SPARQL (a query language for RDF), and OWL (an ontology language which extends the semantics of RDF).

Some of the challenges for the Semantic Web include vastness, vagueness, uncertainty, inconsistency, and deceit. With the increased importance and growth of the Semantic Web on the other hand, research investigations within its technologies and standards become essential.

Several studies have been accomplished concerning semantic web technologies, where some of these studies represent a milestone for RDF. For example in [Schmidt et al., 2010] the authors define the evaluation complexity bounds for different fragments of SPARQL, and

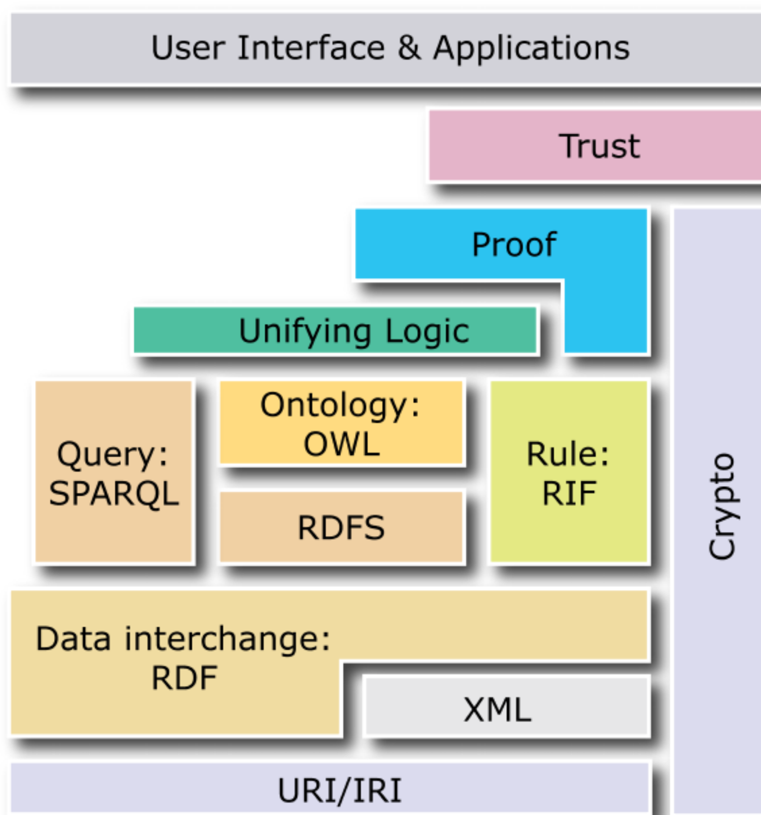

Figure 1.1: Semantic Web Layer Cake
Source: `https://www.w3.org/TR/2014/NOTE-rdf11-primer-20140624/`

many of its sub-fragments. In [Graux et al., 2016a] an optimised implementation of SPARQL queries was provided with distributed data architecture for huge data. In [Pérez et al., 2009] the authors identify a well-designed fragment of SPARQL that has many evaluation and query analysis advantages for improved evaluations. In [Chekol et al., 2012a] the authors proposed a procedure for deciding the containment of SPARQL queries in the presence of RDF Schema (RDFS). Yet the Semantic Web is still being improved, and more research interests become open whenever new technologies are proposed within this community.

Our objectives in this dissertation include studying the interaction between three different technologies within the Semantic Web: RDF, SPARQL, and ShEx. ShEx [Prud'hommeaux, 2017b] is a constraint language for RDF being currently promoted by a W3C working group, providing draft reports like the ShEx Primer draft [Prud'hommeaux, 2017b], and the ShEx Specification draft [Prud'hommeaux et al., 2017]. ShEx would belong to a layer just above the "Data interchange" layer of Fig. 1.1. This technology is designed to be different than RDFS and OWL, which unlike ShEx are not focused on RDF constraints validation. The need of a constraint language for RDF, and the emergence of ShEx as a constraint language open opportunities for investigating its effects, interactions, and the possible improvements within the existing architecture of RDF and SPARQL. To the best of our knowledge this is the first work that consider ShEx with SPARQL in a common framework.

1.2 Summary of Contributions

The contributions in this thesis concern the analysis of SPARQL queries in the presence of ShEx constraints, and can be divided in two sub-directions. First, we consider containment of SPARQL queries in the presence of ShEx constraints. In the second direction we consider optimizing the evaluation of SPARQL queries in the presence of ShEx constraints. The set of contributions can be summarized as follows:

- We propose a sound and complete procedure for the problem of containment with ShEx, for the well-designed OPTIONAL SPARQL fragment extended with external UNIONs. We implemented this procedure and test results were provided.

- We define the complexity of containment with ShEx for the following SPARQL fragment: the BGP fragment (NP-Complete), the well-designed OPTIONAL fragment (NP-Complete), and the well-designed OPTIONAL fragment extended with external UNIONs (Π_2^P-Complete).

- We reduce the SPARQL query containment with ShEx constraints into the First Order Logic (FOL) satisfiability problem. We show that our method can be done in an FOL fragment with only two variables whose satisfiability problem is decidable. While the satisfiability of this fragment is NEXP-Complete, the proposed method allows to consider the containment of a large SPARQL fragment, namely the well-designed OPTIONAL SPARQL fragment extended with external UNIONs, MINUS feature, property path patterns, and a fragment of the FILTER feature. We implemented this procedure and test results were provided.

- We defined a set of well-formed ShEx and RDF pairs, where this set provides interesting characteristics for inferring the relative frequency of occurrence of ShEx types. Based on our previous definition we propose a SPARQL query planning procedure that ranks triple patterns. Ranks are used to decide the order of execution of these triples, which according to our experimentation provides a 25% of query evaluation time gain on average for large amounts of data.

1.3 Thesis Outline

1.3.1 Part I - State-of-the-Art

In Chapter 2 we introduce RDF which is the main data interchange technology for the Semantic Web.

In Chapter 3 we introduce SPARQL which is the standard query language for RDF. We concentrate on the fragments and features that we are going to use this thesis.

In Chapter 4 we describe different Schema languages and Ontology languages for RDF, and we explain the intrinsic differences between these two types of technologies.

In Chapter 5 we introduce First Order Logic and we provide a survey on the different fragments whose satisfiability problem are decidable.

In Chapter 6 we present the most important results on query containment from the literature.

In Chapter 7 we present the most important results on query optimisation from the literature.

1.3.2 Part II - Contribution I

In Chapter 8 we present our procedure for solving query containment for the well-designed OPTIONAL SPARQL fragment extended with UNIONs and in the presence of ShEx constraints. We also define the complexity bounds for the following fragments: the BGP fragment, the well-designed OPTIONAL fragment (NP-Complete), and the well-designed OPTIONAL fragment extended with external UNIONs. We implement the described procedure and we present test results using our implementation.

In Chapter 9 we provide FOL encoding for the problem of SPARQL queries in the presence of ShEx constraints for the the well-designed OPTIONAL SPARQL fragment extended with external UNIONs, MINUS feature, property path patterns, and a fragment of the FILTER feature. We implement the described procedure, we present test results, and we draw a comparison with the results, expressivity and complexity between this procedure and the procedure presented in the previous chapter.

1.3.3 Part III - Contribution II

In Chapter 10 we define ranking procedure for the triple patterns of SPARQL queries based on ShEx constraints. The ranking presented in turns allows for an optimised evaluation of SPARQL queries on large amounts of RDF data.

Finally, in Chapter 11 we conclude our work in this thesis, and we provide the perspectives for future works.

Part I

State-of-the-Art

2

Resource Description Framework (RDF)

Contents

2.1 Introduction **10**

2.2 RDF 1.1 Formal Definition **11**

2.2.1 Abstract Syntax 11

2.3 Interpretation of RDF **13**

2.4 Datatypes for Literals **14**

2.5 RDF Serialization **15**

2.5.1 N-Triples, N-Quads, Turtle and TriG 15

2.5.2 RDF/XML 16

2.5.3 JSON-LD and RDFa 17

2.1 Introduction

The Resource Description Framework (RDF) is a standard model for data interchange on the Web. Resources described by RDF can be anything, including documents, people, physical objects, and abstract concepts [Schreiber and Raimond, 2014].

RDF data describes data by identifying resources, assigning properties to them, and establishing relationships between them. A resource is defined using identifiers - known as Internationalized Resource Identifiers (IRIs) - or literals. A relationship between two resources is known as a triple (`<subject> <predicate> <object>`), making a directed relation from the `<subject>` to the `<object>` using the `<predicate>`. This linking structure forms a directed, labeled multigraph, where the edges represent the named link between two resources, represented by the graph nodes.

Listing 2.1 and Figure 2.1 shows an informal RDF example consisting of triples and its corresponding graph representation respectively.

```
<Bob> <is a> <person>.
<Bob> <is a friend of> <Alice>.
<Bob> <is born on> <the 4th of July 1990>.
<Bob> <is interested in> <the Mona Lisa>.
<the Mona Lisa> <was created by> <Leonardo da Vinci>.
<the video 'La Joconde à Washington'> <is about> <the Mona Lisa>
```

Listing 2.1: RDF example (informal)

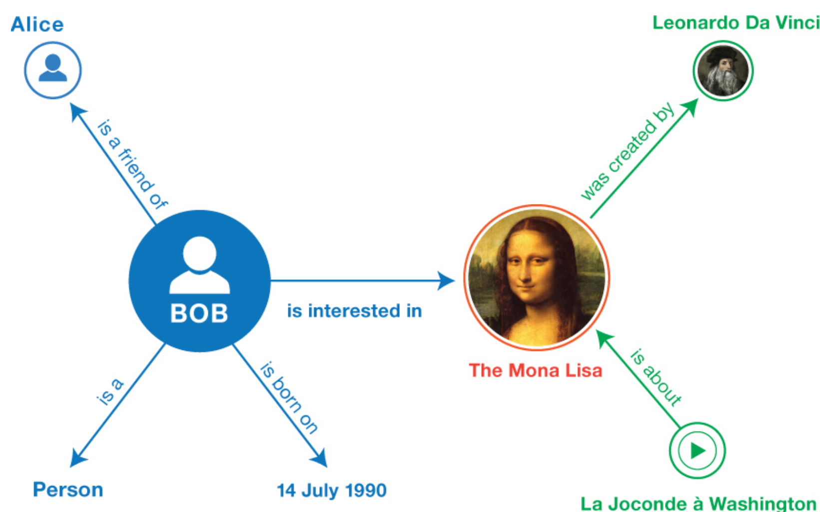

Figure 2.1: RDF graph example
Source: `https://www.w3.org/TR/2014/NOTE-rdf11-primer-20140624/`

RDF provides a standards-compliant way for exchanging data. Using this simple model, it allows structured and semi-structured data to be mixed, exposed, and shared across different applications. Examples of using RDF may be adding machine-readable information to Web

pages, and enriching a dataset by linking it to third-party datasets. RDF has features that facilitate data merging even if the underlying schemas differ, and it specifically supports the evolution of schemas over time without requiring all the data consumers to be changed (i.e. the data always has the triple structure).

RDF consists of a suite of W3C recommendations published in 2004 (known as RDF 1.0). In 2014, a second suite of W3C recommendations was additional Working Group Notes were published defining a more recent version known as RDF 1.1. This latest suite includes (but is not limited to) *RDF 1.1 Concepts and Abstract Syntax* [Cyganiak et al., 2014], *RDF 1.1 XML Syntax*, *RDF 1.1 Semantics* [Patel-Schneider and Hayes, 2014] and other recommendations related to RDF serialisation. The Working Group mission was to extend RDF to include features desirable and important for interoperability, but without a negative effect on deployment.

RDF is currently widely spread on the web and applications exists in several domains. Examples of RDF data available publicly include (but not limited to):

- The project DBpedia which aims at extracting RDF structured content from the information available on the Wikipedia project.
- Bio2RDF which is a biological database based on RDF and semantic web technologies.
- UniProt which provides a comprehensive, high-quality and freely accessible resource of protein sequence and functional information.
- LinkedGeoData which uses the information of the OpenStreetMap project and makes it available in RDF.

In addition, Linking Open Data (LOD) is an initiative and community project to connect related data on the web using RDF (according to the Linked Data format) and making it freely available to everyone.

Figure 2.2 shows datasets that are published in Linked Data format and are interlinked with other datasets in the cloud.

Each node in the diagram of Fig. 2.2 represents an RDF dataset in some domain or special field. These datasets are connected to each where such relation exists. In the current state of the LOD cloud, there are 1,139 interconnected datasets.

2.2 RDF 1.1 Formal Definition

2.2.1 Abstract Syntax

The document [Cyganiak et al., 2014] defines the abstract syntax of RDF 1.1. In this section we introduce the abstract syntax similar to the later.

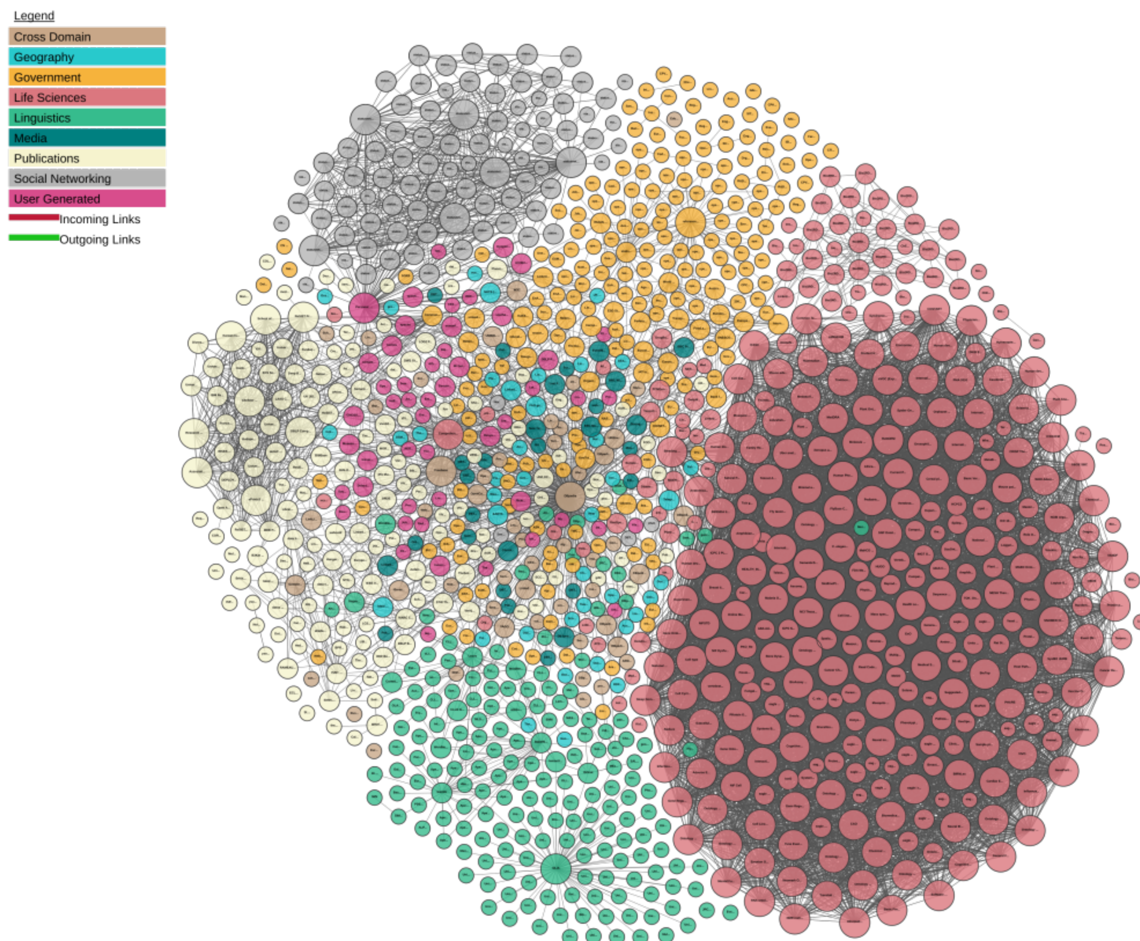

Figure 2.2: The LOD cloud diagram
Source: `http://lod-cloud.net/` (Last updated: 2017-02-20)

We first introduce the terminology for the elements over which RDF graphs are constructed.

Definition 2.2.1 (RDF terminology)**.** *Three pairwise disjoint infinite sets of terms are defined as follows:*

- $\mathcal{I}$*: The set of IRIs*[1] *identifying RDF resources.*
- $\mathcal{L}$*: The set of literals, where a literal may be either a plain or a typed literal.*
- $\mathcal{B}$*: The set of blank nodes, where a blank node is locally scoped for the RDF graph, and does not identify any resource.*

Now we give the definition of an RDF triple whose elements belong to the previously defined sets.

Definition 2.2.2 (RDF triple)**.** *An RDF triple consists of three components:*

[1] An IRI (Internationalized Resource Identifier) within an RDF graph is a Unicode string that conforms to the syntax defined in RFC 3987 [Duerst and Suignard, 2005]. IRIs are a generalization of URIs that permits a wider range of Unicode characters. Every absolute URI and URL is an IRI, but not every IRI is an URI.

- *The subject* $s \in \mathcal{I} \cup \mathcal{B}$
- *The predicate* $p \in \mathcal{I}$
- *The object* $o \in \mathcal{I} \cup \mathcal{L} \cup \mathcal{B}$

An RDF triple is usually written in the order (s, p, o), like in the W3C recommendations for RDF concrete syntaxes.

Definition 2.2.3 (RDF graph)**.** *An RDF graph is a finite set of RDF triples.*

If G is an RDF graph, we use $\mathcal{I}(G)$, $\mathcal{L}(G)$ and $\mathcal{B}(G)$ to respectively denote the set of IRIs, Literals and Blank nodes that appear in at least one triple of G.

As RDF graphs are sets of triples, they can be combined easily, supporting the use of data from multiple sources. Nevertheless, it is sometimes desirable to work with multiple RDF graphs while keeping their contents separate. RDF datasets support this requirement.

Definition 2.2.4 (RDF dataset)**.** *An RDF dataset is a collection of RDF graphs, and comprises:*

- *Exactly one default graph, being an RDF graph. The default graph does not have a name and may be empty.*
- *Zero or more named graphs. Each named graph is a pair consisting of a name* $n \in (\mathcal{I} \cup \mathcal{B})$ *(the graph name), and an RDF graph. Graph names are unique within an RDF dataset.*

2.3 Interpretation of RDF

The semantics of RDF 1.1 is defined in [Patel-Schneider and Hayes, 2014].

In this section we introduce the interpretation of an RDF graph, which is the basis for all other semantic notions discussed for RDF graphs, and also the basis for all the semantics of vocabulary extensions. This interpretation is known as simple interpretation for RDF graphs.

An interpretation of an RDF graph in general is a mapping from IRIs and literals into a set, together with some constraints upon the set and the mapping.

All semantic extensions of any vocabulary or higher-level notation encoded in RDF must conform to the minimal truth conditions defined by the simple interpretation. Other semantic extensions may extend and add to these, but they must not modify or negate them.

Definition 2.3.1 (Simple interpretation)**.** *A simple interpretation* Int *of an RDF graph is a structure consisting of:*

- IR*: The universe of* Int*. A non-empty set of resources.*
- IP*: The set of properties of* Int*.* $(IP \subseteq IR)$

Table 2.1: RDF-compatible XSD types

	Datatype	Value space (informative)
Core types	xsd:string	Character strings (but not all Unicode character strings)
	xsd:boolean	true, false
	xsd:decimal	Arbitrary-precision decimal numbers
	xsd:integer	Arbitrary-size integer numbers
IEEE floating-point numbers	xsd:double	64-bit floating point numbers incl. +-Inf, +-0, NaN
	xsd:float	32-bit floating point numbers incl. +-Inf, +-0, NaN
Time and date	xsd:date	Dates (yyyy-mm-dd) with or without timezone
	xsd:time	Times (hh:mm:ss.sss...) with or without timezone
	xsd:dateTime	Date and time with or without timezone
	xsd:dateTimeStamp	Date and time with required timezone
Recurring and partial dates	xsd:gYear	Gregorian calendar year
	xsd:gMonth	Gregorian calendar month
	xsd:gDay	Gregorian calendar day of the month
	xsd:gYearMonth	Gregorian calendar year and month
	xsd:gMonthDay	Gregorian calendar month and day
	xsd:duration	Duration of time
	xsd:yearMonthDuration	Duration of time (months and years only)
	xsd:dayTimeDuration	Duration of time (days, hours, minutes, seconds only)
Limited-range integer numbers	xsd:byte	-128?+127 (8 bit)
	xsd:short	-32768...+32767 (16 bit)
	xsd:int	-2147483648...+2147483647 (32 bit)
	xsd:long	-9223372036854775808...+9223372036854775807 (64 bit)
	xsd:unsignedByte	0...255 (8 bit)
	xsd:unsignedShort	0...65535 (16 bit)
	xsd:unsignedInt	0...4294967295 (32 bit)
	xsd:unsignedLong	0...18446744073709551615 (64 bit)
	xsd:positiveInteger	Integer numbers >0
	xsd:nonNegativeInteger	Integer numbers ?0
	xsd:negativeInteger	Integer numbers <0
	xsd:nonPositiveInteger	Integer numbers ?0
Encoded binary data	xsd:hexBinary	Hex-encoded binary data
	xsd:base64Binary	Base64-encoded binary data
Miscellaneous XSD types	xsd:anyURI	Absolute or relative URIs and IRIs
	xsd:language	Language tags per [BCP47]
	xsd:normalizedString	Whitespace-normalized strings
	xsd:token	Tokenized strings
	xsd:NMTOKEN	XML NMTOKENs
	xsd:Name	XML Names
	xsd:NCName	XML NCNames

- *A mapping* $IEXT : IP \to 2^{IR \times IR}$
- *A mapping* $IS : \mathcal{I} \to IR$
- *A partial mapping* $IL : \mathcal{L} \to IR$

2.4 Datatypes for Literals

Datatypes are used with RDF literals to represent values such as strings, numbers and dates. The datatype abstraction used in RDF is compatible with XML Schema. These datatypes are known as XSD (XML Schema Definitions) datatypes.

The list of the RDF-compatible XSD types are given in Table 2.1.

2.5 RDF Serialization

RDF 1.1 offers several serialization syntaxes (concrete syntaxes), defined within its W3C recommendations suite. An RDF document in concrete syntax encodes an RDF graph or an RDF dataset which enables their storage and their exchange between systems. However, different ways of writing down the same graph lead to exactly the same triples, and are thus logically equivalent.

The different syntaxes are summarized as follows:

1. Turtle family of RDF languages (N-Triples, N-Quads, Turtle and TriG)
2. RDF/XML (XML syntax for RDF)
3. JSON-LD (JSON-based RDF syntax)
4. RDFa (for HTML and XML embedding)

Since the semantics of RDF is based on assigning IRIs to the RDF terms, the concrete syntaxes also explicitly use IRI definitions for the non-literal nodes of an RDF graph. The example of Fig. 2.1 is annotated with IRIs and is shown in Fig. 2.3. The examples of the concrete syntaxes in this section are based on this annotation.

We notice in Fig. 2.3 that `foaf:`, `xsd:`, `schema:`, `dcterms:`, `wd:` are known as prefixes. Each of them is expanded to form with its successor complete IRIs, but this is a way to make RDF terms more readable. The prefix notion can be utilized in some of the concrete syntaxes as well be seen in the examples that follows.

We also mention that blank nodes may have several representations which are syntax dependent. An explicit occurrence of a blank node (lets say `x`) is given by the following syntax: `_:x`.

2.5.1 N-Triples, N-Quads, Turtle and TriG

These four syntaxes belong to the same family called the Turtle family. Some of these are subsets of others in the same family, the simplest of which is N-Triples, and it belongs to all other syntaxes in the Turtle family.

The example of Fig. 2.3 is given in N-Triples format in Listing 2.2

In N-Triples example, each line (defined by a triple) belongs to an arc in the RDF graph.

N-Quads extends N-Triples to allow encoding RDF datasets instead of only RDF graphs. Instead of triples, it uses quads in each line, the fourth element being the name of the graph. An example of N-Quads is given in Listing 2.3, where there are two named graphs <http://example.org/bob> and <https://www.wikidata.org/wiki/Special:EntityData/Q12418>.

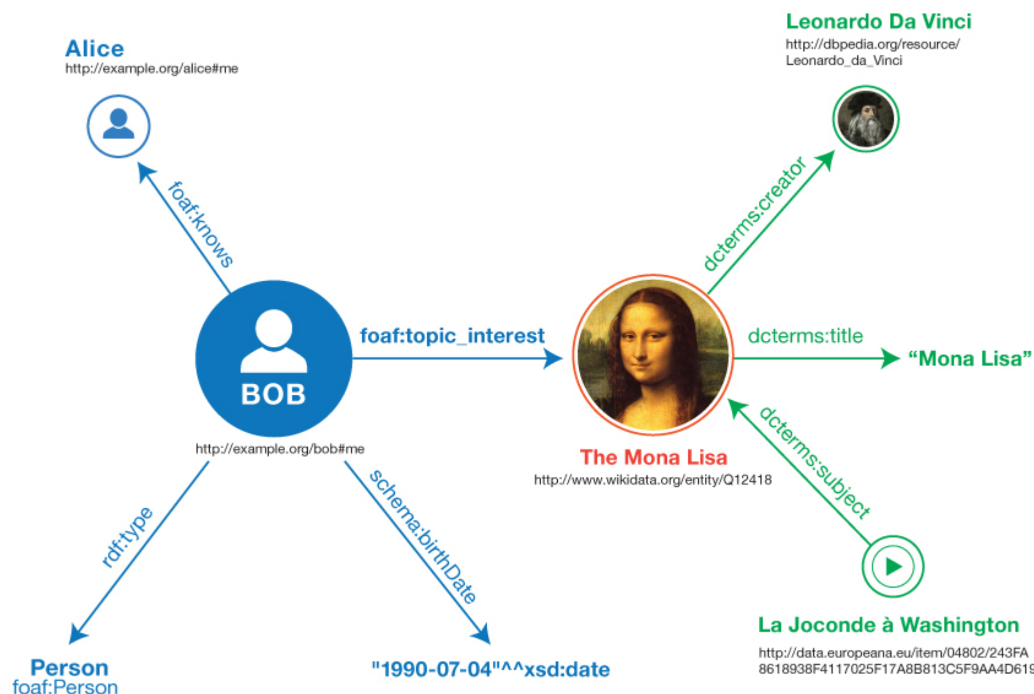

Figure 2.3: RDF graph example
Source: `https://www.w3.org/TR/2014/NOTE-rdf11-primer-20140624/`

```
1 <http://example.org/bob#me> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://xmlns.com/foaf/0.1/Person>.
2 <http://example.org/bob#me> <http://xmlns.com/foaf/0.1/knows> <http://example.org/alice#me> .
3 <http://example.org/bob#me> <http://schema.org/birthDate> "1990-07-04"^^<http://www.w3.org/2001/XMLSchema#date> .
4 <http://example.org/bob#me> <http://xmlns.com/foaf/0.1/topic_interest> <http://www.wikidata.org/entity/Q12418> .
5 <http://www.wikidata.org/entity/Q12418> <http://purl.org/dc/terms/title> "Mona Lisa" .
6 <http://www.wikidata.org/entity/Q12418> <http://purl.org/dc/terms/creator> <http://dbpedia.org/resource/
      Leonardo_da_Vinci> .
7 <http://data.europeana.eu/item/04802/243FA8618938F4117025F17A8B813C5F9AA4D619> <http://purl.org/dc/terms/subject>
       <http://www.wikidata.org/entity/Q12418> .
```

Listing 2.2: RDF concrete syntax (N-Triples)

Turtle and TriG are analogous to N-Triples and N-Quads respectively, but they extend them in oder to allow the definition of prefixes, and alternative triples compact representations. Prefixes make a triple (or a quad) representation more concise and better readable by users. Example of Turtle and TriG are given in Listing 2.4 and Listing 2.5.

2.5.2 RDF/XML

RDF/XML was the only syntax adopted when RDF was originally introduced in 1990s [Schreiber and Raimond, 2014]. It is simply an XML syntax for RDF graph encoding. An example in RDF/XML syntax is given in Listing 2.6.

```
1 <http://example.org/bob#me> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://xmlns.com/foaf/0.1/Person>
      <http://example.org/bob> .
2 <http://example.org/bob#me> <http://xmlns.com/foaf/0.1/knows> <http://example.org/alice#me> <http://example.org/
      bob> .
3 <http://example.org/bob#me> <http://schema.org/birthDate> "1990-07-04"^^<http://www.w3.org/2001/XMLSchema#date> <
      http://example.org/bob> .
4 <http://example.org/bob#me> <http://xmlns.com/foaf/0.1/topic_interest> <http://www.wikidata.org/entity/Q12418> <
      http://example.org/bob> .
5 <http://www.wikidata.org/entity/Q12418> <http://purl.org/dc/terms/title> "Mona Lisa" <https://www.wikidata.org/
      wiki/Special:EntityData/Q12418> .
6 <http://www.wikidata.org/entity/Q12418> <http://purl.org/dc/terms/creator> <http://dbpedia.org/resource/
      Leonardo_da_Vinci> <https://www.wikidata.org/wiki/Special:EntityData/Q12418> .
7 <http://data.europeana.eu/item/04802/243FA8618938F4117025F17A8B813C5F9AA4D619> <http://purl.org/dc/terms/subject>
       <http://www.wikidata.org/entity/Q12418> <https://www.wikidata.org/wiki/Special:EntityData/Q12418> .
8 <http://example.org/bob> <http://purl.org/dc/terms/publisher> <http://example.org> .
9 <http://example.org/bob> <http://purl.org/dc/terms/rights> <http://creativecommons.org/licenses/by/3.0/> .
```

Listing 2.3: RDF concrete syntax (N-Quads)

```
1  BASE   <http://example.org/>
2  PREFIX foaf: <http://xmlns.com/foaf/0.1/>
3  PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
4  PREFIX schema: <http://schema.org/>
5  PREFIX dcterms: <http://purl.org/dc/terms/>
6  PREFIX wd: <http://www.wikidata.org/entity/>
7
8  <bob#me>
9      a foaf:Person ;
10     foaf:knows <alice#me> ;
11     schema:birthDate "1990-07-04"^^xsd:date ;
12     foaf:topic_interest wd:Q12418 .
13
14 wd:Q12418
15     dcterms:title "Mona Lisa" ;
16     dcterms:creator <http://dbpedia.org/resource/Leonardo_da_Vinci> .
17
18 <http://data.europeana.eu/item/04802/243FA8618938F4117025F17A8B813C5F9AA4D619>
19     dcterms:subject wd:Q12418 .
```

Listing 2.4: RDF concrete syntax (Turtle)

2.5.3 JSON-LD and RDFa

JSON-LD and RDFa are further two new different syntaxes that the previously introduced in this chapter, and we only mention the purpose of these two syntaxes here.

JSON-LD is a JSON syntax for encoding RDF graphs. This also allows to change JSON document into RDF document with minimal changes.

