HAL
open science

Sur l'analyse statique des requêtes SPARQL avec la logique modale

Nicola Guido

To cite this version:

Nicola Guido. Sur l'analyse statique des requêtes SPARQL avec la logique modale. Web. Université Grenoble Alpes, 2015. Français. NNT : 2015GREAM059 . tel-01250984v2

HAL Id: tel-01250984

https://inria.hal.science/tel-01250984v2

Submitted on 28 Apr 2016

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.

UNIVERSITÉ DE GRENOBLE

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE GRENOBLE

Spécialité : **Informatique**

Arrêté ministériel : 7 août 2006

Présentée par

Nicola Guido

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

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

On the Static Analysis for SPARQL Queries using Modal Logic

Thèse soutenue publiquement le **3 décembre 2015**

Mme Sihem Amer-Yahia
Directrice de Recherche CNRS, Université de Grenoble-Alpes, Présidente
M. Olivier Curé
Maitre de conférences HDR, Université de Paris-Est, Rapporteur
M. Fabien Gandon
Directeur de Recherche INRIA - Sophia Antipolis Méditerranée, Rapporteur
M. Olivier Corby
Chercheur INRIA - Sophia Antipolis Méditerranée, Examinateur
M. Mohand-Said Hacid
Professeur - Université Claude Bernard Lyon 1 LIRIS, Examinateur
M. Pierre Genevès
Chercheur CNRS - Université de Grenoble-Alpes, Co-Directeur de thèse
Mme Cécile Roisin
Professeur - Université de Grenoble-Alpes, Directrice de thèse

Abstract:
Static analysis is a core task in query optimization and knowledge base verification. We study static analysis techniques for SPARQL, the standard language for querying Semantic Web data. Specifically, we investigate the query containment problem and the query-update independence analysis. We are interested in developing techniques through reductions to the validity problem in logic.

We address SPARQL query containment with optional matching. The optionality feature is one of the most complicated constructors in SPARQL and also the one that makes this language depart from classical query languages such as relational conjunctive queries. We focus on the class of well-designed SPARQL queries, proposed in the literature as a fragment of the language with good properties regarding query evaluation. SPARQL is interpreted over graphs, hence we encode it in a graph logic, specifically the modal logic K interpreted over label transition systems. We show that this logic is powerful enough to deal with query containment for the well-designed fragment of SPARQL.

We show how to translate RDF graphs into transition systems and SPARQL queries into K-formulae. Therefore, query containment in SPARQL can be reduced to unsatisfiability in K. One advantage of this approach is to open the way for implementations using off-the-shelf satisfiability solvers for K. This makes it possible to benefit from years of research in optimization of modal logic satisfiability solvers in the context of SPARQL static analysis.

We also report on a preliminary overview of the SPARQL query-update problem. A query is independent of an update when the execution of the update does not affect the result of the query. Determining independence is especially useful in the contest of huge RDF repositories, where it permits to avoid expensive yet useless re-evaluation of queries. While this problem has been intensively studied for fragments of relational calculus, no works exist for the standard query language for the semantic web. We report on our investigations on how a notion of independence can be defined in the SPARQL context.

Résumé:
L'analyse statique est une tâche essentielle dans l'optimisation des requêtes et la vérification de la base de graphes RDF. Nous étudions des techniques d'analyse statique pour SPARQL, le langage standard pour l'interrogation des données du Web sémantique. Plus précisément, nous étudions le problème d'inclusion des requêtes et de l'analyse de l'indépendance entre les requêtes et la mise à jour de la base de graphes RDF.

Dans cette thèse, nous nous sommes intéressés au développement de techniques d'analyse statique d'inclusion de requêtes qui ramène le problème à un problème de satisfaisabilité dans la logique K.

Nous nous traitons le problème d'inclusion des requêtes SPARQL en présence de l'opérateur OPTIONAL. L'optionalité est l'un des constructeurs les plus compliqués dans SPARQL et aussi celui qui rend ce langage plus expressif que les langages de requêtes classiques, comme SQL. Nous nous concentrons sur la classe de requêtes appelée "well-designed SPARQL", proposée dans la littérature comme un fragment du langage avec de bonnes propriétés en matière d'évaluation des requêtes incluant l'opération OPTIONAL. À ce jour, l'inclusion de requêtes a été implémentée à l'aide de différentes techniques: homomorphisme de graphes, bases de données canoniques, techniques de la théorie des automates et réduction au problème de la validité dans une logique. Dans cette thèse, nous utilisons cette dernière technique pour tester l'inclusion des requêtes SPARQL avec OPTIONAL utilisant une logique expressive appelée «logique K». En utilisant cette technique, il est possible de régler le problème d'inclusion des requêtes pour plusieurs fragments de SPARQL, même en présence de contraintes sur les graphes. Cette extensibilité n'est pas garantie par les autres méthodes.

Nous montrons comment traduire un graphe RDF en un système de transitions, ainsi qu'une requête SPARQL en une formule K. Avec ces traductions, l'inclusion des requêtes dans SPARQL peut être réduite au test de la validité d'une formule logique. Un avantage de cette approche est d'ouvrir la voie pour des implémentations utilisant des solveurs efficaces de satisfaisabilité pour K.

Avec notre procédure de traduction, nous présentons un banc d'essais de tests d'inclusion pour les requêtes SPARQL avec OPTIONAL. Nous avons effectué des expériences pour tester et comparer des solveurs d'inclusion de l'état de l'art.

Nous présentons également un aperçu préliminaire du problème d'indépendance entre requête et mise à jour. Une requête SPARQL est indépendante de la mise à jour d'une base RDF lorsque l'exécution de la mise à jour ne modifie pas le résultat de la requête. Bien que ce problème ait été intensivement étudié pour des fragments de calcul relationnel, il n'existe pas de travaux pour le langage de requêtes SPARQL. Nous proposons une définition de la notion de l'indépendance dans le contexte de SPARQL et nous établissons des premières pistes de l'analyse statique dans certaines situations d'inclusion entre une requête et une mise à jour.

Acknowledgments

I thank Sihem Amer-Yahia and Mohand-Said Hacid for accepting to be part of my thesis committee.

I express my heartfelt gratitude to my reviewers, Olivier Curé, Fabien Gandon and Olivier Corby, who provided me encouraging and constructive feedback. They took the time to very precisely read and criticize my work and I am grateful for their thoughtful and detailed comments. They greatly helped me improve the quality of this thesis.I would like to thank Nabil for his invaluable availability and support.

I thank my supervisors Cécile Roisin and Pierre Genevès for directing my work. They gave me the opportunity to join TYREX team at INRIA as their student. I also thank them for always having had confidence in me, and letting me freely choose my research directions and the way of investigating them. In the TYREX team, I was fortunate to meet Nabil Layaïda. Special thanks to the members of TYREX teams.

To Stefano, thanks for all the support you have given me.

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

Contents

List of Figures xi

1 Introduction 1
1.1 Background . . . 2
1.2 Summary of Contributions . . . 4
1.3 Thesis Outline . . . 4

I State-of-the-art 7

2 Preliminaries 9
2.1 General Notations . . . 10
2.2 The Semantic Web . . . 11
2.3 Resource Description Framework (RDF) . . . 12
2.3.1 RDF Concepts and Abstract Syntax . . . 13
2.3.2 RDF Schema(RDFS) . . . 13
2.4 Web Ontology Language (OWL) . . . 15
2.4.1 No Unique Name Assumption . . . 15
2.4.2 Open World Assumption . . . 15
2.4.3 Features of OWL . . . 16
2.5 SPARQL Protocol and RDF Query Language . . . 17
2.5.1 Anatomy of a SPARQL Query . . . 17
2.5.2 Abstract Syntax for Patterns . . . 19
2.5.3 Standard Semantics of SPARQL . . . 19
2.5.4 Relational Algebra Semantics of SPARQL . . . 21
2.5.5 A Survey on Complexity of SPARQL Evaluation . . . 25
2.6 SPARQL Update . . . 25
2.6.1 Abstract Syntax for DELETE/INSERT Operations . . . 26
2.6.2 Semantics of DELETE/INSERT Operations . . . 27

3 Static Analysis of SPARQL Queries 29
3.1 Introduction . . . 30
3.2 Query Containment Problem . . . 30
3.2.1 Techniques Overview . . . 32
3.2.2 A Survey on the Complexity of Query Containment . . . 38
3.2.3 SPARQL Containment Problem . . . 39
3.3 Conclusion . . . 41

4 An Overview of the Modal Logic 43
4.1 Introduction . . . 44
4.2 History and Motivations . . . 45
4.2.1 Motivations . . . 48
4.3 Basic Modal Logic K and its Extensions . . . 49
4.3.1 Modal Logic K . . . 49

4.3.2 Modal Logic K with Multiple Modalities: $\mathtt{K}_n$ . . . 51
4.3.3 Modal Logic K with Multiple and Backward Modalities: $\mathtt{K}_{(n,back)}$ . . . 52
4.4 Mu-calculus . . . 53
4.4.1 Fixpoints as Recursion . . . 53
4.4.2 Syntax . . . 54
4.4.3 Semantics . . . 54
4.4.4 Notion of Satisfiability . . . 55
4.4.5 Small Model Property . . . 55
4.4.6 Alternation-Free Mu-Calculus: $\mathtt{AF}_\mu$ . . . 55
4.5 State-of-the-art of the Solver . . . 56
4.5.1 KBDD Solver . . . 56
4.5.2 $\mathtt{AF}_\mu$ Solver . . . 57
4.5.3 Tree Solver . . . 58
4.5.4 MSPASS . . . 58
4.6 Conclusion . . . 59

5 A Decision Procedure for $\mathtt{K}_{(n,back)}$ 61
5.1 Introduction . . . 62
5.2 Preliminaries . . . 62
5.2.1 Syntax and Semantics . . . 62
5.2.2 Closure, Lean and Types . . . 64
5.3 Algorithm . . . 67
5.3.1 Algorithm Scheme . . . 67
5.3.2 Correctness . . . 68
5.4 Implementation . . . 70
5.4.1 Implicit Representation of Set of Types . . . 70
5.5 Conclusion . . . 71

II Contribution 73

6 Containment of Well-designed SPARQL queries 75
6.1 Introduction . . . 76
6.2 Well-Designed Graph Patterns . . . 76
6.3 Pattern Trees . . . 78
6.4 Containment of Well-Designed Graph Patterns . . . 82
6.4.1 Bag Semantics and Set Semantics . . . 82
6.4.2 Subsumption Relation . . . 83
6.5 Logical Encoding . . . 84
6.5.1 RDF Graphs as Transition Systems . . . 84
6.5.2 Encoding Queries as K-Formulæ . . . 92
6.5.3 Containment Problem . . . 96
6.6 Reducing Query Containment to Unsatisfiability . . . 97
6.7 Experimentation . . . 100
6.7.1 Benchmark . . . 100
6.8 Query Containment Solvers . . . 103
6.8.1 MSPASS . . . 103

6.8.2 SPARQL-Algebra . . . 104
6.8.3 Experimentation . . . 104
6.9 Conclusion . . . 105

7 Towards Query-Update Independence Analysis for SPARQL 107
7.1 Introduction . . . 108
7.2 Requirements for Independence Analyses . . . 108
7.3 Notion of Independence . . . 110
7.3.1 Definition of Independence . . . 111
7.3.2 A Condition for Independence . . . 112
7.3.3 Direction and Approaches. . . . 113
7.4 Syntactic checking . . . 113
7.5 Containment Approach . . . 117
7.5.1 From Updates to Select Queries . . . 117
7.5.2 From Select Queries to Graphs . . . 119
7.5.3 Homomorphism Notions . . . 119
7.5.4 Containment between Queries and Updates . . . 120
7.6 Conclusion . . . 123

8 Conclusion and Perspectives for Research 125
8.1 Conclusion . . . 126
8.2 Summary of the Publications . . . 127
8.3 Perspectives . . . 127
8.3.1 Containment under Schema . . . 128
8.3.2 Query-Update Independence under Schema . . . 128
8.3.3 Relational Algebra Based Approach . . . 129
8.4 Summary . . . 132

Bibliography 133

A Appendix A 141
A.1 Conjunctive Queries (CQs) . . . 141
A.1.1 Datalog Notation . . . 141
A.1.2 From SQL to Datalog Rule . . . 143

Appendices 141

B Appendix B 145
B.1 Benchmark . . . 145

C Résumé étendu 149
C.1 Motivation et objectifs . . . 149
C.1.1 Cadre des travaux . . . 149
C.1.2 Démarche . . . 151
C.2 Principales contributions . . . 151
C.3 Conclusion et perspectives . . . 152

List of Figures

2.1 Semantic web layer cake [Horrocks et al., 2005] 11
2.2 A simple RDF graph 13
2.3 The anatomy of a SPARQL query [Domingue et al., 2011] 18
2.4 Relational representation of a query 21
2.5 Relational operator † 22
2.6 Functions genCond and genPR [Chebotko et al., 2009] 23
2.7 Relational algebra based semantics of SPARQL [Chebotko et al., 2009] 24

3.1 Query answering using view 31
3.2 Scheme: Reduction to satisfiability test 37
3.3 Query containment complexity for relational fragments 38

4.1 $\mathcal{L}_3$ logic [Łukasiewiez, 1953] 47
4.2 Carnap's semantics 47
4.3 Semantics of modal formulæ in K 50
4.4 Satisfiability of modal formulæ in K 50
4.5 *Modal depth* of modal formulæ in K 51
4.6 Algorithm's principle: progressive bottom-up reasoning [Genevès et al., 2015] 58

5.1 Satisfiability of modal formulæ in $\mathtt{K}_{(n,back)}$ 63
5.2 Closure of the formula ψ 65
5.3 Formal description of the Top-Down Algorithm 68

6.1 Representation of pattern trees 79
6.2 Pattern trees that are not well-designed 80
6.3 An RDF graph where a predicate appears as a node [Chekol, 2012] 85
6.4 Transition system encoding the RDF graph as bipartite graph [Chekol, 2012] 85
6.5 Encoding of triples 86
6.6 Encoding of triples in α relation 87
6.7 Encoding of triples in α-β relations 87
6.8 The β tree structure 88
6.9 Transition system encoding the RDF graph G. 88
6.10 Transition system encoding the RDF graph G. The node in W'' is a black anonymous node; nodes in W' are the other nodes 90
6.11 Restricted transition system 91
6.12 General workflow of query containment tests 101
6.13 Experimental setup for our query containment test. The tester (dashed red rectangle) parses queries and passes them to a solver wrapper (dotted blue rectangle) 103
6.14 Results for test suite (logarithmic scale). 104

7.1 RDF graph of Example 7.3.1 120
7.2 SCENARIO A. $\phi(\mathbf{G_{del}}) \subseteq \mathbf{G_q}$ 121
7.3 SCENARIO B. $\phi(\mathtt{G_q}) \subset \mathtt{G_{del}}$. 121

7.4 SCENARIO C. $\phi(\mathtt{G}_{\mathtt{ins}}) \subset \mathtt{G}_{\mathtt{q}}$. 122
7.5 SCENARIO D. $\phi(\mathtt{G}_{\mathtt{q}}) \subseteq \mathtt{G}_{\mathtt{ins}}$ 123

8.1 Relational representation of a delete update 130
8.2 Relational algebra based semantics of `SPARQL UPDATE` 131

A.1 Graph representation of the rule 142

1

Introduction

Contents

1.1 Background 2
1.2 Summary of Contributions 4
1.3 Thesis Outline 4

1.1 Background

The term "Semantic Web" refers to a Web of data that can be readable and process-able by machines and not just by humans. While the term was coined by Tim Berners Lee, it has largely been an initiative of the World Wide Web Consortium W3C.

In 1999, the *Resource Description Framework* (RDF) was published as a W3C Recommendation, designed as a data model for representing information about World Wide Web resources [Lassila and Swick, 1999].

Jointly with this release, the natural problem of querying and modifying RDF data was raised, and thus the question of how to manage RDF data has been in the focus of the Semantic Web community. A first answer was given with the release of SPARQL as a W3C Recommendation [Prud'hommeaux and Seaborne, 2008]. With time, SPARQL has become the standard query language for RDF. Since then, the amount of RDF data published on the Web has grown constantly, as shown by the popularity of initiatives like "Linked Open Data project" [Bizer et al., 2009] and "DBpedia" [Auer et al., 2007].

This increase in notoriety has drawn the attention of the research community, as RDF and SPARQL offer many new and interesting problems to tackle. Several research efforts are being directed towards understanding the fundamental properties of the language, as well as developing new techniques to handle these problems.

RDF and SPARQL will be formally introduced in Chapter 2. However, as it core, an RDF dataset is a set of triples of the form $(\mathtt{s}, \mathtt{p}, \mathtt{o})$. This syntax is used to describe a directed graph with named arcs; for this reason, an RDF dataset is interchangeably referred to as RDF graph. SPARQL is in essence a graph pattern matching language. Its fundamental component, the SPARQL triple pattern, is an RDF triple which can have variables instead of labels. One could see an RDF graph as merely being a set of tuples; then if one restricts SPARQL to conjunctions of triple patterns (which are basically ternary atoms), then one can see that basic SPARQL queries are essentially relational conjunctive queries. Thus, one can connect this fragment of SPARQL to the decades of works behind conjunctive queries of the database community.

In these works for the relational fragment, two problems are well-studied: static analysis and optimization of queries.

Static analysis refers to the study of queries without any knowledge of the dataset they will be evaluated over. Problems in this area include query containment (which means deciding whether the answer of one query will always be contained in the answer of another query, when evaluated over the same dataset), equivalence (which means deciding whether two different queries will always run the same results when evaluated over the same dataset) and query-update independence problem (which means detecting when the result of the query does not change before and after an update of the dataset).

On the other hand, query optimization refers to the study of how to more efficiently evaluate a given query. When evaluating a query written in a language like SQL, the query is first *parsed*, that is, turned into a parse tree representing the structure of the query. The parse

tree is then transformed into an expression tree of the relational algebra, which is termed a *logical query plan* or just query plan for short. In picking a query plan, one has opportunities to apply many different algebraic operations, with the goal of hopefully producing the best query plan [Levy and Sagiv, 1994]. In reality, obtaining an optimal query plan is not trivial; it is more important to avoid the worst plans and find a good plan. This is a problem which has been studied for several decades, and optimization techniques based on query plans for relational queries are very widespread in the field [Levy and Sagiv, 1994].

In this last decade the static analysis and the optimization of SPARQL queries have received increased attention [Serfiotis et al., 2005, Stocker et al., 2008, Schmidt et al., 2010, Letelier et al., 2012, Chekol et al., 2012b, Pichler and Skritek, 2014]. Most of these works focus on the previously mentioned fragment of SPARQL, taking into account the conjunctive query approach. However, it is SPARQL's ability to work with partial information which makes it an interesting language: since data on the web is inherently incomplete, it is essential for users to be able to request optional information. For example, if one were to request the names, phone numbers and email addresses of a group of people using conjunctive queries, and one person's email address was unknown, then there would be no information regarding that person. SPARQL's OPTIONAL operator enables users to request optional information. In the previous example, if users were to ask for the names of people and their telephone numbers, and optionally their email addresses, then the answer to the query would include all known name and telephone numbers in the dataset, along with email addresses whenever they are available.

Unfortunately, when one goes beyond the basic fragment of SPARQL, things get considerably more complicated [Pérez et al., 2009, Arenas and Pérez, 2011]. The OPTIONAL operator has proven to be particularly complex. Furthermore, its use in practice is substantial, which makes its study essential. [Pérez et al., 2009] showed how the combined complexity of the evaluation problem for SPARQL rises dramatically, from P-membership for the most basic fragment, up to PSPACE-completeness when including the OPTIONAL feature. Nevertheless, the same work also defined a natural and well-behaved fragment of OPTIONAL operator: the class of *well-designed* SPARQL queries. It was shown in [Pérez et al., 2009] that the combined complexity of the evaluation problem for well-designed SPARQL queries is coNP-complete, which is much more tractable than the general case of SPARQL queries with the OPTIONAL operator.

In [Letelier et al., 2012], the authors tackled the problem of static analysis of well-designed SPARQL queries, mostly focusing on query containment and equivalence. To study these problems the authors introduce a new representation of SPARQL graph patterns, called *pattern tree*, that can be viewed as query plans for well-designed SPARQL queries.

In this thesis, we are interested in static analysis for SPARQL. More precisely, we focus on the query containment problem and the query-update independence problem.

Concerning the query containment problem, based on the theoretical complexity results of [Letelier et al., 2012], we introduce a procedure to test the query containment of well-designed SPARQL queries. To date, testing query containment has been performed using different techniques: containment mapping, canonical databases, automata theory techniques and through a reduction to the validity problem in logic. In this thesis, we use the latter

technique to test containment of well-designed SPARQL queries. We show that modal logics are powerful enough to deal with query containment for well-designed SPARQL queries and an efficiently implementation of the whole approach is provided.

Concerning the query-update independence problem, we report on our preliminary investigations of this novel topic for SPARQL.

1.2 Summary of Contributions

We propose a way to solve the query containment problem for a fragment of SPARQL queries with the optional operator. Our approach consists in a linear translation of these queries in terms of formulæ expressed in the modal logic K. One advantage of this approach is to open the way for implementations using off-the-shelf satisfiability solvers for K. This makes it possible to benefit from years of research in optimization of modal logic satisfiability solvers in the context of SPARQL static analysis.

We report on an overview of the query-update independence problem for SPARQL. While this problem has been intensively studied for fragments of relational calculus, to the best of our knowledge, no works exist for the standard query language for the semantic web. We report on our investigations on how a notion of independence can be defined in the new setting brought by SPARQL and RDF. We introduce a condition for detecting the query-update independence for SPARQL and an implementation of this condition is provided. Despite, this first method presents some limitations and more often the query-update dependence is detected. To cope with these limitations, a second approach for qualifying the dependence relations between a query and an update is provided. This methods aims at qualifying the query-update dependence using a containment property over the query and the update graph patterns. When such a containment is detected, a more precise analysis might be allowed.

1.3 Thesis Outline

In Chapter 2, we present the preliminaries that are used along this thesis. We present the technologies standardized by W3C to enable building semantic web applications.

In Chapter 3 we present the principles of the static analysis and optimization techniques for SPARQL and a broad survey of related works.

In Chapter 4, we show a brief overview of the modal logic K and its extensions. We present the syntax, the semantics and the satisfiability problem for each of these logics.

In Chapter 5, we report a decision procedure for the satisfiability problem of K. This decision procedure has been developed during this thesis and details of the implementation are presented in this Chapter.

The Chapter 6 unveils a procedure to translate RDF graphs into transition systems. In doing so, RDF graphs become bipartite graphs with two sets of nodes: triple nodes and subject, predicate, and object nodes where navigation can be done using transition programs. The next task requires encoding queries as modal formulæ and then consequently reducing the query containment test into a validity test in the logic. We then prove that this reduction is sound and complete.

The Chapter 7 reports on preliminary investigations concerning a novel topic: the query-update independence problem for SPARQL.

In Chapter 8, we conclude with a summary of the results of this thesis and draw several perspectives of this work.

Part I

State-of-the-art

2

Preliminaries

This chapter introduces the preliminaries that are used in the development of this thesis. We present the foundations of the semantic web, and provide an overview of the Resource Description Framework, the Web Ontology Language, and the SPARQL Protocol and RDF Query Language.

Contents

2.1 General Notations **10**

2.2 The Semantic Web **11**

2.3 Resource Description Framework (RDF) **12**

2.3.1 RDF Concepts and Abstract Syntax 13

2.3.2 RDF Schema(RDFS) 13

2.4 Web Ontology Language (OWL) **15**

2.4.1 No Unique Name Assumption 15

2.4.2 Open World Assumption 15

2.4.3 Features of OWL 16

2.5 SPARQL Protocol and RDF Query Language **17**

2.5.1 Anatomy of a SPARQL Query 17

2.5.2 Abstract Syntax for Patterns 19

2.5.3 Standard Semantics of SPARQL 19

2.5.4 Relational Algebra Semantics of SPARQL 21

2.5.5 A Survey on Complexity of SPARQL Evaluation 25

2.6 SPARQL Update **25**

2.6.1 Abstract Syntax for DELETE/INSERT Operations 26

2.6.2 Semantics of DELETE/INSERT Operations 27

2.1 General Notations

Let A and B be sets. If $A \cap B = \emptyset$, then $A \uplus B$ denotes $A \cup B$ and recalls the fact that A and B are disjoint.

Complexity theory

In this section we recall notions from complexity theory. We assume the reader has some basic background. For a more detailed treatment, we refer to the textbook of Papadimitriou [Papadimitriou, 1994].