RDFa is used to embed RDF data within HTML and XML documents. They can serve as metadata for search engines while crawling the web.

```
BASE   <http://example.org/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX schema: <http://schema.org/>
PREFIX dcterms: <http://purl.org/dc/terms/>
PREFIX wd: <http://www.wikidata.org/entity/>

GRAPH <http://example.org/bob>
{
    <bob#me>
        a foaf:Person ;
        foaf:knows <alice#me> ;
        schema:birthDate "1990-07-04"^^xsd:date ;
        foaf:topic_interest wd:Q12418 .
}

GRAPH <https://www.wikidata.org/wiki/Special:EntityData/Q12418>
{
    wd:Q12418
        dcterms:title "Mona Lisa" ;
        dcterms:creator <http://dbpedia.org/resource/Leonardo_da_Vinci> .

    <http://data.europeana.eu/item/04802/243FA8618938F4117025F17A8B813C5F9AA4D619>
        dcterms:subject wd:Q12418 .
}

<http://example.org/bob>
    dcterms:publisher <http://example.org> ;
    dcterms:rights <http://creativecommons.org/licenses/by/3.0/> .
```

Listing 2.5: RDF concrete syntax (TriG)

```
<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF
            xmlns:dcterms="http://purl.org/dc/terms/"
            xmlns:foaf="http://xmlns.com/foaf/0.1/"
            xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
            xmlns:schema="http://schema.org/">
    <rdf:Description rdf:about="http://example.org/bob#me">
        <rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
        <schema:birthDate rdf:datatype="http://www.w3.org/2001/XMLSchema#date">1990-07-04</schema:birthDate>
        <foaf:knows rdf:resource="http://example.org/alice#me"/>
        <foaf:topic_interest rdf:resource="http://www.wikidata.org/entity/Q12418"/>
    </rdf:Description>
    <rdf:Description rdf:about="http://www.wikidata.org/entity/Q12418">
        <dcterms:title>Mona Lisa</dcterms:title>
        <dcterms:creator rdf:resource="http://dbpedia.org/resource/Leonardo_da_Vinci"/>
    </rdf:Description>
    <rdf:Description rdf:about="http://data.europeana.eu/item/04802/243FA8618938F4117025F17A8B813C5F9AA4D619">
        <dcterms:subject rdf:resource="http://www.wikidata.org/entity/Q12418"/>
    </rdf:Description>
</rdf:RDF>
```

Listing 2.6: RDF concrete syntax (RDF/XML)

3

SPARQL and Queries

Contents

3.1 Introduction **20**

3.2 Abstract Syntax **20**

3.3 SPARQL Query **21**

3.4 Well-Designed OPT Patterns. **22**

3.5 RPQs, NREs, GXPath, and Property Path Patterns **23**

3.6 SPARQL Semantics **23**

3.7 Preparing Required Fragments for the Rest of this Thesis **25**

3.7.1 Allowed SPARQL Filters 25

3.1 Introduction

In 2004, the W3C launched the Data Access Working Group for designing an RDF query language, called SPARQL. SPARQL 1.0 [Prud'hommeaux and Seaborne, 2008] became an official W3C Recommendation in 2008, and SPARQL 1.1 [Harris and Seaborne, 2013] in 2013.

In this chapter we define the syntax and semantics of the SPARQL query language. SPARQL 1.1 extends SPARQL 1.0 by adding features to the query language such as aggregates, subqueries, negation, property paths, and an expanded set of functions and operators. Any of these extensions, if described in this section, will be mentioned explicitly as belonging to SPARQL 1.1. Otherwise, the syntax and semantics are common between the two versions.

For a complete description of SPARQL, the reader is referred to the SPARQL 1.1 specification document [Harris and Seaborne, 2013] or to [Pérez et al., 2009, Polleres, 2007] for its formal semantics.

3.2 Abstract Syntax

In this section we define an abstract syntax for SPARQL (as commonly found in the literature, for example in [Pérez et al., 2009]) which will be used throughout the remaining part of this dissertation.

First we define query variables. A SPARQL query may contain variables that will be bound to values to give a solution for the query.

Definition 3.2.1 (Query Variable[1]). A query variable is a member of an infinite set that is disjoint from the set of RDF terms. We write $\mathcal{V}$ for the set of variable names.

Definition 3.2.2 (Triple pattern). A triple pattern consists of three components:

- The subject $s \in \mathcal{I} \cup \mathcal{L} \cup \mathcal{V}$
- The predicate $p \in \mathcal{I} \cup \mathcal{V}$
- The object $o \in \mathcal{I} \cup \mathcal{L} \cup \mathcal{V}$

Definition 3.2.3 (Basic Graph Pattern). A Basic Graph Pattern is a set of Triple Patterns.

SPARQL is based around graph pattern matching (Fig. 3.1). Complex graph patterns can be formed by combining smaller patterns in various ways. We define the following graph patterns used in SPARQL:

[1] In SPARQL, a query variable is marked by the use of either "?" or "$"; the "?" or "$" is not part of the variable name. In a query, $abc and ?abc identify the same variable.

- **Basic Graph Pattern (BGP)**, where a set of triple patterns must match. It combines triple patterns by conjunction.
- **Group Graph Pattern**, where a set of graph patterns must all match. It combines graph patterns by conjunction. (For this we use the keyword `AND`)
- **Optional Graph pattern**, where additional patterns may extend the solution. (For this we use the keyword `OPTIONAL` or `OPT` for conciseness)
- **Alternative Graph Pattern**, where two or more possible patterns are tried. It provides a means of combining graph patterns so that one of several alternative graph patterns may match. If more than one of the alternatives matches, all the possible pattern solutions are retrieved. (For this we use the keyword `UNION`)
- **Excluded Graph Pattern (MINUS)**, where an exclusion of results occurs based on removing matches of the evaluation of a graph pattern from another graph pattern. (For this we use the keyword `MINUS`)
- **Filtered Graph Pattern**, where boolean-valued expressions (resembling constraints) limit the number of answers to be returned. Each answer has a set of bindings of variables to RDF terms. Filters restrict solutions to those for which the filter expression evaluates to TRUE. (For this we use the keyword `FILTER`)

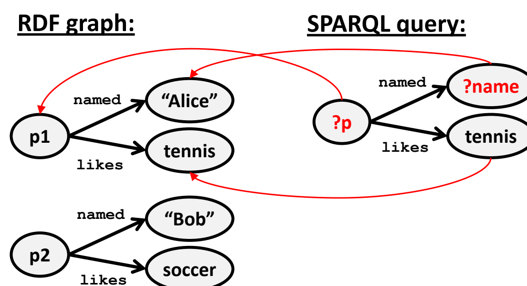

Figure 3.1: SPARQL query matching

Definition 3.2.4 (SPARQL Graph Pattern)**.** A SPARQL graph pattern is defined inductively in the following way:

- every basic graph pattern is a SPARQL graph pattern.
- if P and P' are SPARQL graph patterns and K is a SPARQL constraint, then (P `AND` P'), (P `UNION` P'), (P `OPT` P'), (P `MINUS` P'), and (P `FILTER` K) are SPARQL graph patterns.

3.3 SPARQL Query

SPARQL has four query forms. These query forms use the solutions from pattern matching to form result sets or RDF graphs. The query forms are:

- **SELECT:** Returns all, or a subset of, the variables bound in a query pattern match.
- **CONSTRUCT:** Returns an RDF graph constructed by substituting variables in a set of triple templates.
- **ASK:** Returns a boolean indicating whether a query pattern matches or not.
- **DESCRIBE:** Returns an RDF graph that describes the resources found.

Definition 3.3.1 (SPARQL query)**.** Given a SPARQL graph pattern P, a sequence $\vec{B}$ of variables in P , an IRI μ, and a basic graph pattern Q,

- `ASK FROM` μ `WHERE` P
- `SELECT` $\vec{B}$ `FROM` μ `WHERE` P
- `CONSTRUCT` Q `FROM` μ `WHERE` P
- `DESCRIBE` $\vec{B}$ `FROM` μ `WHERE` P

are SPARQL queries.

In our studies in this thesis we only work with `SELECT` queries, since this type queries are the most relevant to the query static analysis targeted in this thesis, namely the problem of query containment and query evaluation.

3.4 Well-Designed OPT Patterns.

Well-designed OPT patterns define a class of OPTIONAL patterns that have several desired properties [Pérez et al., 2009]. This fragment can be evaluated efficiently, and it provides a pattern that can be easily handled in static analysis tasks.

A query q is well-designed if for every subpattern $q' = (q_1\ OPT\ q_2)$ of q and every variable x occurring in q, it holds that: if x occurs inside q_2 and outside q', then x also occurs inside q_1.

It is also shown in [Pérez et al., 2009] that any well-designed graph pattern can be equivalently rewritten in the normal form:

$(\ldots(t_1\ AND\ \ldots\ AND\ t_k)\ OPT\ O_1)\ OPT\ O_2)\ldots)\ OPT\ O_n)$ *where each t_i is a triple pattern, and each O_j has the same form (also in normal form).*

These normal forms can be represented as pattern trees as described in [Letelier et al., 2012]. For example, a query of the form $((P_1\ OPT\ (P_{11}\ OPT\ P_{111}\ OPT\ P_{112}))\ OPT\ P_{12})\ OPT\ P_{13}$, where each P_i is a BGP, can be represented as a pattern tree as in Fig. 3.2.

We use well-designed OPT patterns as the fragment considered for the study of SPARQL query containment in Chap. 8 and 9.

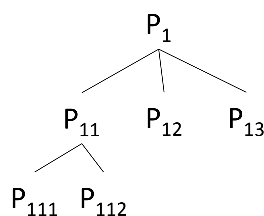

Figure 3.2: Pattern tree example

3.5 RPQs, NREs, GXPath, and Property Path Patterns

RPQs, NREs, and GXPath are graph query languages based on the idea of using regular expressions to specify patterns that must be matched by paths in the input graph. Given a query q, the result of its evaluation over a graph G is always a set of node pairs (v, v') such that v and v' are connected by a path p in G matching the query q. These languages mainly differ in the class of supported regular expressions.

Regular Path Queries (RPQs) [Mendelzon and Wood, 1995] are the most basic language including concatenation, alternative, and recursion operators.

Nested Regular Expressions (NREs) [Pérez et al., 2008] extend RPQs to introduce the ability of traversing edges backwards, as well as the ability of specifying conditions inside paths.

GXPath [Libkin et al., 2013] is essentially an adaptation of XML Path language (XPath) [Robie et al., 2017] to data graphs. With respect to the previous languages, GXPath introduces the complement operator, data tests on the values stored into nodes, as well as counters, which generalize the Kleene star.

NREs form the basis of Property Path Patterns, the path language of SPARQL 1.1. In a SPARQL query triple pattern, the predicate position may be a path (called Property Path) based on regular expressions whose definition is given in Def. 3.5.1.

Definition 3.5.1 (Property Path)**.** A Property Path p is defined inductively as:

$$p ::= I \mid \hat{\ } p \mid !p \mid (p/p) \mid (p|p) \mid p? \mid p*$$

where $I \in \mathcal{I}$, $\hat{\ }p$ is the inverse property, $!p$ is an excluded property, / is a path concatenation symbol, | is an alternative path symbol, ? is optional and $*$ is the Kleene star.

3.6 SPARQL Semantics

In this section we provide a semantic interpretation for SPARQL evaluation. Similar semantic descriptions are common in the literature. Here we mainly follow the semantics

provided in [Pérez et al., 2009], in addition to the semantics of property path patterns from [Chekol, 2016].

Let μ be a mapping (partial function) from $\mathcal{V}$ to the set of RDF terms ($\mathcal{I} \cup \mathcal{L} \cup \mathcal{B}$). dom($\mu$) is the subset of $\mathcal{V}$ where μ is defined. Given a triple t, $\mu(t)$ is the triple obtained by replacing the variables in t according to μ. Similarly we define $\mu(P)$ for a basic graph pattern P.

Two mapping μ_1 and μ_2 are compatible if for all $x \in \text{dom}(\mu_1) \cap \text{dom}(\mu_2)$, $\mu_1(x) = \mu_2(x)$.

Let Ω_1 and Ω_2 be sets of mappings. We adopt the the definition for the join, union, difference, and outer-join as follows:

- $\Omega_1 \bowtie \Omega_2 = \{\mu_1 \cup \mu_2 \mid \mu_1 \in \Omega_1, \mu_2 \in \Omega_2 \text{ and } \mu_1, \mu_2 \text{ are compatible}\}$
- $\Omega_1 \cup \Omega_2 = \{\mu \mid \mu \in \Omega_1 \text{ or } \mu \in \Omega_2\}$
- $\Omega_1 \backslash \Omega_2 = \{\mu \in \Omega_1 \mid \text{ for all } \mu' \in \Omega_2, \mu \text{ and } \mu' \text{ are not compatible mappings}\}$
- $\Omega_1 ⟕ \Omega_2 = (\Omega_1 \bowtie \Omega_2) \cup (\Omega_1 \backslash \Omega_2)$

We use the previous definitions to further define the evaluation of a SPARQL graph pattern in Def. 3.6.1.

Definition 3.6.1 (SPARQL Graph Pattern Evaluation)**.** Let G be an RDF graph. The evaluation of a graph pattern P over G, denoted by $[\![P]\!]_G$, can be defined inductively as follows:

- $[\![(x, I, y)]\!]_G = \{\mu \mid (\mu(x), \mu(I), \mu(y)) \in G\}$
- $[\![(x, p?, y)]\!]_G = \{\mu \mid \mu(x) = \mu(y)\} \cup [\![(x, p, y)]\!]_G$
- $[\![(x, \hat{}p, y)]\!]_G = [\![(y, p, x)]\!]_G$
- $[\![(x, p_1|p_2, y)]\!]_G = [\![(x, p_1, y)]\!]_G \cup [\![(x, p_2, y)]\!]_G$
- $[\![(x, p_1/p_2, y)]\!]_G = \exists n : [\![(x, p_1, n)]\!]_G \bowtie [\![(n, p_2, y)]\!]_G$
- $[\![(x, p*, y)]\!]_G = \{\mu \mid \mu(x) = \mu(y)\} \cup \bigcup_{i \geq 1} [\![(x, p^i, y)]\!]_G$
- $[\![P_1 \texttt{ AND } P_2]\!]_G = [\![P_1]\!]_G \bowtie [\![P_2]\!]_G$
- $[\![P_1 \texttt{ UNION } P_2]\!]_G = [\![P_1]\!]_G \cup [\![P_2]\!]_G$
- $[\![P_1 \texttt{ OPT } P_2]\!]_G = [\![P_1]\!]_G ⟕ [\![P_2]\!]_G$
- $[\![P_1 \texttt{ MINUS } P_2]\!]_G = [\![P_1]\!]_G \backslash [\![P_2]\!]_G$
- $[\![P \texttt{ FILTER } K]\!]_G = \{\mu \in [\![P]\!]_G \mid \mu \models K\}$

Where $I \in \mathcal{I}$, K is a SPARQL constraint, p is a property path, and p^i is the composition of p i times $(p/p/\ldots/p)$.

3.7 Preparing Required Fragments for the Rest of this Thesis

In order to reference different SPARQL fragments later, we define them as follows:

- **BGP**: This is the conjunctive fragment of SPARQL, i.e. the fragment that only allows using the `AND` operator between triples.
- **AND-OPT**: The fragment of SPARQL allowing the `AND` and `OPT` operators only. We particularly consider the well-designed patterns within this fragment (already defined).
- **AND-OPT-(UNION)**: The AND-OPT fragment extended with UNION on the top level only (external).
- **AND-OPT-(UNION)-FILTER**: The AND-OPT-(UNION) fragment extended with the `FILTER` operator for a particular fragment of constraints. These constraints will be defined hereafter (Section 3.7.1).
- **AND-OPT-(UNION)-PP**: The AND-OPT-(UNION) fragment extended with property path patterns from the SPARQL 1.1 syntax. These are regular expressions allowed in the predicate position.
- **AND-OPT-(UNION)-MINUS**: The AND-OPT-(UNION) fragment extended with the `MINUS` operator which puts constraints on situations that must not occur in the results.

3.7.1 Allowed SPARQL Filters

For our up-coming work in this thesis, we are interested in the filter feature to be considered for the SPARQL query containment problem that will be studied. We only consider filters that are decidable for query satisfiability given in [Zhang et al., 2016], which is a necessary requirement for query containment as will be discussed in Chapter 9, Section 9.5.2. Accordingly we define only these filter fragments as follows:

- $FILTER(bound, =, \neq_c)$: The filter fragment only allowing the operators *bound*, $=$, and $\neq_c$.
- $FILTER(bound, \neq, \neq_c)$: The filter fragment only allowing the operators *bound*, $\neq$, and $\neq_c$.

Where $bound(?x)$ means that the variable $?x$ should be bound to a value in the query results. $=/\neq$ are the equality/inequality relations between variables. $\neq_c$ is the inequality of variable with respect to a constant belonging to $(\mathcal{I} \cup \mathcal{L})$.

4

Schema and Ontology Languages for RDF

Contents

4.1 Introduction **28**

4.2 Abstract Syntax **30**

4.3 Semantics **31**

4.4 Other Schema and Ontology Languages for RDF **31**

4.4.1 SHACL 32

4.4.2 Ontology Languages 33

4.5 Conclusion **36**

4.1 Introduction

In this chapter we are going to mainly concentrate on ShEx, a schema language for RDF. Later in this book, we are going to use ShEx for static analysis and optimisation of SPARQL queries. In Sect. 4.4 we introduce another schema language called SHACL and compare it to ShEx. We then introduce ontology languages and explain how they are different from schema languages although they may have common characteristics. We give two examples of ontology languages, RDFS and OWL.

ShEx (or Shape Expressions) is intended to be an RDF constraint language. Logical operators in Shape Expressions such as grouping, conjunction, disjunction and cardinality constraints, are defined to make as closely as possible to their counterparts in regular expressions and grammar languages like BNF [Prud'hommeaux et al., 2014]. Shape Expressions correlate an ordered pattern of pairs of predicate and object classes (called NameClass and ValueClass) and logical operators against an unordered set of edges in a graph. In the example Listing 4.1, $\langle Shape1 \rangle$ is a definition of a shape in ShEx, where a ShEx document contains definitions of several shapes.

```
<Shape1> {
 ex:name xsd:string ,
 ex:phone xsd:string }
```

Listing 4.1: Simple ShEx example

In the previous example, ex:name and ex:phone are NameClasses and xsd:string is a ValueClass. This definition means that for a node belonging to this shape there must strictly exist the predicates ex:name and ex:phone, each once. The objects corresponding to these predicates must be of type xsd:string.

Table. 4.1 gives the ShEx vocabulary with simple examples.

Figure 4.1 shows some ShEx validation examples simple RDF graphs.

ShEx may be serialised using any of three interchangeable concrete syntaxes:

- **Shape Expressions Compact Syntax (ShExC)**, a compact syntax meant for human eyes and fingers. (As in the example of Listing 4.1)
- **ShExJ**, a JSON-LD [Kellogg et al., 2014] syntax meant for machine processing.
- **ShExR**, the RDF interpretation of ShExJ expressed in RDF Turtle syntax.

The complexity of RDF validation against ShEx is shown to be NP-complete in [Staworko et al., 2015].

ShEx is an emerging schema language which has been recently used in several applications including:

Table 4.1: ShEx Vocabulary
Source: `https://www.w3.org/2001/sw/wiki/index.php?title=ShEx&oldid=5008`

Feature	Example	Description
		Matching a Predicate to a NameClass
NameTerm	ex:state	The predicate of any matching triple is the same as the NameTerm IRI.
NameStem	ex:~	The predicate of any matching triple starts with the IRI.
NameAny	. - rdf:type - ex:~	A matching triple has any predicate except those terms NameTerms or NameStems excluded by the '-' operator.
		Matching an Object to a ValueClass
ValueType	xsd:dateTime	The object of any matching triple is the same as the ValueType IRI.
ValueSet	(ex:unassigned ex:assigned)	The object of any matching triple is one of the list of triples in the ValueSet.
ValueStem	ex:~	The object of any matching triple starts with the IRI.
ValueAny		A matching triple has any object except those terms or stems excluded by the '-' operator.
ValueReference	@<UserShape>	The object of a matching triple is an IRI or blank node and the that node is the subject of triples matching the referenced shape expression.
		Rule Types
ArcRule	foaf:givenName xsd:string+	A matching triple matches the NameTerm and the ValueTerm. Cardinality constraints apply.
AndRule	foaf:givenName xsd:string, foaf:familyName xsd:string	Each conjoint matches the input graph.
OrRule	foaf:givenName xsd:string foaf:name xsd:string	Exactly one disjoint matches the input graph.
GroupRule	(x:reproducedBy @<EmployeeShape>, ex:reproducedOn xsd:dateTime)	A matching triple matches the enclosed rule (here an AndRule). Cardinality constraints apply.
		Cardinality
?	foaf:givenName xsd:string?	rule must match 0 or 1 times.
+	foaf:givenName xsd:string+	rule must match 1 or more times.
*	foaf:givenName xsd:string*	rule must match 0 or more times.
{m}	foaf:givenName xsd:string{3}	rule must match m times.
{m,n}	foaf:givenName xsd:string{3,5}	rule must match at least m times and no more than n times.
		Rule Inclusions
&RuleName	& <PersonShape>	Include the referenced rule in place of the include directive.

```
<publication> {
  title xsd:string,
  date xsd:date,
  author @<author>+
  reference @<publication>* }
```

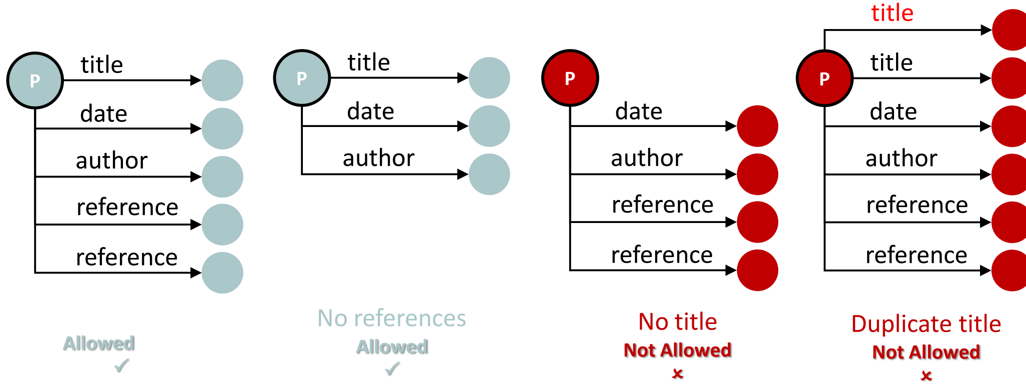

Figure 4.1: ShEx validation examples

- **FHIR** [HL7, 2017], a platform specification that defines a set of capabilities use across the healthcare process, in all jurisdictions, and in lots of different contexts (clinical, diagnostics, medications, workflow, financial...).
- **WebIndex data** [World Wide Web Foundation, 2014], which are data intended as a measure of the World Wide Web's contribution to social, economic and political progress in countries across the world.

4.2 Abstract Syntax

Given a finite set of edge labels Σ and a finite set of types Γ, we define a shape expression e over $\Sigma \times \Gamma$ as follows:

$$e ::= \epsilon \mid \Sigma \times \Gamma \mid e^* \mid (e|e') \mid (e||e')$$

where "$|$" denotes disjunction, "$||$" denotes unordered concatenation, and "$*$" denotes zero or more repetitions. From this definition we also further define the following operators as macros:

- $e^? := (\epsilon \mid e)$ (optional)
- $e^+ := (e \mid\mid e^*)$ (repetition for a positive number of times)
- $e^{[m;n]}$ (e repeated i times with i in the interval from m to n)

which are also parts of the ShEx syntax. In the sequel we write $a :: t$ as a shorthand for $(a, t) \in \Sigma \times \Gamma$.

We denote by $\Psi(\Sigma, \Gamma)$ the set of all possible shape expressions over $\Sigma \times \Gamma$.

A ShEx schema is a tuple $S = (\Sigma, \Gamma, \delta)$, where Σ is a finite set of edge labels, Γ is a finite set of types, and δ is a type definition function that maps elements of Γ to elements of $\Psi(\Sigma, \Gamma)$, i.e. $\delta : \Gamma \to \Psi(\Sigma, \Gamma)$.

We notice that a ShEx shape (or simply a shape) is itself a type. While a shape is considered as a user-defined type, more generally a type may also be a built-in type (like `xsd:string` in the concrete syntaxes of ShEx).

A ShEx schema must have at least one ShEx shape. An empty ShEx schema is a ShEx schema where for each ShEx shape t in it, $\delta(t) = \epsilon$.

4.3 Semantics

[Prud'hommeaux et al., 2014] Semantically, an RDF graph is valid against a ShEx schema if it is possible to assign types to the nodes of the graph in a manner that satisfies the type definitions of the schema.

We assume a fixed graph $G = (V, E)$ which resembles an RDF graph, and a fixed schema $S = (\Sigma, \Gamma, \delta)$. A typing of G w.r.t. S is a function $\lambda : V \to 2^{\Gamma}$ that associates with every node of G a set of types.

Next, the conditions that a typing needs to satisfy are identified. Given a typing λ and a node $n \in V$ we define the neighborhood-typing of n w.r.t. λ as bag (i.e. multiset) over $\Sigma \times 2^{\Gamma}$ as $neighborTyping^{\lambda}_{G}(n) = \{\!\!\{ a :: \lambda(m) \mid (n, a, m) \in E \}\!\!\}$. We note by $L(e)$ the bag language of a shape expression e, i.e. $L(e)$ is the set of bags allowed by the language of e.

Now, λ is a valid typing of S on G if and only if every node satisfies the type definitions of its associated type i.e., for every $n \in V$, $neighborTyping^{\lambda}_{G}(n) \in L(\delta(t))$, for all $t \in \lambda(n)$.

4.4 Other Schema and Ontology Languages for RDF

In this section we are going to describe SHACL, an alternative schema language for RDF. We then describe two ontology languages, RDFS and OWL, and show how they are different from schema languages.

4.4.1 SHACL

Shapes Constraint Language (SHACL) is a W3C recommendation since the 20th of July 2017 [Knublauch and Kontokostas, 2017]. SHACL, similarly to ShEx, is a language for validating RDF graphs against a set of conditions. It is being used/supported by several applications like:

- TopBraid Composer [TopQuadrant, 2001]
- RDFUnit [AKSW, 2016]
- OpenPublicData [iStandUK, 2017]
- Schema.org [Knublauch, 2017].

ShEx can be used as a surface syntax for SHACL. Although ShEx is not yet a W3C recommendation, it is being promoted and expanded by a W3C community group [Kontokostas, 2017]. There are several reasons why ShEx has an advantage over SHACL to be considered for static analysis, we list them here:

1. SHACL can be translated to ShEx, and thus all the static analysis tools for the latter can also be used for the former.
2. ShEx (boolean operators, grammar based operators, and shapes recursion) is more expressive than SHACL (boolean operators).
3. While both ShEx and SHACL can be used for RDF validation, ShEx specifications are designed to be used for a wider set of applications (to generate data, or drive user interfaces). SHACL focuses on validation errors. With ShEx, after validation, an enriched graph is usually obtained; it contains valid nodes and their shapes.
4. ShEx syntax and semantics are based on mathematical concepts, while SHACL original syntax is verbose (Listing 4.3) and its semantics are based on textual descriptions. (Later SHACL adopted a compact syntax copied from ShEx)

For more information on the comparison between ShEx and SHACL refer to [Gayo, 2016, Gayo, 2015].

Consider an example which is written in both ShExC (Listing 4.2) and SHACL (Listing 4.3). An `:Item` shape must have a property `rdf:type` with value `:Item` and either `:source` with an IRI or a `:computation` with a value of shape `:Computation`. The example shows how the ShExC syntax, which is driven by regular expressions intuition, is human-readable and easy to handle for purposes like mathematical static analysis in comparison to SHACL.

On the other hand, Listing 4.4 is a ShExC example that cannot be expressed in SHACL. The example defines a `<Parent>` shape as nodes that have one or more combination of properties `:hasSon` with any value and `:sonBirthDate` with `xsd:date` value. In SHACL we cannot express that equivalent number of `:hasSon` and `:sonBirthDate` properties should occur.

A SHACL to ShExC syntax translator Scala implementation is available at [Gayo, 2017].

```
:Item {
  a (:Item),
  ( :source IRI
  | :computation @:Computation
  )
}

:Computation { a (:Computation) }
```

Listing 4.2: ShExC: Exclusive Or example. **Source:** `https://github.com/labra/ShExcala/wiki/ShExC-vs-SHACL`

4.4.2 Ontology Languages

Ontology languages allow the encoding of knowledge about specific domains and often include reasoning rules that support the processing of that knowledge. Unlike schema languages, RDF ontology languages (like RDFS and OWL) have semantics that provide extended entailment regimes for the SPARQL query language.

Although ontology languages have some capabilities of modeling languages, they have different purposes and characteristics. The main differences in this sense can be summarized as follows:

- Ontology languages are tailored to describe a domain while schema languages are tailored to describe RDF graphs. The domain of discourse of ontology languages is more related to the problem domain in which one is working like people, organizations, etc. while the domain of discourse of schema languages is the RDF graph that one is trying to describe.
- Ontology languages are usually based on Open World Assumption, while schema languages are usually based on Closed World Assumption.
- Ontology languages are usually based on Non-Unique Name Assumption, while schema languages are usually based on Unique Name Assumption.

These inherent differences make it more difficult to use ontology languages as a constraint validation language.

4.4.2.1 RDF Schema (RDFS)

RDFS [Brickley and Guha, 2014] is an ontology language for RDF. It extends the basic RDF vocabulary to allow, for example, semantic distinctions between classes and properties, structuring classes into hierarchies, and constraining the domains and ranges of properties within these classes and other basic RDF datatypes.

```
:Item a sh:Shape ;
 sh:property [
  sh:predicate rdf:type ;
  sh:hasValue :Item ;
  sh:minCount 1 ;
  sh:maxCount 1
 ] ;
 sh:constraint [
 a sh:OrConstraint ;
 sh:shapes (
   [ sh:property [
     sh:predicate :source ;
     sh:nodeKind sh:IRI ;
     sh:minCount 1 ;
     sh:maxCount 1 ;
   ]]
   [ sh:property [
     sh:predicate :computation ;
     sh:valueShape :Computation ;
     sh:minCount 1 ;
     sh:maxCount 1 ;
   ]]
  )
 ] ;
 sh:constraint [
  a sh:NotConstraint ;
  sh:shape [
  sh:constraint [
  a sh:AndConstraint ;
  sh:shapes (
   [ sh:property [
     sh:predicate :source ;
     sh:nodeKind sh:IRI ;
     sh:minCount 1 ;
     sh:maxCount 1
   ]]
   [ sh:property [
     sh:predicate :computation ;
     sh:valueShape :Computation ;
     sh:minCount 1 ;
     sh:maxCount 1 ;
   ]
   ]
  )
  ]
  ]
 ]
.

:Computation a sh:Shape ;
 sh:property [
  sh:predicate rdf:type ;
  sh:hasValue :Computation ;
  sh:minCount 1 ;
  sh:maxCount 1
] .
```

Listing 4.3: SHACL: Exclusive Or example. **Source:** `https://github.com/labra/ShExcala/wiki/ShExC-vs-SHACL`

With RDFS entailment, the same set of RDF triples inherits an extended meaning when a SPARQL query is executed on it. To illustrate the differences between simple, RDF, and RDFS entailment, consider the data of Listing 4.5 and its graphical representation in Figure 4.2.

In RDFS, the vocabularies `rdf:type`, `rdfs:subClassOf`, and `rdfs:range` are attributed a special meaning, and thus allows to extend the set of data as shown in Figure 4.2. That is to say that new implicit RDF triples are added to form the intended meaning which is inferred from the vocabulary.

4.4.2.2 Web Ontology Language (OWL)

OWL [OWL, 2012] is also an ontology language that provides an extended entailment regime to SPARQL queries. OWL has richer vocabulary than RDFS, and is not limited to the

```
<Parent> {
  ( :hasSon .,
    :sonBirthDate xsd:date ) +
}
```

Listing 4.4: ShExC: Groupings and cardinalities

```
ex:book1 rdf:type ex:Publication .
ex:book2 rdf:type ex:Article .
ex:Article rdfs:subClassOf ex:Publication .
ex:publishes rdfs:range ex:Publication .
ex:MITPress ex:publishes ex:book3 .
```

Listing 4.5: Simple RDF example

general semantical structure. For example it allows to define property restrictions that are local to a class. For example:

- It can define cardinalities.
- It can express that if "`A isMarriedTo B`" then this implies "`B isMarriedTo A`".
- It can express that if "`A isAncestorOf B`" and "`B isAncestorOf C`" then "`A isAncestorOf C`".
- It can express that two things are the same, this is very helpful for joining up data expressed in different schemas. You can say that relationship "sired" in one schema is owl:equivalentProperty "fathered" in some other schema. You can also say that two things are the same using owl:sameAs, such as the "Elvis Presley" on wikipedia is the same one on the BBC.

Although the rich vocabulary of OWL allows to express a wide set of constraints, it is not designed to be used as a constraint validation language. To show the point the same set of constraints are expressed in ShExC (Listing 4.6) and in OWL (Listing 4.7).

```
:UserShape {  foaf:name xsd:string
           |  foaf:givenName xsd:string+, foaf:familyName xsd:string
}
```

Listing 4.6: UserShape example in ShExC. **Source:** `https://www.w3.org/2001/sw/wiki/index.php?title=ShEx/ShEx_vs_OWL&oldid=4753`

So, although in OWL all the needed constraints (in the given example) are possible to be expressed, yet it is clear how the concentration on many constraints makes the process complicated. Also because it is based on Open World Assumption, the last two lines are required to insist on the functionality of our properties.

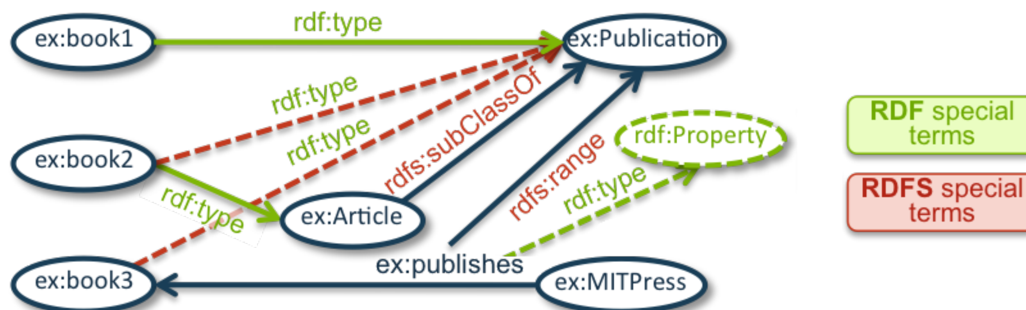

Figure 4.2: A graphical representation of the RDF graph for the example where green dashed lines indicate RDF-entailed triples and red dashed lines indicate triples that are also RDFS-entailed.
Source: `https://www.w3.org/TR/sparql11-entailment/`
Note: This is a modified version of the original figure in the source, since it was reported as an erratum.

4.5 Conclusion

In this chapter we first described ShEx which is a constraint language for RDF. This schema language is emerging, and among others, it is interesting to study this language and its applicabilities. Particularly we are going to use ShEx for static analysis and optimisation of SPARQL queries. We compared ShEx to another schema language that seems to be less interesting for our purpose. We also showed that although ontology languages may be interesting as well, they serve different purposes than schema languages. Studies on SPARQL containment with RDFS and OWL has been done in the literature. We are going to consider ShEx for our studies with SPARQL queries, which is to our best of knowledge, the first work considering ShEx with queries.

```
:UserShapeClass owl:equivalentClass [
  rdf:type owl:Class ;
  owl:unionOf (
    :HasNameClass
    :HasGivenNameFamilyNameClass
  )
] .

:HasNameClass owl:equivalentClass [
 a owl:Class ;
 owl:intersectionOf (
  [ rdf:type owl:Restriction ;
    owl:onProperty foaf:name ;
    owl:someValuesFrom xsd:string ;
  ]
  [ rdf:type owl:Restriction ;
    owl:onProperty foaf:name ;
    owl:cardinality 1
  ]
 )
] .

:HasGivenNameFamilyNameClass owl:equivalentClass [
 a owl:Class ;
 owl:intersectionOf
  ([ rdf:type             owl:Restriction ;
     owl:onProperty       foaf:givenName ;
     owl:someValuesFrom   xsd:string
   ]
   [ rdf:type             owl:Restriction ;
     owl:onProperty       foaf:familyName ;
     owl:someValuesFrom   xsd:string
   ]
   [ rdf:type             owl:Restriction ;
     owl:onProperty       foaf:familyName ;
     owl:cardinality      1
   ]
  )
 ] .

foaf:name a owl:FunctionalProperty .
foaf:familyName a owl:FunctionalProperty .
```

Listing 4.7: UserShape example in OWL. **Source:** `https://www.w3.org/2001/sw/wiki/index.php?title=ShEx/ShEx_vs_OWL&oldid=4753`

5

First Order Logic (FOL)

Contents

5.1 Introduction 40
5.2 FOL Survey 40
5.3 FOL^2 40
5.4 FOL Theorem Provers Implementations and Serialisations 42
5.5 Conclusion 42

5.1 Introduction

FOL (First Order Logic) is in general a logic that in addition to the classical truth values ($\top$,$\bot$) and logical connectives (conjunction $\wedge$, disjunction $\vee$, implication $\rightarrow$...), allows negation, predicate symbols, function symbols, variables, and quantification over variables ($\exists$ and $\forall$).

In this chapter we are going to present our survey done on the characteristics of several FOL fragments, and then particularly introduce FOL with two variables (FOL^2) which we analyze to be suitable (sufficient expressivity and good performance) for the encoding used in our work to solve SPARQL query containment with ShEx constraints.

5.2 FOL Survey

Some fragments of FOL are decidable, while others are not. On the other hand, decidable FOL fragments have different satisfiability complexities. Our interest in a decidable logic with rich features while keeping the satisfiability complexity acceptable for applications led us to make a survey on the decidable FOL fragments from the literature.

In the Table 5.1 we give a summary of the decidable fragments of FOL, with their characteristics and their known satisfiability complexity bounds.

5.3 FOL^2

FOL^2 (FOL with two variables) is a decidable fragment of FOL that allows no more than 2 interacting variables. This fragment is attractive from a theoretical point of view due to its expressivity, adding that it has been proven that it is still decidable with equality/inequality and counting quantifiers (e.g. $\exists^{<n}$). [Grädel et al., 1997][Grädel and Otto, 1999][Etessami et al., 2002]

The presence of more than two variables in a formula is not an evidence that the formula does not belong to FOL^2. Thus the expressivity of FOL^2 may not be intuitive. Consider the following example expression that can be expressed in FOL^2:
"There exists a path of length 4"

- **Form 1:** $\exists x \exists y \exists z \exists w \exists v (R(x,y) \wedge R(y,z) \wedge R(z,w) \wedge R(w,v))$
- **Form 2:** $\exists x \exists y (R(x,y) \wedge \exists z (R(y,z) \wedge \exists w (R(z,w) \wedge \exists v\ R(w,v))))$
- **Form 3:** $\exists x \exists y (R(x,y) \wedge \exists x (R(y,x) \wedge \exists y (R(x,y) \wedge \exists x\ R(y,x))))$

The three forms are equivalent to each other, and they belong to FOL^2.

Table 5.1: Survey on the decidable FOL fragments

Decidable FOL Fragment (description)	**Functions allowed/ arity**	**Predicate arity**	**# of vars**	**Counting quantifiers**	**(In)equality allowed**	**Finite model property**	**Tree model property**
FOL guarded fragment The fragment with only guarded quantification: $\forall \bar{y}\big(\alpha(\bar{x},\bar{y}) \rightarrow \varphi(\bar{x},\bar{y})\big)$ and $\exists \bar{y}\big(\alpha(\bar{x},\bar{y}) \wedge \varphi(\bar{x},\bar{y})\big)$ where guards α: atomic formulae containing all free variables of φ. **SAT(FOL):** 2EXPTIME-complete **SAT(FOL):** EXPTIME-complete for formulae of bounded width. *References (with equality): [Ganzinger 1999]*		∞	∞		✓	✓	✓ (g)
FOL with only 2 variables (L^2) **SAT(FOL):** NEXPTIME-complete *References: [Vardi 1997]*		∞	2		✓	✓	
FOL with only 2 variables (C^2) Is an extension of (L^2) with counting quantifiers *References: [Baader 2002]*		∞	2	✓	✓		
Monadic predicate logic The fragment in which all relation symbols in the signature are monadic (that is, they take only one argument) *References: [Löwenheim 1915], [Kalmár 1929]* *References (with equality): [Bachmair 1993]*	1	1	∞		✓	✓	
The prefix class ($\exists^*\forall^*$) *References: [Bernays, Schönfinkel 1928], [Ramsey 1932]*		∞	∞	✓	✓		
The prefix class ($\exists^*\forall\exists^*$) *References:[Ackermann 1928], [Gurevich 1973]* *References (with equality): [Fermuller 1993]*	✓	∞	∞	✓	✓		
The prefix class ($\exists^*\forall^2\exists^*$) *References:[Gödel 1932], [Kalmár 1933], [Schütte 1934]*		∞	∞				
The prefix class ($\exists^*$) *References: [Gurevich 1976]*	✓	∞	∞	✓	✓		
Propositional Modal Logic - Is a fragment of the (guarded fragment) - Is a fragment of the (2 variable fragment L^2) **SAT(FOL):** PSPACE-complete *References: [Hustadt 2000]*						✓	✓

5.4 FOL Theorem Provers Implementations and Serialisations

The CADE ATP System Competition (CASC) can be considered as the world championship for automated theorem proving. CASC is held at each CADE and IJCAR conference. CASC evaluates the performance of sound, fully automatic, classical logic Automated Theorem Proving systems.

There are many FOL Theorem Provers, which are highly optimised, and participate every year in CASC. Examples of such systems are: SPASS, E Theorem Prover, Vampire, iProver, and others. For a complete list of competing systems and the annual competition results please refer to the CASC website [Sutcliffe, 2017a].

All competing systems uses a syntax called TPTP (Thousands of Problems for Theorem Provers) [Sutcliffe, 2017b].

A TPTP problem is a set of formulae. A formula has the form:

$$language(name, role, formula)$$

- *language*: The languages currently supported are THF - Typed Higher-order Form, TFF - Typed First-order Form, FOF - (full) First-Order Form, and CNF - in Clause Normal Form.
- *name*: The name is a word starting lower case, e.g., original_f1, or a single quoted word, e.g., 'A crazy $ name'.
- *role*: The role is typically one of *axiom* for axioms, *conjecture* for conjectures, or *negated_conjecture* for negated conjectures. A full list of the possible roles is in the TPTP syntax [Sutcliffe, 2017b].
- *formula*: The logical formula uses a consistent and easily understood notation that can be seen in the Backus-Naur form (BNF) [Sutcliffe, 2017b].

In THF, TFF, and FOF the formulae are typically any number of axioms and one conjecture. In CNF the formulae are typically any number of axioms and one or more negated conjectures.

An example of a TPTP problem in the FOF language is given with its description at the heading (% is used for comments) in Listing 5.1.

5.5 Conclusion

In this chapter we investigated several FOL decidable fragments, and we further commented on FOL^2 which will be used in our work for encoding the problem of SPARQL query containment with ShEx constraints. We also presented the implementations available for

```
%------------------------------------------------------------------------
% All humen are created equal. John is a human. John got an F grade.
% There is someone (a human) who got an A grade. An A grade is not
% equal to an F grade. Grades are not human. Therefore there is a
% human other than John.

fof(all_created_equal,axiom,(
    ! [H1,H2] :
      ( ( human(H1)
        & human(H2) )
     => created_equal(H1,H2) ) )).

fof(john,axiom,(
    human(john) )).

fof(john_failed,axiom,(
    grade(john) = f )).

fof(someone_got_an_a,axiom,(
    ? [H] :
      ( human(H)
      & grade(H) = a ) )).

fof(distinct_grades,axiom,(
    a != f )).

fof(grades_not_human,axiom,(
    ! [G] : ~ human(grade(G)) )).

fof(someone_not_john,conjecture,(
    ? [H] :
      ( human(H)
      & H != john ) )).
%------------------------------------------------------------------------
```

Listing 5.1: TPTP problem in the FOF language. **Source:** `http://www.cs.miami.edu/~tptp/TPTP/QuickGuide/Problems.html`

FOL, and a syntax called TPTP that will be used in our work as well for experimenting our encoding methods.

6

Query Containment and Query Satisfiability

Contents

6.1 Introduction 46
6.2 Containment of the SPARQL OPT Fragment 47
6.3 Containment of SPARQL with RDFS Entailment 48
6.4 Conclusion 49

6.1 Introduction

Query containment is the problem of statically deciding, given two queries, whether the results of one of them is always contained in the results of the other. Query satisfiability on the other hand is the problem of deciding wether a single query might always return an empty result (for example if it contains a contradiction). Unsatisfiable queries are always contained in any arbitrary query. Query equivalence is yet another problem, that can also be reduced form query containment by checking if two queries are contained in each other.

All these query static analysis tasks are useful for query optimization. Figure 6.1 gives a basic example of benefiting from a containment relation between two queries, where Query 1 can be executed on a smaller set of data.

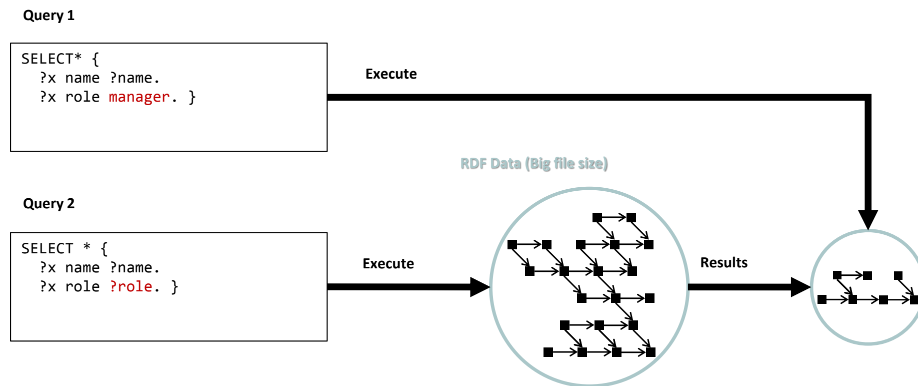

Figure 6.1: An example benefit for query containment

In [Colazzo and Sartiani, 2015], the authors proposed a schema language for edge-labeled data graphs (like RDFs), and then studied the satisfiability of 3 different classes of query languages (RPQs, NREs, and CRPQs) when such constraints are considered, but this study did not include containment. In [Genevès and Layaïda, 2006, Genevès and Layaïda, 2007] the authors studied static analysis aspects of XPath using μ-calculus and Monadic Second-order Logic respectively, then the authors provided in [Genevès and Layaïda, 2010] a tool related to their studies. XPath is a query language on tree structures.

The work in [Chekol et al., 2011] studied containment of PSPARQL, an extension of SPARQL 1.0 with paths and path constraints. In [Kostylev et al., 2015], the authors explored the complexity of containment and evaluation problems for fragments of SPARQL 1.1 property paths. The study in [Pichler and Skritek, 2014] provides complexity analysis for several fragments of SPARQL. Additionally, in [Letelier et al., 2012] the containment of well-designed OPT queries is investigated. None of these works consider schemas in the study of containment.

The works in [Chekol, 2016, Chekol et al., 2012a, Chekol et al., 2012b] study the containment problem with ontology languages and entailment regimes (SHI, RDFS, OWL...). Ontology languages put constraints on data, like schemas, but also allows for entailment of implicit data relations. The works on containment with ontology languages focus on

entailment regimes employed in these languages, but this study considers a respectively small fragment without OPT patterns.

Major results on query containment are summarised in Table 6.1. A study on the satisfiability of RPQs, NREs and GXPATH was done in [Colazzo and Sartiani, 2015] with PTIME complexity. On the study of query containment, one of the first studies on Conjunctive Queries (CQs) was done in [Chandra and Merlin, 1977] with complexity bounds proven to be NP-Complete. The containment of the the well-designed SPARQL fragment (with OPTs) was shown to be Π_2^P-Complete in [Letelier et al., 2012]. Containment of CQs and Unions of CQs in the presence of OWL DL can be done in EXPTIME without inverse and in 2EXPTIME with inverse in [Lutz, 2008]. Containment of SPARQL (AND and UNION fragment) in the presence of RDFS (`rdfs:subclass`, `rdfs:subproperty`, `rdfs:domain` and `rdfs:range`) was shown to be solved using μ-calculus (+nominals +converse) satisfiability in 2EXPTIME [Chekol et al., 2012b]. The RDFS fragment mentioned can be translated to the $\mathcal{SHI}$ Description Logic fragment.

Table 6.1: Query containment complexity results from the literature

Features	**Complexity**
- **Satisfiability only:** RPQs (sound and complete), NREs (sound), GXPath (sound). - New schema language, constraints on incoming/outgoing edges, polynomial equations.	PTIME
- CQs (conjunctive queries)	NP-Complete
- Well-designed OPTIONAL SPARQL	Π_2^P-Complete
- CQs, Unions of CQs. - OWL DL.	EXPTIME (no inverse) 2EXPTIME (with inverse)
- SPARQL (AND and UNION). - RDFS $\mathcal{SHI}$. - μ-calculus(+nominals+converse).	2EXPTIME

In the rest of this chapter we are going to present the containment procedure for the well-designed OPTIONAL SPARQL fragment [Letelier et al., 2012] (since this is the main fragment of our work in Chap. 8), and the containment procedure based on encoding to μ-calculus with OWL constraints [Chekol et al., 2012b] (since we are going also to use logical encodings, FOL encodings in our case, in Chap. 9).

6.2 Containment of the SPARQL OPT Fragment

In [Letelier et al., 2012], the authors studied the problem of SPARQL query containment from the fragment comprising the AND and OPT operators. Their procedure is based on the well-designed OPT patterns described in Section 3.4. The procedure is based on homomorphism between pattern trees, and the authors proved that their procedure is sound and complete. The same authors then showed that the problem for the considered fragment

is NP-complete and for the same fragment extended with external UNION is Π_2^P-complete [Pichler and Skritek, 2014].

We present the containment procedure in the following lemma taken from [Letelier et al., 2012]. The lemma provides the necessary and sufficient conditions for the deciding containment of a well-designed OPT SPARQL queries. The conditions are formulated in terms of pattern trees.

Lemma 6.2.1. *Consider two well-designed pattern trees $\mathcal{T}_1$ and $\mathcal{T}_2$ with roots r_1 and r_2, respectively. Then $\mathcal{T}_1$ is contained in $\mathcal{T}_2$, written as $\mathcal{T}_1 \sqsubseteq \mathcal{T}_2$, if and only if for every subtree $\mathcal{T}_1'$ of $\mathcal{T}_1$ rooted at r_1, there exists a subtree $\mathcal{T}_2'$ of $\mathcal{T}_2$ rooted at r_2 such that:*

1. *$vars(\mathcal{T}_1') \subseteq vars(\mathcal{T}_2')$, and*
2. *there exists a homomorphism from the triple patterns in $\mathcal{T}_2'$ to the triple patterns in $\mathcal{T}_1'$ that is the identity over $vars(\mathcal{T}_1')$.*

We notice that a pattern tree containment relation $\mathcal{T}_1 \sqsubseteq \mathcal{T}_2$ also yields the query containment corresponding to these pattern trees (lets say $q_1 \sqsubseteq q_2$) [Letelier et al., 2012].

6.3 Containment of SPARQL with RDFS Entailment

In [Chekol et al., 2012a] the authors proposed a procedure for deciding the containment of some classes of SPARQL queries with RDF Schema (RDFS) entailment using the modal μ-calculus.

The modal μ-calculus is a formal logic that extends the classical propositional modal logic with least fixpoint and greatest fixpoint. The syntax of a μ-calculus formula φ can be defined inductively as follows:

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

where p is an atomic proposition, X is a variable, a is a program (for navigation), and μ and ν are the least and greatest fixpoint operators respectively.

The μ-calculus is an expressive logic, and its usage for solving containment allows for expressing an expressive fragment of SPARQL (e.g. Property Path Patterns).

The SPARQL containment decision procedure in [Chekol et al., 2012a] is given by Theorem 6.3.1, where $q \sqsubseteq_{rdfs}^{S} q'$ means that q is contained in q' in the presence of a schema S under RDFS entailment regime.

Theorem 6.3.1 ([Chekol et al., 2012a])**.** *Given SPARQL queries q and q' and a schema S, $\Phi(S, q, q')$ is unsatisfiable if and only if $q \sqsubseteq_{rdfs}^{S} q'$.*

where $\Phi(S, q, q')$ is a μ-calculus formula comprising the encoding of the two queries ($\mathcal{A}(q)$ and $\mathcal{A}(q', m)$), the RDFS schema (Φ_S), the RDFS inference rules(Φ_R), and finally a restriction formula (φ_r). As a complete formula it is given as follows:

$$\Phi(S, q, q') = \mathcal{A}(q) \wedge \neg\mathcal{A}(q', m) \wedge \Phi_S \wedge \Phi_R \wedge \varphi_r$$

The authors also proved that their procedure is sound and complete.

6.4 Conclusion

In this chapter we introduced the problem of query containment which has been the subject of several previous studies in the literature. We presented two different containment procedures proposed in [Letelier et al., 2012] and [Chekol et al., 2012a]. The first study considers the well-designed OPTIONAL SPARQL fragment, while the second study does not consider OPTIONAL operators. The later however consider RDFS constraints and entailment rules. None of the studies available in the literature consider both OPTIONAL operators and typing constraints in the same time. This will be the topic of our contribution in Chap. 8 and 9.

7

Query Optimization

Contents

7.1 Introduction 52

7.2 Data Indexing and Query Planning (H_2RDF+) 53

7.3 Typed Data 55

7.4 Distributed RDF Store and Triple Pattern Ordering (SPARQLGX) 56

7.1 Introduction

Query optimization for the RDF data model has been studied with various approaches in the literature. Most existing works are based on scanning the data a priori and either saving new pieces of information about it, or providing alternative data representations. The works in [Goasdoué et al., 2013, Huang et al., 2011, Lee and Liu, 2013, Neumann and Weikum, 2008, Papailiou et al., 2013] are based on techniques that mainly focus on join optimizations by indexing the data. These works do not consider structured data and data typing. Another approach that also does not consider the query structure, yet provides optimized query processing, is vertical data partitioning [Abadi et al., 2007].

The works in [Fernández et al., 2013, Joshi et al., 2013, Pan et al., 2015, Pham et al., 2015] also provide query optimization techniques, by proposing new data representations that are more compact after scanning the data. Additionally, in [Zhang et al., 2013, Zou et al., 2011, Kim et al., 2017] the authors study the structure of the data and provide structural summaries or representative schemas. None of these works is based on a given schema, and thus they require an extensive data scan.

For works based on typed data, an approach was proposed in [Benzaken et al., 2013] that considers typed XML data trees. Unlike RDF, the tree data-type model of XML allows for extremely efficient subtree pruning. In [Aberer and Fischer, 1995] semantic query optimization for object-oriented databases is considered. In [Schmidt et al., 2010, Serfiotis et al., 2005] the authors consider query optimization for typed RDF graphs. These works are mainly oriented towards schema violations.

In [Graux et al., 2016a] the authors provide a SPARQL query evaluator, SPARQLGX, that relies on a translation of SPARQL queries into executable Spark code that adopts evaluation strategies according to the storage method used and statistics on data. Within the system, optimized joins are considered by reordering BGP triple patterns by combining those with common variables. Their approach scales better than the state-of-the-art systems they compare with (RYA, S2RDF, CliqueSquare, PigSPARQL, RDFHive, SDE) [Graux et al., 2016a].

In [Schmidt et al., 2010] the authors give Figure 7.1 with the evaluation complexity bounds for different fragments of SPARQL, where $\mathcal{A}$, $\mathcal{U}$, $\mathcal{O}$, $\mathcal{F}$ are the AND, UNION, OPT, and FILTER fragments respectively. π as a superscript means the corresponding fragment with projection, and the subscript define the number of variables where such constraint holds. This work is a milestone for any optimization efforts within these fragments.

In the rest of this chapter we are going to present two main query optimization techniques. The first is based on data indexing and query planning [Papailiou et al., 2013], because we will work on query planning for our own optimizations but based on ShEx constraints instead. The second work we will discuss is SPARQLGX [Graux et al., 2016a], over which we will implement our optimization technique.

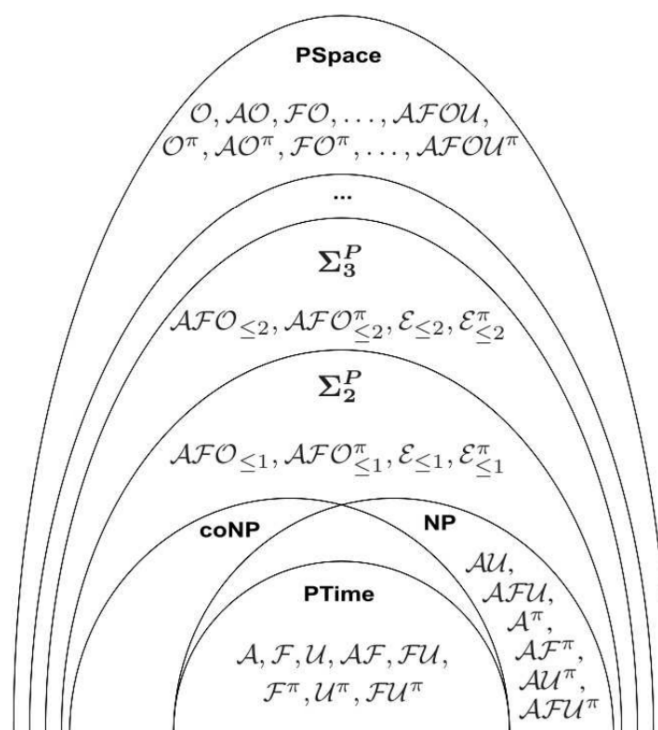

Figure 7.1: SPARQL query evaluation complexity summary

7.2 Data Indexing and Query Planning (H_2RDF+)

An example of a query evaluation systems that performs query planning based on data indexing is H_2RDF+ [Papailiou et al., 2013]. The main contribution of this system is to deal and scale well with substantially complex, non-selective joins in queries which normally result in exponential growth of execution times. H_2RDF+ utilizes distributed MapReduce processing and HBase [Foundation, 2007] indexes.

H_2RDF+ plans the execution order of the different joins so as to minimize the total query execution time. To find the optimal join order one have to consider the different combinations in which the joins can be performed, whose quantity grows exponentially to the number of joining variables, making the problem computationally expensive. H_2RDF+ uses a greedy, cost-based planner that chooses the join that will be executed in every step of the query.

There are two cost models used by H_2RDF+ depending on whether intermediate results exists or not. The two cost models are given in [Papailiou et al., 2013] as follows:

Merge join cost model: (no intermediate results)

$$MJcost(Q) = \sum_{i \in Q} ReadKeys(Q,i)/thr$$

$$ReadKeys(Q,i) = \min\{(\min_{j \in Q} n_j).o_i.SeekOverhead, n_i o_i\}$$

Sort-Merge join cost model: (with intermediate results)

$$SMJcost(Q, I) = (2\sum_{i\in I} n_i o_i + \sum_{i\in Q} ReadKeys(Q \cup I, i))/thr$$

where

- Q: input query.
- I: input intermediate results
- n_i: number of join variable's bindings for the ith query.
- o_i: average bindings of the non-joining variables corresponding to one join variable binding. Refers to the ith query.
- thr: the scan throughput.
- $SeekOverhead$: the seek overhead (6,400 key-values)
- $ReadKeys(Q, i)$: the number of key-values that will be read from the ith query.

H_2RDF+ tries to store intermediate data that maintain grouped results as much as possible. Figure 7.2 shows how grouped results are stored.

?university	?department	?student
Univ0	Dep0	St1
Univ0	Dep0	St2
Univ0	Dep0	St3
Univ0	Dep1	St1
Univ0	Dep1	St2
Univ0	Dep1	St3
Univ1	Dep2	St4
Univ1	Dep2	St5
Univ1	Dep2	St6
Univ1	Dep3	St4
Univ1	Dep3	St5
Univ1	Dep3	St6

Row Oriented Results

?university	Univ0
?department	Dep0, Dep1
?student	St1, St2, St3

?university	Univ1
?department	Dep2, Dep3
?student	St4, St5, St6

Grouped Results

Figure 7.2: Grouped intermediate results [Papailiou et al., 2013]

Figure 7.3 shows the join procedure on the grouped results. In the map phase, it splits the group according to the join variable, so it creates one group for each department. In the reduce phase groups of professors are retrieved from the index and are merged with the inputs to form the resulting outputs.

The indexing scheme used by H_2RDF+ HBase. HBase is a distributed, NoSQL key-value store that can handle large amounts of data. H_2RDF+ uses an HBase table for each index. It uses IDs instead of IRIs and long literals, and keep two separate HBase tables that work as dictionaries to translate string values to IDs and vice versa. The indexes generated by H_2RDF+ includes information about the count of combinations of each two terms occurring together as s-p, s-o, and o-p. It also includes the average of the count of the third term that occurs in RDF triples.

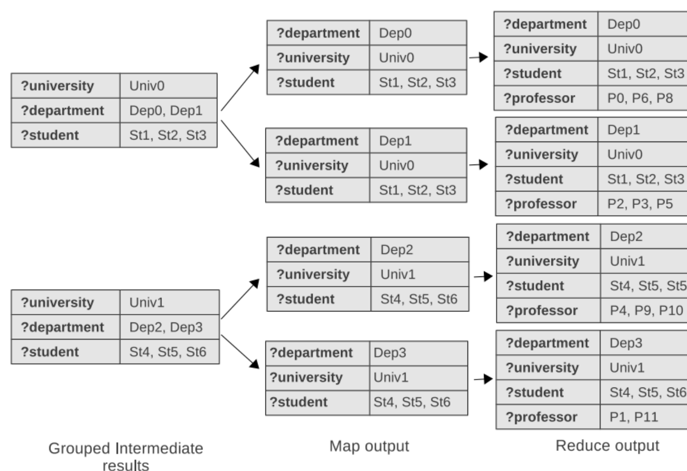

Figure 7.3: Join on grouped intermediate results [Papailiou et al., 2013]

7.3 Typed Data

An example of SPARQL query optimization with typed data is proposed in [Schmidt et al., 2010]. Typed data in other words mean data with constraints to which they comply. The study in [Schmidt et al., 2010] did not consider a specific schema or constraint language RDF data, rather it considered any constraint that can be expressed in First Order Logic (FOL).