- $\mathtt{P}$: The class of languages that can be accepted by some polynomially time bounded deterministic Turing machine.
- $\mathtt{NP}$: The class of languages that can be accepted by some polynomially time bounded non-deterministic Turing machine $\mathbb{T}$, i.e., there exists some polynomial $p(n)$ such that $\mathbb{T}$ halts on every input of length n after at most $p(n)$ steps.
- $\mathtt{coNP}$: The class of languages obtained by taking the complement of any language in $\mathtt{NP}$.
- $\Sigma_{\mathtt{n}}^{\mathtt{P}}$: The class of languages which are accepted by a polynomial time alternating Turing machine $\mathbb{T}$ [1] which begins in a existential state and has at most $(n-1)$ alternations.
- $\Pi_{\mathtt{n}}^{\mathtt{P}}$ The class of languages which are accepted by a polynomial time alternating Turing machine $\mathbb{T}$ which begins in a universal state and has at most $(n-1)$ alternations.
- $\mathtt{PH}$: The polynomial hierarchy $\mathtt{PH}=\bigcup_n \Sigma_{\mathtt{n}}^{\mathtt{P}} = \bigcup_n \Pi_{\mathtt{n}}^{\mathtt{P}}$.
- $\mathtt{PSPACE}$: The class of languages that can be accepted by some polynomially space bounded deterministic Turing machine [2].
- $\mathtt{EXP}$: The class of languages that can be accepted by some exponentially time bounded deterministic Turing machine $\mathbb{T}$, i.e. there exists some polynomial $p(n)$ such that $\mathbb{T}$ halts on every input of length n after at most $2^{p(n)}$ steps.
- $\mathtt{2EXP}$: The class of languages that can be accepted by some doubly exponentially time bounded deterministic Turing machine $\mathbb{T}$, i.e. there exists some polynomial $p(n)$ such that $\mathbb{T}$ halts on every input of length n after at most $2^{2^{p(n)}}$ steps.
- m$\mathtt{EXP}$: The class of languages that can be accepted by some m-fold exponentially time bounded deterministic Turing machine $\mathbb{T}$, i.e. there exists some polynomial $p(n)$ such that $\mathbb{T}$ halts on every input of length n at most $\textsc{unfolded}(m)$ steps, where $\textsc{unfolded}(0)=p(n)$ and $\textsc{unfolded}(k)=2^{\textsc{unfold}(k-1)}$ for each $k \geq 1$.

[1] An alternating Turing machine (ATM) is a non-deterministic Turing machine (so it has two transition functions, one of which is non-deterministically chosen in each step of its computation) in which every state is labeled with either $\exists$ and $\forall$

[2] From Savitch's theorem [Savitch, 1969] it directly follows that $\mathtt{PSPACE}$ equals the class of languages accepted by some polynomially space bounded non-deterministic Turing machine.

2.2 The Semantic Web

The term Semantic Web was coined by Tim Berners-Lee for a web of data that can be processed by machines [Berners-Lee et al., 2001].

"The Semantic Web is not a separate Web but an extension of the current one, in which information is given well-defined meaning, better enabling computers and people to work in cooperation".

The purpose and ideal of the Semantic Web is to allow machines, along with humans, to make better use of the information that can be found in the Web and to infer meaning and knowledge from that information in order to assist human users in their daily activities.

This is not a new vision, already having been hinted in the first World Wide Web Conference (1994) by Tim Berners-Lee as stated in [Shadbolt et al., 2006], and later in Weaving the Web [Berners-Lee and Fischetti, 1999].

The vision was that the semantic annotation of web resources would allow computers to automatically reason about the underlying data and enable them to make reliable and logically founded conclusions. In an effort to bring the vision of the Semantic Web to a reality, necessary measures are being taken. A strong force behind this are the `W3C` working groups, highly involved with standardization and creating a means for researchers to communicate their innovations and to share ideas.

The Semantic Web Layer Cake.

The Semantic Web layer cake [Horrocks et al., 2005], depicted in Figure 2.1, is an informal stack frequently used to illustrate the different core concepts of the Semantic Web.

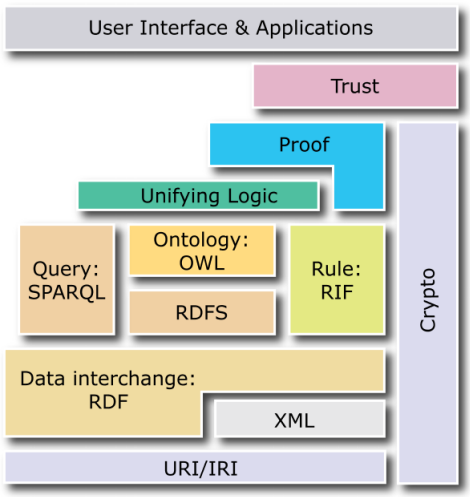


Figure 2.1: Semantic web layer cake [Horrocks et al., 2005]

The diagram is an illustration of the hierarchy of languages, where each layer exploits and uses capabilities of the layers below.

The bottom layers contain technologies that are well known from hypertext web and that without change provide basis for the Semantic Web (URI, Unicode, XML and Namespace).

The middle layers contain technologies standardized by W3C to enable building Semantic Web applications. *Resource Description Framework* (RDF) is a framework for creating statements in a form of so-called triples. It enables to represent information about resources in the form of graph - the Semantic Web is sometimes called Giant Global Graph. *Resource Description Framework Schema* (RDFS) provides basic vocabulary for RDF. Using RDFS it is for example possible to create hierarchies of classes and properties. *Web Ontology Language* (OWL) extends RDFS by adding more advanced constructs to describe semantics of RDF statements. It allows stating additional constraints, such as for example cardinality, restrictions of values, or characteristics of properties such as transitivity. It is based on description logic and so brings reasoning power to the Semantic Web. *SPARQL Protocol and RDF Query Language* (SPARQL) is an RDF query language - it can be used to query any RDF-based data (i.e., including statements involving RDFS and OWL). Querying language is necessary to retrieve information for Semantic Web applications. RIF is a rule interchange format. It is important, for example, to allow describing relations that cannot be directly described using description logic used in OWL.

The top layers contain technologies that are not yet standardized or contain just ideas that should be implemented in order to realize the Semantic Web.

In this thesis, we concentrate mainly on the middle layers, a detailed discussion on the rest of the layers of the Semantic Web layer cake can be found in [Gerber et al., 2008]. The next sections provide an overview of the Resource Description Framework (RDF), RDF Schema (RDFS), the Web Ontology Language (OWL), and the SPARQL Protocol and RDF Query Language (SPARQL).

2.3 Resource Description Framework (RDF)

The *Resource Description Framework* is the core data model of all Semantic Web-based applications. Any information added by higher-level components like RDF Schema and OWL is modeled within RDF.

The current RDF specification is split into six W3C Recommendations.The most important document is the RDF Primer, which introduces the basic concepts of RDF. It is a summary of the other documents and contains the basic information needed to effectively use RDF. The RDF Primer also describes how to define vocabularies using the RDF Vocabulary Description Language (also called RDF Schema).

2.3.1 RDF Concepts and Abstract Syntax

RDF is a graph data format for the representation of information in the Web. An RDF statement is a subject-predicate-object structure, called RDF triple, intended to describe resources and properties of those resources.

We present a compact formalization of RDF [Hayes, 2004]. Let U, B, L be three disjoint infinite sets denoting the set of URIs (identifying a resource), blank nodes (denoting an unidentified resource) and literals (a character string or some other type of data) respectively. We abbreviate any union of these sets as for instance, $UBL = U \cup B \cup L$. More formally, an RDF triple is a tuple $(s, p, o) \in UB \times U \times UBL$, where s is the subject, p the predicate and o the object. A set of RDF triples is often referred to as an RDF graph.

Example 2.3.1. *Consider 10 triples about employers and their information (all identifiers correspond to URIs):*

$$\begin{aligned}\{&(\mathtt{A}_1, \mathtt{name}, \mathtt{Joe}), && (\mathtt{A}_1, \mathtt{phone}, 1111),\\ &(\mathtt{A}_2, \mathtt{name}, \mathtt{Martin}), && (\mathtt{A}_2, \mathtt{fax}, 2222),\\ &(\mathtt{A}_3, \mathtt{name}, \mathtt{Louisa}), && (\mathtt{A}_3, \mathtt{phone}, 3333),\\ &(\mathtt{A}_4, \mathtt{name}, \mathtt{John}), && (\mathtt{A}_4, \mathtt{web}, \mathtt{www.John.fr}),\\ &(\mathtt{A}_4, \mathtt{email}, \mathtt{John@p.fr}), && (\mathtt{A}_4, \mathtt{phone}, 4444) \qquad \}\end{aligned}$$

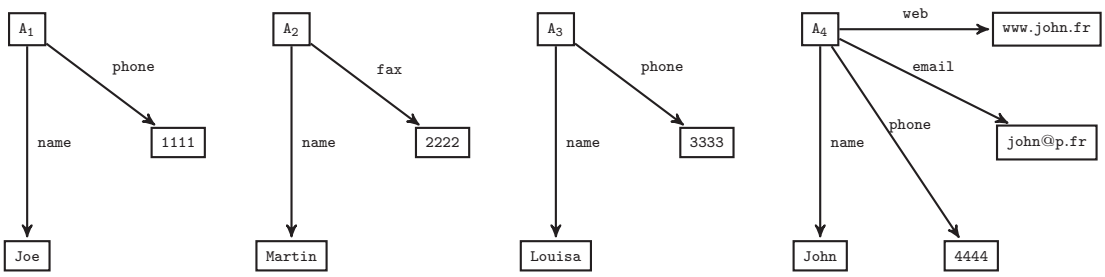


Figure 2.2: A simple RDF graph

2.3.2 RDF Schema(RDFS)

Resource Description Framework Schema (RDFS) is the most basic schema language commonly used in the Semantic Web technology stack. It is lightweight and very easy to use and get started with. RDFS is a semantic extension of RDF. It provides mechanisms for describing groups of related resources and the relationships between these resources.

It may be considered as a simple ontology language expressing subsumption relations between classes or properties [Hayes 2004].

The RDFS class and property system is similar to the type systems of object-oriented programming languages such as Java. RDFS differs from many such systems in that instead of

defining a class in terms of the properties its instances may have, RDFS describes properties in terms of the classes of resource to which they apply. This is the role of the *domain* and *range* mechanisms. For example, we could define the *author* property to have a domain of *Document* and a range of *Person*, whereas a classical object oriented system might typically define a class *Book* with an attribute called *author* of type *Person*.

Technically, RDFS is an RDF vocabulary used for expressing axioms constraining the interpretation of RDF graphs. Hence, schemas are themselves RDF graphs. The RDFS vocabulary and its semantics are given in [Hayes, 2004].

The RDFS vocabulary is defined in a namespace, identified by the IRI

http://www.w3.org/2000/01/rdf-schema#

and it is conventionally associated with the prefix rdfs:. The W3C specifications introduce another important namespace, the RDF namespace

http://www.w3.org/1999/02/22-rdf-syntax-ns#

conventionally associated with the prefix rdf:. These namespaces comprise a set of terms with predefined meaning. In Table 2.1, we present some of the predefined vocabulary terms:

TERMS	MEANING
rdf : type	used for typing entities
rdfs : subClassOf	used to describe subclass relationships between classes
rdfs : subPropertyOf	used to describe subproperty relationships between properties
rdfs : Class	used to assign a logical type to URIs
rdf : Property	used to assign a logical type to URIs
rdfs : domain	used to specify the domain of properties
rdfs : range	used to specify the range of properties
rdfs : Resource	all things described by RDF are instances of this class.

Table 2.1: RDFS - Core vocabulary

Example 2.3.2. *A possible schema for the predicate* name *of Example 2.3.1 is the following:*

: name	rdf : type	rdf : Property
: name	rdfs : domain	: ID
: name	rdfs : range	: Person
: ID	rdf : type	rdfs : Class
: Person	rdf : type	rdfs : Class;

where name *is a property with domain in* ID *and range in Person.* ID *and* Person *are RDF-classes.*

2.4 Web Ontology Language (OWL)

Although RDFS allows the definition of classes, properties and restrictions on these elements, it may not be enough to model all intended situations. W3C identified a number of user cases [Heflin, 2004], where the expressivity provided by RDFS does not suffice. For example, RDFS cannot specify that the Person and City classes are disjoint, or that a StringQuartet has exactly four musicians as members.

The *Web Ontology Language* OWL is a W3C recommendation which defines a family of knowledge representation languages for writing ontologies for the web. The language is similar to Marvin Minsky's Frames language [Minsky, 1974]. The most important differences and characteristic will be explained in the following.

2.4.1 No Unique Name Assumption

The *Unique Name Assumption* UNA is a general concept of description logic, which states that different names always refer to different entities in the world. By contrast to Frames, OWL does not inherit from this assumption.

For example, in OWL it can be assumed by default that the two resources

$<$ http : //www.nicola.it/foaf.rdf#me $>$

and

$<$ http : //people.org/NicolaGuido $>$

are distinct entities. Actually, it could be that they refer to the same entity and hence, the information about both resources could be merged from different sources on the Web. Asserting equality of resources is done with the owl : sameAs property. Explicitly asserting inequality for different resources is possible with the owl : differentFrom property.

2.4.2 Open World Assumption

The *Open World Assumption* OWA is a general concept of formal logics, which admits that the given knowledge is incomplete and everything is true unless it is asserted otherwise.

For example, give only the fact

$<$ http : //www.nicola.it/foaf.rdf#me $>$ rdf : type foaf : Person

it cannot be assumed, that < http : //www.nicola.it/foaf.rdf#me > is not an ex : Student also. SQL would return false when asking whether the resource is a student. With the OWL entailment the result is just unknown.

The UNA and OWA are fundamental principles of the Semantic Web. Within the Semantic Web it is always assumed that local information is not complete. There may be some more information somewhere else. Together with URIs as global identifiers for things in the real worlds and the simple RDF graph model, this core data model of the Semantic Web is very powerful and enables us to merge information from an arbitrary number of distributed resources.

2.4.3 Features of OWL

Depending on the required expressiveness of an application, there are basically three different OWL variants:

- OWL Lite
- OWL DL (Description Logic)
- OWL Full

This separation was introduced because the more expressiveness is required, the more rules have to be applied by reasoners, and worse are the resulting computational properties.

OWL Lite has the lowest formal complexity. It adds a few features to RDFS as for example equality and inequality constraints for classes and individuals or cardinality constraints for properties. Specifically, OWL Lite allows to express:

- **RDFS elements**: classes, individuals (instances), and properties; domain and range of properties, subclass and subproperty relationships, datatypes.
- **Equality/Inequality** equivalent class, property, and individual; different individuals.
- **Property characteristics**: inverse, transitive, symmetric, functional, inverse functional property relationship.
- **Restriction on quantification of property values** universal and existential quantification.
- **Cardinality restriction**: similar to quantification of property values in combination of a specified class, the cardinality can be restricted by a lower and upper bound (min/max) as well as an exact value (for example, to specify that a soccer team exactly requires 11 players to be valid).
- **Class intersection**: additional classes can be defined as the intersection of other classes.

Futher features are available in OWL Full and in OWL DL:

- **Enumerated classes**: definition of a class based on an enumeration of individuals, e.g. the class weekdays=(Monday; Tuesday; Wednesday; Thursday; Friday; Saturday; Sunday).
- **Property value restriction**: property restriction on a specific value, e.g. the class Italian is all persons that have a property country with the value Italy.
- **Disjointness of classes**: it is possible to assert the disjointness of classes.
- **Set-based class definition**: definition of a class based on set-combination of other defined classes (union, intersection, complement).

OWL DL was designed for maximal expressiveness while retaining computational complexity and decidability.

OWL Full does not add any restrictions to the available language constructs (for example, classes can be instances of other classes at the same time which is not allowed in OWL DL). It provides the maximum expressiveness but does not guarantee decidability.

2.5 SPARQL Protocol and RDF Query Language

In the Semantic Web, querying RDF documents and OWL ontologies is done mainly use same font for SPARQL. SPARQL was made a standard by the RDF Data Access Working Group (DAWG) of the World Wide Web Consortium (W3C), and is recognized as one of the key technologies of the Semantic Web. On 15 January 2008, SPARQL 1.0 became an official W3C Recommendation [Prud'hommeaux and Seaborne, 2008], and SPARQL 1.1 in March, 2013 [Garlik and Seaborne, 2013].

SPARQL can be used to express queries across diverse data sources, whether the data is stored natively as RDF or viewed as RDF via middleware. SPARQL contains capabilities for querying required and optional graph patterns along with their conjunctions and disjunctions. SPARQL also supports aggregation, subqueries, negation, creating values by expressions, extensible value testing, and constraining queries by source RDF graph. The results of SPARQL queries can be result sets or RDF graphs [Garlik and Seaborne, 2013].

In this section, we present the foundations of SPARQL.

2.5.1 Anatomy of a SPARQL Query

SPARQL has an SQL-like syntax. In this section, we present the standard W3C recommended syntax. The anatomy of a SPARQL query [Domingue et al., 2011] is shown in Figure 2.3.

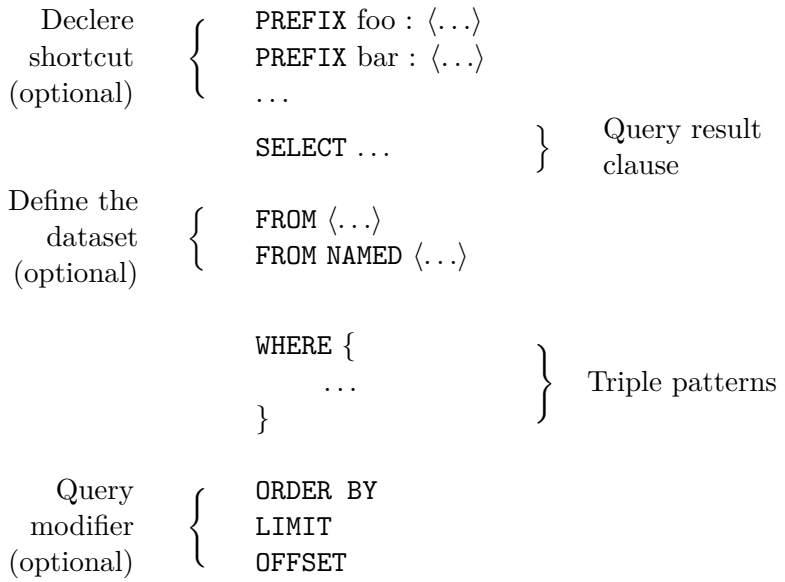


Figure 2.3: The anatomy of a SPARQL query [Domingue et al., 2011]

In the following, we briefly discuss the different parts of a query. For a detailed introduction to SPARQL see [Garlik and Seaborne, 2013].

The optional PREFIX declarations introduce short-cuts for long IRIs as normally done when working with XML namespaces.

The query result clause specifies the form of the results. A SPARQL query can take four forms: SELECT, ASK, CONSTRUCT and DESCRIBE.

- SELECT queries provide answers in a tabular form as if the SPARQL query were a SQL query executed against a relational database.
- ASK form checks whether the SPARQL endpoint can provide answers at least one result; if it does, the answer to the query is YES, otherwise the answer is NO;
- The CONSTRUCT form is similar to the SELECT form, but it provides the answer to the query as an RDF graph;
- The DESCRIBE form returns a single result RDF graph containing RDF data about resources. This data is not prescribed by a SPARQL query, where the query client would need to know the structure of the RDF in the data source, but, instead, is determined by the SPARQL query processor.

The optional set of FROM and FROM NAMED clauses define the dataset against which the query is executed.

The WHERE clause is the core of a SPARQL query. It is specified in the terms of a set of triple patterns. As extensively explained in he following sections, these triple patterns are used to select the triples composing the result.

The set of optional query modifiers operate over the triples selected by the WHERE clause, before generating the result. As in SQL, the clause ORDER BY orders the results set, the LIMIT and OFFSET allow getting results in chunks.

2.5.2 Abstract Syntax for Patterns

SPARQL is based on the notion of query patterns defined inductively from the triple patterns. A tuple $t \in UBV \times UV \times ULBV$, with V a set of variables disjoint from UBL, is called triple pattern. Triple patterns grouped together using SPARQL operators AND, UNION, OPTIONAL form query patterns (or graph patterns). In the rest of this thesis, we consider the following fragment of SPARQL.

Definition 2.5.1 (Query Pattern)**.** *A query pattern q is inductively defined as follows:*

$$q \ ::= \ t \mid q \text{ AND } q \mid q \text{ OPTIONAL } q \mid q \text{ UNION } q$$

In this thesis, we focus on SELECT queries which are the core of SPARQL.

Definition 2.5.2 (SPARQL Select Query)**.** *A SPARQL Select query is a query of the form $q(\overrightarrow{\omega})$ where $\overrightarrow{\omega}$ is a tuple of variables which are called distinguished variables and q is a query pattern. The arity of the query, $|\overrightarrow{\omega}|$, is the number of variables which appear in the result clause of that query. If all variables that appear in a query are distinguished, we can simply write q.*

Example 2.5.1 (Select query)**.** *Consider the query:*

```
SELECT   ?a ?e ?w
WHERE    {(?a, email, ?e) OPTIONAL (?a, web, ?w)}
```

that queries an RDF graph for an email ?e of a person ?a and, if available, for a webpage ?w of ?a, where ?a,?e,?w are distinguished variables and email and web are URIs.

2.5.3 Standard Semantics of SPARQL

SPARQL queries are evaluated under bag semantics, since duplicate tuples are not eliminated unless explicitly specified in the syntax using the SELECT DISTINCT construct. The reason for not eliminating duplicate tuple in SPARQL is that the values of aggregate operators, such as AVG and COUNT, depend on the multiplicity of the tuples in the graph. In the following, we present the standard "mapping-based" semantics of SPARQL [Pérez et al., 2006].

Definition 2.5.3 (Mapping-based representation of a SPARQL query solution)**.**
Let a mapping $\varrho : V \rightarrow UBL$ be a partial function that assigns RDF terms of an RDF graph to variables of a SPARQL query. The domain $\mathtt{dom}(\varrho)$ of ϱ is the subset of V over which ϱ is defined. The empty mapping $\varrho_\emptyset$ is the mapping with empty domain. Then, the mapping-based representation of a SPARQL query solution is the set Ω of mappings ϱ.

We define Σ as an infinite set of all possible mapping-sets, each of which represents a SPARQL query solution.

Example 2.5.2 (Mapping-based representation of a SPARQL query solution)**.** *Consider a graph pattern*

$$(?a, \texttt{phone}, ?p) \text{ OPTIONAL } (?a, \texttt{email}, ?e)$$

that queries the RDF graph in Figure 2.2 for a telephone number ?p of a person ?a and, if available, for an email ?e of ?a, where ?a, ?p, ?e are variables and phone *and* email *are URIs. The graph pattern solution is represented as follows:*

$$\Omega = \boxed{\begin{array}{lll} \varrho_1: & ?a \rightarrow \texttt{A}_1 & ?p \rightarrow 1111 \\ \varrho_2: & ?a \rightarrow \texttt{A}_3 & ?p \rightarrow 3333 \\ \varrho_3: & ?a \rightarrow A_4 & ?p \rightarrow 4444 \quad ?e \rightarrow \texttt{john@p.fr} \end{array}}$$

ϱ_1 and ϱ_2 are the result of the successful match of the triple pattern $(?a, phone, ?p)$ against triples $(A_1, phone, 1111)$ and $(A_3, phone, 3333)$ ϱ_3 is the result of the successful match of the triple patterns $(?a, phone, ?p)$ OPTIONAL $(?a, email, ?r)$ against triples $(A_4, phone, 4444)$ and $(A_4, email, john@p.fr)$, respectively.

Two mappings ϱ_1 and ϱ_2 are compatible when for all $x \in \texttt{dom}(\varrho_1) \cap \texttt{dom}(\varrho_2)$, $\varrho_1(x) = \varrho_2(x)$; mappings with disjoint domains are always compatibles; and $\varrho_\emptyset$ is compatible with any mapping. Let Ω_1 and Ω_2 be sets of mappings, in [Pérez et al., 2006], the following operators are defined between Ω_1 and Ω_2:

$$\begin{aligned} \Omega_1 \bowtie \Omega_2 &= \{\ \varrho_1 \cup \varrho_2 \ \mid\ \varrho_1 \in \Omega_1,\ \varrho_2 \in \Omega_2 \text{ are compatible mappings}\} \\ \Omega_1 \cup \Omega_2 &= \{\ \varrho \ \mid\ \varrho \in \Omega_1 \text{ or } \varrho \in \Omega_2\} \\ \Omega_1 \setminus \Omega_2 &= \{\ \varrho \in \Omega_1 \ \mid\ \forall \varrho' \in \Omega_2,\ \ \varrho \text{ and } \varrho' \text{are not compatible}\} \\ \Omega_1 :\bowtie \Omega_2 &= \{\ (\Omega_1 \bowtie \Omega_2) \cup (\Omega_1 \setminus \Omega_2)\ \} \end{aligned}$$

The mapping-based semantics of SPARQL is defined as a function $[\![.]\!]_G$ which takes a graph pattern expression of a SPARQL query and an RDF graph G and return a set of mappings.

The definition of $[\![.]\!]_G$ is the following:

(1) $[\![t]\!]_G = \{\varrho \ \mid\ \texttt{dom}(\varrho) = \texttt{var}(t) \text{ and } \varrho(t) \in G\}$

where $\texttt{var}(t)$ is the set of variables occurring in the triple pattern t and $\varrho(t)$ is the triple obtained by replacing the variable in t according to ϱ.