The optimization proposed in the later study leads to the elimination of some query triple patterns based on static analysis. Elimination only occurs when such triple patterns does not affect the results according to the given FOL constraints.

An example given in [Schmidt et al., 2010] uses entailment rules of RDFS encoded in FOL, which we also present here. Consider the following two RDFS triples:

```
foaf:knows rdfs:domain  foaf:Person
foaf:knows rdfs:range  foaf:Person
```

Listing 7.1: RDFS Example

RDFS entailment rules that is applied for `rdfs:domain` and `rdfs:range` can be expressed in FOL as follows:

$$\varphi_d := \forall p, c, x, y(T(p, rdfs{:}domain, c), T(x, p, y) \rightarrow T(x, rdf{:}type, c))$$

$$\varphi_r := \forall p, c, x, y(T(p, rdfs{:}range, c), T(x, p, y) \rightarrow T(y, rdf : type, c))$$

Now consider the SPARQL query of Listing 7.2.

The optimization suggested for this query is by eliminating the 2nd and the 3rd triple patterns. The query of Listing 7.2 is logically equivalent to the query of Listing 7.3 and will return the same results according to the RDFS constraints.

```
SELECT * WHERE
{
  ?p1 foaf:knows ?p2 .
  ?p1 rdf:type foaf:Person .
  ?p2 rdf:type foaf:Person .
}
```

Listing 7.2: SPARQL query example

```
SELECT * WHERE
{
  ?p1 foaf:knows ?p2 .
}
```

Listing 7.3: SPARQL query example (optimized)

The optimization in this system is based on eliminations, similar to the elimination given in the previous example.

7.4 Distributed RDF Store and Triple Pattern Ordering (SPARQLGX)

SPARQLGX is a SPARQL query evaluation system that shows to be more efficient than other competing query evaluation systems for the GBP fragment on large-scale datasets. The optimization resulting from this system relies in the storage method used, the query translation, and data statistics collected [Graux et al., 2016a].

SPARQLGX stores RDF data in a distributed architecture based on Apache Spark [Zaharia et al., 2012]. It uses existing Hadoop [Foundation, 2014] infrastructures to evaluate SPARQL queries. Where there are many other query evaluation systems based on distributed datastores, the main optimization resulting from this system relies on the following:

- **Storage Model:** SPARQLGX takes advantage of a vertical partitioning architecture for RDF datasets, introduced by [Abadi et al., 2007]. This vertical partitioning is based on storing a triple (*spo*) in a file named *p* whose contents stores only the values of *s* and *o*. Thus RDF triples having a common predicate are stored in a common file. This method of storage takes advantage of the fact that often in RDF datasets the number of distinct predicates are few relative to the number of distinct subjects and objects. Therefore the storage model leads to a compression of data since the predicate column is removed, while the conversion to this model is straightforward and linear.
- **SPARQL query translation:** SPARQLGX compiles SPARQL conjunctive queries and generates Scala code that is directly executed by the Spark infrastructure (Spark Scala API). Spark is set up to use the Hadoop Distributed File System (HDFS). Given a conjunctive SPARQL query, SPARQLGX translates the first triple pattern then

searches for common a common variable in the other triple patterns to make a join. This allows to obtain optimizations based on join commutativity.

- **Data statistics:** SPARQLGX also collects data statistics that allows to define a selectivity rank for each triple pattern, and thus ordering triple patterns according to their selectivity values before they are translated.

Figure 7.4 presents the experimental results of SPARQLGX in comparison to other query evaluation systems. In the experimental setup described in [Graux et al., 2016b] a dataset of 109 Million RDF triples was used from the WatDiv benchmark its 20 different queries.

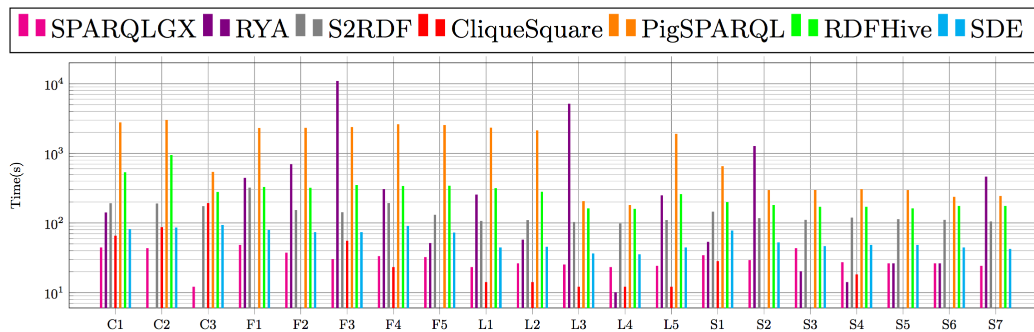

Figure 7.4: Query response times with WatDiv1k [Graux et al., 2016b]

Part II

Contribution I - SPARQL Query Containment with ShEx Constraints

8

Reducing SPARQL Query Containment with ShEx into ShEx Validation and SPARQL Containment

Contents

8.1 Introduction . . . 62

8.2 Containment Procedure Overview . . . 63

8.3 Query Transformation . . . 64

8.3.1 BGP Transformation . . . 64

8.3.2 AND-OPT Transformation . . . 65

8.3.3 AND-OPT-(UNION) Transformation . . . 67

8.4 Query Containment with ShEx . . . 67

8.5 Complexity . . . 69

8.5.1 SPARQL AND-(OPT)-(UNION) Fragments . . . 69

8.6 Implementation and Experimentation . . . 71

8.7 Conclusion . . . 72

8.1 Introduction

In this chapter we investigate the SPARQL query containment with ShEx constraints. Given two SPARQL queries, and a set of ShEx constraints, our purpose is to statically analyze such queries, namely determining the containment relation between them before being actually executed on the data.

For the fragments of SPARQL including OPTIONAL patterns, the containment of queries is normally investigated with the notion of subsumption because queries with OPTIONAL patterns may return results where not all the query variables are bound to values [Arenas and Pérez, 2011]. A solution mapping is a mapping from a set of variables to a set of values, thus designating an answer for a query. A solution mapping σ_1 is subsumed by another solution mapping σ_2 written as $\sigma_1 \sqsubseteq \sigma_2$ if all the variables of σ_1 are also in σ_2 and have the same mapping values. Given a set of mappings Ω_1 (resembling a SPARQL query solution), it is subsumed by another set of mappings Ω_2 written as $\Omega_1 \sqsubseteq \Omega_2$ if for every $\sigma_1 \in \Omega_1$ there exists $\sigma_2 \in \Omega_2$ such that $\sigma_1 \sqsubseteq \sigma_2$.

The consideration of ShEx constraints in query containment is important, because such constraints may affect the results of containment checking. For instance consider the following two SPARQL query graph patterns:

Q_1: `{?x :producer :p1 .  ?x :feature "feature1"}`
`OPT {?x :feature "feature2" .  ?x :expiryDate ?d}`
Q_2: `{?x :producer ?y .  ?x :feature "feature1"}`

Without constraints, no containment relation holds between these two queries. However, consider the following ShEx constraints defined for a $\langle Product \rangle$ node type:

```
<Product> {
:name xsd:string ,
:expiryDate xsd:date ?  ,
:producer @<Company> + ,
:feature xsd:string }
```

The previous ShEx shape definition means that a node of type “Product” should have a name of type string, optionally have an expiry date, have at least one producer which belongs to another ShEx shape $\langle Company \rangle$, and have exactly one feature of type string. Given that these ShEx constraints apply to the data, we can deduce that a containment relation $Q_1 \sqsubseteq Q_2$ holds between the two queries. This is due to the constraint that a “feature” predicate is allowed to occur only once, and thus in query Q_1 the right hand side of the optional pattern will never return results. In such case, we can deduce that the containment relation $Q_1 \sqsubseteq Q_2$ holds between the two queries.

There are several kinds of ShEx constraint violations that may lead to a new conclusion about the containment of two queries. These include (1) cardinality constraint violations, (2) basic data type constraint violations (like xsd:string, xsd:data ...), and (3) ShEx type definition violations (like @$\langle Company \rangle$ type).

Data on the web are getting larger, and distribution of data is getting more applicable. Different data sources are often being managed by different authorities. The need of schemas becomes increasingly necessary in order to manage the big amounts of data. While different sources in the same domain may share the same vocabulary, their constraints on data may vary. While these slight differences in data shapes may become a hassle for users to track individually, the use of OPT patterns in SPARQL provides a way to ask for constraints that are not necessarily applicable, and that is why the study of the optional fragment is particularly interesting.

In this chapter we define a sound and complete procedure for containment of several SPARQL query fragments in the presence of a ShEx schema, based on the usage of ShEx validators and query containment solvers that don't consider any schema constraints. We also study the complexity of the problem. The results vary from NPTIME-Complete to Π_2^P-Complete according to the fragment considered.

8.2 Containment Procedure Overview

The containment problem we are considering takes two input queries and an input ShEx schema.

In order to perform our containment, the procedure we propose requires a ShEx validator and query containment solver. The ShEx validator is used in order to transform the input queries, a process which also depends on an input ShEx schema. Parts of the queries are first transformed into RDF triples, and validated against the ShEx document. We use the validation results to transform the original queries by eliminating their invalid parts. The two new queries obtained are then given to the query containment solver in order to decide the containment result. The procedure is described in the diagram of Figure 8.1.

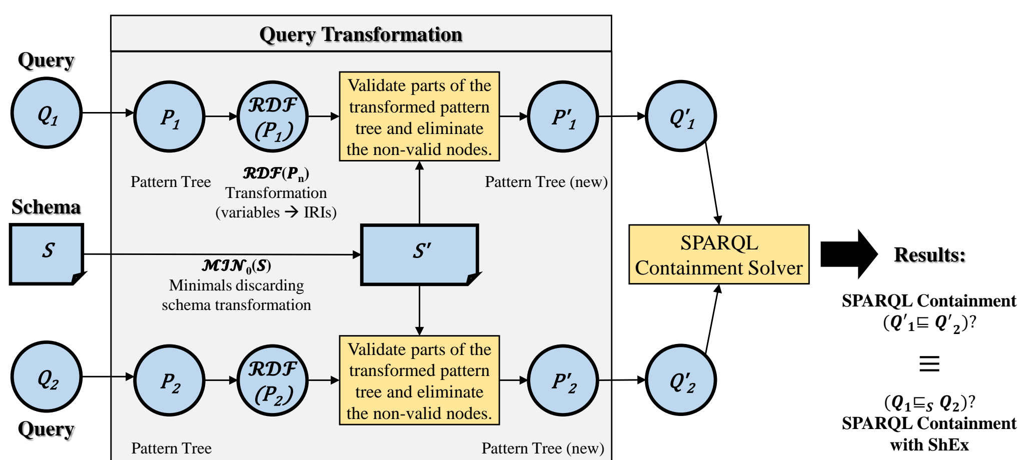

Figure 8.1: Containment Procedure Diagram

In Section 8.3 we describe the query transformation procedure. Then in Section 8.4 we explain how to use the transformed queries, and we prove that our method is sound and complete. In Sect. 8.5 we study the complexity of SPARQL containment with ShEx with respect to different SPARQL fragments up to AND-OPT-(UNION). In Sect. 8.6 we present our implementation of the procedure and the results of our experimentation.

8.3 Query Transformation

Query transformation under ShEx constraints is a process in which we rewrite a query, where the resulting query is equivalent to the original query given that the ShEx constraints hold on the data sets. Two queries are considered to be equivalent if they always give the same execution results.

The resulting query transformations defined in this section have several utilizations, namely for optimization purposes, especially that they are equivalent to and smaller than the original queries. We use them in this chapter particularly for defining containment in Section 8.4.

Before defining the transformation procedures, we give some preliminary definitions.

Definition 8.3.1. *Given a set of triple patterns P, $\mathcal{RDF}(P)$ is a function that yields a set of RDF triples by replacing each variable in P by a fresh IRI. The replacement is unique for each variable name.*

According to the previous definition, there always exists a homomorphism from the triples graph of P to the triples graph of $P' = \mathcal{RDF}(P)$. In fact, P' is an RDF data set that can be validated against a ShEx schema as will be seen later.

Definition 8.3.2. *Given two sets of RDF triples D_1 and D_2 and a ShEx schema S, we say that D_2 is a complement of D_1 w.r.t. S, if:*

1. $D_1 \subseteq D_2$
2. *D_2 is valid w.r.t. S*

Definition 8.3.3. *Given a ShEx schema S, the minimals discarding ShEx schema of S is given by the function $\mathcal{MIN}_0(S)$, and is defined by replacing all minimal cardinality constraints of S by zeros. (i.e. all cardinality constraints $[m, n]$, $+$ and 1 respectively, are replaced with $[0, n]$, $*$ and ? (optional) respectively).*

8.3.1 BGP Transformation

Query transformation of a BGP query is based on the RDF document validation. RDF validation against ShEx is defined with its NP-complete complexity in [Staworko et al., 2015].

Definition 8.3.4 (Query Transformation)**.** *For a BGP SPARQL query Q and a ShEx schema S, the query transformation function $\mathcal{T}_S$ is defined as follows:*

$$\mathcal{T}_S(Q) = \begin{cases} Q, & \textit{if } \mathcal{RDF}(Q) \textit{ is valid w.r.t. } \mathcal{MIN}_0(S) \\ \textit{empty query}, & \textit{otherwise} \end{cases}$$

The validation against $\mathcal{MIN}_0(S)$ is due to the fact that the query triples do not catch the complete data structure. Indeed, queries by nature are just partial representations of the constraints on the data that should be extracted.

8.3.2 AND-OPT Transformation

We extend the BGP transformation to a more interesting SPARQL fragment for our problem, the AND-OPT fragment. The results in this case will be a modified AND-OPT query that is equivalent to the original query, by applying two steps: (1) Eliminating non-valid OPT patterns, and (2) replacing some OPT operators with AND operators.

For the step (1), if we find out that some OPTIONAL patterns will never return results due to the ShEx constraints, the new query that results from this transformation is obtained by omitting these OPTIONAL patterns.

Consider the following SPARQL query:

Q: `{:p1 :producer ?y} OPT {:p1 :review ?z}`

and the following ShEx schema (a minimals discarding ShEx schema):

```
<product> {
:name xsd:string ?  ,
:expiryDate xsd:date ?  ,
:producer @<company> * ,
:feature xsd:string ?  }
```

We consider two triple sets for validation against the ShEx schema, `{:p1 :producer :y}` which is valid, and `{:p1 :producer :y.  :p1 :review :z}` - the optional graph pattern with its parent graph pattern - which is not valid. As a result of this validation step, we rewrite the query by removing the optional pattern which corresponds to the triple set which is not valid, and thus we get:

Q': `{:p1 :producer ?y}`

Query Q' is always equivalent to the original query Q for a dataset using the previous ShEx definition, due to the validation process we applied before eliminating non-valid parts.

In general, given a well-designed OPT pattern in normal form, we apply a validation step for each node of its pattern tree representation. The validation of a node considers its own triples, and all the triples of its ancestors up to the root. Now consider the pattern tree example of Fig. 8.2.

- If the BGP P_1 is not valid, then the whole query will never return results.
- If the BGP $(P_1 \cup P_{11})$ is not valid, then node P_{11} and all its descendants must be eliminated.
- If the BGP $(P_1 \cup P_{11} \cup P_{111})$ is not valid, then node P_{111} must be eliminated, and so on...

We notice that the union operator used above ($\cup$) is the set union operator, and should not be misinterpreted as the SPARQL's `UNION` operator. In the previous examples the union ($\cup$) of two sets of triple patterns (each is a BGP) is a bigger set of triple patterns (also a BGP). After eliminating non-valid OPT patterns, a new well-designed pattern tree will be obtained. The new resulting pattern tree resembles the graph pattern of the new query on which we proceed for the following steps of our procedure.

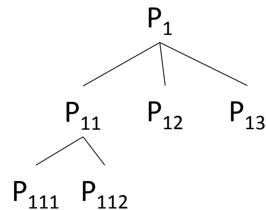

Figure 8.2: Pattern tree example (well-designed OPT pattern in normal form)

For step (2), we check if it is possible to replace some OPT operators with the AND operator. Considering the pattern tree representation of a query this operation can be described by uniting two directly connected nodes into one node, one of which is a child node, and the other is its parent node.

To show this by example, consider the following query:

Q: `{?x :name ?n} OPT {?x :phone ?p}`

and the following ShEx shape:

```
<Person> {
:name xsd:string ,
:phone xsd:string }
```

According to this ShEx shape definition, we know that :name and :phone will always occur together. Thus the right hand side of the OPT pattern will always occur with the left hand side of it. We therefore deduce that the previous query Q is equivalent to another query Q' without an OPT pattern.

Q': `{?x :name ?n. ?x :phone ?p}`

Two nodes in a query pattern tree must be merged into one node (the parent node), if and only if the triples of the child node will necessarily return results whenever the parent node returns results. We apply this check on every pair of parent-child nodes in the query pattern tree in order to get the final transformation of the query.

The transformations described in the latter examples for the AND-OPT SPARQL fragment are given formally in Definition 8.3.6.

Definition 8.3.5. *Given a pattern tree $\mathcal{P}$, and a node n of P, we define $\mathcal{R}_{\mathcal{P}}(n)$ to be the union of the set of triples of n and the set of triple of all its parent nodes up to the root node.*

Definition 8.3.6 (Query Transformation)**.** *For an AND-OPT SPARQL query Q, its pattern tree representation $\mathcal{P}$, and a ShEx schema S, the query transformation function $\mathcal{T}_S$ is defined by the following steps:*

1. *For each node n of $\mathcal{P}$, if $\mathcal{R}_{\mathcal{P}}(n)$ is not valid w.r.t. $\mathcal{MIN}_0(S)$, then eliminate n and all its descendants from $\mathcal{P}$. Let $\mathcal{P}'$ be the new pattern tree after the validation of all the nodes of $\mathcal{P}$.*

2. *For each pair of nodes n_1 and n_2 of $\mathcal{P}'$, such that n_1 is the parent of n_2, if it is necessary for every complement of $\mathcal{RDF}(n_1)$ to include the RDF triples of $\mathcal{RDF}(n_2)$ according to S, then merge n_1 and n_2 into one node. Let P'' be the new pattern tree obtained.*

We define $\mathcal{T}_S(\mathcal{P}) = \mathcal{P}''$.

8.3.3 AND-OPT-(UNION) Transformation

For the AND-OPT SPARQL fragment extended with UNION at the top level, the same procedure can be applied on each UNION pattern separately.

Consider a query pattern Q having the following structure:

$$Q = Q_1 \texttt{ UNION } Q_2 \texttt{ UNION } \dots \texttt{ UNION } Q_n$$

UNION at top level means that every Q_i in Q is a query pattern in the AND-OPT fragment. Now we define the query transformation of the AND-OPT-(UNION) SPARQL fragment.

Definition 8.3.7 (Query Transformation)**.** *For an AND-OPT-(UNION) SPARQL query pattern $Q = Q_1$ UNION Q_2 UNION $\dots$ UNION Q_n, the pattern tree representation $\mathcal{P}$ for Q, and $\mathcal{P}_i$ for each Q_i, and a ShEx schema S, the query transformation function $\mathcal{T}_S$ for $\mathcal{P}$ is defined as follows:*

$$\mathcal{T}_S(\mathcal{P}) = \mathcal{T}_S(\mathcal{P}_1) \texttt{ UNION } \mathcal{T}_S(\mathcal{P}_2) \texttt{ UNION } \dots \texttt{ UNION } \mathcal{T}_S(\mathcal{P}_n)$$

8.4 Query Containment with ShEx

In this section we show how SPARQL query containment with ShEx can be done by benefiting from the transformations of Sect. 8.3.

We first apply the transformation procedure on the two queries to be checked for containment based on a given ShEx schema. The resulting transformations are then checked for containment without considering the ShEx document using query containment solvers as the one proposed in [Pichler and Skritek, 2014]. If the containment of the query transformations holds, then the containment of the original queries with the consideration of ShEx holds.

Definition 8.4.1. *Given two queries q_1 and q_2, we define the relation $q_1 \sqsubseteq_S q_2$ to mean that $q_1 \sqsubseteq q_2$ holds in the presence of a ShEx schema S.*

Theorem 8.4.1. *Given two queries q_1 and q_2, and their corresponding transformations q_1' and q_2' according to a ShEx schema S, the containment relation $q_1 \sqsubseteq_S q_2$ holds if and only if $q_1' \sqsubseteq q_2'$ holds.*

Proof. The soundness of our procedure is evident from the fact that the transformations are equivalent to the original queries in the presence of the ShEx constraints. We use the empty schema as a transformation, or we eliminate parts of the queries only when we are sure that these parts will not return results according to the given schema.

For the completeness of the procedure, we prove it according to the corresponding fragment for each case.

1. For the BGP SPARQL fragment, assume we have two BGP queries q_1 and q_2 and their corresponding transformations q_1' and q_2' according to a ShEx schema S. For the completeness of the procedure, our purpose now is to prove that if $q_1' \not\sqsubseteq q_2'$, then $q_1 \not\sqsubseteq_S q_2$. For the case where q_1 is transformed into the empty query, $q_1' \sqsubseteq q_2'$ always holds, since the empty query is contained in every other query, and therefore the assumption condition can never happen. For the case where q_1' is not empty and q_2' is empty, that means that also q_2 will never return results due to a violation to the ShEx rules. No query can be contained in a query that does not return results except the empty query, and since we know that q_1 may return results due to the absence of any ShEx violation, then $q_1 \not\sqsubseteq_S q_2$ always holds. The final case is when both q_1' and q_2' are kept exactly the same as q_1 and q_2. In the latter case, if $q_1' \not\sqsubseteq q_2'$, then there exists no homomorphism from q_2' to q_1'. Given that the triples of q_1' don't violate the ShEx schema rules, then there exists a data set D which is a complement of $\mathcal{RDF}(q_1')$ w.r.t. S. BGP query solving is based on homomorphism from the set of query triple patterns to the set of RDF triples ([Seaborne and Harris, 2013]). A solution for q_1 necessarily exists in the proposed data set since the homomorphism exists by our proposal. Now we assume that the same solution also holds for q_2 and conclude a contradiction. If the same solution holds for q_2, then there exists a homomorphism from its triples patterns to D. Since all variables of q_1 are replaced with fresh IRIs, then a homomorphism is also necessary to hold from the triple patterns of q_2' to the triple patterns of q_1', and thus we conclude a contradiction because this homomorphism is a sufficient condition for deriving that the containment $q_1' \sqsubseteq q_2'$ holds (condition from [Pichler and Skritek, 2014]).

2. For the AND-OPT SPARQL fragment, assume we have two AND-OPT queries q_1 and q_2 and their corresponding transformations q_1' and q_2' according to a ShEx schema S. We show that for a transformation q' of any query q as proposed in Sect. 8.3, the

containment $q' \sqsubseteq q$ always holds. This follows from the fact that our transformation includes only elimination of optional parts of the query and transformation of other optional conditions into necessary conditions (transformation of OPT operators into AND operators). Both transformations make the query more restrictive in the meaning that they eliminate some solutions but never add solutions to the original query. Assume $q'_1 \not\sqsubseteq q'_2$, our purpose is to show that $q_1 \not\sqsubseteq_S q_2$. Since $q'_1 \not\sqsubseteq q'_2$, then for some subtree $\mathcal{T}'_1$ of q'_1, there doesn't exist a subtree of q'_2 with the homomorphism condition of Lemma 6.2.1. On the other hand, there exists a data set D which is a complement of $\mathcal{RDF}(\mathcal{T}'_1)$ w.r.t. S. A solution for q'_1 necessarily exists in D. If this solution is also a solution for q'_2, and thus for q_2, then a homomorphism must hold from $\mathcal{T}'_2$ of q'_2 to the D, and thus there exists a homomorphism from $\mathcal{T}'_2$ to $\mathcal{T}'_1$, that necessarily doesn't hold due to the fact that $q'_1 \not\sqsubseteq q'_2$, and therefore a contradiction is derived.

3. For the AND-OPT-(UNION) SPARQL fragment, the same proof holds as for the AND-OPT fragment, except that instead of proposing a complement data set that has a solution for q_1, leading to a contradiction when assuming it to have a solution for q_2, we alternatively propose multiple data sets, each corresponding to a top level UNION part of the query, and deriving a contradiction for each of the proposed data sets.

□

8.5 Complexity

In this section we study the complexity of SPARQL query containment with ShEx with respect to different SPARQL fragments.

We show that the complexity varies from NP-complete to Π_2^P-complete for the SPARQL fragments BGP, AND-OPT, and AND-OPT-(UNION). We also extend these fragments to include filter, property path patterns, and the MINUS operator whose containment problem is in the NEXP time complexity class, yet this is not shown to be a lower bound.

8.5.1 SPARQL AND-(OPT)-(UNION) Fragments

Theorem 8.5.1. *Containment with ShEx for the SPARQL BGP fragment is NP-complete.*

Proof. The complexity of containment of the SPARQL BGP fragment is NP-complete [Chandra and Merlin, 1977]. In the presence of ShEx constraints, a sufficient procedure to check containment is to first validate the BGP of each of the considered queries against the ShEx document. RDF validation against ShEx is NP-complete. An invalid query will return no results, and thus is contained in any other query. Otherwise, the normal containment procedure (without ShEx) is applied. Then the BGP fragment containment with ShEx is also in NP.

To prove the NP-hardness of the problem, it is sufficient to show that the containment with ShEx is at least as hard as containment without ShEx which is shown to be NP-complete for the considered fragment. A reduction from the containment problem to the containment with ShEx problem can be easily shown by assuming an empty schema. ☐

Theorem 8.5.2. *Containment with ShEx for the well-designed OPT SPARQL fragment is NP-complete.*

Proof. In [Pichler and Skritek, 2014], the authors studied the problem of containment of well-designed OPT SPARQL queries. The authors provide a procedure for solving the problem, and show the complexity of the problem to be NP-complete for this fragment.

The procedure we follow for deciding query containment of this SPARQL fragment with ShEx is based on both the query transformation described previously in this chapter, and the query containment procedures of [Pichler and Skritek, 2014]. Given two SPARQL queries in the well designed OPT fragment, their containment with ShEx can be decided by the two following steps:

1. Transform both queries. The results of these transformations are two new queries equivalent to the original queries respectively.
2. The two new resulting queries from the first step are used as an input of a general SPARQL containment solver (like the solver described in [Pichler and Skritek, 2014] for this fragment).

Validation of an RDF document against a ShEx document is NP-complete [Staworko et al., 2015]. Therefore, step (1) of the procedure is a series of ShEx validations each of which is in NP. The number of validations considered is polynomial since for a given query pattern tree, the validation occurs on all possible branches, rather than subtrees. While the number of subtrees is exponential in a pattern tree, the number of branches is polynomial. As each branch is validated independently, so the result of each branch validation doesn't affect the ones of other branches. Step (2), which is the query containment problem, is NP-complete for the well-designed OPT fragment. Thus the complexity of containment with ShEx is in NP for this fragment.

To prove the NP-hardness of the problem, it is sufficient to show that the containment with ShEx is at least as hard as the containment without ShEx which is shown to be NP-complete for the considered fragment. A reduction from the containment problem to the containment with ShEx problem can be easily shown by assuming an empty schema. ☐

Theorem 8.5.3. *Containment with ShEx for the well-designed OPT SPARQL fragment extended with top level UNION is* Π_2^P*-complete.*

Proof. In [Pichler and Skritek, 2014], the authors also studied the problem of containment of the AND-OPT-(UNION) fragment. The authors provide a procedure for solving the problem, and show the complexity of the problem to be Π_2^P-complete.

The procedure we follow to for deciding query containment of this fragment with ShEx is similar to the one followed for the AND-OPT fragment, except that in step (2) we use

the solver designed for the corresponding fragment. The usage of such solver will rise the complexity to Π_2^P.

To prove the Π_2^P-hardness of the problem, it is sufficient to show that the containment with ShEx is at least as hard as containment without ShEx which is shown to be Π_2^P-complete for the considered fragment. A reduction from the containment problem to the containment with ShEx problem can be easily shown by assuming an empty schema. □

Table 8.1 summarises our complexity results for the containment problem studied for the different fragments in this chapter.

Table 8.1: Containment complexity

SPARQL Fragment		Without(ShEx)	With(ShEx)
BGP	*[Chandra, 1977]*	NPTIME-Complete	NPTIME-Complete
AND-OPT	*[Pichler, 2014]*	NPTIME-Complete	NPTIME-Complete
AND-OPT-(UNION)	*[Pichler, 2014]*	Π_2^P-Complete	Π_2^P-Complete

8.6 Implementation and Experimentation

In order to implement the procedure described in this chapter for SPARQL containment with ShEx, we needed a ShEx validator and a containment solver:

- We used a ShEx validator provided by the authors of [Boneva et al., 2014].
- We used a containment solver for well-designed SPARQL queries (called SPAM tool) provided by the authors of [Pichler and Skritek, 2014].

The whole procedure, which implements the query transformation and the query containment, is combined into a common Java framework, while using Jena ARQ [The Apache Software Foundation, 2011] as a SPARQL parser.

For the testing of our containment tool, we prepared a test set of 5 pairs of different queries among which we want to check the containments. The containment is considered for each pair against the empty schema (equivalent to containment without schema), in addition to 4 prepared different ShEx schemas.

The queries used are a collection from the Berlin SPARQL Benchmark [Bizer and Schultz, 2009], where the 4 different schemas are constructed by us with varying type and cardinality constraints in order to study the effect of such constraints.

The queries are presented in the appendix Figure A.1 and the schemas used in our benchmark are presented in Figure A.2. The expected (valid) results are shown in Table 8.2.

The results with our implementation are shown in Table 8.3. The total run-time varies between 756 ms and 995 ms for the cases considered in the test. It is also obvious form the results that the majority of time is spent on the transformation of the queries.

Table 8.2: Experiment containment results

	No Schema	S1	S2	S3	S4
Q1a $\sqsubseteq$ Q1b	✓	✓	✓	✓	✓
Q2a $\sqsubseteq$ Q2b	×	×	×	×	✓
Q3a $\sqsubseteq$ Q3b	×	✓	×	×	×
Q4a $\sqsubseteq$ Q4b	×	×	✓	×	×
Q5a $\sqsubseteq$ Q5b	×	×	×	✓	×

8.7 Conclusion

In this chapter we studied the problem of SPARQL query containment with ShEx constraints, and the OPT patterns were shown to be particularly interesting for this study, due to their flexibility with constraints. We showed how transformation of queries can be done based on customized validation procedures. Then we proposed a procedure for the problem of containment with ShEx, and the complexity related to the AND-OPT SPARQL fragment was shown to be NP-complete, and that of the AND-OPT SPARQL fragment extended with external UNION to be Π_2^P-complete. With our proposed query transformation, no complexity

Table 8.3: Containment execution results with respect to different stages

	Cont. without Schema (ms)	Queries Transformation Time (ms)	Con. with Schema (ms)	Total (Valid+Con.) (ms)
QaQb1:S1	45	723	44	767
QaQb1:S2	45	714	42	756
QaQb1:S3	45	738	41	779
QaQb1:S4	45	724	42	766
QaQb2:S1	51	837	53	890
QaQb2:S2	51	851	52	903
QaQb2:S3	51	840	57	897
QaQb2:S4	51	870	53	923
QaQb3:S1	44	740	44	784
QaQb3:S2	44	725	43	768
QaQb3:S3	44	766	44	810
QaQb3:S4	44	754	44	798
QaQb4:S1	47	786	43	829
QaQb4:S2	47	736	0	736
QaQb4:S3	47	815	45	860
QaQb4:S4	47	861	45	906
QaQb5:S1	59	897	57	954
QaQb5:S2	59	913	57	970
QaQb5:S3	59	861	47	908
QaQb5:S4	59	942	53	995

addition over the original problem of containment is required. In the next chapter we show how query containment with ShEx can be encoded into an FOL satisfiability problem. The encoding of the problem with FOL allows for further extensions in the SPARQL fragment considered to include the FILTER, MINUS, and the property path patterns features.