(2) $[\![gp_1 \text{ AND } gp_2]\!]_G = [\![gp_1]\!]_G \bowtie [\![gp_2]\!]_G$

(3) $[\![gp_1 \text{ OPTIONAL } gp_2]\!]_G = [\![gp_1]\!]_G :\bowtie [\![gp_2]\!]_G$

(4) $[\![gp_1 \text{ UNION } gp_2]\!]_G = [\![gp_1]\!]_G \cup [\![gp_2]\!]_G$

(5) $[\![\texttt{SELECT } (v_1, v_2, \ldots, v_n) \texttt{ WHERE } (gp)]\!]_G = \{\varrho_{|v_1,v_2,\cdots,v_3} \ \mid\ \varrho \in [\![gp]\!]_G\}$,

where $\varrho_{|v_1,v_2,\ldots,v_n}$ is a mapping such that $\texttt{dom}(\varrho_{|v_1,v_2,\ldots,v_n}) = \texttt{dom}(\varrho) \cap (v_1, v_2, \ldots, v_n)$ and $\varrho_{|v_1,v_2,\ldots,v_n}(\texttt{?x}) = \varrho(\texttt{?x})$, $\forall \texttt{?x} \in \texttt{dom}(\varrho) \cap (v_1, v_2, \ldots, v_n)$

2.5.4 Relational Algebra Semantics of SPARQL

The relational algebra semantics is an alternative semantics for SPARQL. In [Chebotko, 2008], the author proves its equivalence to the mapping-based semantics. Expressing SPARQL queries in relational algebra has several benefits, i.e. it makes a large body of work on static analysis and query optimization available to SPARQL. Based on the relational semantics for SPARQL, we will draw new perspectives for addressing the query-update problem in the conclusion Chapter 8.

Definition 2.5.4 (Relational representation of SPARQL query solution [Chebotko, 2008])**.** *Let a tuple $r : UVL \rightarrow UBL \cup \{\text{NULL}\}$ be a total function that assigns RDF terms of a RDF graph to URIs, literals and variables*[3] *of a SPARQL query, i.e a URI or a literal is mapped or to itself or to $\{\text{NULL}\}$, and a variable is mapped to an element of the set $UBL \cup \{\text{NULL}\}$, where $\{\text{NULL}\}$ denotes an undefined or unbound value, then, the relational representation of a SPARQL query solution is a set R of tuples r or simply a relation R. The schema of R, denoted as $\xi(R)$, is the subset of UBL over which each tuple $r \in R$ is defined; abusing the notation, we denote a tuple schema as $\xi(r) \equiv \xi(R)$ for all $r \in R$. We defined $\mathcal{R}$ as an infinite set of all possible relations, each of which represents a SPARQL query solution.*

Example 2.5.3 (Relational representation of a SPARQL query solution)**.** *Following the example 2.5.2, consider the same graph pattern*

$$(?a, phone, ?p) \text{ OPTIONAL } (?a, email, ?e)$$

Its solution over the RDF graph of Figure 2.2, is represented as follows.

R =

$\xi(R)$:	?a	phone	?p	email	?e
r_1:	A_1	phone	1111	NULL	NULL
r_2:	A_2	phone	3333	NULL	NULL
r_3	A_4	phone	4444	email	john@p.fr

Figure 2.4: Relational representation of a query

To relate the relational representation and the mapping-based representation, we use an interpretation function Λ as follows.

Definition 2.5.5 (Interpretation function Λ)**.** *We define the interpretation function $\Lambda : \mathcal{R} \rightarrow \Sigma$ as the function that takes a relation $R \in \mathcal{R}$ and returns a mapping-set $\Omega \in \Sigma$, such that each tuple $r \in R$ is assigned a mapping $\varrho \in \Omega$ in the following way: if $r \in \xi(R)$, $x \in V$ and $r(x)$ is not NULL, then $x \in dom(\varrho)$ and $\varrho(x) = r(x)$.*

Example 2.5.4 (Interpretation function Λ)**.** *. Given the solution Ω from the example 2.5.2 and the solution R from Example 2.5.3, one can verify that $\Lambda(R) \equiv \Omega$*

R

?a	phone	?p	email	?e
A_1	phone	1111	NULL	NULL
A_2	phone	3333	NULL	NULL
A_4	phone	4444	email	john@p.fr

$\xrightarrow{\Lambda}$

Ω

ϱ_1 :	$?a \rightarrow A_1$	$?p \rightarrow 1111$	
ϱ_2 :	$?a \rightarrow A_3$	$?p \rightarrow 3333$	
ϱ_3 :	$?a \rightarrow A_4$	$?p \rightarrow 4444$	$?e \rightarrow$ john@p.fr

[3]Blank nodes act as variable [Garlik and Seaborne, 2013]

Before we define the relational algebra based semantics of `SPARQL`, we need to introduce the following notations:

- R, R_1, R_2 and R_3 denote relations
- $\xi(R)$ denotes the schema of a relation R
- $\bowtie$ denotes an inner join
- $:\bowtie$ denotes a left outer join
- $\cup$ denotes an union
- $\uplus$ denotes an outerunion
- $\setminus$ denotes a set difference
- ρ, σ and π denote renaming, selection and projection operators of the relational algebra, respectively.

In additional, we have a relational operator $\dagger$ and the two auxiliary functions, ***genCond*** and ***genPR***. The relational operator $\dagger$ is defined as follows.

Definition 2.5.6 (Relational operator $\dagger$ [Chebotko et al., 2009])**.** *Given a relation R with schema $\xi(R)$, two distinct relational attributes $a, b \in \xi(R)$, and a relational attribute $c \notin \xi(R) \setminus \{a, b\}$, the relational operator $\dagger_{(a,b)\to c}(R)$ merges attributes a and b of relation R into one single attribute c in the following way: for each tuple $r \in R$, if $r(a)$ is not* NULL *then $r(c) = r(a)$, else $r(c) = r(b)$.*

Example 2.5.5. *Consider the following evaluation of $\dagger_{(a,b)\to c}(R)$ baed on Definition 2.5.6*

$\dagger_{(a,b)\to c}$

?a	?b	?x
a1	b1	x1
NULL	b2	x2
NULL	NULL	x3

$=$

?c	?x
a1	x1

$\cup$

?c	?x
b2	x2
NULL	x3

$=$

?c	?x
a1	x1
b2	x2
NULL	x3

Figure 2.5: Relational operator $\dagger$

In [Chebotko et al., 2009], it is shown that $\dagger$ can be derived from existing relational operators.

$$\dagger_{(a,b)\to c}(R) = \rho_{a\to c}\pi_{\xi(R)\setminus\{b\}}(\sigma_{a \text{ is NOT NULL}}(R)) \bigcup \rho_{b\to c}\pi_{\xi(R)\setminus\{a\}}(\sigma_{a \text{ is NULL}}(R))$$

Further, we extend the definition of the $\dagger$ operator to multiple attribute pair merging.

Definition 2.5.7 (Extended relational operator † [Chebotko et al., 2009]). *Given a relation R with schema $\xi(R)$, n pairs (a_1, b_1), (a_2, b_2), ..., (a_n, b_n), where $a_1, b_1, a_2, b_2, \ldots, a_n, b_n \in \xi(R)$ are all distinct relational attributes, and n distinct relational attributes $c_1, c_2, \ldots, c_n \notin \xi(R) \setminus \{a_1, b_1, a_2, b_2, \ldots, a_n, b_n\}$, the relational operator $\dagger_{(a_1,b_1)\rightarrow c_1,(a_2,b_2)\rightarrow c_2,(a_n,b_n)\rightarrow c_n})(R)$ is defined recursively as:*

$$\dagger_{(a_1,b_1)\rightarrow c_1,(a_2,b_2)\rightarrow c_2,(a_n,b_n)\rightarrow c_n}(R) = \dagger_{(a_1,b_1)\rightarrow c_1}\Big(\dagger_{(a_2,b_2)\rightarrow c_2,(a_n,b_n)\rightarrow c_n}(R)\Big).$$

The two auxiliary functions, genCond and genPR, are shown in Figure 2.6.

```
Function genCond
Input: triple pattern tp, RDF triple t
Output: boolean expression cond which is true iff
tp matches an RDF triple t
Begin
        cond = (tp.sp∈ V ∨ tp.sp=t.s)
        cond += ∧(tp.pp∈  V ∨ tp.pp=t.p) ∧
        cond += ∧(tp.op∈ V ∨ tp.op=t.o)
        If (tp.sp = tp.pp) then cond+=∧(t.s=t.p) End If
        If (tp.sp = tp.op) then cond+=∧(t.s=t.o) End If
        If (tp.op = tp.pp) then cond+=∧(t.o=t.p) End If
Return cond
End Function


Function genPR
Input: triple pattern tp, relation R
Output: relational algebra expression which projects
only those attributes of relation R with schema ξ(R)=
(s,p,o)that correspond to distinct tp.sp,tp.pp and tp.op
and renames the projected attributes as s→tp.sp and
p →tp.pp and o→tp.op
Begin
        project-list = s
        rename-list = s→ tp.sp
        If (tp.pp≠tp.sp) then
                project-list += p, rename-list = p→ tp.pp;
        End If
        If (tp.op≠tp.sp and tp.op≠tp.pp) then
                project-list += o, rename-list = o→ tp.op;
        End If
Return ϱ_rename−list φ_project−list(R)
End Function
```

Figure 2.6: Functions genCond and genPR [Chebotko et al., 2009]

Function genCond. Given a triple tp, function genCond generates a boolean expression which is evaluated to **true** if and only if tp matches an RDF triple t. The boolean expression ensures that either $tp.sp$ is a variable and thus can match any RDF term or $tp.sp = t.s$; similar conditions are introduces for $tp.pp$, and $tp.op$. Also, if $tp.sp = tp.pp$, then for tp to match t, it must be true that $t.s = t.p$; similarly for the cases when $tp.sp = tp.op$ and $tp.op = tp.pp$.

Function genPR. Let relation R with schema $\xi(R) = (s, p, o)$ store the subset of triples of G that match triple pattern tp. We define function *genPR* that given a triple pattern tp, generates a relational algebra expression which projects only those attributes of relation R that correspond to distinct $tp.sp$, $tp.pp$, $tp.op$ and renames the projected attributes as $s \rightarrow tp.sp$, $p \rightarrow tp.pp$, $o \rightarrow tp.op$. $R.s$ is always projected and renamed into $tp.sp$, $R.p$ is projected and renamed $tp.pp$ if $tp.pp \neq tp.sp$, and $R.o$ is projected and renamed into $tp.op$ if $tp.op \neq tp.sp$ and $tp.op \neq tp.pp$. This projection procedure ensures that, after attribute renaming, the schema of the resulting relation does not have duplicate attribute names.

The relational algebra based semantics of SPARQL is defined as a function eval which takes a graph pattern expression or a SPARQL query and returns a resulting relation. In Figure 2.7, eval is defined as a set of premise-conclusion rules.

$$\frac{R(s,p,o) = \{(t.s, t.p, t.o) | t \in G \ \wedge \ \texttt{genCond}(tp, t)) \qquad R_2 = \texttt{genPR}(tp, R)}{\texttt{eval}(tp, G) = R_2} \tag{1}$$

$$\frac{\begin{array}{c} R_1 = \texttt{eval}(gp_1, G) \qquad R_2 = \texttt{eval}(gp_2, G) \\ R_3 = \dagger_{\gamma(a_i)}(R_1 \bowtie_{\bigwedge_{[a_i | a_i \in \xi(R_1) \cap \xi(R_2)](R_1.a_i = R_2.a_i \ \vee \ R1.a_i \ is \ \text{NULL} \ \vee \ R_2.a_i \ is \ \text{NULL})}} R_2) \end{array}}{\texttt{eval}(gp_1 \ \text{AND} \ gp_2, G) = R_3} \tag{2}$$

$$\frac{\begin{array}{c} R_1 = \texttt{eval}(gp_1, G) \qquad R_2 = \texttt{eval}(gp_2, G) \\ R_3 = \dagger_{\gamma(a_i)}(R_1 :\bowtie_{\bigwedge_{[a_i | a_i \in \xi(R_1) \cap \xi(R_2)](R_1.a_i = R_2.a_i \ \vee \ R1.a_i \ is \ \text{NULL} \ \vee \ R_2.a_i \ is \ \text{NULL})}} R_2) \end{array}}{\texttt{eval}(gp_1 \ \text{OPTIONAL} \ gp_2, G) = R_3} \tag{3}$$

where $\gamma(a_i) = [(R_1.a_i, R_2.a_i) \ \rightarrow \ a_i \mid a_i \in \xi(R_1) \cap \xi(R_2)]$

$$\frac{R_1 = \texttt{eval}(gp_1, G) \qquad R_2 = \texttt{eval}(gp_2, G) \qquad R_3 = R_1 \cup R_2}{\texttt{eval}(gp_1 \ \text{UNION} \ gp_2, G) = R_3} \tag{4}$$

$$\frac{R = \texttt{eval}(gp, G)}{\texttt{eval}(\texttt{SELECT} \ (v_1, v_2, \ldots, v_n) \ \texttt{WHERE} \ (gp), G) = \pi_{v_1, v_2, \ldots, v_n} R} \tag{5}$$

Figure 2.7: Relational algebra based semantics of SPARQL [Chebotko et al., 2009]

2.5.5 A Survey on Complexity of SPARQL Evaluation

A fundamental issue in every query language is the complexity of query evaluation and the influence of each component of the language in this complexity. We discuss several fragments of SPARQL built incrementally and present complexity results for each such fragment.

As it is customary when studying the complexity of the evaluation problem for a query language [Vardi, 1982], we consider its associated decision problem. We denote this problem by evaluation and we define it as follows:

```
Input:  An RDF graph G, a graph pattern P and a mapping ϱ
Output: Does ϱ ∈ [[P]]_G?
```

Starting with the graph pattern expression constructed by using only the operators AND, the complexity of evaluation is $\Theta(\mid P \mid \cdot \mid G \mid)$ [Pérez et al., 2006], where $\mid G \mid$ (resp. $\mid P \mid$) is the size of G (resp. P).

The complexity of evaluation rises to NP-Complete [Pérez et al., 2006], when the AND-fragment is extended with the UNION operator.

The complexity of the full SPARQL is PSPACE-Complete [Pérez et al., 2006]. The OPTIONAL operator was identified as one of the main sources of complexity. Indeed, it was shown in [Schmidt et al., 2010] that the PSPACE-Complete of SPARQL query evaluation holds even if we restrict SPARQL to the AND and OPTIONAL operator.

2.6 SPARQL Update

SPARQL UPDATE [Garlik and Seaborne, 2013] is intended to be a standard language for specifying and executing updates to RDF graphs in a Graph Store.

The reuse of the SPARQL syntax, in both style and detail, reduces the learning curve for developers and reduces implementation costs. SPARQL UPDATE supports two categories of update operations on a Graph Store:

Graph Update - addition and removal of triples from one of the graphs in the Graph Store

Graph Management - creation and deletion of graphs within the Graph Store

Multiple operations can be packed into requests. The operations of a requests are executed in the order given.

Graph Update. Graph update operations change existing graphs in the Graph Store but do not explicitly delete nor create them.

`SPARQL UPDATE` provides the following graph update operations:

- `INSERT DATA` - this operation adds some triples, given inlined in the request, into a graph. For example:

  `INSERT DATA` A_5 `name Nicola`

- `DELETE DATA` - this operation removes some triples, given inlined in the request, if the respective graph contains these triples. For example:

  `DELETE DATA` A_5 `name Nicola`

- `DELETE/ INSERT` - these actions consist of groups of triples to be deleted and groups of triples to be added.
- `LOAD` - this operation reads the contents of a document representing a graph and adds it as new graph in the Graph Store.
- `CLEAR` - this operation removes all the triples in (one or more) graphs.

Graph Management. Graph management operations allow creating, destroying, moving and copying named graphs in the Graph Store, or adding the contents of one graph to another.

`SPARQL UPDATE` provides these graph management operations:

- `CREATE` - this operation creates a new graph in stores that support empty graphs.
- `DROP` - this operation removes a graph and all of its contents.
- `COPY` - this operation modifies a graph to contain a copy of another.
- `MOVE` - this operation moves all of the data from one graph into another

2.6.1 Abstract Syntax for `DELETE/INSERT` Operations

In this thesis, we focus mostly on `DELETE/INSERT` operations over a graph that constitute the fundamental pattern-based actions for graph updates. The specification of triples is based on quad patterns.

Definition 2.6.1 (Quad Pattern). *A quad pattern q_u is inductively defined as follows:*

$$q_u ::= tp \mid q_u \text{ AND } q_u \mid q_u \text{ UNION } q_u \mid q_u \text{ OPTIONAL } q_u$$

where $tp \in UBV \times UV \times UBLV$. A quad pattern is a query pattern that does not allow blank nodes2 in a `DELETE` *clause.*

The `DELETE/INSERT` operation can be used to remove or add triples from/to a graph based on bindings for a quad pattern specified in a `WHERE` clause. This operation identifies data with the `WHERE` clause, which will be used to compute solution sequences of bindings for a set of variables. The bindings for each solution are then substituted into the `DELETE` template to remove triples, and then in the `INSERT` template to create new triples. If any solution produces a triple containing an unbound variable or an illegal `RDF` construct, such as a literal in a subject or predicate position, then that triple is not included when processing the operation.

Definition 2.6.2 (`SPARQL UPDATE` operation [Ahmeti and Polleres, 2013]). *Let q_d and q_i, and q_w be quad patterns, then $u(q_d, q_i, q_w)$, an update operation, has the form:*

$$u(q_d, q_i, q_w) = \textit{DELETE } q_d \quad \textit{INSERT } q_i \quad \textit{WHERE } q_w$$

Example 2.6.1 (`DELETE/INSERT` query). *Consider the query:*

```
DELETE  {?a phone 1111}
INSERT  {?a phone 5555}
WHERE   {?a phone 1111}
```

that updates the default graph, changing all phone numbers `1111` *into* `5555`

If the `DELETE` clause is omitted, then the operation, denoted by $u(\_, q_i, q_w)$, only inserts data. If the `INSERT` clause is omitted, then the operation, denoted by $u(q_d, \_, q_w)$, only removes data.

2.6.2 Semantics of DELETE/INSERT Operations

Intuitively, the semantics of the execution of $u(q_d, q_i, q_w)$ on G, denoted as $G_{u(q_d,q_i,q_w)}$ or simply $u(G)$ is defined by interpreting both q_d and q_i as "templates" to be instantiated with the solution $\Omega = [\![q_w]\!]_G$

Definition 2.6.3 (Naïve Update Semantics [Ahmeti and Polleres, 2013]). *Let G be a triple store, and $u(q_d, q_i, q_w)$ an update operation, then, a naïve update of G with $u(q_d, q_i, q_w)$, denoted $G_{u(q_d,q_i,q_w)}$ is defined as $(G \setminus A_d) \cup A_i$, where $A_d = \bigcup_{\varrho \in [\![q_w]\!]_G} \varrho(q_d)$ and $A_i = \bigcup_{\varrho \in [\![q_w]\!]_G} \varrho(q_i)$.*

We have introduced the preliminaries that are used in the development in this thesis. In the rest of the thesis, we focus on the static analysis for SPARQL. There are some important challenges regarding querying in the Semantic Web, i.e. optimization of queries over large dataset, satisfiability test of queries evaluated at remote endpoints, materialized query evaluations in case of queries containment. To tackle these challenges, static analysis of queries is absolutely essential. In the following chapters, the task of static analysis of SPARQL queries has been addressed by reductions to satisfiability test in a modal logic.

3

Static Analysis of SPARQL Queries

This chapter reports static analysis techniques for `SPARQL` *(the standard language for querying Semantic Web data). Static analysis is a core task in query optimization and knowledge base verification. We review static analysis techniques of the literature, considering the peculiarities of* `SPARQL` *and, in particular we concentrate on the containment problem.*

Contents

3.1 Introduction **30**

3.2 Query Containment Problem **30**

3.2.1 Techniques Overview 32

3.2.2 A Survey on the Complexity of Query Containment 38

3.2.3 SPARQL Containment Problem 39

3.3 Conclusion **41**

3.1 Introduction

Static Analysis is a conflated term and serves as an umbrella for the varied approaches to analyse source code before it is executed [Nielson et al., 1999]. These approaches vary because not all languages have the same features, i.e. some languages are strongly typed while others are weakly typed.

Static Analysis techniques help developers gain a better understanding of their programs, i.e. by detecting potential errors before executing them. Using Static Analysis techniques, developers can discover errors like semantic errors, type errors, memory errors, logic errors, interface and include errors and various security errors associated with each.

In the context of data complexity, Static Analysis is a fundamental task in query optimization and knowledge based verification. For example, queries that are evaluated many times, deserve to be optimized towards fast evaluation. Even queries that are evaluated once on very large data deserve optimizations.

Many optimization tasks rely on three simple questions:

- Does query q ever produce a non-empty result?
- Does query q_1 always produce the same result as query q_2?
- Does query q_1 always produce a subset of the results of query q_2?

We refer to the first question as the satisfiability problem (or non-emptiness problem), to the second as the equivalence problem, and to the third as the containment problem.

Containment, equivalence and satisfiability problems are well studied for relational database query languages. Equivalence and satisfiability can be derived from containment, thus, we mainly concentrate on the containment problem.

In this chapter, we present an overview of the query containment problem and techniques to solve it. We conclude with a complexity survey of the problem for different query language fragments.

3.2 Query Containment Problem

The basic form of reasoning on queries is checking containment. Query containment is defined as the problem of determining if the result of a query is included in the result of another query for any given dataset.

For example, checking containment is a relevant task in the following context. Let q be a query, G be a dataset, $V = q(G)$ be a view of the answers of q over G and let q' be another query, then q' can be answered using V provided q' is contained in q. In this scenario, the possibility of answering q' over the view V is much less costly than answering over G

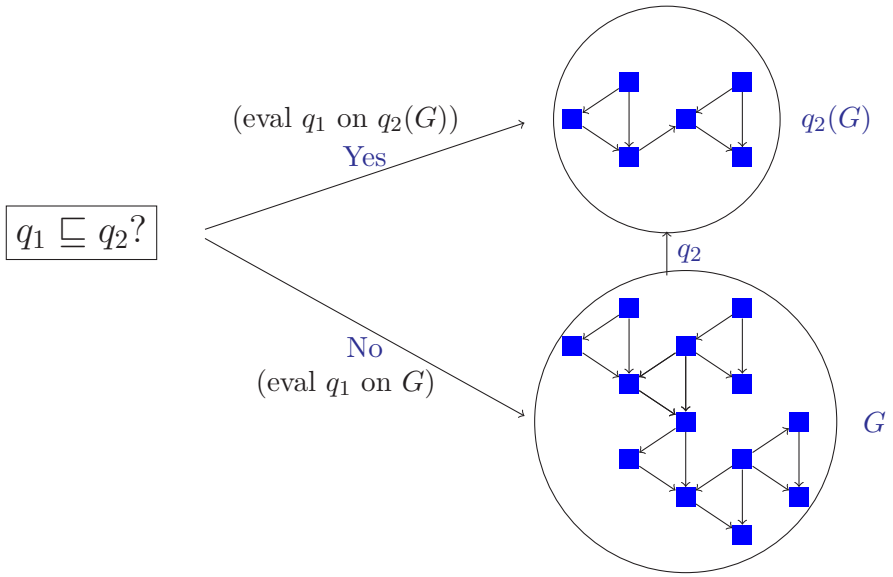


Figure 3.1: Query answering using view

Query containment is crucial in several other contexts, such as information integration [Ullman, 1997, Calvanese et al., 1998, Millstein et al., 2003], knowledge base verification [Levy and Rousset, 1996] and query optimization [Abiteboul et al., 1994, Aho et al., 1979a].

Data integration Data integration is the problem of combining data residing at different sources, and providing the user with a unified view of these data. Several recent works point out that query containment is essential for this purpose, as [Ullman, 1997, Calvanese et al., 1998, Millstein et al., 2003]. A number of ideas concerning information-integration tools can be thought of as constructing answers to queries using views that represent the capabilities of information sources. In [Ullman, 1997], query containment algorithms connect to information integration via a concept called "synthesizing queries from views", an idea originally studied by [Yang and Larson, 1987, Goldstein and Larson, 2001].

Query optimization. Query optimization is one of the central topics of database systems. It aims at improving the efficiency of query answering, and largely benefits from the possibility of performing various kinds of comparisons between query expressions. In [Aho et al., 1979a], the authors discuss the difficulty of optimizing queries based on the relational algebra operation select, project and join, and they propose a matrix, called a tableau, a useful device for representing the value of a query. Optimization of queries is couched in terms of finding a minimal tableau equivalent to a given one. The key idea in the equivalence test is what they call a containment mapping, defined as a mapping from the rows of one tableau to another that preserves distinguished variables and constants and does not map

any symbol to two different symbols. Static analysis and optimisation of SPARQL queries have received significant attention in recent years. A key ingredient for query optimization is the availability of a comprehensive catalogue of equivalence-preserving transformation rules for SPARQL patterns.

knowledge base verification. Building complex knowledge based applications requires encoding large amounts of domain knowledge. After acquiring knowledge from domain experts, much of the effort in building a knowledge base goes into verifying that the knowledge is encoded correctly. A knowledge base is verified if it can be shown that certain constraints always hold between the inputs and the outputs. In [Levy and Rousset, 1996], the authors present an approach to reduce certain constraints in terms of query containment. This approach to the verification problem is based on showing a close relationship to the problem of query containment and the advantage of using a collection of algorithms developed for query containment.

Needless to say, query containment is undecidable if the expressive power of the query language is not limited. Suitable query languages have to be designed for retaining decidability. The same is true in databases, where the notion of conjunctive query is the basic one in the investigation on reasoning on queries [Chandra and Merlin, 1977].Most of the results on query containment concern conjunctive queries and their extensions.

3.2.1 Techniques Overview

The problem of syntactically characterizing containment and equivalence of conjunctive queries was solved in the late 1970s by Chandra and Merlin [Chandra and Merlin, 1977] and by the tableaux work of Aho, Sagiv and Ullman [Aho et al., 1979a]. It is known that, for conjunctive queries, query answering and containment are equivalent problems. There is a well-known reduction of query containment for conjunctive queries to the problem of querying a database in [Chandra and Merlin, 1977], that involves asserting the query Q_2 as a database, and asking the query Q_1 in this database.

Unfortunately, these techniques do not suffice to solve the containment problem for query languages with ontologies and regular expressions, in the context of semi-structured knowledge bases. That is why for semistructured data query languages (referred as regular path queries) automata theoretic notions are often employed to address containment and other problems [Calvanese et al., 2000]. In addition to using automata, containment has been addressed by a reduction to satisfiability test. In this direction, queries are translated into formulæ in a particular logic that supports the query languages features and then the overall problem is reduced into satisfiability test. Several works exist that developed and used this technique [Calvanese et al., 1998, Genevès et al., 2007, Calvanese et al., 2008, Chekol et al., 2011, Chekol et al., 2012b, Chekol et al., 2013] which also inspired this thesis.

In the following, we review these methods in more details.

3.2.1.1 Containment Mappings

This approach has been proposed by Chandra and Merlin in [Chandra and Merlin, 1977], who provide an NP-completeness complexity for the containment of conjunctive queries.

In [Chandra and Merlin, 1977], authors restricted their attention to a fragment of the relational calculus, introducing the class of conjunctive queries CQs, that coincides with the class of queries that use selection, projection, and join only. CQs are queries of first-order logic that are built from atomic formulæ by means of conjunctions and existential quantifications only. Thus, the generic conjunctive query takes the form

$$(\exists x_1) \ldots (\exists x_k)(R_1 \wedge \ldots \wedge R_q)$$

where $R_1, \ldots, R_q$ are atomic formulæ built from the relations of the database with the variables $x_1, \ldots, x_k$ (see Appendix A).

In [Chandra and Merlin, 1977], authors introduced also two key concepts to detect the queries containment: canonical databases and the folding of conjunction queries.

Suppose $Q = (\exists x_1) \ldots (\exists x_k)\varphi$ is a conjunctive query, where φ is a conjunction of atoms of the form $R(x_i, \ldots, x_{i_r})$. The canonical database of Q, denoted by A_Q, is a database defined over a finite set of attributes $\{x_1, \ldots, x_k\}$, and for all $R(x_i, \ldots, x_{i_r})$ atom of φ, A_Q contains the tuple $(x_i, \ldots, x_{i_r})$.

Based on the notion of canonical databases of a conjunctive query, the operation of folding is defined as follows. A conjunctive query Q' is a folding of an other conjunctive query Q, if the set of variable of Q' is a subset of the set of variable Q, and there exists a homomorphism $\phi : A_Q \rightarrow A_{Q'}$ such that $A_{Q'} = \phi(A_Q)$. Note that Q' is contained in Q [written $Q_1 \subseteq Q_2$] and in that case Q' is a subquery of Q.

Conjunctive queries can be represented using different notations that are equivalent. In the following examples, we use the most common Datalog rules (deductive databases notation), à la [Ullman, 1997]. For those who are not familiar with rules, a briefly introduction of the Datalog notation for CQs is reported in Appendix A.

Example 3.2.1 (Containment mappings)**.** *Let Q_1 and Q_2 be:*

$$Q_1 : p(X, Z) : -a(X, Y), a(Y, Z).$$

$$Q_2 : p(X, Z) : -a(X, U), a(V, Z).$$

the containment mapping $\phi : A_{Q_2} \rightarrow A_{Q_1}$, such that $Q_1 \subseteq Q_2$ is the follows.

$$\phi : \{X \rightarrow X,\ U \rightarrow Y,\ V \rightarrow Y,\ Z \rightarrow Z\}$$

A generalization of this method is presented in [Ramakrishnan et al., 1989]. To test whether $Q_1 \subseteq Q_2$, the following steps are required.

1. *freeze* the body of Q_1 (by turning each of its subgoals into facts in the database. That is, replace each variable in the body by a distinct constant, and treat the resulting subgoals as the only tuples in the database.
2. Apply Q_2 to this canonical database.
3. If the frozen head of Q_1 is derived by Q_2, then $Q_1 \subseteq Q_2$. Otherwise, not; in the fact the canonical database is a counterexample to the containment, since surely Q_1 derives its own frozen head from this database.

Example 3.2.2. *Let Q_1 and Q_2 be the* `CQs` *of the Example 3.2.1:*

$$Q_1 : p(X, Z) : -a(X, Y), a(Y, Z).$$

$$Q_2 : p(X, Z) : -a(X, U), a(V, Z).$$

We use integers starting at 0 as the constants that "freeze" the `CQ` *Q_1. Thus, the canonical database A_{Q_1}, constructed from Q_1 consists of two tuples $a(0, 1)$ and $a(1, 2)$, and nothing else. The frozen head of Q_1 is $p(0, 2)$. If we apply Q_2 to D, the substitution $X \to 0$, $U \to 1$, $V \to 1$, and $Z \to 2$ yields $p(0, 2)$ in the head of Q_2. Since this fact is the frozen head of Q_1, we have $Q_1 \subseteq Q_2$.*

3.2.1.2 Tableau Method

This approach has been proposed in [Aho et al., 1979a]. Authors propose tableaux as a two-dimensional representation of queries. Tableaux are similar to conjunctive queries, and every queries that use select, project and join, only, can be represented by a tableau.

A tableau is a matrix consisting of a summary, the first row of the matrix, and a set of rows, that are to be exclusively called rows (the summary is not referred to as a "row"). The columns of a tableau correspond to the attributes of the query in a fixed order. The symbols appearing in a tableau are chosen from:

- distinguished variables, usually denoted by subscripted a's;
- nondistinguished variables,usually denoted by subscripted b's;
- constants, which are drawn from the domains of the attributes;
- blanks.

The summary of a tableau may contain only distinguished variables, constants and blank. The rows of a tableau may contain variables (distinguished and nondistinguished) and constants. The same variable cannot appear in two different columns of a tableau, and that a distinguished variable not appear in a column unless it also appears in the summary of that column.

Example 3.2.3 (Tableau representation of queries)**.** *Let Q_1 and Q_2 be the* `CQs` *of the Example 3.2.1:*

$$Q_1 : p(X, Z) : -a(X, Y), a(Y, Z).$$

$$Q_2 : p(X, Z) : -a(X, U), a(V, Z).$$

their tableaux are:

$T_1 =$

	X	Y	Z
	a_1		a_2
ω_1	a_1	b_1	b_2
ω_2	b_3	b_1	a_2

$T_2 =$

	X	U	V	Z
	a_1			a_2
ω_3	a_1	b_4	b_5	b_6
ω_4	b_7	b_8	b_9	a_2

Table 3.1: Tableau representation of a conjunctive query

The key idea in the containment test, is what Aho et al. in [Aho et al., 1979a] call a "containment mapping", which is defined as a mapping from rows of one tableau to another that preserves distinguished variables and constants and does not map any symbol to two different symbols.

Formally, let T_1 and T_2 be tableaux, and let h be a mapping from the rows of T_1 to the rows of T_2. h is a containment mapping if:

- For each row_i of T_1, if row_i has a distinguished variable in some column A, then $h(row_i)$ of T_2 also has a distinguished variable in column A.
- If row_i of T_1 has a constant c in column A, then $h(row_i)$ has c in column A.
- If row_i and row_j of T_1 have the same nondistinguished variable in column A, then $h(row_i)$ and $h(row_j)$ have the same symbol in that column.

The techniques of [Chandra and Merlin, 1977], can be used to show that T_1 is contained in T_2 if and only if their summaries are the same (up to renaming of distinguished variables), and there is a containment mapping from T_2 to T_1.

Example 3.2.4. *Let T_1 and T_2 be the two tableaux of the previous example 3.2.3. We show the containment of T_1 in T_2, by defining the following mapping of symbols (an homomorphism in [Chandra and Merlin, 1977]) from T_2 to T_1.*

in T_2	in T_1
a_1	a_1
b_4	b_1
b_1	b_1
b_1	b_2
b_7	b_3
b_8	b_1
b_9	b_1
a_2	a_2

Table 3.2: Mapping of symbols h from T_1 to T_2

Using the mapping of Table 3.2, we can map ω_3 to ω_1 and ω_4 to ω_2. Note that the values of the column U and V of T_2 are mapped on the same value of the column Y of T_1.

3.2.1.3 Automata Method

In [Calvanese et al., 2000], the authors address the problem of query containment in the context of semistructured knowledge bases KB. The basic querying mechanism on a KB is that of regular path queries (`RPQs`). An `RPQ` R is expressed as a regular expression or a finite automaton, and computes the set of pairs of nodes of the KB connected by a path that conforms to the regular language $\mathcal{L}(R)$ defined by R. Semistructured data are usually modelled as labeled graphs and methods for extracting information from semistructured data incorporate special querying mechanisms that are not common in traditional database system. The authors present a technique to check containment of queries for regular path queries based on the use of two-way finite automata. Differently from standard finite state automata, two-way automata are equipped with a head that can move back and forth on the input string. A transition of these kinds of automata indicates not only the new state, but also whether the head should move left, right, or stay in place. Our technique shows the power of two-way automata in dealing with the inverse operator and with the variables in conjunctive queries. In particular, we describe an algorithm that checks containment of two queries by checking nonemptiness of a two-way automaton constructed from the two queries. The algorithm works in exponential space, and therefore has the same worst-case complexity as the best algorithm known for the case of conjunctive regular path queries without inverse

The basic idea is to represent a semistructured knowledge base by several words and given a regular path query, an automata is built such that this automata accepts a word if and only if the regular expression has a non-empty answer on the knowledge base. In [Calvanese et al., 2000] authors provides `EXPSPACE` worst-case.

3.2.1.4 Reduction to Satisfiability Test

In this approach, containment is addressed by a reduction to satisfiability test. In this direction queries are translated into formulæ, in a particular logic that supports the query language features, and then the overall problem is reduced to satisfiability test. Some works are presented in [Calvanese et al., 1998, Genevès et al., 2007, Calvanese et al., 2008, Chekol et al., 2012b, Chekol et al., 2011, Chekol et al., 2012a, Chekol et al., 2013]

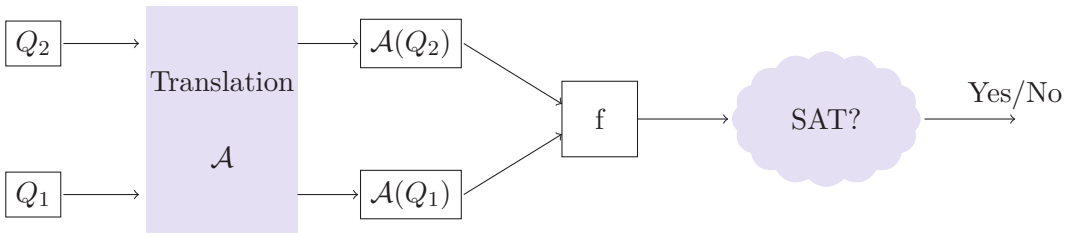


Figure 3.2: Scheme: Reduction to satisfiability test
min-imization

All these works share the following steps:

1. A target logic is chosen and it supports at least the query language features.

2. A query Q_i is encoded in the target logic; for example, by applying a translation function $\mathcal{A}$, such that $\varphi_{Q_i} = \mathcal{A}(Q_i)$;

3. Let φ_{Q_1} and φ_{Q_2} be two formulæ, encoding of the queries Q_1 and Q_2. The set-theoretic inclusion $Q_1 \subseteq Q_2$ is regarded as implication between formulæ:

$$\varphi_{Q_1} \rightarrow \varphi_{Q_2} \quad \equiv \quad \neg\varphi_{Q_1} \vee \varphi_{Q_2}$$

4. The negation of the previous formula is tested for unsatisfiability.

$$\texttt{unsat}(\varphi_{Q_1} \wedge \neg\varphi_{Q_2})$$

5. If the formula is not satisfiable then $Q_1 \subseteq Q_2$, otherwise $Q_1 \not\subseteq Q_2$

A common scheme for this approach is shown in Figure 3.2.

3.2.1.5 Bilan

We have presented four techniques for solving the query containment problem. The first two techniques syntatically characterizing containment and equivalence of conjunctive queries. Unfortunately, these techniques do not suffice to solve the containment problem for query languages in the context of semistructured knowledge base, where the basic querying mechanism, namely regular path queries, asks for all pairs of objects that are connected by a path conforming to a regular expressions. In this context, the automata method and the reduction to satisfiability test have been introduced.

In this thesis we address the query containment problem for `SPARQL`. Since `SPARQL` queries semistructured knowledge bases, we have decided to use logics to solve the query containment problem. One advantage of using a generic and well-characterized logical language, as opposed to ad-hoc methods, is extensiblity. Using this technique, if the logic chosen is powerful enough, it is possible to deal with the query containment problem for several fragments of `SPARQL`, even in presence of schemas. In the next chapters, we present the target modal logic chosen and a decision procedure for testing its satisfiability that admits efficient implementations.

3.2.2 A Survey on the Complexity of Query Containment

Needless to say, query containment is undecidable ([Trahktenbrot, 1950]), if we do not limit the expressive power of the query language.

`NP`-completeness has been established for conjunctive queries in [Chandra and Merlin, 1977] and for the union of conjunctive queries in [Sagiv and Yannakakis, 1980]. In [Klug, 1988, van der Meyden, 1992] Π_n^P-completeness of containment of conjunctive queries with inequality is proved. In [Deutsch and Tannen, 2001], `EXPSPACE`-completeness of containment of conjunctive queries with regular expression is shown and in [Florescu et al., 1998] the `PSPACE`-hardness is provide for the containment of conjunctive queries with some restriction on the regular expression. For various classes of Datalog queries with inequalities, decidability and undecidability results are presented in [Chaudhuri and Vardi, 1992] and [van der Meyden, 1992], respectively.

Fragment	Complexity
`CQs`	`NP-Complete` [Chandra and Merlin, 1977]
Union of `CQs`	`NP-Complete` [Sagiv and Yannakakis, 1980]
`CQs` with inequality	$\Pi_n^P - \texttt{Complete}$ [van der Meyden, 1992]
`CQs` with regular expression	`EXPSPACE-Complete` [Deutsch and Tannen, 2001]
First order queries	`undecidable` [Trahktenbrot, 1950]

Figure 3.3: Query containment complexity for relational fragments

3.2.3 SPARQL Containment Problem

In this section we review works that have previously established results for query languages related to SPARQL.

In [Gutierrez et al., 2011],we found a first work that explores the notion of query evaluation and of query containment for a (Datalog-style) RDF rule based query language. In particular, the authors established the NP-completeness of query containment over simple RDF graphs (graphs that do not mention the RDFS vocabulary). This result is also published in the RDF semantics document [Hayes, 2004]. Despite, the query language analysed in this work is rather simple compared to SPARQL and no constraints were assumed for the problem.

In [Serfiotis et al., 2005], Serfiotis et al. address the problem of the Semantic Query Optimization (SQO).

"*The* SQO *is the process of increasing the potential for an efficient evaluation of queries by using intentional information about the contents of a database. The essential idea is to use knowledge about the data to reformulate a query into a more efficient but semantically equivalent one.*"[Serfiotis et al., 2005].

[Serfiotis et al., 2005] provides algorithms for the containment and minmization of RDFS query patterns utilizing concept and property hierarchies for the query language RQL (RDF Query Language). RQL, indeed, allows both exact and extended matching, while SPARQL allows only exact matching. The NP-completeness is established for query containment concerning conjunctive and union of conjunctive queries.

In line with this, a work in [Polleres, 2007] shows how to translate SPARQL queries into non-recursive Datalog with negation. In this work is proved that SPARQL has equal expressive power as the relational calculus [Polleres, 2007], hence the containment and equivalence are undecidable for full SPARQL. However the work does not take in consideration decision procedures for the query containment problem.

As for the relational calculus, containment and equivalence have extensively been studied for SPARQL so far, i.e in [Chekol et al., 2011, Chekol et al., 2012a, Letelier et al., 2012, Chekol et al., 2012b, Chekol et al., 2013, Pichler and Skritek, 2014] for interesting fragments of SPARQL.

A complexity survey for the query containment problem for SPARQL is reported in Table 3.3.

Graph pattern	Complexity
AND	NP-Complete [Chandra and Merlin, 1977]
UNION-AND	NP-Complete [Sagiv and Yannakakis, 1980]
well-designed (UNION-)OPTIONAL-AND	Π_2^P [Letelier et al., 2012]
UNION-OPTIONAL-AND	undecidable [Trahktenbrot, 1950]

Table 3.3: Query containment complexity for SPARQL fragments

The line of research found in[Chekol et al., 2011, Chekol et al., 2012a, Chekol et al., 2012b, Chekol et al., 2013] also provide some complexity results. They mainly provide a 2EXPTIME upper-bound for the containment problem for a fragment of SPARQL queries without the optional operator, but in the presence of RDFS/OWL constraints. The work found in [Chekol et al., 2013] provides a benchmark for the static analysis of SPARQL queries (without the optional operator), and report experimental results that, overall, confirm that SPARQL containment solvers are still in early stage. Their results, in presence of schemas, are resumed in Table 3.4

Graph Pattern	Schema Language	Complexity
AND	$\mathcal{ALCH}$	2EXPTIME
AND-UNION	$\mathcal{ALCH}$	2EXPTIME
AND-UNION	RDFS-entailment	EXPTIME
AND-UNION	OWL/$\mathcal{ALCH}$-entailment	EXPTIME
PSPARQL	$\mathcal{ALCH}$	2EXPTIME

Table 3.4: Containment complexity for SPARQL fragments with schema [Chekol, 2012]

The line of research followed in [Letelier et al., 2012, Pichler and Skritek, 2014] concentrate on complexity results, and provide complexity bounds for the containment problem with a variety of query language fragments. They start to study the fundamental fragment of well-designed SPARQL restricted to the AND and OPTIONAL operator and then extended this fragment with the UNION operator (outside the scope of other operators) and/or the projection π. Their comprehensive complexity analysis is reported in Table 3.5. We use the following notations: wd to denote the fragment of the well-designed queries, $\cup$ the UNION operator and π the projection. For instance, the entry in the last line, first column states that the containment problem of $Q_1 \subseteq Q_2$ is NP-Complete if Q_1 is allowed to use AND and OPTIONAL together with UNION and projection, while Q_2 is restricted to AND and OPTIONAL.

$Q_1 \setminus Q_2$	wd	$wd + \{\cup\}$	$wd + \{\pi\}$	$wd + \{\cup, \pi\}$
wd	NP-Complete	Π_2^P − Complete	undecidable	undecidable
$wd + \{\cup\}$	NP-Complete	Π_2^P − Complete	undecidable	undecidable
$wd + \{\pi\}$	NP-Complete	Π_2^P − Complete	undecidable	undecidable
$wd + \{\cup, \pi\}$	NP-Complete	Π_2^P − Complete	undecidable	undecidable

Table 3.5: Containment complexity for well-designed SPARQL fragments [Pichler and Skritek, 2014]

Note, in Table 3.5, containment problem displays a surprising asymmetry. For instance, testing $Q_1 \subseteq Q_2$ is NP-Complete if Q_1 uses projection and Q_2 is restricted to AND and UNION. However, the problem becomes undecidable if we want to test $Q_2 \subseteq Q_1$.

3.3 Conclusion

These problems, containment, equivalence, and satisfiability, are collectively called static analysis of queries.

Equivalence and satisfiability problems can be derived from containment, thus, we mainly concentrate on the containment problem.

Query containment is a well-studied problem for relational database query languages and has been addressed for different query languages, since.

We have reported four methods to detect query containment:

- the containment mappings method
- the tableaux method
- the automata method
- the reduction to satisfiability test

In this thesis we address the query containment problem for `SPARQL` using the latter method. One advantage of using a generic and well-characterized logical language, as opposed to ad-hoc methods, is extensiblity. Using this technique if the logic chosen is powerful enough, it is possible to deal with the query containment problem for several fragment of `SPARQL`, even in presence of schemas. In the next chapters, we present the target modal logic chosen and a decision procedure for testing its satisfiability that admits efficient implementations.

4

An Overview of the Modal Logic

This chapter introduces modal logic. Modal logic is the study of modal propositions and the logical relationships that they bear to one another. The most well-known modal propositions are propositions about what is necessarily the case and what is possibly the case. Hence, modal logic extends classical logic with the ability to express not only "p is true", but also statements like "p is possibly true" or "p is necessarily true". In this chapter, we present the syntax and semantics for several modal logics and discuss their satisfiability problem.

Contents

4.1 Introduction **44**

4.2 History and Motivations **45**

4.2.1 Motivations 48

4.3 Basic Modal Logic K and its Extensions **49**

4.3.1 Modal Logic K 49

4.3.2 Modal Logic K with Multiple Modalities: $\mathtt{K}_n$ 51

4.3.3 Modal Logic K with Multiple and Backward Modalities: $\mathtt{K}_{(n,back)}$ 52

4.4 Mu-calculus **53**

4.4.1 Fixpoints as Recursion 53

4.4.2 Syntax 54

4.4.3 Semantics 54

4.4.4 Notion of Satisfiability 55

4.4.5 Small Model Property 55

4.4.6 Alternation-Free Mu-Calculus: $\mathtt{AF}_\mu$ 55

4.5 State-of-the-art of the Solver **56**

4.5.1 KBDD Solver 56

4.5.2 $\mathtt{AF}_\mu$ Solver 57

4.5.3 Tree Solver 58

4.5.4 MSPASS 58

4.6 Conclusion **59**

4.1 Introduction

The term "modal logic" denotes a broad family of languages that vary in expressiveness, complexity, and application areas.

A *modality* qualifies the truth of a judgment. Necessarily and possibly are the most important and best known modalities. They are called "alethic" modalities, from the Greek word for "truth". In traditional terminology, `Necessarily p` is an "apodeictic judgment", `Possibly p` is a "problematic judgment", and `p` by itself, an "assertoric judgment" [Fitting and Mendelsohn, 1988].

For example, the following are all modal judgments:

- It is possible that it will snow tomorrow.
- It is possible for humans to travel to Pluto
- It is necessary that either it is snowing here now or it is not snowing here now

The operators *it is possible that* and *it is necessary that* are "modalities". These qualifiers indicate the mode in which the proposition is said to be true. There are other modalities, however. For example, *it once was the case that*, *it will once be the case that*, and *it ought to be the case that*. Not all modalities give rise to modal logics. Our investigation is grounded in judgments to the effect that certain modal propositions logically imply others.

In this thesis, we treat modal logics with the ability to express only the possibly and the necessarily of propositions. Modal logic has been applied to numerous areas of computer science, including artificial intelligence, program verification, hardware verification, database theory and distributed computing. In these applications, deciding satisfiability of a modal formula is one of the most basic reasoning problems, and various techniques have been developed and optimized to decide it.

The satisfiability problem is the problem of finding out whether a formula is satisfiable, i.e., whether there exists a model and a state of the model where the formula is **true**. Many typical reasoning tasks for modal logic can be reduced to the satisfiability problem. When speaking of the decidability or complexity of a logic, one usually means the decidability or complexity of its satisfiability problem.

In this thesis, we are especially interested in the satisfiability of the smallest normal logic `K`, that is `PSPACE-Complete` [Ladner, 1977, Stockmeyer, 1976, Halpern and Moses, 1992], and of a fragment of the propositional modal μ-calculus (a.k.a. $\mathtt{AF}_\mu$), that admits exponential time satisfiability solvers [Tanabe et al., 2008, Genevès, 2008].

4.2 History and Motivations

The modern interest in modal logic begins with Aristotle [1]. In *De Interpretatione*, Aristotle noticed not simply that necessity implies possibility (and not vice versa), but that the notions of necessity and possibility were interdefinable.

- The proposition φ *is possible* may be defined as: *not-*φ *is not necessary*
- The proposition φ *is necessary* may be defined as: *not-*φ *is not possible*

Aristotle wished to formalize the logical relationships between what is, what is necessary, and what is possible. Unfortunately, his treatment of modality suffered from a number of confusions, and modal logic was dismissed as a failure. Philosophers after Aristotle added other interesting observations to Aristotle's catalog of implications. We found contributions in works performed by the Megarians [2] and the Stoics [3], among others. The subject was almost entirely forgotten until Lewis reconsidered it. [4]

Bull and Segerberg [Bull and Segerberg, 1984] describe the twentieth century development of modal logic along the lines of three different traditions: syntactic, algebraic and model-theoretic. The syntactic tradition is the oldest and is characterised by the lack of explicit semantics. Then we have the algebric tradition with a semantics of sorts in algebraic terms. Finally there is the model theoretic tradition, whose semantics is in terms of models.