9

Reducing SPARQL Query Containment with ShEx into FOL Satisfiability

Contents

9.1 Introduction **76**

9.2 ShEx Document Encoding **77**

9.3 Query Encodings **78**

9.3.1 Predicates as Variables 79

9.3.2 “Containing Query” Variables 80

9.4 Implementation and Experimentation **80**

9.5 Extensions **82**

9.5.1 Property Path Patterns 82

9.5.2 Filters 83

9.5.3 Minus 84

9.6 Conclusion **85**

9.1 Introduction

In this chapter we provide an alternative method for solving our SPARQL query containment with ShEx constraints. We encode the problem into an FOL formula and we check its satisfiability with existing automated theorem provers.

Encoding the problem into a logical formula allows the use of existing and highly optimized automated theorem provers. It also allows for extending the SPARQL fragment with additional features, which will be discussed in this chapter.

Since FOL satisfiability is not decidable in general, we propose an encoding based on FOL^2 (FOL with 2 variables only), which is a decidable fragment. After a survey done on the decidable fragments of FOL and presented in Chap. 5, we found that FOL^2 provides the sufficient conditions for encoding our problem.

The problem encoding is divided into two formulas. An axiom formula which resembles the encoding of the ShEx document constraints (ζ), and another conjecture formula expressing the problem of the containment of two well-designed OPT queries ($\mathcal{A}(Q_1) \wedge \neg\mathcal{A}(Q_2)$). These two formulas (the axiom and the conjecture) can be given to an automated theorem prover to check the validity of the whole problem. The procedure is described in the diagram of Figure 9.1.

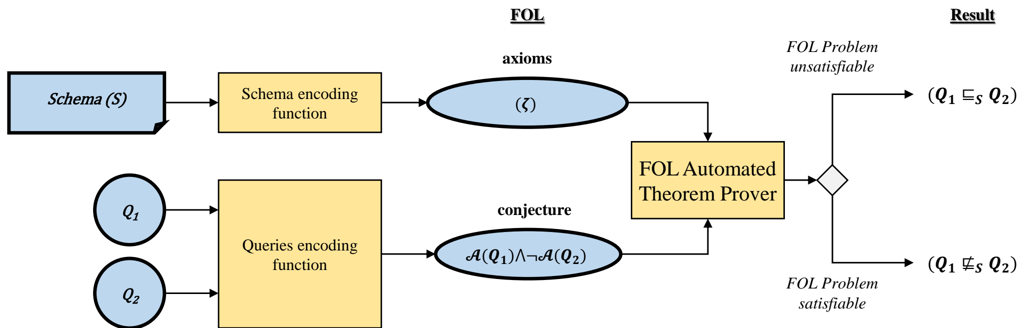

Figure 9.1: Containment Procedure Diagram by Encoding to FOL

We notice that if no input ShEx document is given, the set of axioms in our problem would be empty. The same procedure can be used without axioms (i.e. only the conjecture) to check the containment of queries without ShEx.

We explain the ShEx document encoding in Section 9.2, and the queries encoding in Section 9.3. In Section 9.4 we describe our implementation of this procedure and the test results obtained. In Section 9.5 we discuss the possible extensions in the SPARQL fragment within this containment procedure. Finally in Section 9.6 we give a conclusion that summarizes the results of this chapter and provides a comparison with the procedure of Chap. 8 where applicable.

9.2 ShEx Document Encoding

To start defining the encoding, we first define for each ShEx shape a primitive relation $S_i(x)$ meaning that the RDF term x has the shape S_i.

Let P_i be the set of predicates allowed in a shape S_i. Let R be a relation of arity 3 for expressing triple pattern relations (subject, predicate, object). Let O_p be a relation of arity 1, describing the type of restriction of the predicate p on the object (e.g. String, Date, IRIInstance . . .). For each shape S_i we define the following implication rule:

$$\forall x \Big(S_i\left(x\right) \rightarrow \left(\forall y \left(S_{Allowed}\left(x, y\right)\right) \wedge S_{Cardinality}\left(x\right)\right) \Big)$$

Where:

$$S_{Allowed}(x, y) = \bigwedge \{R(x, p, y) \rightarrow O_p(y) | p \in P_i\}$$

The cardinality formula depends on the form of the shape definition expression, and thus must be defined inductively taking into account the ShEx connectives that may occur.

$$S_{Cardinality} \equiv \delta(e)$$

Where $\delta(e)$ is the expansion of the shape expression e, defined recursivly as follows:

$$\begin{aligned}
\delta((p :: o)^{[m;n]}) &= \exists^{\leq n} y \; R(x, p, y) \\
\delta((p :: o)^{*}) &= \top \\
\delta(e_1 \mid\mid e_2) &\equiv \delta(e_1) \wedge \delta(e_2) \\
\delta(e_1 \mid e_2) &\equiv \delta(e_1) \vee \delta(e_2) \\
\delta((e_1 \mid\mid e_2)^{*}) &\equiv \delta((e_1)^{*}) \wedge \delta((e_2)^{*}) \\
\delta((e_1 \mid e_2)^{*}) &\equiv \delta((e_1)^{*}) \wedge \delta((e_2)^{*}) \\
\delta(((p :: o)^{[m;n]})^{*}) &\equiv \delta((p :: o)^{*})
\end{aligned}$$

Notice that we only care about maximal cardinality constraints, because if this number is exceeded in the query, we know that the query will not return results for this part of the query. While in the case of minimal cardinality constraints, it does not matter for the query (the constraints should apply on the RDF document, not the query).

We also notice that Kleene closure will be transmitted from complex shape expressions into the atomic symbols of the expression as shown in the definition of $\delta(e)$. Using this transmission has a semantical loss for the original expression. For example if we have $((p1 :: o1) \mid\mid (p2 :: o2))^{*}$, this signifies that the number of occurrences of $p1$ and $p2$ should

be the same. This piece of information is lost after the expansion of the expression by δ, yet this information should be discarded when dealing with queries, such constraint should apply on the RDF document complete data, not the query.

Now we define the FOL axiom expressing the ShEx constraints by the disjunction of all the shapes:

$$\zeta \equiv \forall x \Big(S_1(x) \vee S_2(x) \vee \cdots \vee S_n(x) \Big)$$

The encoding described previously is always within the FOL^2 fragment. There is no case where more than two variables are needed.

To show an example of the encoding in practice, consider the following ShEx schema having a single shape:

```
<Product> {
:name xsd:string ,
:expiryDate xsd:date ?  ,
:producer xsd:string * ,
:feature xsd:string }
```

This will be translated into the following FOL axiom:

$$\begin{gathered}
\forall x \ \texttt{Product}(x) \wedge \\
\texttt{Product}(x) \rightarrow \Big(\forall y \\
R(x, \texttt{name}, y) \rightarrow \texttt{string}(y) \wedge \\
R(x, \texttt{expiryDate}, y) \rightarrow \texttt{date}(y) \wedge \\
R(x, \texttt{producer}, y) \rightarrow \texttt{string}(y) \wedge \\
R(x, \texttt{feature}, y) \rightarrow \texttt{string}(y) \wedge \\
\Big(\exists^{\leq 1} y \ R(x, \texttt{name}, y) \Big) \wedge \\
\Big(\exists^{\leq 1} y \ R(x, \texttt{expiryDate}, y) \Big) \wedge \\
\Big(\exists^{\leq 1} y \ R(x, \texttt{feature}, y) \Big) \\
\Big)
\end{gathered}$$

9.3 Query Encodings

In [Letelier et al., 2012] the authors provide a method for well-designed OPT query containment based on homomorphism between the two pattern trees corresponding to the two

queries in question, and they provide a proof for the correctness of their method. In this section we formulate the homomorphism between the two pattern trees in the FOL logic, allowing us to use the formula as a conjecture in our problem to check the containment in the presence of the ShEx constraints axiom. The final containment conjecture formula, similar to the work in [Chekol, 2016], is given as the following combined encoding, where $\mathcal{A}(Q_1)$ is the encoding of the first query, and $\mathcal{A}(Q_2)$ is the encoding of the second query:

$$\mathcal{A}(Q_1) \wedge \neg\mathcal{A}(Q_2)$$

In the following, we define the encoding $\mathcal{A}(Q)$ of a query Q.

Given a query, each variable in it will be given an IRI referring to its name. This encoding will not use FOL variables, it is just a set of axioms. We define the encoding of a query $\mathcal{A}(Q)$ as follows, where P is a BGP, and each W_i is a well-designed OPT pattern:

$$\begin{aligned} \mathcal{A}(P) &= \bigwedge \{R(s,p,o) \mid \{s,p,o\} \in P\} \\ \mathcal{A}(P\ OPT\ W_1\ OPT \ldots OPT\ W_n) &\equiv \mathcal{A}(P) \vee (\mathcal{A}(P) \wedge (\mathcal{A}(W_1) \vee \ldots \vee \mathcal{A}(W_n))) \end{aligned}$$

The idea from the second formula of the encoding, i.e. for queries with OPT patterns, is to expand the pattern with $\vee$'s (disjunctions) at each level, because when used for containment checking with another query, it suffices that each branch of the first query pattern tree is contained in at least one branch of the second query pattern tree. The disjunction provides this condition at each level. The previous equation is actually recursive, and for deeper pattern trees more recursion and thus expansion will occur.

To show a simple example of the encoding in practice, consider the following SPARQL query:

Q_1: `{:p1 :producer ?y} OPT {:p1 :review ?z}`

This will be translated into the following FOL formula as follows:

$$R(\texttt{p1},\texttt{producer},\texttt{y}) \vee (R(\texttt{p1},\texttt{producer},\texttt{y}) \wedge R(\texttt{p1},\texttt{review},\texttt{z}))$$

The previous FOL formula represents $\mathcal{A}(Q_1)$. With another query Q_2 we can form our conjecture $\mathcal{A}(Q_1) \wedge \neg\mathcal{A}(Q_2)$ to check the containment $Q_1 \sqsubseteq Q_2$.

The query encoding described until now assumes that there are no variables in the predication position. We next explain how such variables can be adopted in our encoding.

9.3.1 Predicates as Variables

Normally, if a predicate variable is substituted by an IRI as followed for other variables, this IRI will not match to any of the predicates of the ShEx document, and thus will be treated as a violation of the ShEx rules.

This problem can be solved by substitution. For the set of predicate variables, we substitute them by all the possible combinations of the predicate IRIs that are mentioned in the ShEx document. This means that the containment test should be repeated several times, as much as there are different combinations. While it is possible to be expressed (by disjunctions), yet the size of the encoding will increase dramatically.

Without using substitution, expressing predicate variables as variables in the logic can serve our need except that there is a limitation by the FOL^2 fragment - it does not allow more than two variables. Some encodings in this case may give a formula out of the scope of FOL^2, and thus may make our problem undecidable.

Although several solutions may be adopted, we propose a restriction on the query form that keeps the encoding size with no expansion at all. The restriction states that the predicate variables may only appear in the predicate positions in the query. In this case, it would be possible to treat these variables as variables in FOL and existentially quantify them ($\exists p_1 \exists p_2 \exists p_3 \dots$). Although more than two predicate variables may occur in the query, yet the encoding still belongs to FOL^2 because these variables are not interacting in a single relation R, and thus can be separated into sets of triples each including only a single predicate variable.

Yet a less tight restriction can be used instead, which states that a predicate variable can be connected to at most two other predicate variables by the relation R. Connected to at most 2 other variables, this allows us to represent the connections as a chain, and thus make it possible to be expressed in FOL^2.

9.3.2 "Containing Query" Variables

By containing query, we mean the query on the right hand side of the containment symbol ($Q_{left} \sqsubseteq Q_{right}$). The variables of Q_{right} which also appear in Q_{left} will be substituted by IRIs corresponding to them. For other variables of Q_{right} we substitute them by all the possible combinations from the IRIs of Q_{left}.

To reduce the number of combinations to be considered, we limit the substitutes of a variable relative to the position it occurs on (subject, predicate, or object position).

9.4 Implementation and Experimentation

In order to implement the procedure described in this chapter for SPARQL containment with ShEx, we need to provide an FOL encoding for the queries and the schemas:

- We used Jena ARQ as a query parser in Java.

- We write our query encoding as a problem conjecture in the FOF TPTP syntax, which is an FOL syntax accepted by the majority of automated theorem provers.
- For the ShEx constraints document encoding into a TPTP axiom set, we use a ShEx syntax expressed in JSON, where there are several tools available for converting from/to ShEx JSON syntax to/from ShEx compact syntax [Prud'hommeaux, 2017a]. (FOF TPTP encoding of our test schemas are given in Chap. A)
- The resulting TPTP axioms and conjecture are then given to an automated theorem prover. (In our experiments we use two different automated theorem provers, namely SPASS [Weidenbach et al., 2009] and E Theorem Prover [Schulz, 2013], to validate our encodings in practice)

For the testing of our containment tool which is based on FOL encoding, we used the same set of query pairs and schema that are used in the experiment of Chap. 8 Sect. 8.6.

Tables 9.1, 9.2, 9.3, and 9.4 show our results with the tool built for the procedure with FOL theorem provers.

Table 9.1: Queries encoding time

	Queries encoding time (ms)
QaQb1	367
QaQb2	371
QaQb3	366
QaQb4	368
QaQb5	368

Table 9.2: Schemas encoding time

	Schemas encoding time (ms)
Schema 1	25
Schema 2	25
Schema 2	25
Schema 2	25

Table 9.3: Formula execution time on SPASS

	Formula execution time on SPASS (ms)
QaQb1	$\approx$ 10 (with all schemas)
QaQb2	$\approx$ 20 (with all schemas)
QaQb3	$\approx$ 10 (S0, S1) $\approx$ 20 (with other schemas)
QaQb4	$\approx$ 20 (with all schemas)
QaQb5	$\approx$ 20 (S0) $\approx$ 30 (with other schemas)

The results shows that a better performance is achieved for the cases considered with the FOL implementation than with the former implementation (Chap. 8). We notice that although the theoretical execution of the second method is NEXPTIME-Complete, the Satisfiability problem of the FOL formula, yet in practice we get better execution time by almost half the execution time compared to the other method which is theoretically in Π_2^P. We expect

Table 9.4: Formula execution time on E Theorem Prover

	Formula execution time on E Theorem Prover (ms)				
	No Schema	**S1**	**S2**	**S3**	**S4**
QaQb1	7	8	8	8	8
QaQb2	10	12	11	12	12
QaQb3	9	10	10	10	10
QaQb4	11	12	12	13	13
QaQb5	15	18	17	18	17

such results due to the high optimization techniques used in the automated theorem provers, and due to the fact that realistic queries (even relatively big queries) are small with respect to the problem considered. Validation of a whole set of data in contrast with FOL may be tedious to the huge amount of data deployed in such cases, while by nature, queries are relatively concise. It is also clear from the previous results that most of the time is spent on parsing the queries and encoding them, and not in the automated theorem prover itself.

9.5 Extensions

In this section we are going to study the potentials of extending our SPARQL fragment for our containment problem. The method based on FOL provides better expressivity, than the former solving method, for dealing with the new fragments.

We study the extension of 3 features separately: Property Path Patterns from SPARQL 1.1, Filter operator, and Minus operator.

9.5.1 Property Path Patterns

In order to adopt property path patterns into our system, we are going to modify the proposed query encoding. The modification will take place only on the first rule of the query encoding. Given a BGP P the encoding becomes as follows:

$$
\begin{aligned}
\mathcal{A}(P) &\equiv \bigwedge \{\mathcal{A}(\{s,p,o\}) \mid \{s,p,o\} \in P\} \\
\mathcal{A}(\{s,p,o\}) &\equiv R(s,p,o) \text{ , if } p \text{ is simple} \\
&\equiv \mathcal{A}(\{s,p_1,o\}) \vee \mathcal{A}(\{s,p_2,o\}) \text{ , if } p = p_1|p_2 \\
&\equiv \exists z\ \mathcal{A}(\{s,p_1,z\}) \wedge \mathcal{A}(\{z,p_2,o\}) \text{ , if } p = p_1/p_2 \\
&\equiv \mathcal{A}(\{o,q,s\}) \text{ , if } p = \hat{}q \\
&\equiv \neg\mathcal{A}(\{s,q,o\}) \text{ , if } p = !q \\
&\equiv \mathcal{A}(\{s,q,o\}) \wedge \\
&\quad\ \mathcal{A}(\{s,q,o\}) \rightarrow (s = o) \vee \Big(\exists z\ \mathcal{A}(\{s,q,z\}) \wedge \bar{\mathcal{A}}(\{z,q,o\})\Big) \text{ , if } p = q^*
\end{aligned}
$$

Where we define the encoding $\bar{\mathcal{A}}$ similarly, but by using another relation $\bar{R}$ (which corresponds to closure patterns) as follows:

$$
\begin{aligned}
\bar{\mathcal{A}}(\{s,p,o\}) &\equiv \bar{R}(s,p,o) \text{ , if } p \text{ is simple} \\
&\equiv \bar{\mathcal{A}}(\{s,p_1,o\}) \vee \bar{\mathcal{A}}(\{s,p_2,o\}) \text{ , if } p = p_1|p_2 \\
&\equiv \exists z\ \bar{\mathcal{A}}(\{s,p_1,z\}) \wedge \bar{\mathcal{A}}(\{z,p_2,o\}) \text{ , if } p = p_1/p_2 \\
&\equiv \bar{\mathcal{A}}(\{o,q,s\}) \text{ , if } p = \hat{}q \\
&\equiv \neg\bar{\mathcal{A}}(\{s,q,o\}) \text{ , if } p = !q \\
&\equiv \bar{\mathcal{A}}(\{s,q,o\}) \wedge \\
&\quad\ \bar{\mathcal{A}}(\{s,q,o\}) \rightarrow (s=o) \vee \Big(\exists z\ \mathcal{A}(\{s,q,z\}) \wedge \bar{\mathcal{A}}(\{z,q,o\})\Big) \text{ , if } p = q^*
\end{aligned}
$$

9.5.2 Filters

In [Zhang et al., 2016] the satisfiability of SPARQL queries with filters was studied. Consider the following unsatisfiable SPARQL query pattern:

$((?x,a,?y)\ UNION\ (?x,b,?z))\ FILTER\ (bound(?y)\ \&\ bound(?z))$

We know that such example is unsatisfiable due to the inappropriate usage of the FILTER. This result will also affect the results of containment study, and for this reason we study the fragments with FILTER operations.

For some FILTER constructs, the authors showed the satisfiability of SPARQL is undecidable. We know from our study of containment that deciding satisfiability is necessary for deciding containment. This can be shown by giving a query that is undecidable for satisfiability, and another query that we know is unsatisfiable. In this case deciding the satisfiability of the first query is important for us to know if it is contained in the second query, otherwise our containment procedure is undecidable.

In the following table we present the FILTER fragments that we will consider, by keeping the decidability results from satisfiability. For example, in the table below, Filter($bound, =, \neq_c$) means the fragment of SPARQL with filters only allowing the constructs *bound*, $=$ (variables equality), and $\neq_c$ (variable inequality to constants).

Filter fragment	**Decidable**	**FOL^2 encoding complexity**
Filter($bound, =, \neq_c$)	yes	PTIME
Filter($bound, \neq, \neq_c$)	yes	PTIME
Filter($\neg bound$)	no	$\times$
Filter($=_c$)	no	$\times$
Filter($=, \neq$)	no	$\times$

Now we explain how we can extend our encoding for SPARQL containment to adopt filters using the FOL method. We first associate to every pattern P a set $\Gamma(P)$ of sets of constraints. ($\mathcal{V}$ is the set of variables)

- If P is a triple pattern (s, p, o), then $\Gamma(P) := \{\{s, p, o\} \cap \mathcal{V}\}$
- $\Gamma(P_1\ UNION\ P_2) := \Gamma(P_1) \cup \Gamma(P_2)$
- $\Gamma(P_1\ AND\ P_2) := \{S_1 \cup S_2 \mid S_1 \in \Gamma(P_1) \text{ and } S_2 \in \Gamma(P_2)\}$
- $\Gamma(P_1\ OPT\ P_2) := \Gamma(P_1\ AND\ P_2) \cup \Gamma(P_1)$
- $\Gamma(P_1\ FILTER\ C) := \{S' = \delta(S, C) \mid S \in \Gamma(P_1)\}$, where $\delta(S, C)$ is defined as follows:
  - $\delta(S, C) = \delta(S, C_1) \cup \delta(S, C_2)$, if $C = C_1 \& C_2$
  - $\delta(S, C) = \delta(S, C_1), \delta(S, C_2)$, if $C = C_1 | C_2$
  - $\delta(S, C) = equal(x, c)$, if C is $?x = c$ and $?x \in S$
  - $\delta(S, C) = \neg equal(x, c)$, if C is $?x \neq c$ and $?x \in S$
  - $\delta(S, C) = equal(x, y)$, if C is $?x =?y$ and $?x, ?y \in S$
  - $\delta(S, C) = \neg equal(x, y)$, if C is $?x \neq?y$ and $?x, ?y \in S$
  - $\delta(S, C) = equal(x, any)$, if C is $bound(?x)$ and $?x \in S$
  - Finally: $\delta(S, C) = \delta(S, C) \cup \{equal(x, any)\ |?x \text{ is not in } C \text{ and } ?x \in S\}$

The final step is applied even if a filter condition does not exist on a studied pattern. A FILTER with the empty condition C_{empty} is considered in this case.

Now the results of the previous procedure are sets whose members will be considered as relations in the FOL encoding (example: $equal(x, y)$). We present the new query encodings based on these associated sets as follows:

$$\mathcal{A}(P) = \bigwedge \{R(s, p, o) \mid \{s, p, o\} \in P\} \wedge (\bigvee \{\alpha(S) \mid S \in \Gamma(P\ FILTER\ C_{empty})\})$$
$$\text{where: } \alpha(S) = \bigwedge \{c \mid c \in S\}$$
$$\mathcal{A}(P\ OPT\ P_1\ OPT..OPT\ P_n) \equiv \mathcal{A}(P) \vee (\mathcal{A}(P) \wedge (\mathcal{A}(P_1) \vee ... \vee \mathcal{A}(P_n)))$$

9.5.3 Minus

In this section we write down a straightforward method for dealing with the $MINUS$ operator of SPARQL. The $MINUS$ operator is semantically described as a negation based on testing whether a pattern exists in the data, given the bindings already determined by the query pattern. Starting from our first simplest encoding, we can adopt this operator in the FOL query encoding as follows:

$$
\begin{aligned}
\mathcal{A}(P) &= \bigwedge\{R(s,p,o) \mid \{s,p,o\} \in P\} \\
\mathcal{A}(P_1\ MINUS\ P_2) &\equiv \mathcal{A}(P_1) \wedge \neg\mathcal{A}(P_2) \\
\mathcal{A}(P\ OPT\ P_1\ OPT..OPT\ P_n) &\equiv \mathcal{A}(P) \vee (\mathcal{A}(P) \wedge (\mathcal{A}(P_1) \vee ... \vee \mathcal{A}(P_n)))
\end{aligned}
$$

9.6 Conclusion

In this chapter we showed how the SPARQL containment problem can be solved by generating a corresponding logical formula, using the FOL^2 fragment, and checking its validity.

In a wider comparison, we combine the results of this chapter with the previous chapter (Chap. 8). We developed two radically different approaches for solving the problem and we evaluated them. The first approach relies on the joint use of a ShEx validator and a tool for checking query containment without constraints. In a second approach, we show how the problem can be solved by a reduction to a fragment of first-order-logic with two variables. This alternative approach allows to take advantage of any of the many existing FOL theorem provers in this context. According to the evaluation of how the two approaches compare experimentally, it was shown that the FOL implementation has a better practical performance for the cases considered although it has a higher theoretical complexity.

In addition, with the FOL satisfiability approach, it was possible to adopt extensions (FILTER, MINUS, and property path patterns) that are not supported in the other procedure. Table 9.5 gives an upper bound complexity summary of the results of containment procedures from Chap. 8 and In Chap. 9, and the supported fragments within each.

Table 9.5: Comparing the upper bound complexities of the two different containment methods for the supported fragments

SPARQL Fragment	Method from Chap. 8 **Without/With(ShEx)**	Method from Chap. 9 (FOL) **Without/With(ShEx)**
BGP	NP	NEXP
AND-OPT	NP	NEXP
AND-OPT-(UNION)	Π_2^P	NEXP
AND-OPT-(UNION)-Minus	× [not supported]	NEXP
AND-OPT-(UNION)-FILTER	× [not supported]	NEXP
AND-OPT-(UNION)-PP	× [not supported]	NEXP
AND-OPT-(UNION)-MINUS-FILTER-PP	× [not supported]	NEXP

Part III

Contribution II - Optimising SPARQL Evaluation with ShEx

10

Selectivity Estimation for SPARQL Triple Patterns

Contents

10.1 Introduction **90**
10.2 Definitions **91**
10.2.1 Abstract Syntax of the Considered ShEx Fragment 91
10.2.2 Preliminary Definitions 92
10.3 Well-Formed Data-Schema Pairs **92**
10.3.1 Cardinality Constraints 94
10.3.2 Shape Distinction 94
10.3.3 Data Nodes Isolation 95
10.4 Shape Relation Graph **95**
10.5 Ranking **96**
10.5.1 Hierarchical Relations between ShEx Shapes 97
10.5.2 Predicate Distributions Among ShEx Shapes 99
10.5.3 SPARQL Query Triple Rankings 100
10.6 Evaluation **100**
10.6.1 Experiment 1: With Web Index Schema 101
10.6.2 Experiment 2: With LDBC SNB Schema and gMark Queries 103
10.7 Conclusion **106**

10.1 Introduction

In this chapter we investigate how the evaluation of SPARQL queries [Seaborne and Harris, 2013] can be optimized in the presence of ShEx constraints. We propose a method for optimizing the order of evaluation of subqueries, by taking advantage of the information on the data described in ShEx. While SPARQL query optimization by static analysis is important and well-studied, the emergence of constraint languages (such as ShEx) raises new questions on how the knowledge of additional constraints can be effectively leveraged as a part of the static analysis and optimization. In this work, we focus on the logical query structure and in particular on subquery ordering that can be automatically inferred from a set of data constraints. More specifically, we consider SPARQL basic graph patterns (BGPs), and we focus on the order of execution of triple patterns by ranking them, which aims to minimize the overall execution cost of the query.

We postulate that ShEx constraints contain useful information for selecting the order of execution of triple patterns. Optimization opportunities arise from the presence of joins between query triple patterns, and common variables. In several situations, the order of execution of triple patterns can be rearranged so that the size of intermediate results for join variables are minimized. Consider the arbitrary query example of Listing 10.1 with 3 triple patterns and a join on the variable $?x$.

```
1 SELECT ?x WHERE {
2  ?x :p1000 :a .
3  ?x :p700 :b .
4  ?x :p1 :c .
5 }
```

Listing 10.1: SPARQL query example

Assume that we know that the triple with predicate $:p1000$ will return 1000 values for $?x$, that of $:p700$ will return 700, and that of $:p1$ will return 1. The join between the first two triple patterns may give up to 700 values which should be reserved in memory for another join with the third triple. A wise choice in this case is to reorder the triple patterns execution, knowing that the third triple is more selective than the other two triples. Executing the third triple first will guarantee that at most one value will be reserved in the memory for the next join. Such an order will provide an optimized execution of the whole query.

Hence, our main purpose in this work is to infer better execution orders for triple patterns in more general queries, based on the knowledge extracted from a ShEx document. We first define a set of well formed ShEx schemas, that possess interesting characteristics for deciding optimal execution orders. We then define our optimization method by exploiting information extracted from a ShEx schema. To the best of our knowledge, this is the first work addressing SPARQL query optimization based on ShEx. We implemented our procedure on the top of SPARQLGX, which is one of the most efficient engine for distributed SPARQL evaluation and known to outperform many competitors in the field [Graux et al., 2016a]. SPARQLGX already implements various query optimization techniques including reordering

triple patterns [Graux et al., 2016a], but without considering schema constraints. We show that our technique further improves the efficiency of query execution times. With large amounts of data, the optimization may exceed 85% for queries that are highly affected by the triple patterns execution order, and by 25% on average.

The rest of the chapter is organised as follows: In Sect. 10.2 we introduce some preliminaries necessary for understanding the rest of the paper. In Sect. 10.3 we define well-formation rules for a data-schema pair that are of interest for our optimization process. In Sect. 10.4 we define a graph representation of a ShEx schema. In Sect. 10.5 we formally define our optimization process based on ranking triple patterns. In Sect. 10.6 we report on experimental results with our optimization technique. Finally, we conclude in Sect. 10.7.

10.2 Definitions

10.2.1 Abstract Syntax of the Considered ShEx Fragment

In this section we define a variation of the abstract syntax of ShEx. The definition presented here is similar to the abstract syntax defined in Chap. 4 Sect. 4.2, except that here a shape expression e is defined inductively in terms of e^+, and then e^* is defined as a syntactic sugar, while in the former definition the inverse was adopted (i.e. e was defined in terms of e^*, and then e^+ was defined as a syntactic sugar). The purpose of adopting this variation in this chapter is to allow us to design Def. 10.2.1 and Def. 10.2.2 in terms of the semantics of e^+, which would be otherwise complicated to define in terms of the semantics of e^*. Thus we recall the definition of the abstract syntax with the adopted variation.

Given a finite set of edge labels Σ and a finite set of types Γ, we define a shape expression e over $\Sigma \times \Gamma$ as follows:

$$e ::= \epsilon \mid \Sigma \times \Gamma \mid e^+ \mid (e|e') \mid (e\|e')$$

where "$|$" denotes disjunction, "$||$" denotes unordered concatenation, and " $+$ " denotes repetition for a positive number of times. From this definition we also further define the following operators as macros:

- $e^? := (\epsilon \mid e)$ (optional)
- $e^* := (\epsilon \mid e^+)$ (unordered Kleene star)
- $e^{[m;n]}$ (e repeated i times with i in the interval from m to n)

which are also parts of the ShEx syntax. In the sequel we write $a :: t$ as a shorthand for $(a, t) \in \Sigma \times \Gamma$.

A shape expression schema (ShEx), or simply schema, is a tuple $S = (\Sigma, \Gamma, \delta)$, where Σ is a finite set of edge labels, Γ is a finite set of types, and δ is a type definition function that

maps elements of Γ to shape expressions e over $\Sigma \times \Gamma$. If the δ is not defined for some type $t \in \Gamma$, the default definition is $\delta(t) = \epsilon$.

10.2.2 Preliminary Definitions

In accordance with the abstract syntax, we define the following shape expression inclusion relation which we use in the rest of the chapter.