Syntactic Tradition

The syntactic tradition started with a pioneering paper from Lewis, "*Implication and the Algebra of Logic*", published in Mind in 1912 [Lewis, 1912]. The original analysis of Lewis concerns disjunction. Consider, he says the following two propositions:

1. α = Either Mozart died, or Mozart is a pen.
2. β = Either Giulietta does not love me or she does

Both these propositions are of the form $A \vee B$

Yet, Lewis argues, there are more important differences between the two. For example, we know that α is true since we know that Mozart is dead but we know that β is true without knowing which of the disjuncts is true. Thus β exhibits a 'purely logical or formal character' and an 'independence of facts' that is lacking in α.

Lewis thinks that we can express the difference between the disjunction of α and β and he proposed to call the former 'extensional or' and the latter 'intensional or'. While the

[1] The historical information in this section are from [Kneale and Kneale, 1962, Fitting and Mendelsohn, 1988]

[2] The founder of the Megarian school was Euclides, a disciple of Socrate and elder contemporary of Plato.

[3] The founder of the Stoic school, which originates from the teaching of the Megarians, was Zeno of Citium.

[4] Kant (1724-1804) and Frege (1848-1925) believed that no more or less could be derived from the modal form of a statement φ that from φ itself. This claim has come to be seen as false.

extensional disjunction is rendered by the traditional, truth-value functional operator $\vee$, a novel sort of operatos is needed to render intensional disjuction. Lewis introduced a new symbol for it, denoted by $\boxed{\vee}$.

The same problem also concerns other connectives, as the implication. The extensional kind of implication is the material implication of ordinary truth value $\rightarrow$, and the intensional kind of implication, called `strict implication`, denoted by the 'fish-hook' $\prec$. This operator is not found, nor definable, in classical logic, and so Lewis proposed to develop a calculus of strict implication [Lewis, 1914].

The method chosen by Lewis in his search for a calculus of strict implication was the axiomatic one. There is no formal semantics in Lewis' work; semantics is left at an informal level. His method can be described as follows.

- A formal language is defined, including symbols to deal with the observed features you want to treat;
- at the beginning, the formulæ have no meaning at all;
- From the set of all formulæ, a number of them are somehow selected for testing against one's intuition. Some are accepted as valid, some are rejected as non-valid, some may be difficult to decide.
- The valid one give a finite description of an infinite scene.

In this syntactic tradition many axiomatization were born: Lewis gave us five of them, namely, the so-called system S1 through S5.

Of modal logicians working in the same vein as Lewis, Oskar Becker and H.Von Wright are remembered. Von Write should be named the second most important author in the syntactic tradition. In his work [von Wright, 1951], Von Wright remarks that modal logic is the logic of the modes of being. In this works, he sets out to explore modal logic in a wider sense, the logic of the modes of knowledge, belief, norms and similar concepts; this wider sense of the term has since gained currency. This work marks the beginning of much work in epistemic, doxastic and deontic logic.

Algebraic Tradition

The algebraic tradition started with Łukasiewiez, around 1918. He answered the question

Is it possible to understand the modal operators in terms of simple truth-conditions?

Since there are only four unary truth-functions (identity, negation, tautology and contradiction), so if necessity or possibility (denoted by $\Box$ and $\triangle$, respectively) is to be truth-functional, it would have to be one of them, which is absurd in presence of the ordinary truth values 1 (truth) and 0 (false). Łukasiewiez, with the introduction of a third truth value $\frac{1}{2}$ (possibility of (some kind)), was able to define the modal operators as unary truth functions over the enhanced set of truth values. The truth-tables of connectives are shown in Fig. 4.1.

$\wedge$	1	1/2	0
1	1	1/2	0
1/2	1/2	1/2	0
0	0	0	0

$\vee$	1	1/2	0
1	1	1	1
1/2	1	1/2	1/2
0	1	1/2	0

$\rightarrow$	1	1/2	0
1	1	1/2	0
1/2	1	1	1/2
0	1	1	1

$\neg$	
1	0
1/2	1/2
0	1

$\Box$	
1	1
1/2	0
0	0

$\triangle$	
1	1
1/2	1
0	0

Figure 4.1: $\mathcal{L}_3$ logic [Łukasiewiez, 1953]

Let the resulting logic be called $\mathcal{L}_3$. $\mathcal{L}_3$ is a subsystem of the classical propositional calculus (removing the new truth-value $\frac{1}{2}$, we get the old classical one back).

Also in the origins of this algebraic tradition are the works done by Tarski and Jónsson, in 1951-1952. They proved relational completeness of modal logic with respect to general Kripke models via duality, several years before Kripke invented his famous semantic for modal logics [Jónsson and Tarski, 1951, Jónsson and Tarski, 1952].

Model-Theoretic Tradition

If algebraic semantics is discounted, then Rudolf Carnap was the first to provide a semantics for modal logic. It is generally understood that in Carnap three intellectual sources join: Frege, Leibniz and Wittgenstein.

- From Frege, he took his ideas and interest in semantics, including the distinction between intension and extension.
- From Leibniz, Carnap took the idea of interpreting necessity as truth in a possible world, consequently giving us a specific semantics for S_5 (shown in Fig. 4.2).
- From Wittgenstein, he took the very general ideas about descriptions of states and the concept of a world of atoms.

Carnap takes a universe of "descriptions of states" containing sets of atomic proposition. Later on, his descriptions of state become in the literature "worlds". Given a collection W of state descriptions and a particular description of state, s, the atomic proposition p is true at s of W, if and only if p is a member of s. Let $V(p)$ be the set of description of states whose p is a member, we briefly presents the Carnap's semantics in Fig. 4.2

$\models_s p$	iff	$s \in V(p)$ and p is an atomic proposition
$\models_s \neg\varphi$	iff	not $\models_s \varphi$
$\models_s \varphi \wedge \psi$	iff	$\models_s \varphi$ and $\models_s \psi$
$\models_s \varphi \vee \psi$	iff	$\models_s \varphi$ or $\models_s \psi$
$\models_s \varphi \rightarrow \psi$	iff	if $\models_s \varphi$ then $\models_s \psi$
$\models_s \Box\varphi$	iff	for all $t \in W$, $\models_t \varphi$
$\models_s \triangle\varphi$	iff	for some $t \in W$, $\models_t \varphi$

Figure 4.2: Carnap's semantics

It does not look like modern modal logic semantics: possible worlds are missing. According to Hintikka [Hintikka, 1957]: 'Carnap came extremely close to the basic ideas of possible-worlds semantics, and yet apparently did not formulate them, not even to himself'.

The next step of importance within the semantic tradition was taken by Arthur Prior. Prior, whose interests lay in temporal notions, founded tense logic, now also known as temporal logic. In his book *Time and Modality* [Prior, 1957], Prior models time as set ω of natural numbers. ω replaces the possible worlds of Carnap W by a set of points of time. In [Prior, 1957], attention is focused on the operation defined by the conditions:

$$\begin{array}{lll} \models_t \Box\varphi & \text{iff} & \forall u \geqq t, \models_u \varphi \\ \models_t \triangle\varphi & \text{iff} & \exists u \geqq t, \models_u \varphi \end{array}$$

In retrospective, it seems that Carnap and Prior together supplied all the necessary ingredients for modal logic as we know it at present [Bull and Segerberg, 1984]. The modern notion of a model for a modal logic is a triple $\mathcal{M} = (W, R, V)$, where W is a set of possible worlds, R the accessibility relation or the alternativeness relation (Hintikka [Hintikka, 1957]), and V a valuation. As we say, W and V were contributed by Carnap and R is obtained by generalizing over Prior.

But this semantics, also so-called *possible-worlds semantics* or *Kripke semantics*, is commonly attributed to S.A. Kripke. In [Kripke, 1959, Kripke, 1963], Kripke introduced a domain of possible worlds and regarded the modal prefix *it is necessary that* as a quantifier over worlds. Kripke [Kripke, 1963] introduced *Kripke structure* $\mathcal{M}$ as a relational model for a *possible-worlds semantics* for the modal logic of necessity and possibility. All of the modal logics treated in this thesis are based on this semantics, called also *relational semantics*.

4.2.1 Motivations

As we said, we devoted this chapter to the relational or Kripke semantics for modal logic. This is the best known and the best explored style of modal semantics. It is also, arguably, the most intuitive. Over the years modal logic has been applied in many different ways. It has been used as a tool for reasoning about time, beliefs, computational systems, necessity and possibility, and much else besides. These application, though diverse, have something important in common: the key ideas they employ can all be represented as simple graph-like structure. And as we shall see, modal logic is an interesting tool for talking about such structure.

Given the sub-graph matching nature of `SPARQL` and its evaluation over graphs, `SPARQL` can be encoded in a modal logic where the interpretation of that logic is over a graph. In the rest, of the section we introduce the basic modal logic `K` and its extension. For each logics, we report their syntax and their semantics and define the notion of satisfiability of a formula. We start with the basic modal logic `K` that we extend with multiple modalities, backward modalities and fixpoints, in the order.

4.3 Basic Modal Logic K and its Extensions

The *basic modal logic* with a relational semantics is called K. K extends propositional logic by allowing existential and universal quantification over the direct successors of a state (with respect to a transition system) using *diamond* and *box* formulæ.

4.3.1 Modal Logic K

In this section we introduce the *basic modal language* and its relational semantics. We define basic modal syntax, introduce models and frames, and give the satisfaction definition [5].

4.3.1.1 Syntax

Given a collection of atomic propositions $\mathbb{AP} = \{q, q', q'', \dots\}$ and a set of modality symbols $\mathbb{MOD} = \{m\}$ containing a single element, we define the *basic modal language* K as follows:

$$\varphi \ ::= \ \top \mid \bot \mid q \mid (\varphi) \mid \neg\varphi \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \varphi \rightarrow \psi \mid \varphi \leftrightarrow \psi \mid \Box\varphi \mid \triangle\varphi$$

where q is in $\mathbb{AP}$. We read $\top, \bot, \wedge, \vee, \neg, \rightarrow, \leftrightarrow, \Box, \triangle$ as '`true`', '`false`','`and`', '`or`', '`not`', '`implies`', '`equals`', '`box`' and '`diamond`', respectively.

The set of K formulæ is the smallest set containing $\mathbb{AP}$ that is closed under the Boolean operators $\wedge, \vee, \neg, \rightarrow, \leftrightarrow$ and the unary modalities $\Box$ and $\triangle$. There is redundancy in the way we have defined the modal language: all these connectives are not primitives, some of them will follow from the satisfaction definition given below.

Abbreviation	Formula
$\varphi \ \wedge \ \psi$	$\neg(\neg\varphi \ \vee \ \neg\psi)$
$\Box\varphi$	$\neg\triangle\neg\varphi$
$\varphi \rightarrow \psi$	$\neg\varphi \ \vee \ \psi$
$\varphi \leftrightarrow \psi$	$(\varphi \wedge \psi) \vee (\neg\varphi \wedge \neg\psi)$

[5] Syntax and Semantics of modal logics are presented à la [Blackburn et al., 2006]

4.3.1.2 Semantics

A model $\mathcal{M}$ for the basic modal language K is a triple $\mathcal{M} = (W, R, V)$ where:

- W is a non-empty set, whose elements we usually call *possible worlds* or simply **points**;
- $R \subseteq W \times W$ is a transition relation that must be total, that is for every point $w \in W$, there exists a point $w' \in W$ such that $(w, w') \in R$;
- $V : \mathbb{AP} \to \mathcal{P}(W)$ is a function (*the valuation*) that assigns to each atomic proposition $q \in \mathbb{AP}$ a subset $V(q) \subseteq W$; think of $V(q)$ as the set of points in $\mathcal{M}$ where q is **true**.

The first two components (W, R) of $\mathcal{M}$ are called the frame underlying the model.

$\mathcal{M}$ is also known as a Kripke structure for K. This Kripke model can be seen as a graph or transition system. The semantics of a formula φ, in terms of a transition system $\mathcal{M} = (W, R, V)$, is denoted by $[\![\varphi]\!]_{\mathcal{M}}$. The semantics of modal formulæ of K is defined as follows:

$$
\begin{array}{lcl}
[\![\top]\!]_{\mathcal{M}} & = & W \\
[\![\bot]\!]_{\mathcal{M}} & = & \emptyset \\
[\![q]\!]_{\mathcal{M}} & = & V(q) \\
[\![\neg\varphi]\!]_{\mathcal{M}} & = & W \setminus [\![\varphi]\!]_{\mathcal{M}} \\
[\![\varphi \wedge \psi]\!]_{\mathcal{M}} & = & [\![\varphi]\!]_{\mathcal{M}} \cap [\![\psi]\!]_{\mathcal{M}} \\
[\![\varphi \vee \psi]\!]_{\mathcal{M}} & = & [\![\varphi]\!]_{\mathcal{M}} \cup [\![\psi]\!]_{\mathcal{M}} \\
[\![\Diamond\varphi]\!]_{\mathcal{M}} & = & \{w \in W \mid \exists w' \in W.\ (w, w') \in R \text{ and } w' \in [\![\varphi]\!]_{\mathcal{M}}\} \\
[\![\Box\varphi]\!]_{\mathcal{M}} & = & \{w \in W \mid \forall w' \in W.\ (w, w') \in R \ \Rightarrow\ w' \in [\![\varphi]\!]_{\mathcal{M}}\}
\end{array}
$$

Figure 4.3: Semantics of modal formulæ in K

4.3.1.3 Notion of Satisfiability

Suppose w is a point in a model $\mathcal{M} = (W, R, V)$. Then we inductively define the notion of a formula φ being satisfied (or true) in $\mathcal{M}$ at point w as follows:

$$
\begin{array}{lll}
\mathcal{M}, w \models \top & & \text{always} \\
\mathcal{M}, w \models \bot & & \text{never} \\
\mathcal{M}, w \models q & \text{iff} & w \in V(q) \\
\mathcal{M}, w \models \neg\varphi & \text{iff} & \mathcal{M}, w \not\models \varphi \\
\mathcal{M}, w \models \varphi \wedge \psi & \text{iff} & \mathcal{M}, w \models \varphi \text{ and } \mathcal{M}, w \models \psi \\
\mathcal{M}, w \models \varphi \vee \psi & \text{iff} & \mathcal{M}, w \models \varphi \ \text{ or } \ \mathcal{M}, w \models \psi \\
\mathcal{M}, w \models \Diamond\varphi & \text{iff} & \text{there exists } w' \in W \text{ with } (w, w') \in R \text{ and } \mathcal{M}, w' \models \varphi \\
\mathcal{M}, w \models \Box\varphi & \text{iff} & \text{for all } w' \in W, \text{ if } (w, w') \in R \text{ then } \mathcal{M}, w' \models \varphi
\end{array}
$$

Figure 4.4: Satisfiability of modal formulæ in K

A formula φ is *satisfiable* if there exists $\mathcal{M}, w$ with $\mathcal{M}, w \models \varphi$. In this case, $\mathcal{M}$ is called a *model* of φ.

4.3.1.4 Finite Tree Model Property

For a given logic, the finite model property states that if a formula is satisfiable, then it is satisfiable by a model of some bounded size. Provided that model-checking (the problem of determining the truth value of a formula φ in a given Kripke structure $\mathcal{M}$, $\mathcal{M}, w \models \varphi$) is decidable which is the case for almost all modal logics, this immediately gives decidability of satisfiability for the logic, as one need simply check every transition system up to the size bound.

Theorem 4.3.1 (Finite-tree-model property [Blackburn et al., 2001])**.** *K has the finite-tree-model property, i.e. every satisfiable formula φ has a model $\mathcal{M}, w$ such that R is a finite tree with root w_0 and $\mathcal{M}, w_0 \models \varphi$*

A formula φ has a finite tree model that is only as deep as its *modal depth*, which we define as follows:

- if $q \in \mathbb{AP}$, then depth(q)=0;
- if $\varphi = \neg\varphi'$, then depth(φ)=depth(φ');
- if $\varphi = \varphi' \wedge \varphi''$ or $\varphi = \varphi' \vee \varphi''$, then depth($\varphi$)= max{depth($\varphi'$),depth($\varphi''$)};
- if $\varphi = \Diamond\varphi'$ or $\varphi = \Box\varphi'$, then depth(φ)=depth(φ')+1.

Figure 4.5: *Modal depth* of modal formulæ in K

4.3.2 Modal Logic K with Multiple Modalities: $\mathtt{K}_n$

The modal logic $\mathtt{K}_n$ extends K by allowing a finite set of modalities or transition programs $\mathbb{MOD} = \{m, m', m'' \ldots\}$, that allows a more complex navigation into the transition system.

Close to the definition in [Blackburn et al., 2006], the modal language $\mathtt{K}_n$ is defined as follows:

$$\varphi \ ::= \ \top \mid \bot \mid q \mid (\varphi) \mid \neg\varphi \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \varphi \rightarrow \psi \mid \varphi \leftrightarrow \psi \mid [m]\varphi \mid \langle m \rangle \varphi$$

where $q \in \mathbb{AP}$ and $m \in \mathbb{MOD}$ is a transition program.

The semantics of $\mathtt{K}_n$ is given over a transition system $\mathcal{M} = (W, \{R^m\}_{m \in \mathbb{MOD}}, V)$, where W and V are defined as usually, and $\{R^m\}_{m \in \mathbb{MOD}}$ is a set of binary relations on W. Each R^m

with $m \in \mathbb{MOD}$ is a transition relation defined as for the $\mathtt{K}$ logic. The semantics of $\mathtt{K}_n$ is different from the $\mathtt{K}$ semantics, in the following cases:

$$\begin{array}{lcl} [\![\langle m\rangle\varphi]\!]_{\mathcal{M}} & = & \{w \in W \mid \exists w' \in W.\ (w, w') \in R^m \ \mathit{and}\ w' \in [\![\varphi]\!]_{\mathcal{M}}\} \\ [\![[m]\varphi]\!]_{\mathcal{M}} & = & \{w \in W \mid \forall w' \in W.\ (w, w') \in R^m \ \Rightarrow\ w' \in [\![\varphi]\!]_{\mathcal{M}}\} \end{array}$$

Suppose w is a point in a model $\mathcal{M} = (W, R, V)$. Then we inductively define the notion of a formula φ, prefixed by diamond or box, being satisfied (or true) in $\mathcal{M}$ at point w as follows:

$$\begin{array}{lll} \mathcal{M}, w \models \langle m\rangle\varphi & \text{iff} & \text{there exists } w' \in W \text{ with } (w, w') \in R^m \text{ and } \mathcal{M}, w' \models \varphi \\ \mathcal{M}, w \models [m]\varphi & \text{iff} & \text{for all } w' \in W, \text{ if } (w, w') \in R^m \text{ then } \mathcal{M}, w' \models \varphi \end{array}$$

4.3.3 Modal Logic K with Multiple and Backward Modalities: $\mathtt{K}_{(n,back)}$

The modal logic $\mathtt{K}_{(n,back)}$ extends $\mathtt{K}_n$ by allowing backward modalities. Let m be a modality, we call $\overline{m}$ backward modality. We assume that the function $^{-} : \mathbb{MOD} \to \mathbb{MOD}$ is defined and that $\overline{\overline{m}} = m$, for each $m \in \mathbb{MOD}$.

Close to the definition in [Blackburn et al., 2006], the modal language $\mathtt{K}_{(n,back)}$ is defined as follows:

$$\varphi \ ::= \ \top \mid \bot \mid q \mid (\varphi) \mid \neg\varphi \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \varphi \to \psi \mid \varphi \leftrightarrow \psi \mid [m]\varphi \mid \langle m\rangle\varphi \mid [\overline{m}]\varphi \mid \langle\overline{m}\rangle\varphi$$

where $q \in \mathbb{AP}$ and $m \in \mathbb{MOD}$. A modality can be in forward $\langle m\rangle$ or backward $\langle\overline{m}\rangle$ form.

The semantics of $\mathtt{K}_{(n,back)}$ is given over a transition system $\mathcal{M} = (W, \{R^m\}_{m\in\mathbb{MOD}}, V)$. Let $(w, w') \in R^m$, we have $(w', w) \in R^{\overline{m}}$. The semantics of $\mathtt{K}_{(n,back)}$ extends the $\mathtt{K}_n$ as follows:

$$\begin{array}{lcl} [\![\langle\overline{m}\rangle\varphi]\!]_{\mathcal{M}} & = & \{w' \in W \mid \exists w \in W.\ (w', w) \in R^{\overline{m}} \ \mathit{and}\ w \in [\![\varphi]\!]_{\mathcal{M}}\} \\ [\![[\overline{m}]\varphi]\!]_{\mathcal{M}} & = & \{w' \in W \mid \forall w \in W.\ (w', w) \in R^{\overline{m}} \ \Rightarrow\ w \in [\![\varphi]\!]_{\mathcal{M}}\} \end{array}$$

Suppose w is a point in a model $\mathcal{M} = (W, R, V)$. Then we inductively define the notion of a formula φ, prefixed by diamond or box, being satisfied (or true) in $\mathcal{M}$ at point w as follows:

$$\begin{array}{lll} \mathcal{M}, w' \models \langle\overline{m}\rangle\varphi & \text{iff} & \text{there exists } w \in W \text{ with } (w', w) \in R^{\overline{m}} \text{ and } \mathcal{M}, w \models \varphi \\ \mathcal{M}, w' \models [\overline{m}]\varphi & \text{iff} & \text{for all } w \in W, \text{ if } (w', w) \in R^{\overline{m}} \text{ then } \mathcal{M}, w \models \varphi \end{array}$$

4.4 Mu-calculus

In 1983, Dexter Kozen published a study [Kozen, 1983] of a logic that combined simple modalities, as in $\mathtt{K}_n$, with fixpoint operators to provide a form of recursion. As we will see next, it has a simple syntax, an easily given semantics, and yet the fixpoint operators provide immense expressive power.

4.4.1 Fixpoints as Recursion

Suppose that W is the set of "*possible worlds*" (or **points**) of some transition system $\mathcal{M}$. As we have seen, one way to provide semantics over a transition system is to map modal formulæ φ to a set of possible worlds $V(\varphi)$. V tells us at which states each formula holds (it is true). If the logic contains variables with interpretation ranging over $\mathcal{P}(W)$, then the semantics of a formula with a free variable $\varphi(X)$ becomes a function $f : \mathcal{P}(W) \to \mathcal{P}(W)$. If the lattice structure on $\mathcal{P}(W)$ is given by set inclusion, and if f is a monotonic function, then by the Knaster-Tarski theorem f has fixed points (a unique maximal fixpoint operator ν and a unique minimal μ). So the modal logic can be extended with a minimal fixpoint operator μ, so that $\mu X.\varphi(X)$ or simply $\mu X.\varphi$ is a formula whose semantics is the least fixed point of f; and similar a maximal fixpoint operator ν, so that $\nu X.\varphi(X)$ or $\nu X.\varphi$ is a formula whose semantics is the greatest fixed point of f.

The standard theory says that if f is a monotonic function on a lattice, we can construct the least fixed point of f by iterating f in the bottom element $\perp$ of the lattice to form an increasing chain whose limit is the fixed point. The length of the chain is in general transfinite, but is bounded at worst by the cardinal after the cardinality of the lattice. So if f is monotonic on $\mathcal{P}(W)$, we have

$$\mu f = \cup_{\alpha<k} f^{\alpha}(\perp)$$

and similarly

$$\nu f = \cap_{\alpha<k} f^{\alpha}(W)$$

where k is at worst $|W| + 1$ for finite W, or $\mathbb{N}$ for countable W.

So for a minimal fixpoint $\mu X.\varphi$, if a world w satisfies the fixpoint, it satisfies some approximant, suppose the $\beta + 1$ s.t $w \models \varphi^{\beta+1}(\perp)$. If we unfold this formula once, we get $w \models \varphi(\varphi^{\beta}(\perp))$. The unfolder cannot continue *ad infinitum* because the ordinals are well-founded. This is the strict meaning behind the slogan *μ means finite looping.*

For a maximal fixpoint $\nu X.\varphi$, there is no such decreasing chain $w \models \nu X.\varphi(X)$ iff $w \models \varphi(\nu X.\varphi(X))$ and we may loop for ever. This is the strict meaning behind the slogan *ν means looping.*

4.4.2 Syntax

Let $\mathbb{AP} = \{q, q', q'', \dots\}$,$\mathbb{V} = \{X, Y, Z, \dots\}$ and $\mathbb{MOD} = \{m, m', m'' \dots\}$ be countable sets of atomic propositions, variables and modality symbols, respectively. The set of μ-formulæ are defined as follows:

$$\varphi \ ::= \ \top \mid \bot \mid q \mid X \mid (\varphi) \mid \neg\varphi \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \varphi \rightarrow \psi \mid \varphi \leftrightarrow \psi \mid [m]\varphi \mid \langle m \rangle \varphi \mid \mu X.\varphi \mid \nu X.\varphi$$

where $q \in \mathbb{AP}$, $X \in \mathbb{V}$ and $m \in \mathbb{MOD}$. As for K, there is redundancy in the way we have defined the fixpoint operators. μ and ν are inter-definable:

Abbreviation	Formula
$\nu X.\varphi$	$\neg\mu X.\neg\varphi(\neg X)$

Note the triple use of negation in μ, which is required to maintain the positivity. A formula is said to be in *positive form* if it is written with the derived operators so that $\neg$ only occurs applied to atomic proposition. It is in *positive normal form* if in addition all bound variables are distinct. Any formula can be put into *positive normal form* by use of De Morgan laws and α-conversion. So we shall often assume positive normal form.

4.4.3 Semantics

The semantics of the μ-calculus is given over a transition system $\mathcal{M} = (W, \{R^m\}_{m \in \mathbb{MOD}}, V)$ together with a function $\rho : \mathbb{V} \rightarrow \mathcal{P}(W)$, called a *valuation* for $\mathcal{M}$. The semantics is define in the standard manner as follows.

$$\begin{array}{lcl}
[\![\top]\!]^{\rho}_{\mathcal{M}} & = & W \\
[\![\bot]\!]^{\rho}_{\mathcal{M}} & = & \emptyset \\
[\![q]\!]^{\rho}_{\mathcal{M}} & = & V(q) \\
[\![X]\!]^{\rho}_{\mathcal{M}} & = & \rho(X) \\
[\![\neg\varphi]\!]^{\rho}_{\mathcal{M}} & = & W \setminus [\![\varphi]\!]^{\rho}_{\mathcal{M}} \\
[\![\varphi \wedge \psi]\!]^{\rho}_{\mathcal{M}} & = & [\![\varphi]\!]^{\rho}_{\mathcal{M}} \cap [\![\psi]\!]_{\mathcal{M}} \\
[\![\varphi \vee \psi]\!]^{\rho}_{\mathcal{M}} & = & [\![\varphi]\!]^{\rho}_{\mathcal{M}} \cup [\![\psi]\!]^{\rho}_{\mathcal{M}} \\
[\![\langle m \rangle \varphi]\!]^{\rho}_{\mathcal{M}} & = & \{w \in W \mid \exists w' \in W.\ (w, w') \in R^m \ \mathit{and}\ w' \in [\![\varphi]\!]^{\rho}_{\mathcal{M}}\} \\
[\![[m]\varphi]\!]^{\rho}_{\mathcal{M}} & = & \{w \in W \mid \forall w' \in W.\ (w, w') \in R^m \ \Rightarrow \ w' \in [\![\varphi]\!]^{\rho}_{\mathcal{M}}\} \\
[\![\mu X.\varphi]\!]^{\rho}_{\mathcal{M}} & = & \bigcap\{W' \subseteq W \mid [\![\varphi]\!]^{\rho[X \mapsto W']}_{\mathcal{M}} \subseteq W'\} \\
[\![\nu X.\varphi]\!]^{\rho}_{\mathcal{M}} & = & \bigcup\{W' \supseteq W \mid [\![\varphi]\!]^{\rho[X \mapsto W']}_{\mathcal{M}} \supseteq W'\}
\end{array}$$

Where $\rho[X \mapsto W']$ is an extension of ρ defined as follows.

$$\rho[X \mapsto W'](Y) = \begin{cases} W' & \mathit{if}\ Y = X \\ \rho(Y) & \mathit{otherwise} \end{cases}$$

4.4.4 Notion of Satisfiability

We write $\mathcal{M}, \rho, w \models \varphi$ if $w \in [\![\varphi]\!]^{\rho}_{\mathcal{M}}$. We write $\mathcal{M} \models \varphi$ if $\mathcal{M}, \rho, w \models \varphi$ holds for any valuation ρ and world w.

Formula φ and φ' are equivalent ($\varphi \equiv \varphi'$) if $[\![\varphi]\!]^{\rho}_{\mathcal{M}} = [\![\varphi']\!]^{\rho}_{\mathcal{M}}$ for any Kripke structure $\mathcal{M}$ and valuation ρ.

A formula φ is *valid* if it is equivalent to `true`, i.e. it is satisfied by all $\mathcal{M}$.

A formula φ is *satisfiable*, denoted by $\mathcal{M}, \rho, w \models \varphi$, if there exists a model $\mathcal{M}$ and a valuation ρ such that $[\![\varphi]\!]^{\rho}_{\mathcal{M}} \neq \emptyset$

4.4.5 Small Model Property

We recall that, a logic is said to have the finite model property if every satisfiable formula is satisfied by a finite model. The small model property is stronger: every satisfiable formula must be satisfied by a model whose size is bounded by some function on the size of the formula (i.e., the number of operators and propositional variables appearing in it). It is known that that the modal μ-calculus has the small model property [Kozen, 1983]; particularly, every satisfiable formula is true in a model whose size is exponential in the length of the formula.

4.4.6 Alternation-Free Mu-Calculus: $\mathtt{AF}_{\mu}$

The alternation-free fragment of the modal μ-calculus ($\mathtt{AF}_{\mu}$) allows less complex decision procedure for satisfiability judgment than the full μ-calculus, yet it still has strong expressive power.

A formula φ is *alternation-free* if the following conditions are satisfied:

- For any subformula ψ of φ in the form of $\mu X.\psi'$ and for any subformula $\eta \in \psi'$ in the form of $\nu Y.\eta'$, X does not occur freely in η'.

- For any subformula ψ of φ in the form of $\nu X.\psi'$ and for any subformula $\eta \in \psi'$ in the form of $\mu Y.\eta'$, X does not occur freely in η'.

Known decision procedures for the modal μ-calculus are complex. In [Kozen, 1983], a exponential-time decision procedure was given for full μ-calculus. The alternation-free restriction does not reduce the theoretical complexity [Bonatti et al., 2006] (both are `EXPTIME-Complete`), but it does lead to efficient implementation using binary decision diagramma (`BDD`) [Tanabe et al., 2008]. Meanwhile, the restricted fragments are still sufficiently powerful [Gutierrez et al., 2012] and adopted in several works [Genevès, 2008, Chekol et al., 2012b, Genevès et al., 2015].

4.5 State-of-the-art of the Solver

As modal logic extends propositional logic, the study in modal satisfiability is deeply connected with that of propositional satisfiability. For example, tableau-based decision procedure are presented in [Ladner, 1977, Halpern and Moses, 1992, Patel-Schneider and Horrocks, 1999]. Such methods are built on top of the propositional tableau construction procedure by forming a fully expanded propositional tableau and generating successor nodes "on demand".

Non propositional methods take a different approach to the problem. It has been shown that, by embedding modal logics into first order logic, a first-order theorem prover can be used for deciding modal satisfiability [Hustadt and Schmidt, 2000, Areces et al., 2000]. These translation-based approaches have the advantage of leveraging the tremendous implementation progress that has occurred over in first-order theorem proving. Soundness and completeness are ensured by the soundness and completeness of the resolution prover (once the soundness and completeness of the translation have been shown), while a decision procedure is automatically obtained for any modal logic that can be translated into a decidable fragment of first-order logic (such as the two-variable fragment [Grädel et al., 1997]).

In the following, we present some of the solvers available in the literature. We start to discuss a `BDD`-Based Decision Procedure implementation for the Modal Logic `K` [Pan et al., 2006]. Then, we present an $\mathtt{AF}_\mu$ solver that implements algorithms found in [Tanabe et al., 2008]. We discuss the Tree-Solver developed by Pierre Genevès [Genevès, 2008]. And finally we present a first-order theorem prover `SPASS` and its enhancement with a translator of modal formulæ `MSPASS` [Hustadt and Schmidt, 2000].

The first three solvers are inspired by the automata-theoretic approach for logics, while the latter by the translation-based approach.

4.5.1 KBDD Solver

Pan et al. propose a `BDD`-based decision procedure for the modal logic `K`, and show that it can be implemented rather efficiently [Pan et al., 2006]. They also note that their method is inspired by and closely related to the automata approach for logics with the tree-model property [Vardi, 1996].

In that approach, one proceeds in two step:

1. An input formula is translated to a tree automaton that accepts all tree models of the formula.
2. The automaton is tested for non-emptiness, i.e. whether it accepts some tree.

In Pan's algorithms, these steps are combined to carry out the non-emptiness test without explicitly constructing the automaton, as point out in [Baader and Tobies, 2001].

The automaton's non-emptiness test for the logic K consists in a single fixpoint computation, which starts with a set of states and then repeatedly applies a monotone operator until a fixpoint is reached. In [Pan et al., 2006], two approaches are presented to perform the fixpoint computation.

Top-down approach This approach is closely related to the type elimination used for more complex modal logics, see e.g. [Halpern and Moses, 1992]. Starting from the set of all states, repeatedly inconsistent states (states contain unwitnessed negated box formulæ) are removed, until all remaining nodes are consistent.

Bottom-up approach In contrast, this approach starts with a small set of states (those without negated box formulæ) and repeatedly adds those states whose negated box formulæ are witness in the current set.

Note, this solver supports only the modal logic K. In [Pan et al., 2006], authors mention that their approach can be easily extended to K_n but they did not provide an implementation. The presence of a finite number of modalities in our application, prevents us to directly use this solver.

Implementation KBDD is an implementation of the BDD-Based decision procedure and its variants of [Pan et al., 2006]. It is in C++ using the CUDD2.3.1 package for BDD [Somenzi, 1998], with a formula simplification in OCaml.

It is available here http://www.cs.rice.edu/CS/Verification/Software/kbdd.tar

4.5.2 AF_μ Solver

AF_μ Solver (Alternation Free two-way μ-calculus) [Tanabe et al., 2008] is a satisfiability solver for the alternation-free fragment of the μ-calculus. It is a prototype implementation which determines the satisfiability of a μ-calculus formula by producing a yes-or-no answer. The method proposed is a based tableau method. Although, the size of the tableau set maintained in the method might be large for complex formulæ, the set and the operations on it are expressed using BDD. Like the KBDD's top-down approach, the main part of Tanabe et al.'s procedure repeatedly computes subsets of states of some fixed set of states, removing inconsistent states, until a fixpoint is reached. At the end of the computation, if the formula is present in a node of the fixpoint, then the formula is satisfiable.

The time complexity of the satisfiability-testing algorithm is $2^{\Theta(n \ log \ n)}$, in terms of the formula size n (i.e., the number of operators and propositional variables appearing in the formula).

Implementation It is programmed in Java, and the decision procedure is implemented using JavaBDD [Whaley, 2007]. The solver is not available online.

4.5.3 Tree Solver

The TreeSolver is an implementation, through BDD techniques, of the satisfiability-testing for an alternation-free modal μ-calculus with converse, where formulæ are cycle-free and are interpreted over finite ordered trees. The time complexity of the satisfiability-testing algorithm is optimal $2^{\Theta(n)}$, in terms of the formula size n.

The algorithm works by enumerating all partially satisfiable states, in a bottom-up order: it first considers all states where leaves (atomic propositions of the formula) are partially satisfied, then incrementally uses the already known partially satisfiable states and the compatibility relation (two states are related according to a modality) to find states which require a deeper tree to be partially satisfied, as illustrated by Fig. 4.6.

At the end of the enumeration of all partially satisfiable states, if the formula is satisfied on the root of the tree then the tree is a model for the formula. Otherwise the algorithm concludes that the formula does not have a tree model (i.e. the formula is unsatisfiable).

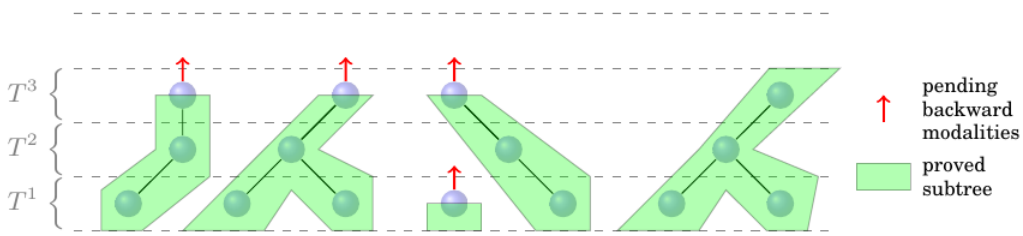


Figure 4.6: Algorithm's principle: progressive bottom-up reasoning [Genevès et al., 2015]

Implementation The system has been implemented as a web application. Interaction with the system is offered through a user interface in a web browser. The tool is available online from: `http://tyrex.inria.fr/websolver/`

4.5.4 MSPASS

MSPASS [Hustadt and Schmidt, 2000] is an enhancement of the first-order theorem prover SPASS [Weidenbach et al., 2009] with a translator of modal formulæ, formulæ of description logics, and formulæ of the relational calculus into first-order logic with equality. SPASS is a sound and complete theorem prover for first-order logic with equality. It is one of the fastest and most sophisticated theorem provers for first-order logic with equality, and its performance compares well with special purpose theorem provers for modal logics, description logics and first-order logic.

The input language of SPASS was extended to accept, in additional to first-order formulæ, also modal, relational and description logic formulæ as input with the enhancement MSPASS. Reasoning in MSPASS is performed in three stages [Hustadt and Schmidt, 2000]:

1. translation of a given set of modal and relation formulæ into a a set of first-order formulæ;
2. transformation into clausal form;
3. saturation-based resolution.

Using certain flag settings MSPASS is known to provide decision procedures for a range of logics, including the modal logic $\mathtt{K}_n$ [Hustadt and Schmidt, 1998, Schmidt, 1999].

Implementation. - MSPASS (free open-source reasoner) is implemented in ANSI C and differs from SPASS only in the extended input language and the translation routines. Starting from July 2007, MSPASS has been incorporated into SPASS, available at the following address: http://www.spass-prover.org/download/.

4.6 Conclusion

In this chapter we have introduced some modal logics. We started to discuss the basic modal logic K, and in an incremental way, we have described how features can be added such as multiple modalities, backward modalities and fixpoints.

For all of these modal logics, we have discussed state-of-the-art solvers of the literature. In the next chapter, we review in more details the BDD based decision procedure for K, reported in [Pan et al., 2006]. Then, we show how this decision procedure can be extended with multiple and backward modalities. The implementation of this extended decision procedure will be used in Chapter 6 for solving the SPARQL containment problem.

5

A Decision Procedure for $\mathsf{K}_{(n,back)}$

The chapter 5 describes BDD-based decision procedure for the modal logic $K_{(n,back)}$ with backward modalities. The procedure is inspired by the BDD-Based Decision Procedures for the Modal Logic K [Pan et al., 2006]. This chapter extends their approaches allowing multiple and backward modalities.

Contents

5.1 Introduction **62**

5.2 Preliminaries **62**

5.2.1 Syntax and Semantics 62

5.2.2 Closure, Lean and Types 64

5.3 Algorithm **67**

5.3.1 Algorithm Scheme 67

5.3.2 Correctness 68

5.4 Implementation **70**

5.4.1 Implicit Representation of Set of Types 70

5.5 Conclusion **71**

5.1 Introduction

In the last years, modal logic has been applied to numerous areas of computer science. Deciding satisfiability of a modal formula is one of the most basic reasoning problems and various techniques have been developed and optimized to decide it.

The satisfiability for the basic normal logic $\mathtt{K}_{(n,back)}$ is PSPACE-complete [Stockmeyer, 1976, Ladner, 1977].

There exist different techniques, but no one is able to always outperform others for inputs of different characteristics.

In this chapter, we focus on Pan et al.'s work [Pan et al., 2006]. We propose an algorithm for deciding satisfiability of formulæ, in presence of multiple and backward modalities.

An implementation of the algorithm is provided and uses Binary Decision Diagrams [Bryant, 1986], a symbolic representation techniques.

5.2 Preliminaries

In this chapter, we adopt the syntax and the semantics of the modal logic $\mathtt{K}_{(n,back)}$ introduced in the previous chapter (Section 4). To facilitate the reading of this chapter, we briefly report them below.

5.2.1 Syntax and Semantics

The set of $\mathtt{K}_{(n,back)}$ formulæ is constructed from a finite set of atomic propositions $\mathbb{AP}$ and a finite set of modalities $\mathbb{MOD}$ and is the least set containing $\mathbb{AP}$ that is closed under the boolean operators $\wedge, \vee, \neg$ and the modalities operators $\langle\alpha\rangle, [\alpha]$ with $\alpha \in \mathbb{MOD}$.

$$\varphi \ ::= \ \top \mid \bot \mid q \mid \neg\varphi \mid \varphi' \wedge \varphi'' \mid \varphi \vee \varphi'' \mid \langle\alpha\rangle\varphi \mid [\alpha]\varphi$$

with α a forward modality, i.e. $\alpha = m$, or a backward modality, i.e. $\alpha = \overline{m}$, such that $\alpha = \overline{\overline{\alpha}}$.

A formula ψ in $\mathtt{K}_{(n,back)}$ is interpreted in a Kripke structure $\mathcal{M} = (W, \{R^\alpha\}_{\alpha\in\mathbb{MOD}}, V)$, where W is a set of possible worlds, each $R^\alpha \subseteq W \times W$ is the accessibility relation on worlds for the modality α, and $V : \mathbb{AP} \to \mathcal{P}(W)$ is a labeling function for each world w. Note that $R^{\overline{\alpha}}(w', w) = (R^\alpha(w, w'))^{-1}$

The notion of formula ψ being satisfied in a world w of a Kripke structure $\mathcal{M}$ (written $\mathcal{M}, w \models \psi$) is inductively defined in Figure 5.1

A formula ψ is satisfiable if there exist $\mathcal{M}, w$ with $\mathcal{M}, w \models \psi$. In this case, $\mathcal{M}$ is called a *model* of ψ. Two formulæ φ and ψ are said to be equivalent if, for all structure $\mathcal{M}$ and all worlds $w \in W$, $\mathcal{M}, w \models \varphi$ if and only if $\mathcal{M}, w \models \psi$.

$\mathcal{M}, w \models \top$		always
$\mathcal{M}, w \models \bot$		never
$\mathcal{M}, w \models q$	iff	$w \in V(q)$
$\mathcal{M}, w \models \neg\varphi$	iff	$\mathcal{M}, w \not\models \varphi$
$\mathcal{M}, w \models \varphi' \wedge \varphi''$	iff	$\mathcal{M}, w \models \varphi'$ and $\mathcal{M}, w \models \varphi''$
$\mathcal{M}, w \models \varphi' \vee \varphi''$	iff	$\mathcal{M}, w \models \varphi'$ or $\mathcal{M}, w \models \varphi''$
$\mathcal{M}, w \models \langle\alpha\rangle\varphi$	iff	there exists $w' \in W$ with $(w, w') \in R^\alpha$ and $\mathcal{M}, w' \models \varphi$
$\mathcal{M}, w \models [\alpha]\varphi$	iff	for all $w' \in W$, if $(w, w') \in R^\alpha$ then $\mathcal{M}, w' \models \varphi$

Figure 5.1: Satisfiability of modal formulæ in $\mathtt{K}_{(n,back)}$

For our purpose in the rest of the chapter, we restrict our attention to formulæ in a certain normal form. If not stated otherwise, we assume all formulæ to be in *negation normal form* NNF.

Definition 5.2.1 (Negation normal form)**.** *A formula ψ in $\mathtt{K}_{(n,back)}$ is in negation normal form (NNF), if ψ satisfies the following condition:*

All negations ($\neg$) in ψ appear immediately before atomic proposition.

The set of all formulæ in negation normal form, is defined as follows.

$$\varphi \ ::= \ \top \mid \bot \mid q \mid \neg q \mid \varphi' \wedge \varphi'' \mid \varphi' \vee \varphi'' \mid \langle\alpha\rangle\varphi \mid [\alpha]\varphi$$

with α be a backward or a forward modality.

We restricted to this set of formulæ for avoiding that in a formula can appear different subformulæ equivalent, i.e. $\langle\alpha\rangle\neg p$ and $\neg[\alpha]p$. This situation can complicate our decision procedure, by increasing the dimension of the structures used and introducing dependencies among subformulæ that should be explicitly managed. Nevertheless, each $\mathtt{K}_{(n,back)}$ formula can be converted into an equivalent one in NNF that is of linear size [Niwiǹski and Walukiewicz, 1996]. The conversion can be done by applying the equivalence rules of Table 5.1.

$\neg\top$	$=$	$\bot$
$\neg\bot$	$=$	$\top$
$\neg(\varphi' \wedge \varphi'')$	$=$	$\neg\varphi' \vee \neg\varphi''$
$\neg(\varphi' \vee \varphi'')$	$=$	$\neg\varphi' \wedge \neg\varphi''$
$\neg[\alpha]\varphi$	$=$	$\langle\alpha\rangle\neg\varphi$
$\neg\langle\alpha\rangle\varphi)$	$=$	$[\alpha]\neg\varphi$

Table 5.1: Set of equivalence rules for converting a formula into an equivalent one in NNF

Example 5.2.1 (Negation normal form)**.** *Let ψ be the formula:*

$$\psi = p \wedge (\neg[m]p \vee \neg(p \wedge q))$$

by applying the equivalence rules of Table 5.1, we obtain the NNF *formula φ_{NNF}*

$$\psi_{NNF} = p \wedge (\langle m\rangle\neg p \vee (\neg p \vee \neg q))$$

that is equivalent to ψ

5.2.2 Closure, Lean and Types

In the following, we report three fundamental notions in the developing of our decision procedure.

We firstly introduce the Fischer and Lander closure of a formula ψ [Fischer and Ladner, 1979]. The closure is the set of all subformulæ of ψ, including ψ itself.

Then we identify a subset of the closure, called lean of ψ. The lean contains all the subformulæ of ψ, that are neither conjunctions nor disjunctions.

We finally introduce the notion of ψ-types or simple types. A type is a subset of the lean and it is used for establishing the truth status of a formula.

5.2.2.1 Closure

The Fisher-Lander closure $\mathtt{cl}(\psi)$ [Fischer and Ladner, 1979] of a formula ψ is defined as the set of all subformula of ψ.

Specifically, the relation $\rightarrow_{\mathtt{cl}}$ on the set of all formulæ in NNF is defined as the least relation that satisfies the following:

- $\varphi_1 \wedge \varphi_2 \rightarrow_{cl} \varphi_1,\ \varphi_1 \wedge \varphi_2 \rightarrow_{\mathtt{cl}} \varphi_2$
- $\varphi_1 \vee \varphi_2 \rightarrow_{\mathtt{cl}} \varphi_1,\ \varphi_1 \vee \varphi_2 \rightarrow_{\mathtt{cl}} \varphi_2$
- $\langle\alpha\rangle\varphi \rightarrow_{\mathtt{cl}} \varphi$
- $[\alpha]\varphi \rightarrow_{\mathtt{cl}} \varphi$

Definition 5.2.2 (Closure)**.** *The closure $\mathtt{cl}(\psi)$ is the smallest set $\mathcal{S}$ that contains ψ and closed under the relation $\rightarrow_{cl}$, i.e. if $\varphi_1 \in \mathcal{S}$ and $\varphi_1 \rightarrow_{cl} \varphi_2$ then $\varphi_2 \in S$*

Example 5.2.2 (Closure)**.** *Consider the formula* $\psi = \langle m \rangle p \wedge ([\overline{m}]p \vee \neg q)$*, the closure* $\mathtt{cl}(\psi)$ *of the formula* ψ *is shown in Figure 5.2.*

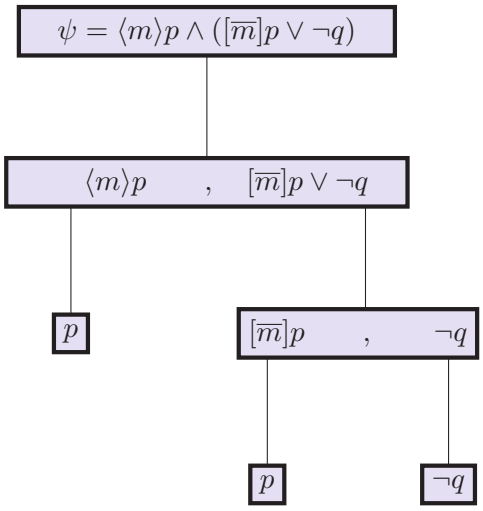


Figure 5.2: Closure of the formula ψ

The closure of ψ *is the following:*

$$\mathtt{cl}(\psi) = \{\psi,\ \langle m \rangle p,\ [\overline{m}]p \vee \neg q,\ p,\ [\overline{m}]p,\ \neg q\}$$

5.2.2.2 Lean

The lean $\mathtt{Lean}(\psi)$ of a formula ψ is the subset of the Fischer-Lander closure of ψ composed of atomic and modal subformualae of ψ [Pan et al., 2006].

Definition 5.2.3 (Lean)**.** *The lean* $\mathtt{Lean}(\psi)$ *of a formula* ψ *is a subset of* $\mathtt{cl}(\psi)$ *defined as follows:*

$$\{\varphi \in \mathtt{cl}(\psi) \mid \varphi \in \mathbb{AP} \text{ or } \varphi \text{ is in the form of } \langle \alpha \rangle \varphi' \text{ or } [\alpha]\varphi'\}$$

Example 5.2.3 (Lean)**.** *Given the formula* ψ *of the previous example 5.2.2, the Lean of* ψ *is the following:*

$$\mathtt{Lean}(\psi) = \{\langle m \rangle p,\ p,\ [\overline{m}]p,\ \neg q\}$$

Note the lean of a formula ψ can be seen as a compact representation of ψ itself. Specifically, let $\mathtt{cl}^*(\psi)$ be the extended closure of a formula ψ, defined as $\mathtt{cl}^*(\psi) = \mathtt{cl}(\psi) \cup \{\neg\varphi \mid \varphi \in \mathtt{cl}(\psi)\}$. Every formula $\varphi \in \mathtt{cl}^*(\psi)$ can be seen as a boolean combination of formulæ in the Lean of φ [Pan et al., 2006].