Definition 10.2.1. *Given a shape expression e, an edge label p and a ShEx shape s. The inclusion relation $(p, s) \in e$ is defined inductively as follows:*

- $(p, s) \in e$ *if* $e = (p, s)$
- $(p, s) \in e$ *if* $e = e_1^+$ *and* $(p, s) \in e_1$
- $(p, s) \in e$ *if* $e = e_1 \mid e_2$*, and* $(p, s) \in e_1$ *or* $(p, s) \in e_2$
- $(p, s) \in e$ *if* $e = e_1 \mid\mid e_2$*, and* $(p, s) \in e_1$ *or* $(p, s) \in e_2$

We also define the following shape expression optional condition.

Definition 10.2.2. *Given a shape expression e, an edge label p and a ShEx shape s, we say that the atomic shape expression (p, s) is optional in e, written as $(p, s) \in_{opt} e$ if:*

- $e = e_1 \mid e_2$ *and* $(p, s) \notin e_1$ *and* $(p, s) \in e_2$*, OR*
- $e = e_1 \mid e_2$ *and* $(p, s) \in e_1$ *and* $(p, s) \notin e_2$*, OR*
- $e = e_1 \mid e_2$ *and* $(p, s) \in_{opt} e_1$ *or* $(p, s) \in_{opt} e_2$*, OR*
- $e = e_1 \mid\mid e_2$ *and* $(p, s) \in_{opt} e_1$ *and* $(p, s) \in_{opt} e_2$*, OR*
- $e = e_1^+$ *and* $(p, s) \in_{opt} e_1$

10.3 Well-Formed Data-Schema Pairs

Representing data using RDF, as well as with other semantic web technologies or database systems, is a design issue. The same pieces of information may be represented in different ways according to the designer choices. In addition, given a definite data graph representation, different point of views may be raised on how it is constrained. Thus different ShEx schemas, all of which correctly describe the same data graph, may be suggested as the schemas constraining them. Accordingly, we introduce a notion of well-formed data-schema pairs. The set of rules that will be introduced on the data-schema pairs in this section will help us to better identify efficient SPARQL query designs, by static analysis of the schema.

The rules for well-formation warantee that the necessary information needed for our ranking can be deduced from the ShEx schema, yet the ranking procedure is not deterministic. For

some shapes, the relations attached to them are not indicative for the selectivity of those shapes. We also define in this section the schema formation rules that make our shape ranking procedure deterministic.

Definition 10.3.1 (Well-Formed Data-Schema Pair)**.** *A data-schema pair* (G, S) *is well-formed if and only if the following rules hold.*

1. ***Cardinality rule:*** *Every m-to-n relation between two schema shapes in* S*, where* $m > n$ *or* m *is not bound, is modeled from the* m*-sided shape to the* n*-sided shape.*
2. ***Shape distinction rule:*** *For every 4 schema shapes* $s_1, s_2, so_1, so_2 \in S$ *(not necessarily distinct), and for every 2 predicates* p_a *and* p_b *(not necessarily distinct),* so_1 *and* so_2 *are distinct if the following conditions hold:*
   - $(p_a, so_1) \in \delta(s_1)$
   - $(p_b, so_2) \in \delta(s_2)$
   - O_1 *is the set of nodes of* G *which occur as objects of* p_a *whose subject belongs to the shape* s_1
   - O_2 *is the set of nodes of* G *which occur as objects of* p_b *whose subject belongs to the shape* s_2
   - $O_1 \cap O_2 = \emptyset$

The well-formation rules impose a schema design that gives preference to some constraints among others, all of which are respected in the data-schema pair, but it does not force new constraints to be added. Well-formed data-schema pairs provide the maximal set of desired information that can be inferred from the ranking procedure described in Sect. 10.5, without adding new constraints.

Although the well-formation rules are sufficient for optimization of real life data-schema examples, the ranking system is not totally deterministic. A restrictive set of rules on a data-schema pair that make our ranking system deterministic are given in the following definition.

Definition 10.3.2 (Ranking-Deterministic Data-Schema Pair)**.** *A data-schema pair* (G, S) *is ranking-deterministic if and only if the following rules hold.*

1. ***Well-formedness:*** (G, S) *is well-formed.*
2. ***Cardinality rule:*** *There is no closure cardinality (+,*) in* S*.*
3. ***Shape distinction rule:*** *For every 3 shapes* $s_o, s_1, s_2 \in S$ *(not necessarily distinct), if there exist 2 predicates* p_a *and* p_b *(not necessarily distinct) where* $(p_a, s_o) \in \delta(s_1)$ *and* $(p_b, s_o) \in \delta(s_2)$*, then* s_1 *and* s_2 *refers to the same shape, and* p_a *and* p_b *refers to the same predicate.*
4. ***Data nodes isolation rule:*** *For every data IRI instance* d*, every 2 shapes* $s_1, s_2 \in S$*, and every predicate* p*, if* $(p, s_2) \in \delta(s_1)$ *and* d *belongs to the shape* s_2*, then there exists a data IRI instance* d' *such that the RDF triple* $\langle d', p, d\rangle \in G$*.*

In the following subsections (10.3.1, 10.3.2, and 10.3.3), we give examples and additional descriptions of the well-formedness and ranking-deterministic rules, aiding to understand how they contribute to our ranking procedure.

10.3.1 Cardinality Constraints

Example 10.3.1. *Assume we want to model a schema describing the relation between students and schools. If we know that the relation in the data between schools and students will be 1-to-many, then the following two schema examples are legitimate, but only the first one is well-formed w.r.t. the data.*

Schema proposition 1: (Well-Formed)

```
<Student> { :name xsd:string , :school @<School> }
<School> { :name xsd:string }
```

Schema proposition 2: (Not Well-Formed)

```
<Student> { :name xsd:string }
<School> { :name xsd:string , :student @<Student> * }
```

As it is evident from Example 10.3.1, the *well-formation cardinality rule* tries to avoid the usage of positive and Kleene closures ($+$, $*$). Formally, the semantics of the two proposed schemas are different. Schema proposition 1 is more restrictive. Schema proposition 2 misses the restriction that a student should belong to 1 and only 1 school, although it is still an acceptable schema even if this restriction is inherent in the data.

Indeed, the *well-formation cardinality rule* helps us to determine the relative quantity of shape occurrences in the data. For example Schema proposition 1 allows us to know that the $\langle Student \rangle$ instances definitely occur in the data more than $\langle School \rangle$ instances.

10.3.2 Shape Distinction

A shape in a ShEx schema can be as general as allowing any node in any RDF graph to belong to it. The more the shape has restrictions, the more it describes a specific type of nodes. The *well-formation shape distinction rule* puts restrictions on shapes that seem to be too general that they surely miss expressing some constraints that are inherent in the data.

Example 10.3.2. *Assume we want to model a schema describing the relation between students and researchers to their corresponding schools and research companies. Knowing that schools are not research companies, then the following two schema examples are legitimate, but only the first one is well formed w.r.t. the data.*

Schema proposition 1: (Well Formed)

```
<Student> { :name xsd:string , :school @<School> }
<Researcher> { :name xsd:string , :company @<Company> }
```

Schema proposition 2: (Not Well Formed)
```
<Student> { :name xsd:string , :school @<Establishment> }
<Researcher> { :name xsd:string , :company @<Establishment> }
```

In Example 10.3.2, Schema proposition 2 will not allow us to determine the relative quantity of $\langle Student \rangle$ instances to those of $\langle Establishment \rangle$ instances in the data, while with Schema proposition 1 we are sure that the quantity of $\langle Student \rangle$ instances are more than that of $\langle School \rangle$ instances (and similarly between $\langle Researcher \rangle$ and $\langle Company \rangle$).

10.3.3 Data Nodes Isolation

The *data nodes isolation rule* for deterministic ranking states that a data instance shall not be isolated from other data instances unless isolation is required by the given schema.

Example 10.3.3. *Assume we have a schema describing the relation between students and schools, and a set of data as described below.*
Schema proposition:
```
<Student> { :name xsd:string , :school @<School> }
<School> { :name xsd:string }
```
RDF data:
```
:schoolA :name "School example isolated"
:schoolB :name "School example unisolated"
:student1 :name "Alice"
:student2 :name "Bob"
:student3 :name "Alain"
:student4 :name "Marion"
:student1 :school :schoolB
:student2 :school :schoolB
:student3 :school :schoolB
:student4 :school :schoolB
```

In Example 10.3.3, `:schoolA` is isolated, and thus the data-schema pair are not *ranking-deterministic*. If we have many isolated nodes (let us say many isolated shcools), the number of schools in the data may be more than the number of students, which renders our ranking procedure unuseful, since it does not reflect the correct relative frequencies of the data.

10.4 Shape Relation Graph

In this section we define a shape graph representation that we use to assign ranks to shapes in Sect. 10.5. A shape relation graph is a graphical representation focusing only on the relations existing between shapes in a ShEx document, discarding cardinalities.

Definition 10.4.1 (Shape Relation Graph)**.** *Given a ShEx document S, we define a shape relation graph $G = \mathcal{SRG}(S)$ as a tuple (N, E) of set of nodes N, each corresponding to a ShEx shape, and a labelled directed relation E between nodes such that:*

- *$E(n_1, x, n_2)$ defines an edge from n_1 to n_2 labelled with x.*
- *Given any two nodes $n_1, n_2 \in N$, and any predicate p, then $E(n_1, p, n_2)$ if and only if $(p, n_2) \in \delta(n_1)$ and $(p, n_2) \notin_{opt} \delta(n_1)$*
- *Given any two nodes $n_1, n_2 \in N$, and any predicate p, then $E(n_1, p^{opt}, n_2)$ if and only if $(p, n_2) \in_{opt} \delta(n_1)$*

Figure B.1 (appendix) shows the *shape relation graph* of a real life schema used in our experimentation (Section 10.6). User-defined types are shown as ovals while built-in types (like xsd:string) and IRIs are shown as rectangles. For visualization reasons, we also replicate some type nodes in the mentioned figure.

We define the set of root nodes of a *shape relation graph.*

Definition 10.4.2. *Given a shape relation graph $G(N, E)$, we define $\mathcal{R}(G)$ as the set of all root nodes of G. A node $s \in N$ is considered a root node if and only if it has no incoming edges in G.*

We also define the set of cycles of a *shape relation graph.*

Definition 10.4.3. *Given a shape relation graph $G(N, E)$, we define $\mathcal{C}(G)$ as the set of all cycles in G. A cycle is a subgraph of shape relation graph. A subgraph $C(N_C, E_C)$ of G is a cycle if and only if the set of edges E_C defines a directed path that starts and ends with the same node $n \in N$, and N_C is the set of all nodes that can be visited by the set of edges E_C, where $|N_C| = |E_C|$.*

10.5 Ranking

In order to decide the order of execution of query triple patterns, we assign them ranks inferred from the analysis of the ShEx document. These ranks are based on two main concepts: 1) The hierarchical relations between ShEx shapes, and 2) The predicate distributions among ShEx shapes.

The first concept gives rankings to shapes, and the second concept gives ranking to predicates. The ranking of query triple patterns is based on the product of the two rankings.

10.5.1 Hierarchical Relations between ShEx Shapes

In ShEx, the definition of a shape may be based on other shapes defined in the same schema. This notion, called shape inclusion, is explicitly represented by the edges of the *shape relation graph* defined in Sect. 10.4. Such edge relations between shapes allow us to infer information about the relative frequency of data corresponding to these shapes.

Consider Schema proposition 1 in of Example 10.3.1. Representing it as a *shape relation graph*, $\langle School \rangle$ is a child of $\langle Student \rangle$. Each student in the data should have exactly one registered school, and multiple students may be registered in the same school according to the schema. Such a relation between shapes allows us to know that a student instance occurs more in the database than a school instance. Actually the number of schools is at most equivalent to the number of students, where this is a worst case assumption - each student has a unique school. It is evident that this is an extreme case that should not be considered as an average distribution. Thus, it is important to study the hierarchical relations between ShEx shapes. In the example we give the $\langle School \rangle$ shape a priority ranking, since we know that they occur less than the $\langle Student \rangle$ shape, and thus rendering variables corresponding to it more selective.

Concerning cardinality, we notice that a higher cardinality is independent on the actual number of data instances of a shape. For example, if we have `:registeredIn @<School> {1,3}` instead of `:registeredIn @<School>`, that does not necessarily mean an increase in the number of schools; the same set of schools may apply in both cases. For the ranking system it is sufficient to consider the relation structure rather than the structure and cardinalities together, and that is why we ignore explicit cardinalities of edges in the *shape relation graph* defined in Sect. 10.4.

The ranking procedure we propose starts from the root shapes (root nodes as defined in Definition 10.4.2). A root shape will have a ranking of 1. Going down through the descendant shapes from the root shape the ranking increases. If there are two (or more) incoming edges to a shape, the lower ranking is transferred. A problem in such a procedure is when there is a cycle between shapes in the graph representation of the schema, that means that the ranking will propagate forever. In such case there is no preference for any of the shapes in the cycles, and all of them must have the same ranking. In some cases, a cycle has an optional relation(s) within it, given by the cardinalities “?", “*", or “{0,n}". In such case, we know that a cut in the cycle can only occur at these points. For asserting the strength of normal relations against such optional relations, the preference for ranking is to actually cut the cycles at these points and apply the ranking system by avoiding such kind of cycles.

Now we formally define all the procedures described.

10.5.1.1 Schema Graph Adjustment

First, given the *shape relation graph* G of a ShEx schema, we modify it by detecting optional relations and cycles.

1. For each cycle $C_i(N_i, E_i) \in \mathcal{C}(G)$:
   - For all predicate p, if there exist nodes $n_1, n_2 \in N_i$ such that there exists an edge $E(n_1, p^{opt}, n_2)$, then remove this edge. Let the new resulting graph be G_{nor}.
2. For each cycle $C_i(N_i, E_i) \in \mathcal{C}(G_{nor})$:
   - Merge all the nodes $x \in N_i$ into a single node c_i.

10.5.1.2 Schema Shapes Ranking

Now let the output of the *Schema Graph Adjustment* be $G_{adj}(N, E)$. We define the ranking function $\delta_S(x)$ for all $x \in N$ as follows:

1. For each node $r \in \mathcal{R}(G_{adj})$, $\delta_S(r) = 1$
2. For each node $r \in \mathcal{R}(G_{adj})$, apply the procedure $P(r)$ defined next.

Given a node $x \in N$, the procedure $P(x)$ is defined as follows:

- For each $s \in N$, and for each predicate p where there exists an edge $E(x, p, s)$
  1. If $\delta_S(s)$ is not initialised: $\delta_S(s) = \delta_S(x) + 1$
  2. If $\delta_S(x) + 1 < \delta_S(s)$, then $\delta_S(s) = \delta_S(x) + 1$
  3. Apply $P(s)$

Finally, we transmit the cycle rankings to the original nodes.

- For each cycle $C_i(N_i, E_i)$ in G_{nor}:
  - For each node $x \in N_i$, $\delta_S(x) = \delta_S(c_i)$

10.5.2 Predicate Distributions Among ShEx Shapes

In the previous section we ranked shapes according to their relative frequency of occurrences based on relations between them. Such a ranking is not sufficient for deciding rankings of triple patterns in a query since such ranking is also affected by the uniqueness versus globality of predicates within shapes.

Given a predicate p used in the shapes s_1, s_2, ..., s_n. The ranking of p within a shape s, denoted as $\delta_P(p, s)$ is defined as follows.

$$\begin{aligned}
\delta_P(p,s) &= \frac{\delta_S(s)}{\delta_S(s_1) \times \delta_S(s_2) \times \cdots \times \delta_S(s_n)}\ \text{, if } s \in \{s_1, s_2, \ldots, s_n\} \\
\delta_P(p,s) &= \frac{1}{\delta_S(s_1) \times \delta_S(s_2) \times \cdots \times \delta_S(s_n)}\ \text{, otherwise}
\end{aligned}$$

The previous formula works by reducing the ranking of a predicate when it is more global, i.e. when it is used with more shapes. With more shapes the factors in the denominator will increase and thus reducing the overall ranking. Such predicates are frequent, they are used every where, and this means there will be a large set of nodes in the database associated with this predicate. The ranking system tends to leave such predicates to be executed lastly, and that is why the modelled function reduces its ranking. On the other hand, if a predicate is unique for a certain shape, its ranking tends to be bigger by reducing the number of denominators to only one, which is the shape it corresponds to.

We notice that if a predicate p corresponds to only one shape s_m, then the ranking corresponding to it will be always 1, where this value represents the highest ranking possible.

$$\delta_P(p, s_m) = \frac{\delta_S(s_m)}{\delta_S(s_m)} = 1$$

On the other hand, the lowest possible ranking is when the predicate p is used globally in all the shapes defined in the ShEx document, and particularly when the shape considered for the current ranking is a root node, which have the lowest possible shape ranking of 1, and the denominator is the largest possible which is the product of all the shape rankings.

$$\delta_P(p, s_{root}) = \frac{1}{\prod_{\forall i, \text{ where } s_i \text{ is a shape}} \delta_S(s_i)}$$

10.5.3 SPARQL Query Triple Rankings

Now our purpose is to rank the triple patterns given a BGP query. Triple patterns with higher ranking will be executed first. Before ranking triples, we need to validate the BGP against the ShEx document, and for each subject in the triple patterns the ShEx validator will decide to which shapes this subject may belong. A subject may belong to multiple shapes at the same time. Thus, for each subject s occurring in the triple patterns we have a set $C(s)$ of candidate shapes for s. For convenience, given a triple pattern t, we define $C(t) = C(s)$ if s is a subject of t. We also define $p(t)$ as the predicate of the triple t.

To define the triple ranking function, we use the two ranking functions δ_S and δ_P defined previously.

Given a BGP B and a triple pattern $t \in B$, we define the ranking of the triple t w.r.t. a ShEx schema, denoted by $\delta_T(t)$, as follows:

$$\delta_T(t) = Avg\Big[\delta_S(S_i) \times \delta_P(p(t), S_i)\Big]_{\forall i,\ S_i \in C(t)}$$

For a given triple t, the previous function is the average of the product $\delta_S \times \delta_P$ by considering all the possible candidate shapes for the subject of t.

10.6 Evaluation

We prepared experiments with different setups that show the advantageous effects of the optimization procedure described in this chapter. Both experiments use real ShEx schema examples (Web Index [World Wide Web Foundation, 2014] and LDCB SNB [Erling et al., 2015] respectively) and a benchmark data generator for each of them.

In the first experiment, the queries are designed by us to utilize different combinations of the schema types. In addition to showing the results of this experiment, the main purpose of hand-crafting queries is to allow readers to investigate the queries in a clear way, where the variable names in the queries are indicative to the type they belong to.

In the second experiment, the queries are generated by a specific benchmarking tool (gMark [Bagan et al., 2017]). This setup is more realistic and convincing concerning the obtained results. Queries generated by the gMark tool are of different structures and sizes, and thus are suitable for benchmarking purposes.

10.6.1 Experiment 1: With Web Index Schema

In this experiment we generate data according to a real example ShEx schema, using a generator called wiGen (or WebIndex Data Generator) [Gayo et al., 2014]. We make our own set of 12 queries which are of different sizes, and designed to utilise all schema shapes defined in the ShEx schema of wiGen.

10.6.1.1 Generated data

wiGen is a generator of random data that can be used to benchmark RDF schema languages. The data model used by it is inspired by the WebIndex data, which are data intended as a measure of the World Wide Web's contribution to social, economic and political progress in countries across the world [World Wide Web Foundation, 2014]. The ShEx schema utilised by wiGen is represented as a *shape relation graph* in Fig. B.1.

The generator allows the user to define the number of instances needed for each shape in the schema. We generated 15M nodes, and the resulting RDF dataset is about 80M RDF triples.

10.6.1.2 Queries

The set of queries we use for this experimentation is shown in Fig. B.2. The variable names in the queries are hand-crafted to indicate a schema type from the schema of Fig. B.1 (?d: DataSet, ?s: Slice, ?o: Observation, ?c: Country, ?comp: Computation, ?i: Indicator, ?org: Organization). The focus of our study is on BGP queries, and thus all of them are within this fragment. The queries of Fig. B.2 are given in their optimized form i.e. with their order of triple patterns computed by our ranking system. Our purpose is to compare each query with counter part queries, which are just equivalent to the original ones, with different order of their triple patterns. The number of permutations for each query depends on the number of triples. For 4 triple patterns there are 24 different permutations, while for 7 triple patterns there are 5040. In our experiment we generate all the permutations if there are less than 50 of them, and otherwise we generate 50 random permutations.

10.6.1.3 SPARQLGX

An advantage of our optimization technique is that it can be applied on the top of query systems like SPARQLGX [Graux et al., 2016a]. This system in turns is based on SPARK coding with Hadoop underlying infrastructure for evaluating SPARQL queries [Zaharia et al., 2012]. SPARQLGX is known to outperform many competitors in the field concerning conjunctive

queries [Graux et al., 2016a]. In our experiment we show how the application of our technique further decreases the average run time for SPARQLGX in the presence of a ShEx document.

SPARQLGX, in the current state, has a basic triple pattern ordering strategy that is based on grouping triple patterns with common join variables together. This ordering is not deterministic for a set of triple patterns; it also depends on their initial ordering given by the input SPARQL query. In our experiments we show that this ordering itself is important for obtaining improved results, yet we show that using our ranking strategy based on the ShEx information further improves the results.

We define 3 systems that are included in our experiment as follows:

- **S1:** Is SPARQLGX with its ordering strategy turned off (the system itself provides a configuration that stops reordering triple patterns and keeps the original ordering of the query triple patterns).
- **S2:** Is SPARQLGX with its ordering strategy turned on.
- **Optimized:** Is an extension of SPARQLGX. It extends it with the application of our ordering strategy based on the ShEx information.

10.6.1.4 Results

The results of our query evaluations are presented in Figure 10.1 which is explained as follows:

- **The blue area** is the runtime range of system **S2** concerning the different permutations of each test query.
- **avg(S2)** marks the average runtime of all the permutations of each test query with system **S2**.
- **avg(S1)** marks the average runtime of all the permutations of each test query with system **S1**.
- Finally, **the green bars** shows the runtime of each input query with our **optimized system** that has a single deterministic triple patterns ordering for each test query.

For each query run we set an evaluation timeout of 200 seconds. Some queries times out, and the query run time is considered 200 seconds for calculating the average. In our given chart if the average is shown to be 80 seconds (the top of the chart), then this means it is $\geq$ 80 seconds.

We notice that we don't show the runtime range of the queries with system **S1** (as done with system **S2**) since there are always input permutations that times out for the considered test queries, and thus we only show the average for this system.

The average of the improvement of query executions by our system (**Optimized**) compared to **avg(S2)** ranged between 3.7% and 18%. The mean improvement of all test queries

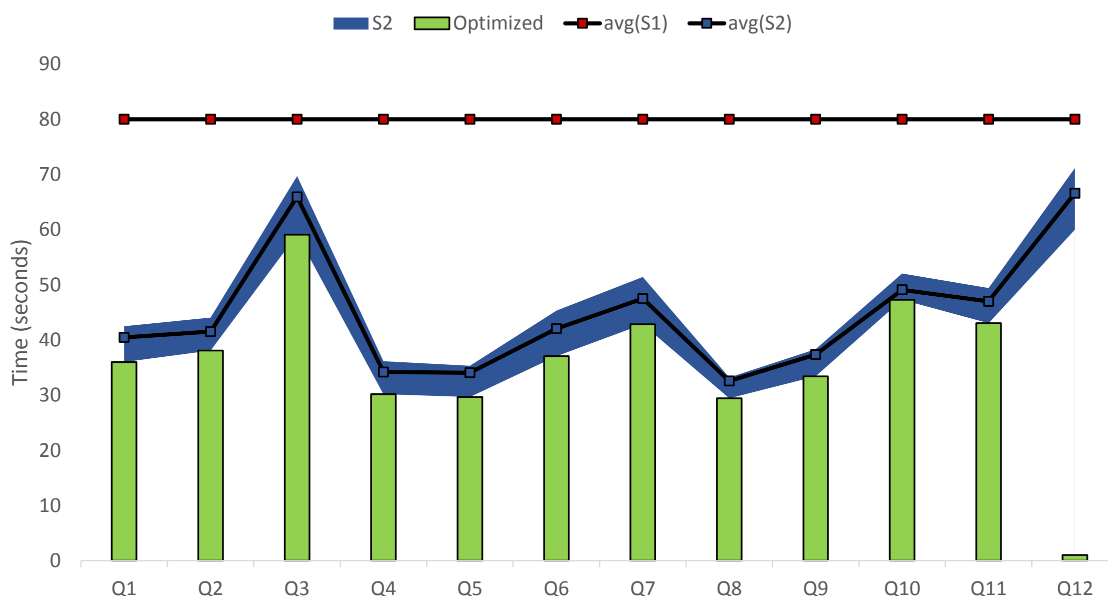

Figure 10.1: Comparing ranking-optimized query evaluation with other systems (WebIndex data)

is 9.8%. We notice that in this calculation we did not consider query Q12 which has a theoretical 98% improvement average. The purpose of Q12 was to show that if the query is unsatisfiable according to the ShEx schema (it violates it), then we do not execute it and the time considered is the validation time of the query. In Q12 the last triple pattern has `foaf:homePage` in the predicate position and the string `"homepageOrg988"` in the object position, while the object of `foaf:homePage` should be an IRI according the ShEx schema.

10.6.2 Experiment 2: With LDBC SNB Schema and gMark Queries

In this experiment we generate data according to the Social Network Benchmark (SNB) schema of the Linked Data Benchmark Council (LDBC). The data and queries are generated by gMark [Bagan et al., 2017]. gMark is a graph and query workload generator based on an input schema. Technically the setup is comparable to that of Experiment 1 except that in Experiment 2 queries are generated by the benchmarking tool rather than being hand-crafted. In addition Experiment 2 is applied on 4 different dataset of different sizes, thus we show 4 different charts corresponding to the datasets, and therefore allowing to further comment on the effect of data size.

10.6.2.1 Generated data

Using the gMark tool, we generated 4 datasets, all according to the LDBC SNB schema (check the schema in Figure B.3 and the corresponding well-formed shape relation graph in Figur B.4). The tool allows users to define the dataset size by indicating the number

of nodes to be generated, in our case 5M, 30M, 50M, and 100M nodes scenarios are used, corresponding to about 11M, 67M, 113M, and 227M RDF triples respectively.

10.6.2.2 Queries

Using gMark, we also generated a set of 12 SPARQL queries based on the LDBC SNB schema. We setup the sizes of queries such that in each query there are between 6 and 10 triple patterns, and there are between 4 and 6 distinguished variables. The choice of the query size is to allow for structures to form within the schema hierarchy, and not to limit it to simple variable relations. Going beyond the size where such hierarchies form is pointless for our evaluation, yet we give a small range to provide a variety of formation choices.

As in Experiment 1, we generated 50 random triple pattern permutations of each query.

10.6.2.3 Results

The results of our query evaluation with the 4 datasets are presented in 4 charts (Figsures 10.2, 10.3, 10.4, and 10.5), where each is similar to the chart described in Experiment 1.

The results show a faster execution of some queries (Q1, Q9, Q10, Q11, and Q12), while it preserves or slightly improves the execution time of the other queries when run using our methodology. Some queries do not show significant improvement due to the structure of the query and its selectivity. For example if a query is asking for the results concerning two pair of variables signifying the relation between countries and languages. The results of such query is small and constant, since the number of countries and languages is constant; they do not vary even when the dataset size is exponentially increased, and thus such results are expected for some of the generated queries. Actually these kinds of queries are intentionally generated by gMark for benchmarking purposes (check [Bagan et al., 2017]).

Concerning the dataset sizes, it is clear from the charts that our optimization is less evident when the 5M nodes dataset is compared to the bigger datasets. Compared to system **S2** in Figure 10.2, the optimized orderings of queries Q1, Q10, and Q12 showed a slight improvement. In Figure 10.3 the improvement is more significant with the latter queries, in addition to the new improvements in queries Q9 and Q11. By further increasing the size of the datasets, the improvement almost stay the same (or precisely it barely increases), which shows a threshold where the gain, although significant, is stabilized.

The average of the improvement of our system (**Optimized**) compared to **avg(S2)** is given as follows:

- **Dataset 5M nodes:** Improvement of queries ranged between 1.2% and 20.5%. The mean improvement of all test queries is 3.8%.
- **Dataset 30M nodes:** Improvement of queries ranged between 1.6% and 87%. The mean improvement of all test queries is 23.5%.

- **Dataset 50M nodes:** Improvement of queries ranged between 1.4% and 84.6%. The mean improvement of all test queries is 25.2%.

- **Dataset 100M nodes:** Improvement of queries ranged between 1.6% and 85.1%. The mean improvement of all test queries is 25.7%.