5.2.2.3 Types

A ψ-*type* or simply a *type* (Hintikka set in the temporal logic literature) is a set $t \subseteq \mathtt{Lean}(\psi)$. A type determines a truth assignment of every formula in $\mathtt{cl}^*(\psi)$ with the relation $\dot{\in}$ defined as follows.

Definition 5.2.4 (Relation $\dot{\in}$)**.** *Let φ be a (non-atomic) formula, then $\varphi \dot{\in} t$ if*

- $\varphi \in \mathtt{Lean}(\psi)$ *and* $\varphi \in t$
- $\varphi = \neg\varphi'$ *and not* $\varphi' \dot{\in} t$
- $\varphi = \varphi' \wedge \varphi''$, $\varphi' \dot{\in} t$ *and* $\varphi'' \dot{\in} t$
- $\varphi = \varphi' \vee \varphi''$ *and* $\varphi' \dot{\in} t$ *or* $\varphi'' \dot{\in} t$.

A formula φ is true at a type t iff $\varphi \dot{\in} t$. The truth status of a formula is now related to the truth assignment of its ψ-*types*.

A compatible relation (or maximal accessibility relation) is now defined between types. This relation establishes which formula must be true in a type in order for it to be a witness for a modal formula.

Definition 5.2.5 (Relation Δ)**.** *For t, t' be two types, a relation $\Delta^\alpha(t, t')$ holds if the following conditions are satisfied:*

- $\forall\langle\alpha\rangle\varphi,\ \langle\alpha\rangle\varphi \in t \Rightarrow \varphi \dot{\in} t'$
- $\forall\langle\overline{\alpha}\rangle\varphi,\ \langle\overline{\alpha}\rangle\varphi \in t' \Rightarrow \varphi \dot{\in} t$
- $\forall[\alpha]\varphi' \in \mathtt{Lean}(\psi),\ [\alpha]\varphi' \in t \Rightarrow \varphi' \dot{\in} t'$
- $\forall[\overline{\alpha}]\varphi' \in \mathtt{Lean}(\psi),\ [\overline{\alpha}]\varphi' \in t' \Rightarrow \varphi' \dot{\in} t$

Given a set of lean types $\mathcal{T} \in \mathtt{Lean}(\psi)$, we can now construct a satisfying Kripke structure $\mathcal{M}_T$ for the formula ψ as follows.

$$\mathcal{M}_T = (T, \{\Delta^\alpha\}_{\alpha \in \mathbb{MOD}}, V)$$

where

- T is the set of possible worlds or simply points;
- $\{\Delta^\alpha\}_{\alpha \in \mathbb{MOD}}$ is a set of binary relations on T;
- $V : \mathbb{AP} \to \mathcal{P}(T)$ is a function that assign to each atomic proposition $q \in \mathbb{AP}$ a subset $V(q) \subseteq T$, such that $t \in V(q)$ iff $q \in t$.

Such a Kripke structure $\mathcal{M}_T$ is almost a canonical model [Blackburn et al., 2001]. The only difference can be seen when trying to prove that $\mathcal{M}_T$ satisfies , for all $\varphi \in \mathtt{cl}(\psi)$.

Consider the following statement:

$$\mathcal{M}_T, t \models \varphi \Leftrightarrow \varphi \dot{\in} t$$

This statement is clearly true for atomic and propositional φ by definition of types and it is also true for $\varphi = \langle\alpha\rangle\varphi$, by construction of $\{\Delta^\alpha\}_{\alpha\in\mathbb{MOD}}$. The only case that fails is the cases $\varphi = \langle\alpha\rangle\neg\varphi' \in t$ with α be a forward or backward modality: it might be the case that $\varphi' \dot{\in} t'$ for all t' with $\Delta^\alpha(t, t')$. In this case we say the formula $\langle\alpha\rangle\neg\varphi'$ in t is **not witnessed** by any t' in T. In the following, we will describe operators on type sets whose fixpoint T then indeed satisfies $\mathcal{M}_T, t \models \varphi \Leftrightarrow \varphi \dot{\in} t$.

5.3 Algorithm

The most important property of $\mathtt{K}_{(n,back)}$, as for $\mathtt{K}$, is the *tree-model property*, which allows automata-theoretic approaches to be applied [Pan et al., 2006]. In fact, $\mathtt{K}_{(n,back)}$ has the strongest *finite-tree-model property*, which will allow both top-down and bottom-up construction of such automata.

In this thesis, we take into consideration a top-down approach. We focus on this approach because it can be extended to decide satisfiability of more powerful logics with recursion, such as the $\mathtt{AF}_\mu$. A decision procedure for such a logic is reported in [Tanabe et al., 2008, Tanabe et al., 2005].

5.3.1 Algorithm Scheme

Our algorithm relies on a top-down tableau method which attempts to construct satisfying Kripke structures for the formula ψ, by a fixpoint computation. Nodes of the tableau are specific subsets of the Lean of ψ. The algorithm starts from the set of all possible nodes, and repeatedly removes inconsistent nodes until a fixpoint is reached. At the end of the computation, if ψ is present in a node of the fixpoint, then it is satisfiable. In this case, the fixpoint contains a satisfying model for the input formula.

Our decision procedure handles a formula in three steps:

1. The formula is converted into a `NNF`.
2. The closure and the lean of the formula are generated,
3. Starting from the set of all possible types, the fixpoint is calculated.

We have already seen the first two steps. In the following, we report the core of our decision procedure.

The algorithm follows the following scheme:

$T = \mathtt{Init}(\psi)$
repeat
  $T' = T$
  $T \ = \mathtt{Upd}(T')$
until $T = T'$
if exists $t \in T$ such that $\psi \dot{\in} t$ **then**
  return "ψ is satisfiable"
else **return** "ψ is not satisfiable"
end if.

Figure 5.3: Formal description of the Top-Down Algorithm

The functions $\mathtt{Init}(\psi)$ and $\mathtt{Upd}(T)$ are defined as follows:

- $\mathtt{Init}(\psi)$ is the set of all possible ψ-*types* (recall types are subsets of the Lean).
- $\mathtt{Upd}(T) := T \setminus \mathtt{bad}(T)$, where $\mathtt{bad}(T)$ is defined as follows:

$$\mathtt{bad}(T) := \{t \in T \mid \text{there exists } \langle\alpha\rangle\neg\varphi \in t \text{ and, for all } t' \in T \text{ with } \Delta^{\alpha}(t,t') \text{ we have } \varphi \dot{\in} t'\}$$

5.3.2 Correctness

In this section we prove the correctness of the satisfiability testing algorithm, by extending the proof reported in [Pan et al., 2006].

Theorem 5.3.1. *The top-down approach algorithm decides satisfiability of $\mathtt{K}_{(n,back)}$ formulæ.*

Proof. The proof is given in three step. We firstly prove the termination of the algorithm and then we prove the soundness and completeness.

Termination.

For $\psi \in \mathtt{K}_{(n,back)}$, since $Cl(\psi)$ is a finite set, $\mathtt{Lean}(\psi)$ is also finite. Since this algorithm is operating with elements in a finite lattice $2^{\mathtt{Lean}(\psi)}$ and uses a monotone $\mathtt{Upd}$ operator, it obviously terminates. In the specific, it will terminate in $\mathtt{depth}(\psi) + 1$ iterations.

Specifically, let T be the set of types that is the fixpoint of the top-down procedure, $\mathtt{Upd}(T) = T$. Let T^0 for $\mathtt{Init}(\psi)$ and T^i for the set of types after i-iterations. Since $\mathtt{Upd}(.)$ is monotone and each T^i is a subset of the finite $\mathtt{Lean}(\psi)$, the top-down algorithm terminates. More precisely, all types in T^i that contain unwitnessed formulæ φ with $depth(\varphi) > i$ are removed in T^{i+1}. As a consequence, the algorithm stops after at most $depth(\psi)+1$ iterations.

To finish the proof, it thus suffices to prove soundness and completeness.

Soundness.

For each type $t \in T$ and formula $\varphi \dot{\in} cl(\psi)$, if $\varphi \dot{\in} t$, then $\mathcal{M}_T, t \models \varphi$.

By induction on the structure of formulæ:

- if $\varphi = q$ or $\varphi = \neg q$ for $q \in \mathbb{AP}(\psi)$, then $\mathcal{M}_T, t \models \varphi$ iff $\varphi \in t$ by construction of V;
- if $\varphi = \varphi' \wedge \varphi''$ or $\varphi = \varphi' \vee \varphi''$, the claim follows immediately by induction and the definition of types;
- if $\varphi = \neg\varphi' \vee \neg\varphi''$, then $\varphi \dot{\in} t$ implies that $\varphi' \dot{\notin} t$ or $\varphi'' \dot{\notin} t$, since, otherwise, $\neg\varphi$ would be in t. This implies that $\neg\varphi' \dot{\in} t$ or $\neg\varphi'' \dot{\in} t$, and thuse we have $\mathcal{M}_T, t' \models \neg\varphi'$ or $\mathcal{M}_T, t' \models \neg\varphi''$ by induction. Hence, $\mathcal{M}_T, t \models \varphi$
- the case $\varphi = \neg\varphi' \wedge \neg\varphi''$ is analogous;
- let $\varphi = \langle\alpha\rangle\varphi' \in t$. The definition of Δ^α implies that $\varphi' \dot{\in} t'$, for all t' with $\Delta^\alpha(t, t')$. By induction, $\mathcal{M}_T, t' \models \varphi'$, for all t' with $\varphi' \dot{\in} t'$, and thus $\mathcal{M}_T, t \models \langle m\rangle\varphi'$;
- if $\varphi = \langle\alpha\rangle\neg\varphi' \in t$, then $t \notin \mathtt{bad}(T)$ because $\mathtt{Upd}(T) = T$, and thus there exists $t' \in T$ with $\Delta^\alpha(t, t')$ and $\varphi' \dot{\notin} t'$. By definition of types, $\neg\varphi' \dot{\in} t'$, and thus we have $\mathcal{M}_T, t' \models \varphi'$ by induction. Hence, $\mathcal{M}_T, t \models \langle m\rangle\neg\varphi'$.

Completeness.

For all $\varphi \in cl(\psi)$, if φ is satisfiable, then there exists some $t \in T$ with $\varphi \dot{\in} t$.

Given a satisfiable formula φ, take a model $\mathcal{M} = (W, \{R^\alpha\}_{\alpha \in \mathbb{MOD}}, V)$ with $\mathcal{M}, w_\varphi \models \varphi$. For a world $w \in W$, we define its type $t(w) = \{\varrho \in \mathtt{Lean}(\psi) \mid \mathcal{M}, w \models \varrho\}$, and we define $T(W) = \{t(w) \mid w \in W\}$. Obviously, due to the semantics of the modality formulæ and the definition of $t(.)$, $(w, w') \in R^m$ implies $\Delta^\alpha(t(w), t(w'))$. Then we show, by induction on i, that $T(W) \subseteq T^i$. Since $\varphi \dot{\in} t(w'_\varphi)$ by construction, this proves the claim.

- $T(W) \subseteq T^0$ since T^0 contains all types $t \subseteq \mathtt{Lean}(\psi)$.
- Let $T(W) \subseteq T^i$ and assume $T(W) \not\subseteq T^{i+1}$. Then there is some $w \in W$ such that $t(w) \in \mathtt{bad}(T^i)$. So there is some $\langle\alpha\rangle\neg\varphi' \in t(w)$ and, for all $t' \in T^i$ with $\Delta^\alpha(t(w), t')$, we have $\varphi' \dot{\in} t'$. Hence, there is no $w' \in W$ with $(w, w') \in R^\alpha$ and $\mathcal{M}, w' \models (\neg\varphi')$, in contradiction to $\mathcal{M}, w \models \langle m\rangle\neg\varphi'$.

$\square$

5.4 Implementation

In this section, we describe how to implement our algorithm using Binary Decision Diagrams (BDDs).

5.4.1 Implicit Representation of Set of Types

Our implementation relies on a symbolic representation and manipulation of sets of ψ-types using Binary Decision Diagrams (BDDs) [Bryant, 1986]. BDDs provide a canonical representation of Boolean functions. Experience has shown that this representation is very compact for very large Boolean functions [Clarke et al., 1996] and that various operations on Boolean functions can be carried out efficiently on their BDD representation.

We introduce a bit-vectors representation of ψ-*types*: for $\mathtt{Lean}(\psi) = \{\varphi_1, \ldots, \varphi_n\}$, we represent a type $t \subseteq \mathtt{Lean}(\psi)$ by a vector $\overrightarrow{t} = \langle t_1, \ldots, t_n \rangle \in \{0,1\}^n$, such that $\varphi_i \in t$ iff $t_i = 1$.

A BDD with n variables is then used to represent a set of such bit vectors.

Let $\varphi \in \mathtt{cl}^*(\psi)$, we define $\varphi \dot{\in} t$ on the bit-vector representation as follows:

$$\mathtt{status}_\varphi(\overrightarrow{t}) = \begin{cases} 1 & \text{if } \varphi = \top \\ 0 & \text{if } \varphi = \bot \\ t_i & \text{if } \varphi \in \mathtt{Lean}(\psi) \\ \mathtt{status}_{\varphi'}(\overrightarrow{t}) \wedge \mathtt{status}_{\varphi''}(\overrightarrow{t}) & \text{if } \varphi = \varphi' \wedge \varphi'' \\ \mathtt{status}_{\varphi'}(\overrightarrow{t}) \vee \mathtt{status}_{\varphi''}(\overrightarrow{t}) & \text{if } \varphi = \varphi' \vee \varphi'' \\ \neg\mathtt{status}_{\varphi'}(\overrightarrow{t}) & \text{if } \varphi = \neg\varphi' \end{cases}$$

We use $a \rightarrow b$ to denote the implication and $a \leftrightarrow b$ to denote the equivalence of two Boolean formualae a and b over bit vectors.

We can now construct the BDD of the relation Δ^α, for $\alpha \in \mathbb{MOD}$. This BDD relates all pairs $(\overrightarrow{t}, \overrightarrow{t}')$ that are consistent w.r.t. the program α, such that $\overrightarrow{t}'$ supports all of $\overrightarrow{t}$'s $[\alpha]\varphi$ formualae, and vice versa $\overrightarrow{t}$ support all of $\overrightarrow{t}'$'s $[\overline{\alpha}]\varphi$ formualae:

$$\Delta^\alpha(\overrightarrow{t}, \overrightarrow{t}') \stackrel{def}{=} \bigwedge_{1 \leq i \leq n} \begin{cases} t_i \rightarrow \mathtt{status}_\varphi(\overrightarrow{t}') & \text{if } \varphi_i = [\alpha]\varphi \\ t'_i \rightarrow \mathtt{status}_\varphi(\overrightarrow{t}) & \text{if } \varphi_i = [\overline{\alpha}]\varphi \\ \top & \text{otherwise} \end{cases}$$

Given a set T of types, we write the corresponding characteristic function as χ_T, and we use $\chi_{\overline{T}}$ for the characteristic function of the complement of T.

The predicate $\mathtt{bad}_i(T)$, for a formula $\varphi_i = \langle\alpha\rangle\neg\varphi$ is defined as follows:

$$\chi_{\mathtt{bad}_i(T)}(\overrightarrow{t}) \stackrel{def}{=} t_i \wedge \forall \overrightarrow{t}' : ((\chi_T(\overrightarrow{t}') \wedge \Delta^\alpha(\overrightarrow{t}, \overrightarrow{t}')) \rightarrow \neg\mathtt{status}_\varphi(\overrightarrow{t}'))$$

and thus $\mathtt{bad}(T)$ can be written as

$$\chi_{\mathtt{bad}(T)}(\overrightarrow{t}) \stackrel{def}{=} \bigvee_{1\leq i\leq n} \chi_{\mathtt{bad}_i(T)}(\overrightarrow{t})$$

In our implementation, we compute each $\chi_{\overline{\mathtt{bad}_i(T)}}$ and we use it in the implementation of the top-down algorithm. It is easy to see that $\chi_{\overline{\mathtt{bad}_i(T)}}(\overrightarrow{t})$ is equivalent to

$$t_i \rightarrow \exists \overrightarrow{t}' : (\chi_T(\overrightarrow{t}') \wedge \Delta^\alpha(\overrightarrow{t}, \overrightarrow{t}') \wedge \mathtt{status}_\varphi(\overrightarrow{t}'))$$

Then, the BDD of the fixpoint computation is initially set to the true constant, and the function `Upd(.)` is implemented as:

$$\chi_{\mathtt{Upd}(T)}(\overrightarrow{t}) \stackrel{def}{=} \chi_T(\overrightarrow{t}) \wedge \bigwedge_{1\leq i\leq m} (\chi_{\overline{\mathtt{bad}_i(T)}}(\overrightarrow{t}))$$

Finally, the solver is implemented as iterations over the sets $\chi_{\mathtt{Upd}(T)}$ until a fixpoint is reached. The satisfiability condition is met if the formula ψ is present in a ψ-*type* of T such that:

$$\exists \overrightarrow{t} \left[\chi_T(\overrightarrow{t}) \wedge \mathtt{status}_\psi(\overrightarrow{t})\right]$$

5.5 Conclusion

In this chapter, we review in the details how satisfiability solvers for K are designed.

We presented our own implementation of a decision procedure for $\mathtt{K}_{(n,back)}$ based on the works of [Pan et al., 2006]. $\mathtt{K}_{(n,back)}$ is a subset of the logic dealt with in the work of [Tanabe et al., 2005].

Extending the expressive power of these logics while preserving decidability is a very challenging task. In particular we know that $\mathtt{K}_{(n,back)}$ with recursion (fixpoints), nominals and counting is undecidable [Bonatti and Peron, 2004].

In the following chapter, we show that $\mathtt{K}_{(n,back)}$ is already enough for checking query containment for a fragment of SPARQL with the optional operator. We also use the satisfiability testing prototype that we have developed and described in Section 5.4 as a basis for the practical experiments conducted in the next chapter.

Part II

Contribution

6

Containment of Well-designed SPARQL queries

This chapter addresses `SPARQL` *query containment with optional matching. The optionality feature is one of the most complicated constructors in* `SPARQL` *and also the one that makes this language depart from classical query languages such as relational conjunctive queries. We focus on the class of well-designed* `SPARQL` *queries, proposed in the literature as a fragment of the language with the optional matching and good properties regarding query evaluation. We investigate an approach based on a logical reduction to satisfiability of modal logic. We translate well-designed* `SPARQL` *queries in terms of modal logic formulae. We then use a satisfiability solver to check for containment. This approach provides a gain of an order of magnitude in the time spent for solving the containment problem. We show that this approach is efficiently implementable and extensible.*

Contents

6.1 Introduction **76**

6.2 Well-Designed Graph Patterns **76**

6.3 Pattern Trees **78**

6.4 Containment of Well-Designed Graph Patterns **82**

6.4.1 Bag Semantics and Set Semantics 82

6.4.2 Subsumption Relation 83

6.5 Logical Encoding **84**

6.5.1 RDF Graphs as Transition Systems 84

6.5.2 Encoding Queries as K-Formulæ 92

6.5.3 Containment Problem 96

6.6 Reducing Query Containment to Unsatisfiability **97**

6.7 Experimentation **100**

6.7.1 Benchmark 100

6.8 Query Containment Solvers **103**

6.8.1 MSPASS 103

6.8.2 SPARQL-Algebra 104

6.8.3 Experimentation 104

6.9 Conclusion **105**

6.1 Introduction

Given two SPARQL queries q and q', the query containment problem amounts to determine whether, for any RDF graph, the result of evaluating q is always included in the result of evaluating q'.

The aim of this chapter is to address SPARQL query containment using modal logic. We focus on the class of well-designed SPARQL queries restricted to the AND and OPTIONAL operators, introduced in the literature by [Pérez et al., 2006]. Well-designed queries form a natural fragment of SPARQL that is very common in practice. This class is defined by imposing a simple and natural syntactic restriction on optional parts.

In [Letelier et al., 2012, Pichler and Skritek, 2014], the authors report the complexity of the query containment problem for several fragments of SPARQL. For the well-designed SPARQL fragment, restricted to AND and OPTIONAL operators, the containment complexity is Π_2^P.

In this chapter, we show that the modal logic $\mathtt{K}_n$ is expressive enough to deal with query containment for well-designed SPARQL queries. Furthermore, this logic admits PSPACE-Complete satisfiability problems [Stockmeyer, 1976, Ladner, 1977] that is implemented in practice, i.e. [Patel-Schneider and Horrocks, 1999, Hustadt and Schmidt, 2000, Tanabe et al., 2008]. Hence, our approach opens a way to take advantage of these implementations.

6.2 Well-Designed Graph Patterns

In [Pérez et al., 2006], the authors identify the so-called well-designed graph patterns. Well-designed patterns form a natural fragment of SPARQL that is very common in practice. Furthermore, well-designed patterns have several interesting features.

Definition 6.2.1 (Well-Designed Graph Pattern [Pérez et al., 2006])**.** *A* UNION*-free graph pattern P is well-designed if for every subpattern $P' = (P_1 \text{ OPTIONAL } P_2)$ of P and for every variable $?x$ occurring in P, the following condition holds*

- *if $?x$ occurs both inside P_2 and outside P', then it also occurs in P_1.*

Example 6.2.1 (Well-Designed Graph Pattern)**.** *Given the following:*

$$P = (((?x, \mathtt{name}, \mathtt{John})) \text{ OPTIONAL } (\underbrace{(?y, \mathtt{email}, ?e) \text{ OPTIONAL } (?x, \mathtt{phone}, ?z)}_{P'}))$$

with $P' = (P_1 \text{OPTIONAL} P_2)$ and $P_1 = ((?y, \mathtt{email}, ?e)$ and $P_2 = (?x, \mathtt{phone}, ?z)$.

By Definition 6.2.1, P is not a well-designed graph pattern, since $?x$ occurs inside P_2 and outside P', but not in P_1.

Another class of graph patterns defined in [Pérez et al., 2006] are well-designed patterns in *opt normal form*. In general, a pattern containing only the operators AND and OPTIONAL is in *opt normal form* if the OPTIONAL operator never occurs in the scope of an AND operator.

Definition 6.2.2 (OPT normal form)**.** *A UNION-free graph pattern P is in opt normal form if either:*

1. *P is constructed by using only the* AND *operators*

2. *$P = (O_1 \text{ OPTIONAL } O_2)$, with O_1 and O_2 patterns in opt normal form.*

In Theorem 4.11 of [Pérez et al., 2009], the authors show that every well-designed pattern can be transformed into *opt normal form* in polynomial time, using the following reordering rules.

Proposition 6.2.1 (Reordering rules)**.** *Let P be a well-designed pattern and P' a pattern obtained from P by using one of the following reordering rules:*

$$(P_1 \text{ AND } (P_2 \text{ OPTIONAL } P_3)) \rightarrow ((P_1 \text{ AND } P_2) \text{ OPTIONAL } P_3) \qquad (a)$$

$$((P_1 \text{ OPTIONAL } P_2) \text{ AND } P_3) \rightarrow ((P_1 \text{ AND } P_3) \text{ OPTIONAL } P_2) \qquad (b)$$

Then, P' is a well-designed pattern equivalent to P.

From proposition 6.2.1 and associativity and commutativity of AND, the following theorem is obtained.

Theorem 6.2.2 ([Pérez et al., 2006])**.** *Every well-designed graph pattern P is equivalent to a pattern in the following opt normal form:*

$$((\ldots(t_1 \text{ AND } t_2 \text{ AND } \ldots \text{ AND } t_k) \text{ OPTIONAL } O_1) \text{ OPTIONAL } O_2) \ldots \text{ OPTIONAL } O_n) \qquad (*)$$

where each t_i is a triple pattern, $n \geq 0$ and each O_n has the same form of $(*)$.

The proof of the theorem is based on term rewriting techniques.

Example 6.2.2. *Consider the well-designed pattern*

$$P = ((\texttt{?x,name,John}) \text{ OPTIONAL } (\texttt{?x,email,?e})) \text{ AND } (\texttt{?x,phone,?z})$$

The OPTIONAL *normalized form is*

$$P' = ((\texttt{?x,name,John}) \text{ AND } (\texttt{?x,phone,?z})) \text{ OPTIONAL } (\texttt{?x,email,?e})$$

Well-designed graph patterns (in *opt normal form*) allow for a natural tree representation, formalized by so-called *pattern tree* in [Letelier et al., 2012].

6.3 Pattern Trees

A rooted tree is defined as a tuple $T = (V, E, r)$, where V is the set of nodes, E is a set of edges, and $r \in V$ is the root of the tree. Trees are assumed to be unordered and undirected. For any node $n \in V$, we write T_n to denote the subree of T rooted at n, composed by all the descendants of n in the tree.

Using this terminology, we can now report the tree representation of `SPARQL` graph patterns presented in [Letelier et al., 2012].

Definition 6.3.1 (Pattern Tree [Letelier et al., 2012])**.** *A pattern tree $\mathcal{T}$ is a pair $\mathcal{T} = (T, \mathcal{P})$, where $T = (V, E, r)$ is a rooted unordered tree, and $\mathcal{P} = (P_n)_{n \in V}$ is a labeling of the nodes of T, such that P_n is a non-empty set of triple patterns, for every $n \in V$.*

For pattern trees, we depict the tree structure with the corresponding labels in every node, as in the following example.