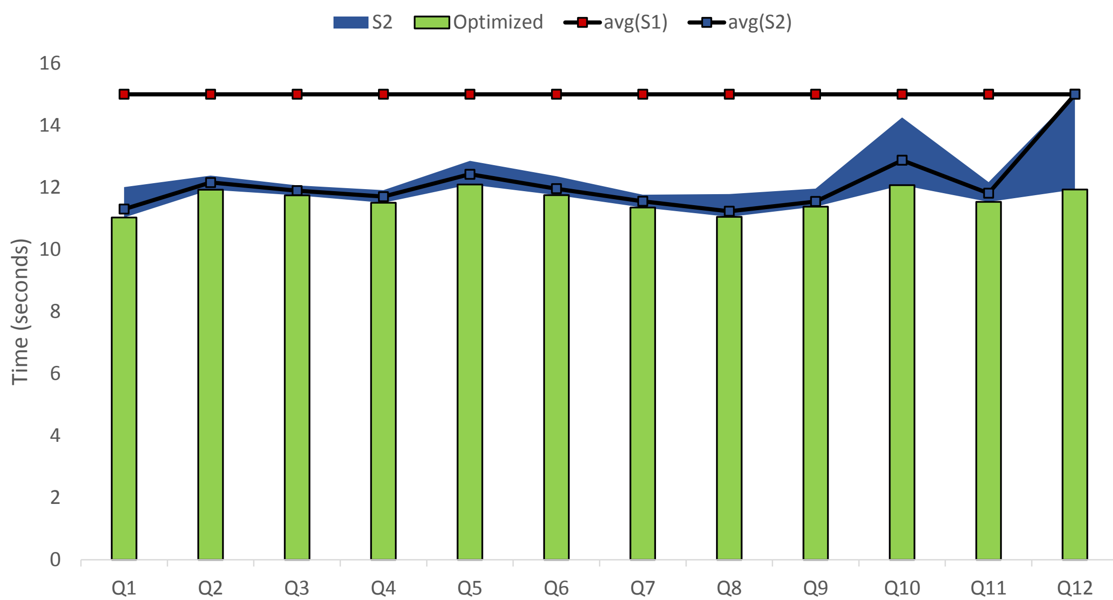

Figure 10.2: Comparing ranking-optimized query evaluation with other systems (SNB data 5M nodes)

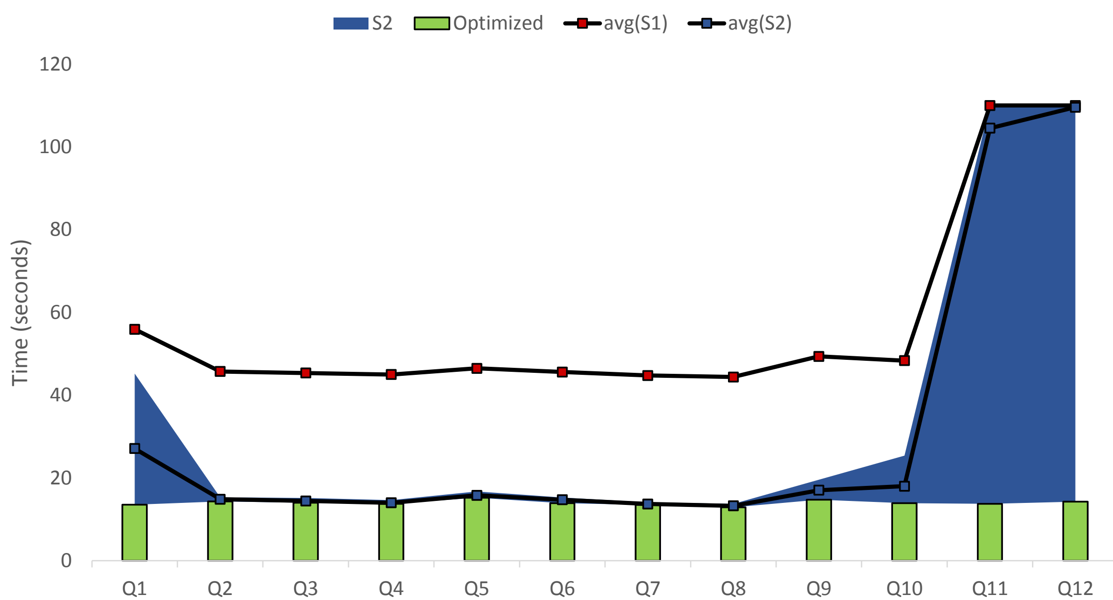

Figure 10.3: Comparing ranking-optimized query evaluation with other systems (SNB data 30M nodes)

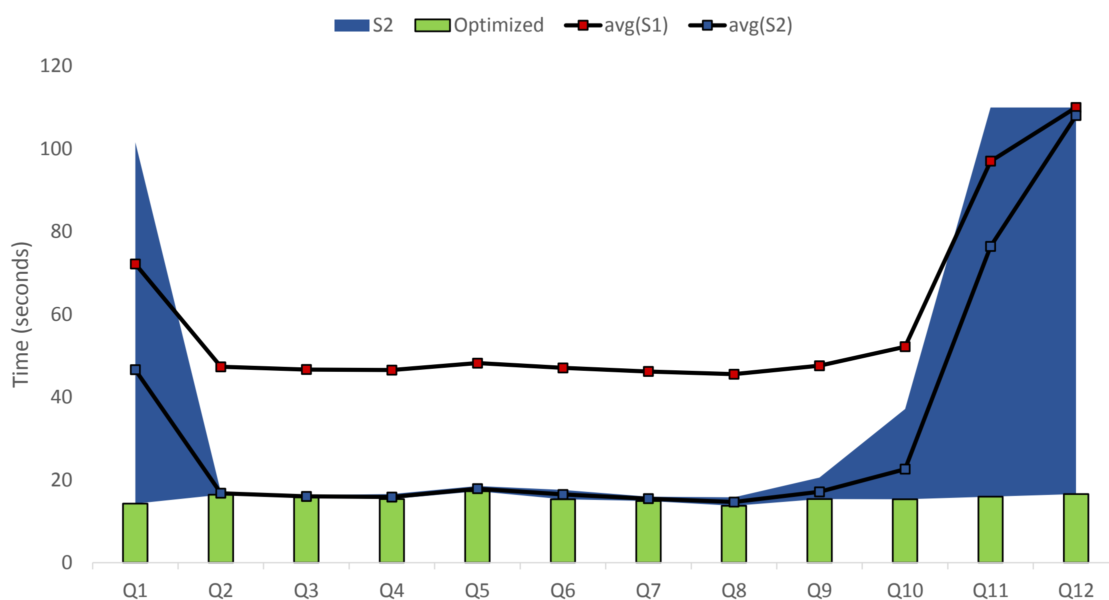

Figure 10.4: Comparing ranking-optimized query evaluation with other systems (SNB data 50M nodes)

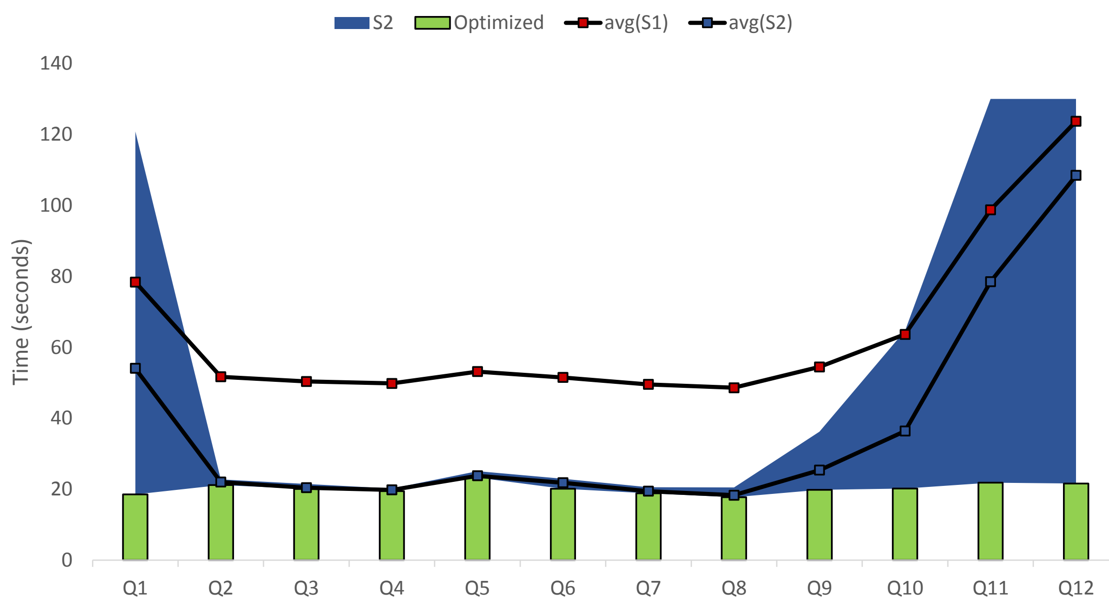

Figure 10.5: Comparing ranking-optimized query evaluation with other systems (SNB data 100M nodes)

10.7 Conclusion

We studied a method for SPARQL query optimization based on ranking triple patterns in order to select their execution order. The originality of our approach is that rankings generated by our system are based on information inferred from a schema expressed in ShEx, which is an emerging schema language for RDF data. To the best of our knowledge, this is

the first attempt of leveraging ShEx constraints for SPARQL query optimization.

We first defined a well-formation notion for data-schema pairs that is useful for inferring quantitative information about data instances. We then defined a procedure for determining rankings. We implemented a prototype of our system on top of the SPARQLGX query evaluation engine, which is known to outperform many competitors in the field. We compared the rankings found by our system, owing to the analysis of ShEx constraints, to the original reordering method of SPARQLGX in terms of query evaluation times, and with datasets of various sizes. Preliminary experimental results indicate that most rankings found by our system lead to improvements in query execution times. This illustrates the interest of considering ShEx constraints for SPARQL query optimization.

11

Conclusion and Perspectives

Contents

11.1 Summary **110**

11.2 Perspectives **111**

11.2.1 SPARQL Containment with the mixture of ShEx and OWL 111

11.2.2 RDF Distributed Data ShEx Validation 112

11.2.3 SPARQL Optimization by Transformations with ShEx 112

11.1 Summary

In this thesis, our objective was to study three layers of the Semantic Web, and the interactions between them, leading to interesting results in query analysis and optimization. The three layers mainly included the RDF standard (a data interchange standard), SPARQL (the standard query language for RDF), and ShEx (a constraint language for RDF being currently promoted by a W3C community group).

In the first part of the thesis we introduced these technologies, as well as main results in the literature about two important issues of the Semantic Web (and databases in general), which are query containment and query optimization.

For the part about our contributions on query containment (Part II), we considered the problem of SPARQL query containment in the presence of ShEx constraints. We first proposed a sound and complete procedure for the problem of containment with ShEx, considering several SPARQL fragments. Particularly our procedure considers OPTIONAL query patterns, that turns out to be an important fragment to be studied with schemas. We then showed that the complexity of our problem for the BGP SPARQL fragment is NP-Complete, for the well-designed OPTIONAL fragment is NPTIME-Complete, and for the well-designed OPTIONAL fragment extended with external UNIONs is Π_2^P-Complete. As an alternative approach, we reduced the SPARQL query containment with ShEx constraints into First Order Logic (FOL) satisfiability problem. We showed that our method can be done in an FOL fragment with only two variables whose satisfiability problem is decidable. While the satisfiability of this fragment is NEXPTIME-Complete, the proposed method allows to consider the containment of a large SPARQL fragment, namely the well-designed OPTIONAL SPARQL fragment extended with external UNIONs, MINUS feature, property path patterns, and a fragment of the FILTER feature.

This is the first work addressing SPARQL query containment in the presence of ShEx constraints, and it is the first work addressing the SPARQL OPTIONAL patterns with any kind of schema or ontology language.

For the part about our contributions on query optimization (Part II), we optimized the evaluation of BGP SPARQL queries, on RDF graphs, by taking advantage of ShEx constraints. Our optimization is based on computing and assigning ranks to query triple patterns, dictating their order of execution. For achieving that we first defined a set of well-formed ShEx schemas, that possess interesting characteristics for SPARQL query optimization, more precisely it allows to infer the relative frequency of occurrence of ShEx types. We then define our optimization method by exploiting information extracted from a ShEx schema. We finally reported on evaluation results performed showing the advantages of applying our optimization on top of SPARQLGX (an existing state-of-the-art query evaluation system). According to our experimentation, with large amounts of data, the optimization with our system may exceed 85% for queries that are highly affected by the triple patterns execution order, and by 25% on average.

11.2 Perspectives

11.2.1 SPARQL Containment with the mixture of ShEx and OWL

In [Chekol, 2016] the authors studied the problem of SPARQL query containment with an OWL entailment fragment. In this thesis on the other hand, we investigated the problem of SPARQL query containment in the presence ShEx constraints.

ShEx and OWL have different purposes. While the first is an RDF constraint language and adopts the Closed World Assumption, OWL is an ontology language associated with an entailment regime and adopts the Open World Assumption. These two languages exhibit different features useful for different purposes and one language is not meant to replace the other.

It was shown in Chap 4 that it is very verbose to express ShEx constraints in OWL. We also discussed how OWL differs from constraint languages like ShEx. On the other hand, these two technologies are not contradictory, and they can be used together in a common framework to serve the targets of each. Moreover, even if it was hard to use OWL as a constraint language, its ability to express constraint makes it interesting to further characterise the intersection of the respective expressive powers of ShEx and OWL.

In situations where both ShEx and OWL are deployed in a common framework, it becomes necessary to consider the problem of query containment from the point of view of the combination of both technologies. Gathering there constraints together is important, because these becomes the whole set of constraints, and their combination may produce more containment cases. Keeping these constraints separate means that we are losing semantics at some place.

In this context, it may be worth explaining the following research questions:

- How can we define the intersections between ShEx and OWL (intersections in terms of expressive power)?

- How can we process the expressive power intersections between ShEx and OWL to serve SPARQL query containment?

- Although it is hard to express ShEx constraints in OWL, is it possible at the first place for the whole ShEx fragment?

- If it is possible to express all ShEx constraints in OWL, is it worth doing it?

11.2.2 RDF Distributed Data ShEx Validation

The ShEx validation semantics are well defined for single source RDF graph. Yet there exist many RDF storage systems where data are distributed [Abadi et al., 2007, Graux et al., 2016a]. Such systems are acquiring increased importance as the databases are growing in size, and the need to handle these large amounts becomes essential. They provide optimized storage and retrieval methods.

In this context it is important to define a validation procedure for these situations.

- Can we find a distributed algorithm for handling distributed data validation with ShEx?
- If the process should be applied in a series (one machine after the other), what is the amount of information that should be transferred within the process? And how can we represent it?

We should even be more aware about the fact that most distributed storage systems keeps replicates of the data.

- How to avoid constraint violations due to data replicates?

All these aspects are implementation dependent (for eg. vertical partitioning versus horizontal partitioning). Yet it is interesting to find a global approach that takes the distribution scheme as an input, and handles the validation problem accordingly.

11.2.3 SPARQL Optimization by Transformations with ShEx

In Chapter 10 we proposed a method for optimizing SPARQL query evaluation based on triple patterns ranks that are retrieved with the help of the ShEx definitions. Yet other options for such optimization may be inferred from the ShEx document.

In Chapter 8 we provided a query transformation procedure that we use for the containment problem solving. One of the transformations defined can be considered itself an optimization procedure, where in that transformation some OPT operators are transformed into AND operators. To recall, consider the following query:

Q: `{?x :name ?n} OPT {?x :phone ?p}`

and the following ShEx shape:

```
<Person> {
:name xsd:string ,
:phone xsd:string }
```

According to this ShEx shape definition, we know that :name and :phone will always occur together. Thus the right hand side of the OPT pattern will always occur with the left hand

side of it. We therefore deduce that the previous query Q is equivalent to another query Q' without an OPT pattern.

Q': `{?x :name ?n.  ?x :phone ?p}`

It is known from [Pérez et al., 2009] that the evaluation of the AND-OPT fragment is in Π_2^P while the evaluation of the BGP fragment is in PTIME. This means that Q evaluation can be optimized by evaluating Q' which is equivalent to it in the context of the ShEx constraints.

One interesting perspective for further work would be to study the following research questions:

- How to leverage the full potential of ShEx constraints:
    1. in order to obtain SPARQL query transformations of different types?
    2. in order to characterise different fragments for which an optimized representation of the original query could be generated?

Bibliography

[OWL, 2012] (2012). OWL 2 web ontology language document overview (second edition). W3C recommendation, W3C. http://www.w3.org/TR/2012/REC-owl2-overview-20121211/.

[Abadi et al., 2007] Abadi, D. J., Marcus, A., Madden, S. R., and Hollenbach, K. (2007). Scalable semantic web data management using vertical partitioning. In *Proceedings of the 33rd International Conference on Very Large Data Bases*, VLDB '07, pages 411–422. VLDB Endowment.

[Aberer and Fischer, 1995] Aberer, K. and Fischer, G. (1995). Semantic query optimization for methods in object-oriented database systems. In *Proceedings of the Eleventh International Conference on Data Engineering*, ICDE '95, pages 70–79, Washington, DC, USA. IEEE Computer Society.

[AKSW, 2016] AKSW (2016). RDFUnit: An RDF Unit Testing Suite. `https://github.com/AKSW/RDFUnit`. [Online; accessed 27-August-2017].

[Arenas and Pérez, 2011] Arenas, M. and Pérez, J. (2011). Querying semantic web data with SPARQL. In *Proceedings of the Thirtieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '11, pages 305–316, New York, NY, USA. ACM.

[Bagan et al., 2017] Bagan, G., Bonifati, A., Ciucanu, R., Fletcher, G. H. L., Lemay, A., and Advokaat, N. (2017). gMark: Schema-driven generation of graphs and queries. *IEEE Transactions on Knowledge and Data Engineering*, 29(4):856–869.

[Benzaken et al., 2013] Benzaken, V., Castagna, G., Colazzo, D., and Nguyen, K. (2013). Optimizing XML querying using type-based document projection. *ACM Trans. Database Syst.*, 38(1):4:1–4:45.

[Berners-Lee and Jaffe, 1994] Berners-Lee, T. and Jaffe, J. (1994). About W3C. `https://www.w3.org/Consortium/`. [Online; accessed 27-August-2017].

[Bizer and Schultz, 2009] Bizer, C. and Schultz, A. (2009). The Berlin SPARQL benchmark. *Int. J. Semantic Web Inf. Syst.*, 5(2):1–24.

[Boneva et al., 2014] Boneva, I., Gayo, J. E. L., Hym, S., Prud'hommeaux, E. G., Solbrig, H. R., and Staworko, S. (2014). Validating RDF with shape expressions. *CoRR*, abs/1404.1270.

[Brickley and Guha, 2014] Brickley, D. and Guha, R. (2014). RDF schema 1.1. W3C recommendation, W3C. http://www.w3.org/TR/2014/REC-rdf-schema-20140225/.

[Chandra and Merlin, 1977] Chandra, A. K. and Merlin, P. M. (1977). Optimal implementation of conjunctive queries in relational data bases. In *Proceedings of the Ninth Annual ACM Symposium on Theory of Computing*, STOC '77, pages 77–90, New York, NY, USA. ACM.

[Chekol, 2016] Chekol, M. W. (2016). On the containment of SPARQL queries under entailment regimes. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, pages 936–942. AAAI Press.

[Chekol et al., 2011] Chekol, M. W., Euzenat, J., Genevès, P., and Layaïda, N. (2011). PSPARQL query containment. In *The 13th International Symposium on Database Programming Languages*.

[Chekol et al., 2012a] Chekol, M. W., Euzenat, J., Genevès, P., and Layaïda, N. (2012a). SPARQL query containment under RDFS entailment regime. In Gramlich, B., Miller, D., and Sattler, U., editors, *Automated Reasoning: 6th International Joint Conference, IJCAR 2012, Manchester, UK, June 26-29, 2012. Proceedings*, pages 134–148, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Chekol et al., 2012b] Chekol, M. W., Euzenat, J., Genevès, P., and Layaïda, N. (2012b). SPARQL query containment under SHI axioms. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*, AAAI'12, pages 10–16. AAAI Press.

[Colazzo and Sartiani, 2015] Colazzo, D. and Sartiani, C. (2015). Typing regular path query languages for data graphs. In *Proceedings of the 15th Symposium on Database Programming Languages*, DBPL 2015, pages 69–78, New York, NY, USA. ACM.

[Cyganiak et al., 2014] Cyganiak, R., Wood, D., and Lanthaler, M. (2014). RDF 1.1 concepts and abstract syntax. W3C recommendation, W3C. `http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/`.

[Duerst and Suignard, 2005] Duerst, M. and Suignard, M. (2005). RFC 3987: Internationalized Resource Identifiers (IRIs). RFC 3987 (Proposed Standard), see `http://www.ietf.org/rfc/rfc3987.txt`.

[Erling et al., 2015] Erling, O., Averbuch, A., Larriba-Pey, J., Chafi, H., Gubichev, A., Prat, A., Pham, M.-D., and Boncz, P. (2015). The ldbc social network benchmark: Interactive workload. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, SIGMOD '15, pages 619–630, New York, NY, USA. ACM.

[Etessami et al., 2002] Etessami, K., Vardi, M. Y., and Wilke, T. (2002). First-order logic with two variables and unary temporal logic. *Inf. Comput.*, 179(2):279–295.

[Fernández et al., 2013] Fernández, J. D., Martínez-Prieto, M. A., Gutiérrez, C., Polleres, A., and Arias, M. (2013). Binary rdf representation for publication and exchange (hdt). *Web Semant.*, 19:22–41.

[Foundation, 2007] Foundation, T. A. S. (2007). Apache HBase. `http://hbase.apache.org/`. [Online; accessed 26-July-2017].

[Foundation, 2014] Foundation, T. A. S. (2014). Apache Hadoop. `http://hadoop.apache.org/`. [Online; accessed 26-July-2017].

[Gayo, 2015] Gayo, J. E. L. (2015). ShExC vs SHACL. `https://github.com/labra/ShExcala/wiki/ShExC-vs-SHACL`. [Online; accessed 27-August-2017].

[Gayo, 2016] Gayo, J. E. L. (2016). ShEx vs SHACL. `https://www.slideshare.net/jelabra/shex-vs-shacl`. [Online; accessed 27-August-2017].

[Gayo, 2017] Gayo, J. E. L. (2017). SHACL/ShEx implementation. `https://github.com/labra/shaclex`. [Online; accessed 27-August-2017].

[Gayo et al., 2014] Gayo, J. E. L., Prud'Hommeaux, E., Solbrig, H., and Rodríguez, J. M. A. (2014). Validating and describing linked data portals using rdf shape expressions. In Knuth, M., Kontokostas, D., and Sack, H., editors, *1st Workshop on Linked Data Quality (LDQ)*, number 1215 in CEUR Workshop Proceedings, Aachen.

[Genevès and Layaïda, 2006] Genevès, P. and Layaïda, N. (2006). A system for the static analysis of xpath. *ACM Trans. Inf. Syst.*, 24(4):475–502.

[Genevès and Layaïda, 2007] Genevès, P. and Layaïda, N. (2007). Deciding XPath containment with MSO. *Data & Knowledge Engineering*, 63(1):108 – 136. Data Warehouse and Knowledge Discovery (DAWAK '05).

[Genevès and Layaïda, 2010] Genevès, P. and Layaïda, N. (2010). Xml reasoning made practical. In *2010 IEEE 26th International Conference on Data Engineering (ICDE 2010)*, pages 1169–1172.

[Goasdoué et al., 2013] Goasdoué, F., Kaoudi, Z., Manolescu, I., Quiané-Ruiz, J., Zampetakis, S., et al. (2013). Cliquesquare: efficient hadoop-based rdf query processing. In *BDA'13-Journées de Bases de Données Avancées*.

[Grädel et al., 1997] Grädel, E., Kolaitis, P. G., and Vardi, M. Y. (1997). On the decision problem for two-variable first-order logic. *Bulletin of Symbolic Logic*, 3(1):53–69.

[Grädel and Otto, 1999] Grädel, E. and Otto, M. (1999). On logics with two variables. *Theoretical Computer Science*, 224(1):73 – 113.

[Graux et al., 2016a] Graux, D., Jachiet, L., Genevès, P., and Layaïda, N. (2016a). Sparqlgx: Efficient distributed evaluation of sparql with apache spark. In Groth, P., Simperl, E., Gray, A., Sabou, M., Krötzsch, M., Lecue, F., Flöck, F., and Gil, Y., editors, *The Semantic Web – ISWC 2016: 15th International Semantic Web Conference, Kobe, Japan, October 17–21, 2016, Proceedings, Part II*, pages 80–87, Cham. Springer International Publishing.

[Graux et al., 2016b] Graux, D., Jachiet, L., Genevès, P., and Layaïda, N. (2016b). SPARQLGX in action: Efficient distributed evaluation of SPARQL with apache spark. In Kawamura, T. and Paulheim, H., editors, *Proceedings of the ISWC 2016 Posters & Demonstrations Track co-located with 15th International Semantic Web Conference (ISWC 2016), Kobe, Japan, October 19, 2016.*, volume 1690 of *CEUR Workshop Proceedings*. CEUR-WS.org.

[Guha, 2013] Guha, R. V. (2013). ISWC 2013 Keynote - Ramanathan V. Guha. `http://iswc2013.semanticweb.org/content/keynote-ramanathan-v-guha.html`. [Online; accessed 27-August-2017].

[Harris and Seaborne, 2013] Harris, S. and Seaborne, A. (2013). SPARQL 1.1 query language. W3C recommendation, W3C. `http://www.w3.org/TR/2013/REC-sparql11-query-20130321/`.

[HL7, 2017] HL7 (2017). FHIR. `http://hl7.org/fhir/index.html`. [Online; accessed 27-August-2017].

[Huang et al., 2011] Huang, J., Abadi, D. J., and Ren, K. (2011). Scalable SPARQL querying of large RDF graphs. Proc. VLDB Endow., 4(21).

[iStandUK, 2017] iStandUK (2017). Open Public Data. `http://www.openpublicdata.com/`. [Online; accessed 27-August-2017].

[Joshi et al., 2013] Joshi, A. K., Hitzler, P., and Dong, G. (2013). *Logical Linked Data Compression*, pages 170–184. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Kellogg et al., 2014] Kellogg, G., Lanthaler, M., and Sporny, M. (2014). JSON-LD 1.0. W3C recommendation, W3C. http://www.w3.org/TR/2014/REC-json-ld-20140116/.

[Kim et al., 2017] Kim, H., Ravindra, P., and Anyanwu, K. (2017). Type-based semantic optimization for scalable RDF graph pattern matching. In *Proceedings of the 26th International Conference on World Wide Web*, WWW '17, pages 785–793, Republic and Canton of Geneva, Switzerland. International World Wide Web Conferences Steering Committee.

[Knublauch, 2017] Knublauch, H. (2017). Schema.org (converted to SHACL by TopQuadrant). `http://datashapes.org/schema`. [Online; accessed 27-August-2017].

[Knublauch and Kontokostas, 2017] Knublauch, H. and Kontokostas, D. (2017). Shapes constraint language (SHACL). W3C recommendation, W3C. https://www.w3.org/TR/2017/REC-shacl-20170720/.

[Kontokostas, 2017] Kontokostas, D. (2017). Shape Expressions Community Group. `https://www.w3.org/community/shex/`. [Online; accessed 27-August-2017].

[Kostylev et al., 2015] Kostylev, E. V., Reutter, J. L., Romero, M., and Vrgoč, D. (2015). SPARQL with property paths. In *Proceedings of the 14th International Conference on The Semantic Web - ISWC 2015 - Volume 9366*, pages 3–18, New York, NY, USA. Springer-Verlag New York, Inc.

[Lee and Liu, 2013] Lee, K. and Liu, L. (2013). Scaling queries over big RDF graphs with semantic hash partitioning. *Proc. VLDB Endow.*, 6(14):1894–1905.

[Letelier et al., 2012] Letelier, A., Pérez, J., Pichler, R., and Skritek, S. (2012). Static analysis and optimization of semantic web queries. In *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, PODS '12, pages 89–100, New York, NY, USA. ACM.

[Libkin et al., 2013] Libkin, L., Martens, W., and Vrgoč, D. (2013). Querying graph databases with XPath. In *Proceedings of the 16th International Conference on Database Theory*, ICDT '13, pages 129–140, New York, NY, USA. ACM.

[Mendelzon and Wood, 1995] Mendelzon, A. O. and Wood, P. T. (1995). Finding regular simple paths in graph databases. *SIAM J. Comput.*, 24(6):1235–1258.

[Neumann and Weikum, 2008] Neumann, T. and Weikum, G. (2008). RDF-3X: A RISC-style engine for RDF. *Proc. VLDB Endow.*, 1(1):647–659.

[Pan et al., 2015] Pan, J. Z., Pérez, J. M. G., Ren, Y., Wu, H., Wang, H., and Zhu, M. (2015). Graph pattern based RDF data compression. In Supnithi, T., Yamaguchi, T., Pan, J. Z., Wuwongse, V., and Buranarach, M., editors, *Semantic Technology: 4th Joint International Conference, JIST 2014, Chiang Mai, Thailand, November 9-11, 2014. Revised Selected Papers*, pages 239–256, Cham. Springer International Publishing.

[Papailiou et al., 2013] Papailiou, N., Konstantinou, I., Tsoumakos, D., Karras, P., and Koziris, N. (2013). H2RDF+: High-performance distributed joins over large-scale RDF graphs. In *2013 IEEE International Conference on Big Data*, pages 255–263.

[Patel-Schneider and Hayes, 2014] Patel-Schneider, P. and Hayes, P. (2014). RDF 1.1 semantics. W3C recommendation, W3C. http://www.w3.org/TR/2014/REC-rdf11-mt-20140225/.

[Pérez et al., 2008] Pérez, J., Arenas, M., and Gutierrez, C. (2008). *nSPARQL: A Navigational Language for RDF*, pages 66–81. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Pérez et al., 2009] Pérez, J., Arenas, M., and Gutierrez, C. (2009). Semantics and complexity of sparql. *ACM Trans. Database Syst.*, 34(3):16:1–16:45.

[Pham et al., 2015] Pham, M.-D., Passing, L., Erling, O., and Boncz, P. (2015). Deriving an emergent relational schema from rdf data. In *Proceedings of the 24th International Conference on World Wide Web*, WWW '15, pages 864–874, Republic and Canton of Geneva, Switzerland. International World Wide Web Conferences Steering Committee.

[Pichler and Skritek, 2014] Pichler, R. and Skritek, S. (2014). Containment and equivalence of well-designed sparql. In *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '14, pages 39–50, New York, NY, USA. ACM.

[Polleres, 2007] Polleres, A. (2007). From SPARQL to rules (and back). In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, pages 787–796, New York, NY, USA. ACM.

[Prud'hommeaux, 2017a] Prud'hommeaux, E. (2017a). Shape expressions (shex) json formats. Technical report, W3C and MIT. http://shex.io/primer/ShExJ.

[Prud'hommeaux, 2017b] Prud'hommeaux, E. (2017b). Shape expressions (shex) primer. Technical report, W3C and MIT. `http://shexspec.github.io/primer/`.

[Prud'hommeaux et al., 2017] Prud'hommeaux, E., Boneva, I., Labra Gayo, J., and Kellogg, G. (2017). Shape expressions language 2.0. Technical report, W3C. `https://shexspec.github.io/spec/`.

[Prud'hommeaux et al., 2014] Prud'hommeaux, E., Labra Gayo, J. E., and Solbrig, H. (2014). Shape expressions: An rdf validation and transformation language. In *Proceedings of the 10th International Conference on Semantic Systems*, SEM '14, pages 32–40, New York, NY, USA. ACM.

[Prud'hommeaux and Seaborne, 2008] Prud'hommeaux, E. and Seaborne, A. (2008). SPARQL query language for RDF. W3C recommendation, W3C. `http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/`.

[Robie et al., 2017] Robie, J., Spiegel, J., and Dyck, M. (2017). XML path language (XPath) 3.1. W3C recommendation, W3C. https://www.w3.org/TR/2017/REC-xpath-31-20170321/.

[Schmidt et al., 2010] Schmidt, M., Meier, M., and Lausen, G. (2010). Foundations of sparql query optimization. In *Proceedings of the 13th International Conference on Database Theory*, ICDT '10, pages 4–33, New York, NY, USA. ACM.

[Schreiber and Raimond, 2014] Schreiber, G. and Raimond, Y. (2014). RDF 1.1 primer. W3C note, W3C. `http://www.w3.org/TR/2014/NOTE-rdf11-primer-20140624/`.

[Schulz, 2013] Schulz, S. (2013). *System Description: E 1.8*, pages 735–743. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Seaborne and Harris, 2013] Seaborne, A. and Harris, S. (2013). SPARQL 1.1 query language. W3C recommendation, W3C. `http://www.w3.org/TR/2013/REC-sparql11-query-20130321/`.

[Serfiotis et al., 2005] Serfiotis, G., Koffina, I., Christophides, V., and Tannen, V. (2005). Containment and minimization of rdf/s query patterns. In *Proceedings of the 4th International Conference on The Semantic Web*, ISWC'05, pages 607–623, Berlin, Heidelberg. Springer-Verlag.

[Staworko et al., 2015] Staworko, S., Boneva, I., Gayo, J. E. L., Hym, S., Prud'hommeaux, E. G., and Solbrig, H. (2015). Complexity and Expressiveness of ShEx for RDF. In Arenas, M. and Ugarte, M., editors, *18th International Conference on Database Theory (ICDT 2015)*, volume 31 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 195–211, Dagstuhl, Germany. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[Sutcliffe, 2017a] Sutcliffe, G. (2017a). The CADE ATP System Competition. `http://www.cs.miami.edu/~tptp/CASC/`. [Online; accessed 27-August-2017].

[Sutcliffe, 2017b] Sutcliffe, G. (2017b). TPTP Format for Problems. `http://www.cs.miami.edu/~tptp/TPTP/QuickGuide/Problems.html`. [Online; accessed 27-August-2017].

[The Apache Software Foundation, 2011] The Apache Software Foundation (2011). Apache jena. `http://jena.apache.org/`. [Online; accessed 17-February-2017].

[TopQuadrant, 2001] TopQuadrant (2001). SHACL Tutorial: Getting Started. `https://www.topquadrant.com/technology/shacl/tutorial/`. [Online; accessed 27-August-2017].

[Weidenbach et al., 2009] Weidenbach, C., Dimova, D., Fietzke, A., Kumar, R., Suda, M., and Wischnewski, P. (2009). SPASS version 3.5. In *Proceedings of the 22Nd International Conference on Automated Deduction*, CADE-22, pages 140–145, Berlin, Heidelberg. Springer-Verlag.

[World Wide Web Foundation, 2014] World Wide Web Foundation (2014). WEB INDEX. `http://thewebindex.org/`. [Online; accessed 2-May-2017].

[Zaharia et al., 2012] Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., Franklin, M. J., Shenker, S., and Stoica, I. (2012). Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, NSDI'12, pages 2–2, Berkeley, CA, USA. USENIX Association.

[Zhang et al., 2013] Zhang, X., Chen, L., Tong, Y., and Wang, M. (2013). Eagre: Towards scalable I/O efficient sparql query evaluation on the cloud. In *2013 IEEE 29th International Conference on Data Engineering (ICDE)*, pages 565–576.

[Zhang et al., 2016] Zhang, X., Van Den Bussche, J., and Picalausa, F. (2016). On the satisfiability problem for sparql patterns. *J. Artif. Int. Res.*, 56(1):403–428.

[Zou et al., 2011] Zou, L., Mo, J., Chen, L., Özsu, M. T., and Zhao, D. (2011). gstore: Answering SPARQL queries via subgraph matching. *Proc. VLDB Endow.*, 4(8):482–493.

A

Appendix: Chapter 8 and 9 Experiments

A.1 Chapter 8 and 9, Experiments: Queries & Schemas

Q1a:
```
SELECT * WHERE {
 :product1 :is :Product .
 ?product :label  "label1".
 ?product :productFeature "ProductFeature1" .
 ?product :productFeature "ProductFeature2" . }
```

Q1b:
```
SELECT * WHERE {
 :product1 :is :Product .
 ?product :label ?label .
 ?product :productFeature "ProductFeature1" .
 ?product :productFeature "ProductFeature2" . }
```

Q2a:
```
SELECT * WHERE {
 :product1 :is :Product .
 :product1 :label ?label .
 :product1 :comment ?comment .
 :product1 :producer ?p .
 ?p :label ?producer .
 :product1 :publisher ?p .
 :product1 :productFeature "ProductFeature1"^^xsd:string .
 :product1 :productPropertyTextual1 ?propertyTextual1 .
 :product1 :productPropertyNumeric1 ?propertyNumeric1 .
 OPTIONAL
 { :product1 :productPropertyTextual2 ?propertyTextual2 }
 OPTIONAL
 { :product1 :productPropertyTextual3 ?propertyTextual3 }
 OPTIONAL
 {:product1 :productPropertyNumeric2 ?propertyNumeric2 .
  ?propertyNumeric2 :value "123" }
}
```

Q2b:
```
SELECT * WHERE {
 :product1 :is :Product .
 :product1 :label ?label .
 :product1 :comment ?comment .
 :product1 :producer ?p .
 ?p :label ?producer .
 :product1 :publisher ?p .
 :product1 :productFeature "ProductFeature1" .
 :product1 :productPropertyTextual1 ?propertyTextual1 .
 :product1 :productPropertyNumeric1 ?propertyNumeric1 .
 OPTIONAL
 { :product1 :productPropertyTextual2 ?propertyTextual2 }
 OPTIONAL
 { :product1 :productPropertyTextual3 ?propertyTextual3 }
}
```

Q3a:
```
SELECT * WHERE {
 ?product :is :Product .
 ?product :label ?label .
 ?product :productFeature "ProductFeature1" .
 ?product :productPropertyNumeric1 ?p1 .
 ?product :productPropertyNumeric3 ?p3 .
 OPTIONAL
 {?product :productFeature "ProductFeature2" .
  ?product :rating ?rating }
}
```

Q3b:
```
SELECT * WHERE {
 ?product :is :Product .
 ?product :label ?label .
 ?product :productFeature "ProductFeature1" .
 ?product :productPropertyNumeric1 ?p1 .
 ?product :productPropertyNumeric3 ?p3 .
}
```

Q4a:
```
SELECT *  WHERE {
 ?review :reviewFor :product1 .
 ?review :title ?title .
 ?review :text ?text .
 ?review :reviewDate ?reviewDate .
 ?review :reviewer ?reviewer .
 ?reviewer :is :Reviewer .
 ?reviewer :name  "Name1".
 ?reviewer :name  "Name2".
 OPTIONAL { ?review :rating ?rating1 . }
 OPTIONAL { ?review :rating ?rating2 . }
 OPTIONAL { ?review :rating ?rating3 . }
 OPTIONAL { ?review :rating ?rating4 . }
}
```

Q4b:
```
SELECT *  WHERE {
 ?review :reviewFor :product1 .
 ?review :title ?title .
 ?review :text ?text .
 ?review :reviewDate ?reviewDate .
 ?review :reviewer ?reviewer .
 ?reviewer :is :Reviewer .
 ?reviewer :name ?reviewerName .
}
```

Q5a:
```
SELECT * WHERE {
 :product1 :is :Product .
 :product1 :label ?productLabel .
 OPTIONAL {
  ?offer :product :product1 .
  ?offer :price ?price .
  ?offer :vendor ?vendor .
  ?vendor :label ?vendorTitle .
  ?vendor :country :FR .
  ?offer :publisher ?vendor .
  ?offer :validTo ?date .
 }
 OPTIONAL {
  ?review :is :Review .
  ?review :reviewFor :product1  .
  ?review :reviewer ?reviewer .
  ?reviewer :name ?revName .
  ?review :title ?revTitle  .
  ?revTitle :value "title1" .
  OPTIONAL { ?review :rating1 ?rating1 . }
  OPTIONAL { ?review :rating2 ?rating2 . }
 }
}
```

Q5b:
```
SELECT * WHERE {
 :product1 :is :Product .
 :product1 :label ?productLabel .
 OPTIONAL {
  ?offer :product :product1 .
  ?offer :price ?price .
  ?offer :vendor ?vendor .
  ?vendor :label ?vendorTitle .
  ?vendor :country :FR.
  ?offer :publisher ?vendor .
  ?offer :validTo ?date .
 }
 OPTIONAL {
  ?review :is :Review .
  ?review :reviewFor :product1  .
  ?review :reviewer ?reviewer .
  ?reviewer :name ?revName .
  ?review :title ?revTitle  .
  ?revTitle :value "title2" .
  OPTIONAL { ?review :rating1 ?rating1 . }
  OPTIONAL { ?review :rating2 ?rating2 . }
  }
}
```

Figure A.1: Queries for SPARQL containment experimentation

Schema 1:	Schema 2:	Schema 3:	Schema 4:

:Product {
 :is [:Product] ?,
 :label xsd:string ? ,
 :productFeature xsd:string ?
}
 	
:Reviewer {
 :is [:Reviewer] ?
}
 	
:Review {
 :is [:Review] ?
}
	:Product {
	 :is [:Product] ?,
	 :label xsd:string ? ,
	 :productFeature xsd:string *
	}
	
	:Reviewer {
	 :is [:Reviewer] ?,
	 :name xsd:string ?
	}
	
	:Review {
	 :is [:Review] ?
		:Product {
		 :is [:Product] ?,
		 :label xsd:string ? ,
		 :productFeature xsd:string *
		}
		
		:Reviewer {
		 :is [:Reviewer] ?,
		 :name xsd:string *
		}
		
		:Review {
		 :is [:Review] ?,
		 :reviewer @:Reviewer ?,
		 :title xsd:string ?
		}
			:Product {
			 :is [:Product] ?,
			 :label xsd:string ?,
			 :productFeature xsd:string *,
			 :productPropertyNumeric2 @:PropertyNumeric ?
			}
			
			:Reviewer {
			 :is [:Reviewer] ?,
			 :name xsd:string *
			}
			
			:Review {
			 :is [:Review] ?
			}
			
			:PropertyNumeric {
			 :is [:PropertyNumeric] ?,
			 :value xsd:integer ?
			}

Figure A.2: Schemas for SPARQL containment experimentation

```
1 fof(schema1, axiom, (! [X] : ( productt(X) | revieww(X) | reviewerr(X) ))) .
2 fof(schema2, axiom, (! [X,Y] : ( ( productt(X) => ( ( is(X,Y) => Y = product ) & ( ~is_max2(X,Y) ) & ( label(X,Y)
     => string(Y) ) & ( ~label_max2(X,Y) ) & ( productFeature(X,Y) => string(Y) ) & ( ~productFeature_max2(X,Y)
    ) ) ) & ( revieww(X) => ( ( is(X,Y) => Y = review ) & ( ~is_max2(X,Y) ) ) ) & ( reviewerr(X) => ( ( is(X,Y)
    => Y = reviewer ) & ( ~is_max2(X,Y) ) ) ) ))) .
3 fof(schema3, axiom, ( ![X] : (isURI(X) => (~string(X) & ~integer(X))))) .
4 fof(schema4, axiom, ( ![X] : (string(X) => (~isURI(X) & ~integer(X))))) .
5 fof(schema5, axiom, ( ![X] : (integer(X) => (~string(X) & ~isURI(X))))) .
```

Listing A.1: Schema 1: Encoding into FOF TPTP Syntax as Axioms

```
1 fof(schema1, axiom, (! [X] : ( productt(X) | revieww(X) | reviewerr(X) ))) .
2 fof(schema2, axiom, (! [X,Y] : ( ( productt(X) => ( ( is(X,Y) => Y = product ) & ( ~is_max2(X,Y) ) & ( label(X,Y)
     => string(Y) ) & ( ~label_max2(X,Y) ) & ( productFeature(X,Y) => string(Y) ) ) ) & ( revieww(X) => ( ( is(X
    ,Y) => Y = review ) & ( ~is_max2(X,Y) ) ) ) & ( reviewerr(X) => ( ( is(X,Y) => Y = reviewer ) & ( ~is_max2(X
    ,Y) ) & ( name(X,Y) => string(Y) ) & ( ~name_max2(X,Y) ) ) ) ))) .
3 fof(schema3, axiom, ( ![X] : (isURI(X) => (~string(X) & ~integer(X))))) .
4 fof(schema4, axiom, ( ![X] : (string(X) => (~isURI(X) & ~integer(X))))) .
5 fof(schema5, axiom, ( ![X] : (integer(X) => (~string(X) & ~isURI(X))))) .
```

Listing A.2: Schema 2: Encoding into FOF TPTP Syntax as Axioms

```
1 fof(schema1, axiom, (! [X] : ( productt(X) | revieww(X) | reviewerr(X) ))) .
2 fof(schema2, axiom, (! [X,Y] : ( ( productt(X) => ( ( is(X,Y) => Y = product ) & ( ~is_max2(X,Y) ) & ( label(X,Y)
     => string(Y) ) & ( ~label_max2(X,Y) ) & ( productFeature(X,Y) => string(Y) ) ) ) & ( revieww(X) => ( ( is(X
    ,Y) => Y = review ) & ( ~is_max2(X,Y) ) & ( reviewerIs(X,Y) => reviewerr(Y) ) & ( ~reviewerIs_max2(X,Y) ) &
    ( title(X,Y) => string(Y) ) & ( ~title_max2(X,Y) ) ) ) & ( reviewerr(X) => ( ( is(X,Y) => Y = reviewer ) & (
     ~is_max2(X,Y) ) & ( name(X,Y) => string(Y) ) ) ) ))) .
3 fof(schema3, axiom, ( ![X] : (isURI(X) => (~string(X) & ~integer(X))))) .
4 fof(schema4, axiom, ( ![X] : (string(X) => (~isURI(X) & ~integer(X))))) .
5 fof(schema5, axiom, ( ![X] : (integer(X) => (~string(X) & ~isURI(X))))) .
```

Listing A.3: Schema 3: Encoding into FOF TPTP Syntax as Axioms

```
1 fof(schema1, axiom, (! [X] : ( productt(X) | revieww(X) | reviewerr(X) | propertyNumericc(X) ))) .
2 fof(schema2, axiom, (! [X,Y] : ( ( productt(X) => ( ( is(X,Y) => Y = product ) & ( ~is_max2(X,Y) ) & ( label(X,Y)
     => string(Y) ) & ( ~label_max2(X,Y) ) & ( productFeature(X,Y) => string(Y) ) & ( productPropertyNumeric2(X,
    Y) => propertyNumericc(Y) ) & ( ~productPropertyNumeric2_max2(X,Y) ) ) ) & ( revieww(X) => ( ( is(X,Y) => Y
    = review ) & ( ~is_max2(X,Y) ) ) ) & ( reviewerr(X) => ( ( is(X,Y) => Y = reviewer ) & ( ~is_max2(X,Y) ) & (
     name(X,Y) => string(Y) ) ) ) & ( propertyNumericc(X) => ( ( is(X,Y) => Y = propertyNumeric ) & ( ~is_max2(X
    ,Y) ) & ( value(X,Y) => integer(Y) ) & ( ~value_max2(X,Y) ) ) ) ))) .
3 fof(schema3, axiom, ( ![X] : (isURI(X) => (~string(X) & ~integer(X))))) .
4 fof(schema4, axiom, ( ![X] : (string(X) => (~isURI(X) & ~integer(X))))) .
5 fof(schema5, axiom, ( ![X] : (integer(X) => (~string(X) & ~isURI(X))))) .
```

Listing A.4: Schema 4: Encoding into FOF TPTP Syntax as Axioms

B

Appendix: Chapter 10 Experiments

B.1 Chapter 10, Experiment 1: Shape Relation Graph

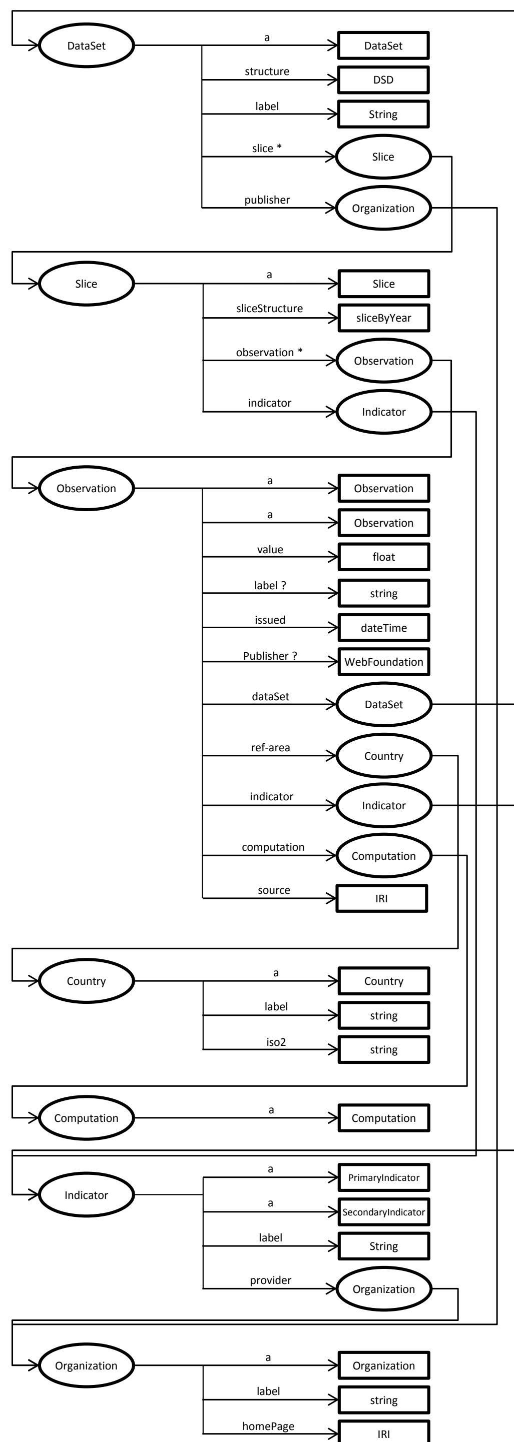

Figure B.1: Shape relation graph (WebIndex)

B.2 Chapter 10, Experiment 1: Queries

Q1 (optimized ordering)
```
SELECT ?o, ?d, ?struct WHERE {
  ?d rdfs:label "dataSet1576" .
  ?d qb:structure ?struct .
  ?o qb:dataSet ?d .
  ?o rdf:type qb:Observation .
}
```

Q2 (optimized ordering)
```
SELECT ?o, ?d, ?org WHERE {
  ?org foaf:homepage
ex:homepageOrg988 .
  ?d dct:publisher ?org .
  ?o qb:dataSet ?d .
  ?o rdf:type qb:Observation .
}
```

Q3 (optimized ordering)
```
SELECT ?o, ?d, ?c, ?i, ?comp,
?org WHERE {
  ?org foaf:homepage
ex:homepageOrg988 .
  ?i wf:provider ?org .
  ?o cex:indicator ?i .
  ?o cex:computation ?comp .
  ?o cex:ref-area ?c .
  ?o qb:dataSet ?d .
  ?o rdf:type qb:Observation .
}
```

Q4 (optimized ordering)
```
SELECT ?s, ?i WHERE {
  ?i wf:provider ex:org825 .
  ?s cex:indicator ?i .
  ?s rdf:type qb:Slice .
}
```

Q5 (optimized ordering)
```
SELECT ?s, ?i WHERE {
  ?i rdfs:label "indicator2006" .
  ?s cex:indicator ?i .
  ?s rdf:type qb:Slice .
}
```

Q6 (optimized ordering)
```
SELECT ?s, ?i, ?org WHERE {
  ?org foaf:homepage
ex:homepageOrg988 .
  ?i wf:provider ?org .
  ?s cex:indicator ?i .
  ?s rdf:type qb:Slice .
}
```

Q7 (optimized ordering)
```
SELECT ?s, ?type, ?i, ?org
WHERE {
  ?org foaf:homepage
ex:homepageOrg988 .
  ?i wf:provider ?org .
  ?s cex:indicator ?i .
  ?s rdf:type ?type .
}
```

Q8 (optimized ordering)
```
SELECT ?d, ?org WHERE {
  ex:obs4830 qb:dataSet ?d .
  ?d dct:publisher ?org .
  ?d rdf:type qb:DataSet .
}
```

Q9 (optimized ordering)
```
SELECT ?d, ?type, ?org WHERE {
  ex:obs4830 qb:dataSet ?d .
  ?d dct:publisher ?org .
  ?d rdf:type ?type .
}
```

Q10 (optimized ordering)
```
SELECT ?s, ?org, ?i, ?hp WHERE
{
  ?s wf:provider ?org .
  ?org foaf:homepage ?hp .
  ?s cex:indicator ?i .
  ?org rdf:type
org:Organization .
  ?s rdf:type qb:Slice .
}
```

Q11 (optimized ordering)
```
SELECT ?s, ?d, ?struct1, ?struct2
WHERE {
  ?s qb:data ?d .
  ?d qb:structure ?struct2 .
  ?s qb:sliceStructure ?struct1 .
  ?d rdf:type qb:DataSet .
  ?s rdf:type qb:Slice .
}
```

Q12 (violating)
```
SELECT ?o, ?d, ?c, ?i, ?comp,
?org WHERE {
  ?o rdf:type qb:Observation .
  ?o qb:dataSet ?d .
  ?o cex:ref-area ?c .
  ?o cex:indicator ?i .
  ?o cex:computation ?comp .
  ?i wf:provider ?org .
  ?org foaf:homepage
"homepageOrg988" .
}
```

Q12 (optimized)
```
SELECT ?o, ?d, ?c, ?i, ?comp,
?org WHERE {

}
```

Figure B.2: Hand-crafted experiment queries

B.3 Chapter 10, Experiment 2: Schema

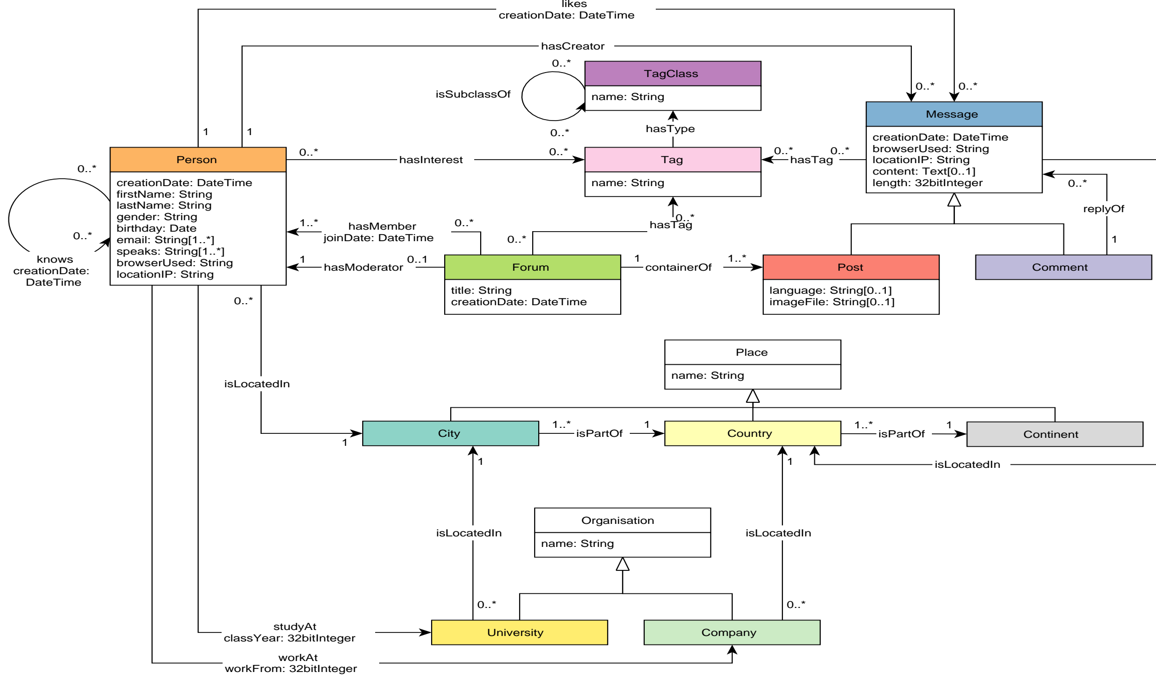

Figure B.3: Schema (LDBC SNB)
Source: `https://github.com/ldbc/ldbc_snb_docs`

B.4 Chapter 10, Experiment 2: Shape Relation Graph

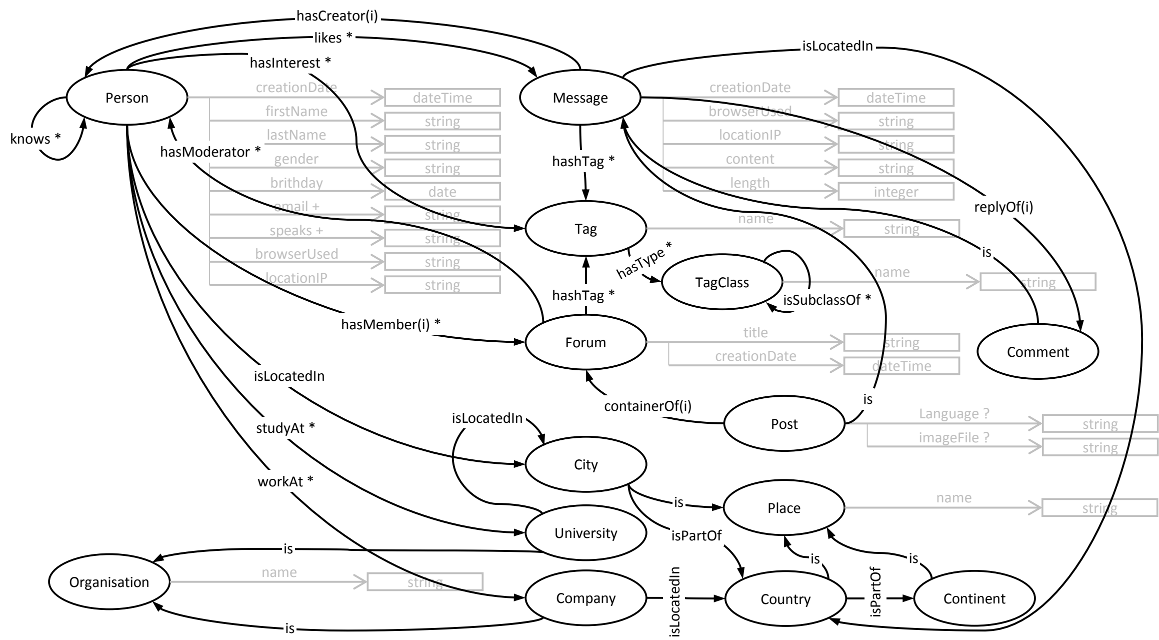

Figure B.4: Shape relation graph (LDBC SNB)

B.5 Chapter 10, Experiment 2: Queries

The following prefix definition follows for all the queries in this chapter:
`PREFIX ex: <http://example.org/gmark/>`

```
SELECT ?x3, ?x2, ?x4, ?x1, ?x0 WHERE {
?x1 ex:pcontainerOf ?x0 .
?x1 ex:phasMember ?x2 .
?x2 ex:pworksAt ?x3 .
?x0 ex:pisSubclassOf ?x4 .
?x5 ex:plikes ?x4 .
?x6 ex:phasMember ?x5 .
?x6 ex:phasMember ?x3 . }
```

Listing B.1: Query (Q1)

```
SELECT ?x4, ?x2, ?x0, ?x3, ?x1 WHERE {
?x0 ex:pisLocatedIn ?x8 .
?x8 ex:pknows ?x1 .
?x9 ex:pisLocatedIn ?x1 .
?x9 ex:pgender ?x2 .
?x3 ex:pemail ?x2 .
?x3 ex:phasInterest ?x4 .
?x0 ex:pbrowserUsed ?x5 .
?x6 ex:pname ?x5 .
?x6 ex:phasType ?x7 .
?x4 ex:pisSubclassOf ?x7 . }
```

Listing B.2: Query (Q2)

```
SELECT ?x0, ?x3, ?x2, ?x1 WHERE {
?x1 ex:pname ?x0 .
?x2 ex:phasType ?x1 .
?x3 ex:phasInterest ?x2 .
?x3 ex:pbirthday ?x4 .
?x5 ex:pcreationDate ?x4 .
?x5 ex:pisLocatedIn ?x6 .
?x6 ex:pname ?x7 .
?x8 ex:pname ?x7 .
?x9 ex:pisPartOf ?x8 . }
```

Listing B.3: Query (Q3)

```
SELECT ?x1, ?x0, ?x2, ?x3 WHERE {
?x1 ex:pemail ?x0 .
?x1 ex:plikes ?x2 .
?x2 ex:pisLocatedIn ?x3 .
?x3 ex:pisPartOf ?x4 .
?x4 ex:pname ?x5 .
?x6 ex:pname ?x5 .
?x7 ex:pstudyAt ?x6 .
?x7 ex:pisLocatedIn ?x8 . }
```

Listing B.4: Query (Q4)

```
SELECT ?x2, ?x1, ?x4, ?x0, ?x3 WHERE {
?x0 ex:phasMember ?x9 .
?x9 ex:pname ?x1 .
?x10 ex:phasModerator ?x1 .
?x2 ex:pspeaks ?x10 .
?x2 ex:phasModerator ?x11 .
?x11 ex:pknows ?x3 .
?x3 ex:pknows ?x4 .
?x5 ex:pisLocatedIn ?x0 .
?x5 ex:pgender ?x6 .
?x7 ex:pspeaks ?x6 .
?x8 ex:phasMember ?x7 .
?x8 ex:phasModerator ?x4 . }
```

Listing B.5: Query (Q5)

```
SELECT ?x4, ?x2, ?x3, ?x5, ?x0, ?x1 WHERE {
?x1 ex:pname ?x0 .
?x1 ex:pname ?x2 .
?x3 ex:pname ?x2 .
?x4 ex:pisPartOf ?x3 .
?x4 ex:pisPartOf ?x5 .
?x5 ex:pname ?x6 .
?x7 ex:pgender ?x6 .
?x7 ex:pgender ?x8 .
?x9 ex:pname ?x8 .
?x9 ex:pname ?x10 . }
```

Listing B.6: Query (Q6)

```
SELECT ?x3, ?x4, ?x5, ?x2, ?x0, ?x1 WHERE {
?x1 ex:pworksAt ?x0 .
?x1 ex:pstudyAt ?x2 .
?x2 ex:pname ?x3 .
?x4 ex:pname ?x3 .
?x4 ex:pname ?x5 .
?x6 ex:pname ?x5 .
?x6 ex:plocationIP ?x7 .
?x8 ex:pbrowserUsed ?x7 . }
```

Listing B.7: Query (Q7)

```
SELECT ?x2, ?x1, ?x3, ?x0 WHERE {
?x0 ex:pisLocatedIn ?x1 .
?x2 ex:pisPartOf ?x1 .
?x3 ex:pisLocatedIn ?x2 .
?x3 ex:pgender ?x4 .
?x5 ex:pname ?x4 .
?x5 ex:pname ?x6 . }
```

Listing B.8: Query (Q8)

```
SELECT ?x0, ?x3, ?x2, ?x1, ?x4 WHERE {
?x1 ex:pname ?x0 .
?x1 ex:pisLocatedIn ?x2 .
?x2 ex:pname ?x3 .
?x4 ex:pname ?x3 .
?x4 ex:pisLocatedIn ?x5 .
?x5 ex:pisPartOf ?x6 .
?x7 ex:pisPartOf ?x6 . }
```

Listing B.9: Query (Q9)

```
SELECT ?x0, ?x2, ?x1, ?x3, ?x5, ?x4 WHERE {
?x1 ex:pisSubclassOf ?x0 .
?x2 ex:phasType ?x1 .
?x3 ex:phasInterest ?x2 .
?x3 ex:pknows ?x4 .
?x5 ex:phasType ?x0 .
?x6 ex:phasInterest ?x5 .
?x6 ex:pknows ?x7 .
?x7 ex:pknows ?x8 .
?x4 ex:phasMember ?x8 . }
```

Listing B.10: Query (Q10)

```
SELECT ?x4, ?x5, ?x0, ?x1, ?x3, ?x2 WHERE {
?x0 ex:pname ?x1 .
?x2 ex:pname ?x1 .
?x3 ex:pisLocatedIn ?x2 .
?x4 ex:pisPartOf ?x0 .
?x5 ex:pisPartOf ?x0 .
?x6 ex:pisPartOf ?x0 .
?x7 ex:pisPartOf ?x3 .
?x8 ex:pisPartOf ?x3 .
?x9 ex:pisPartOf ?x3 . }
```

Listing B.11: Query (Q11)

```
SELECT ?x1, ?x0, ?x2, ?x4, ?x3 WHERE {
?x1 ex:pisLocatedIn ?x0 .
?x1 ex:pisLocatedIn ?x2 .
?x3 ex:pisLocatedIn ?x2 .
?x3 ex:pisLocatedIn ?x4 .
?x5 ex:pisLocatedIn ?x0 .
?x6 ex:phasCreator ?x5 .
?x6 ex:phasTag ?x7 .
?x4 ex:phasTag ?x7 . }
```

Listing B.12: Query (Q12)