Example 6.3.1. *Consider the following graph patterns:*

$$P_A = (\texttt{?a}, \texttt{email}, \texttt{?e})$$

$$P_B = (\texttt{?a}, \texttt{email}, \texttt{?e}) \text{OPTIONAL} (\texttt{?a}, \texttt{web}, \texttt{?w})$$

$$P_C = \Big(\big((\texttt{?a}, \texttt{name}, \texttt{?n}) \text{AND} (\texttt{?a}, \texttt{email}, \texttt{?e})\big) \text{OPTIONAL} (\texttt{?a}, \texttt{web}, \texttt{?w})\Big) \text{OPTIONAL} \Big((\texttt{?a}, \text{phone}, \texttt{?p}) \text{OPTIONAL} (\texttt{?a}, \texttt{fax}, \texttt{?f})\Big)$$

The corresponding pattern trees ($\mathcal{T}_A$, $\mathcal{T}_B$, $\mathcal{T}_C$ respectively) are shown in figure 6.1.

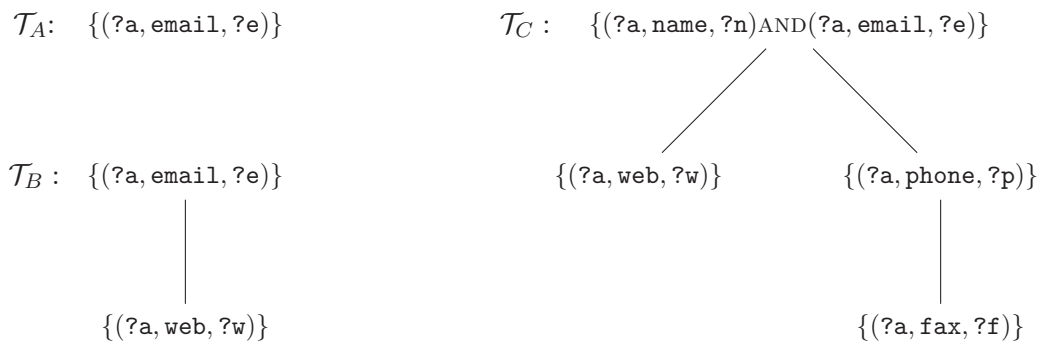


Figure 6.1: Representation of pattern trees

Well-designed graph patterns (in *opt normal form*) consist of conjunctive parts (represented by the nodes of the pattern tree) that are located in a structure of nested OPTIONAL operators (modelling by the tree-structure). The order of child nodes in such a tree does not matter. This is due to the fact that for well-designed graph patterns, the following property holds:

$$((P_1 \text{ OPTIONAL } P_2) \text{ OPTIONAL } P_3) \rightarrow ((P_1 \text{ OPTIONAL } P_3) \text{ OPTIONAL } P_2)$$

6.3.0.1 Syntactic Relationship between Pattern Trees and Graph Patterns

Given a pattern tree $\mathcal{T}$ and a node $n \in V$, a subtree of $\mathcal{T}$ rooted at n, denoted by $\mathcal{T}_n$, is a pattern tree composed of n and a connected subset of all its descendants. We denote with $\texttt{vars}(P_n)$ the set of variables that occur in the triples of P_n, and with $\texttt{vars}(\mathcal{T})$ the set $\bigcup_{n \in V} \texttt{vars}(P_n)$. We denote by $\texttt{and}(P_n)$ the graph pattern obtained by putting in AND all triples in P_n. Consider $P = \{t_1, \ldots, t_j\}$, then $\texttt{and}(P_i) = (t_1 \text{ AND } t_2 \text{ AND } \ldots \text{ AND } tj)$. We denote with $\texttt{and}(\mathcal{T})$ the conjunction of all triples that occur in $\mathcal{T}$.

These notations are used in the following to establish a syntactic relationship between pattern trees and SPARQL graph patterns. Towards this goal, we need the definition of a transformation function TR(_,_,_).

Definition 6.3.2 (Transformation function TR)**.** *Consider a pattern tree $\mathcal{T} = (T, \mathcal{P})$ and a set of ordering functions $\Sigma = \{\sigma_n \mid n \in V\}$, such that σ_n defines an ordering on the children of n. That is, if n has k children, then σ_n is a function from $\{1, \ldots, k\}$ to the set of children of n, such that $\sigma(1)$ is the first child, $\sigma(2)$ the second one, and so on. The transformation function $\text{TR}(\mathcal{T}, n, \Sigma)$ of $\mathcal{T}_n$ is defined as follows.*

Assume n has k children in $\mathcal{T}$, then

$$\text{TR}(\mathcal{T}, n, \Sigma) = (\ldots((\textit{and}(P_n) \text{ OPTIONAL } \text{TR}(\mathcal{T}, \sigma_n(1), \Sigma)) \text{ OPTIONAL } \text{TR}(\mathcal{T}, \sigma_n(2), \Sigma)) \text{ OPTIONAL } \text{TR}(\mathcal{T}, \sigma_n(k), \Sigma))$$

and if n has no children, then $\text{TR}(\mathcal{T}, n, \Sigma) = \textit{and}(P_n)$

Example 6.3.2. *Applying* $\textsc{Tr}(\_,\_,\_)$ *to the pattern trees* $\mathcal{T}_A$, T_B *and* T_C *of Example 6.3.1 with an ordering sibling nodes from left to right* ($\overrightarrow{\Sigma}$)*, we get the following* `SPARQL` *graph patterns.*

$\textsc{Tr}(\mathcal{T}_A, r, \overrightarrow{\Sigma}) = (\texttt{?a,email,?e})$

$\textsc{Tr}(\mathcal{T}_B, r, \overrightarrow{\Sigma}) = (\texttt{?a,email,?e})$ OPTIONAL $(\texttt{?a,web,?w})$

$\textsc{Tr}(\mathcal{T}_C, r, \overrightarrow{\Sigma}) = ((((\texttt{?a,name,?n})$ AND $(\texttt{?a,email,?e}))$ OPTIONAL $(\texttt{?a,web,?w}))$

OPTIONAL $((\texttt{?a,phone,?p})$ OPTIONAL $(\texttt{?a,fax,?f})))$

Note, we get $\textsc{Tr}(\mathcal{T}_A, r, \overrightarrow{\Sigma}) = P_A$, $\textsc{Tr}(\mathcal{T}_B, r, \overrightarrow{\Sigma}) = P_B$ *and* $\textsc{Tr}(\mathcal{T}_C, r, \overrightarrow{\Sigma}) = P_C$. *If we change the order of the sibling nodes for right to left, we get:* $\textsc{Tr}(\mathcal{T}_C, r, \overleftarrow{\Sigma}) \neq P_C$.

Tr establishes a syntactic relationship between pattern trees and `SPARQL` graph patterns. Notice that several (different) `SPARQL` patterns can be obtained from a single pattern tree depending on the ordering functions used in the transformation. Thus, it is not trivial to establish a semantic relationship between a pattern tree $\mathcal{T}$ and an arbitrary transformation of $\mathcal{T}$ [Letelier et al., 2012]. In the following, a well-designedness condition for pattern trees is introduced. This notion is crucial in defining a semantics for pattern trees *à la* [Pichler and Skritek, 2014].

6.3.0.2 Well-Designed Pattern Trees

Definition 6.3.3 (Well-designed pattern tree [Letelier et al., 2012])**.** *A pattern tree* $\mathcal{T}$*, where* $\mathcal{T} = ((V, E, r), \mathcal{P})$*, is a well-designed pattern tree* `wdPT`*, if for every variable* `?x`*, occurring in* $\mathcal{T}$*, the set* $\{n \in V \mid \texttt{?x} \in \texttt{vars}(P_n)\}$ *induces a connected subgraph of* T.

Example 6.3.3. *The pattern trees in Example 6.3.1 are well-designed, while the following pattern trees are not:*

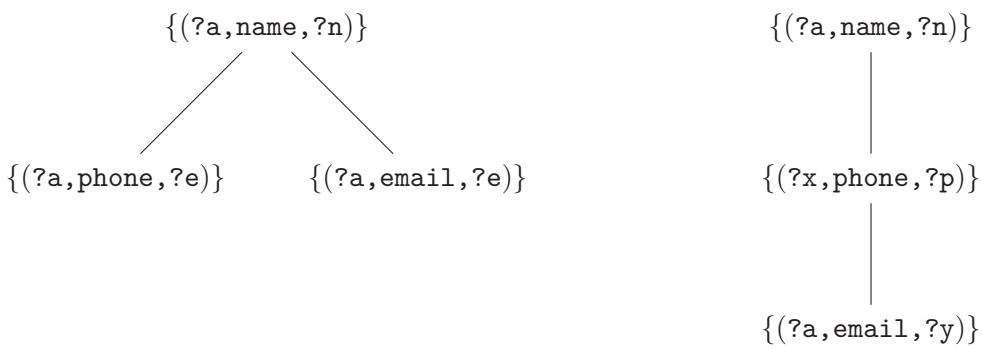


Figure 6.2: Pattern trees that are not well-designed

Variable `?e` *in the tree on the left, and variable* `?a` *in the tree on the right, induce disconnected subgraphs.*

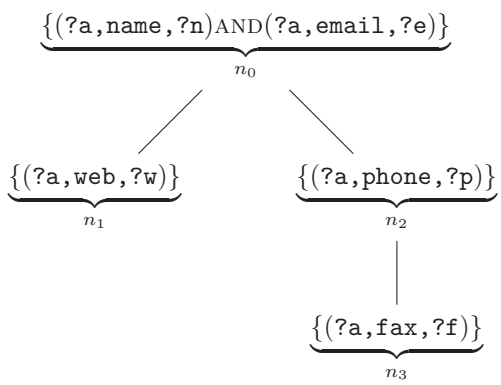
This restriction ensures that the result of the translation from a `wdPT` into a graph pattern is well-designed. Conversely, also every *pattern tree* derived from a well-designed graph pattern in *opt normal form* yields a `wdPT`. This provides a polynomial time transformation from `wdPT`s into equivalent well-designed graph patterns and viceversa.

Normal Form. In [Letelier et al., 2012], the authors introduce a *normal form* for the `wdPT` and it was shown, that every `wdPT` can be efficiently transformed into an equivalent `wdPT` in *normal form*. The next definition uses the following notation of `newvars`. Let $branch(n) = n^1, \ldots, n^K$ with $n^1 = r$ and $n^k = n$ be the unique sequence of nodes from the root r to n, then for nodes n, $\widehat{n}$ is the parent of n, we have $\texttt{newvars}(n) = vars(n) \setminus \texttt{var}(branch(\widehat{n}))$.

Definition 6.3.4 (Well-designed Pattern Trees in Normal Form)**.** *Let $\mathcal{T} = ((V, E, r), \mathcal{P})$ be a* `wdPT`*. $\mathcal{T}$ is also a* `wdPT` *in normal form if and only if*

1. *for every variable* `?x` $\in$ `vars`$(\mathcal{T})$*, there exists a unique node $n \in V$ s.t.* `?x` $\in$ `newvars`(n) *and all other nodes $n' \in V$ with* `?x` $\in$ `vars`(n') *are descendants of n;*
2. `newvars`$(n) \neq \emptyset$ *for every $n \in V$.*

Example 6.3.4. *Consider the* `wdPT`*, $\mathcal{T}_C$ of example 6.3.1:*



$\mathcal{T}_C$ is a `wdPT` *in normal form. In Table 6.1, the functions* `newvars` *and* `vars` *are defined for each node. It easily to check that conditions (1) and (2) hold.*

	newvars	vars
n_0	{?a, ?n, ?e}	{?a, ?n, ?e}
n_1	{?w}	{?a, ?w}
n_2	{?p}	{?a, ?p}
n_3	{?f}	{?a, ?f}

Table 6.1: Function `newvars` and `var` for $\mathcal{T}_C$

Condition (1) holds since each variable is defined in a unique node (there are not multiple occurrences of a variable in the `newvars` *column) . Moreover, the only variable* `?a` *that appears in each node of the tree (*`?a` *is in each row of the column* `vars`*) is defined in the root of the tree, so it is trivial to check that each node is a descendant of the root.*

Condition (2) holds because each node introduces at least one new variable (column `newvars` *does not present empty rows).*

6.3.0.3 Semantics of Pattern Trees

Analogously to graph patterns, the result of evaluating a *wdPT* $\mathcal{T}$ over some RDF graph G is denoted by $[\![\mathcal{T}]\!]_G$.

In [Letelier et al., 2012], the set $[\![\mathcal{T}]\!]_G$ of mappings solutions was defined via a translation to graph patterns (using the translation function Tr). However, for wdPTs in *normal form*, the set of solutions has a nice direct characterization in terms of maximal subtrees of $\mathcal{T}$. A maximal subtree of $\mathcal{T}$, is a subtree of $\mathcal{T}$ whose leaves are also leaves of $\mathcal{T}$.

Lemma 6.3.1 ([Pichler and Skritek, 2014])**.** *Let $\mathcal{T}$ be a wdPT in normal form and G be an RDF graph, then $\rho \in [\![\mathcal{T}]\!]_G$ iff there exists a subtree $\mathcal{T}'$ of $\mathcal{T}$, s.t.*

1. $dom(\rho) = vars(\mathcal{T}')$,
2. *$\mathcal{T}'$ is the maximal subtree of $\mathcal{T}$, s.t.* $\rho \sqsubseteq [\![and(\mathcal{T}')]\!]_G$

It can be easily checked that $\mathcal{T}'$ is uniquely defined by $dom(\rho)$. This tree is referred as $\mathcal{T}_\rho$.

6.4 Containment of Well-Designed Graph Patterns

In the rest of the chapter, we consider well-designed graph patterns (without projections nor filters) à la [Letelier et al., 2012]. Moreover, SPARQL allows the use of blank nodes in graph patterns, which we do not consider here (blank nodes can be replaced by variables [Gutierrez et al., 2011]).

6.4.1 Bag Semantics and Set Semantics

In real database systems, queries are usually evaluated under bag semantics, not set semantics: input database relations may be bags (multisets), and queries may return bags as answers. The same holds for SPARQL. In particular, SPARQL queries are evaluated under bag semantics, since duplicate tuples are not eliminated unless explicitly specified in the syntax using the SELECT DISTINCT construct. The reason for not eliminating duplicate tuples in SPARQL is that the values of aggregate operators, such as AVG and COUNT, depend on the multiplicities of the tuples in the graph. Most of the studies on query containment use set semantics rather than bag semantics. This is due to very high complexity: for instance, containment becomes undecidable (even for) union of conjunctive queries under bag semantics [Ioannidis and Ramakrishnan, 1995].

Nevertheless, for the fragment considered in this chapter (allowing only for AND and OPTIONAL in absence of blank nodes) both semantics coincide [Pérez et al., 2006]. Therefore, in absence of blank nodes in graph patterns, duplicated solutions could be generated only by the use of UNION and GRAPH operators (see [Garlik and Seaborne, 2013] for details).

6.4.2 Subsumption Relation

For the containment of queries, in this chapter we consider the subsumption relation $\sqsubseteq$ rather than the classical relational subset $\subseteq$ [Pérez et al., 2009].

Solutions for queries containing the OPTIONAL operator are essentially incomplete and may possibly bind only a subset of the variables in the query [Garlik and Seaborne, 2013, Pérez et al., 2006], see Example 2.5.2. Hence, in the presence of partial query answers, subsumption is a more natural notion of containment [Arenas and Pérez, 2011].

Definition 6.4.1 (Subsumption relation $\sqsubseteq$)**.** *A mapping ϱ_1 is subsumed by ϱ_2, denoted by $\varrho_1 \sqsubseteq \varrho_2$, if $\mathtt{dom}(\varrho_1) \cap \mathtt{dom}(\varrho_2) = \mathtt{dom}(\varrho_1)$ and for every $?x \in \mathtt{dom}(\varrho_1)$ it holds that $\varrho_1(?x) = \varrho_2(?x)$ (implying that ϱ_1 ϱ_2 are compatible). Moreover, $\varrho_1 \sqsubset \varrho_2$ whenever $\varrho_1 \sqsubseteq \varrho_2$ and $\varrho_1 \neq \varrho_2$.*

Example 6.4.1 (Subsumption relation $\sqsubseteq$)**.** *Consider the following two queries:*

q_1 : *SELECT* $?a\ ?n$
WHERE $\{(?a, \mathtt{name}, ?n)\}$

q_2 : *SELECT* $?a\ ?n\ ?p$
WHERE $\{(?a, \mathtt{name}, ?n)$ OPTIONAL $(?a, \mathtt{phone}, ?p)\}$

and the following RDF graph G

$$G = \{(\mathtt{A}_1, \mathtt{name}, \mathtt{Joe}), (\mathtt{A}_1, \mathtt{phone}, \mathtt{1111})\}$$

containing the first two triples of Example 2.3.1. Then,

$$[\![q_1]\!]_G = \{\varrho_1 = \{?a \to \mathtt{A}_1\ ,\ ?n \to \mathtt{Joe}\}\}$$

while

$$[\![q_2]\!]_G = \{\varrho_2 = \{?a \to \mathtt{A}_1\ ,\ ?n \to \mathtt{Joe}\ ,\ ?p \to \mathtt{1111}\}\}$$

Hence $q_1 \not\sqsubseteq q_2$. This is, however, unintuitive, since the answer to q_2 contains strictly more information than that to q_1, and it is easy to see that for no graph G, pattern q_2 returns fewer bindings than q_1, that $q_1 \sqsubseteq q_2$.

The definition of subsumption of mappings can be extendend to subsumption of sets of mappings. Given sets of mappings Ω_1 and Ω_2, then Ω_1 is subsumed by Ω_2, denoted by $\Omega_1 \sqsubseteq \Omega_2$, if for every $\varrho_1 \in \Omega_1$ there exists a $\varrho_2 \in \Omega_2$ such that $\varrho_1 \sqsubseteq \varrho_2$.

Further, for two wdPTs $\mathcal{T}_1$ is subsumed by $\mathcal{T}_2$, denoted by $\mathcal{T}_1 \sqsubseteq \mathcal{T}_2$, if $[\![\mathcal{T}_1]\!]_G \sqsubseteq [\![\mathcal{T}_2]\!]_G$ holds for every graph G.

In Lemma 4.2 of [Letelier et al., 2012], the authors provide a necessary and sufficient condition to test whether $\mathcal{T}_1 \sqsubseteq \mathcal{T}_2$.

Lemma 6.4.1. *(Necessary and sufficient condition [Letelier et al., 2012]) Consider wdPTs $\mathcal{T}_1$ and $\mathcal{T}_2$ with root r_1 and r_2 respectively. Then $\mathcal{T}_1 \sqsubseteq \mathcal{T}_2$ 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 s.t:*

- $vars(\mathcal{T}_1) \subseteq vars(\mathcal{T}_2)$. (1)
- *there exists a homomorphism from the triple in $\mathcal{T}_2$ to the triples in $\mathcal{T}_1$ that is the identity over $vars(\mathcal{T}_1)$.* (2)

Lemma 4.2 of [Letelier et al., 2012] yields a straightforward Π_2^P procedure to check whether $\mathcal{T}_1 \sqsubseteq \mathcal{T}_2$. Check for every subtree $\mathcal{T}_1'$ of $\mathcal{T}_1$ that there exists a subtree $\mathcal{T}_2'$ of $\mathcal{T}_2$ and a homomorphism satisfying properties (1) and (2). In Theorem 4.3 of [Letelier et al., 2012], it is shown that Π_2^P is also the lowest bound of the problem.

6.5 Logical Encoding

In the following, we explain how pattern trees are encoded as modal logic formulæ. SPARQL is interpreted over graphs, hence we encode it into a graph logic, specifically the $\mathtt{K}_n$ logic, interpreted over label transition systems. In the next section, we show how to translate RDF graphs into transition systems and SPARQL pattern trees into $\mathtt{K}_n$-formulæ. Therefore, query containment in SPARQL can be reduced to unsatisfiability in $\mathtt{K}_n$.

6.5.1 RDF Graphs as Transition Systems

In this section, we show how to translate RDF graphs into labeled transition systems. First of all, translating RDF graphs into transition systems is necessary in order to restrict the models that satisfy $\mathtt{K}_n$-formulæ obtained from the translation of the queries. Additionally, if RDF graphs can be translated into transition systems, then model checking can be used to evaluate SPARQL queries [Mateescu et al., 2009]. Notice the modal logic $\mathtt{K}_n$ admits PSPACE-Complete satisfiability problem, the same complexity of the evaluation problem for the full SPARQL. We refer the reader to Chapter 4 for more details about satisfiability testing for K and its variants.

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

- For each triple $(s, p, o) \in G$, s and o become points of the transition system and p is a transition program, such that an edge $\langle s, o \rangle$ labeled p is built. This approach does not work in general case, for instance predicates or properties p can also be nodes in an RDF graph as shown in Figure 6.3, where `childof` cannot be a transition program.

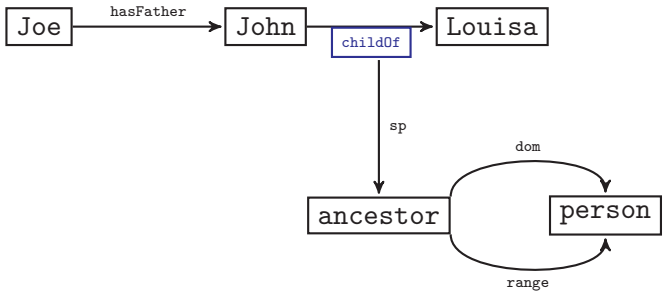


Figure 6.3: An RDF graph where a predicate appears as a node [Chekol, 2012]

- for each triple `t=(s,p,o)` $\in G$, two sets of points are introduced in the transition system: one set for each triple n_t and another set for each element of the triple n_s, n_p and n_o where the atomic propositions `s`,`p` and `o` are set to be `true` respectively. Additionally, there are edges $\langle n_s, n_t \rangle$, $\langle n_t, n_p \rangle$ and $\langle n_t, n_o \rangle$ in the transition system that are accessible through programs s, p, o respectively. This approach overcomes the limitation of the first one.

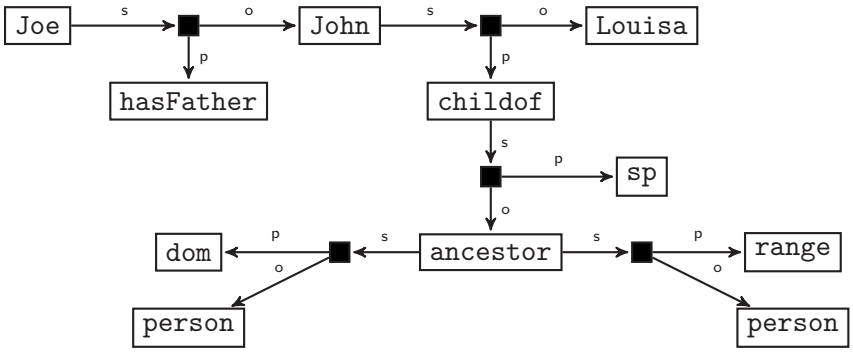


Figure 6.4: Transition system encoding the RDF graph as bipartite graph [Chekol, 2012]

This technique has been introduced in [Baget, 2005, Calvanese et al., 2008] and an application in the RDF context is reported in [Chekol, 2012]. The author of this work introduces an encoding of RDF graphs based on bipartite graphs (as in Figure 6.4). In the following, we discuss in details how this technique works and we introduce our encoding of RDF graphs as bipartite graphs.

6.5.1.1 Encoding of RDF graphs

We propose an encoding of an RDF graph as a transition system in which points correspond both to RDF entities and RDF triples. Different modalities are used for distinguishing subjects, predicates, and objects of triples. Expressing predicates as points, instead of modalities, makes it possible to deal with full RDF expressiveness in which a predicate may also be the subject or object of a statement.

Before formally defining our RDF graph encoding, we explain the basic idea behind the encoding using the following example. Consider the graph G of the Example 6.4.1.

$$G = \{(\mathtt{A}_1, \mathtt{name}, \mathtt{Joe}), (\mathtt{A}_1, \mathtt{phone}, \mathtt{1111})\}$$

1. For each triple of G, there are three points in the transition system, where the subject, the predicate and the object of the triple are true. These points are related by two modalities $\langle s \rangle$ and $\langle o \rangle$ that associate the subject to the predicate and the predicate to the object, respectively (as in Figure 6.5).

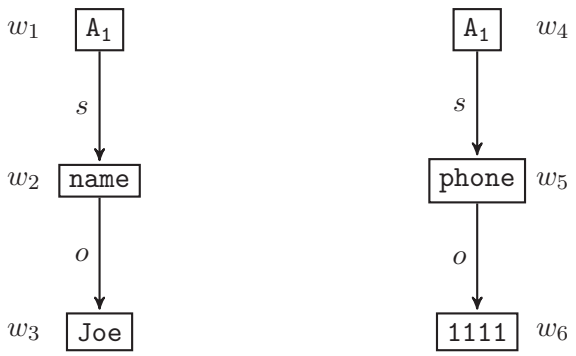


Figure 6.5: Encoding of triples

   Note, the RDF entity $\mathtt{A}_1$ occurs twice in G, hence in the transition system there are two points (w_1 and w_4), labeled $\mathtt{A}_1$, where it is true.

2. Once we have established the encoding of the triples, then there are many ways to relate triples among them in a transition system. We choose one and explain it below. We recall that in this work, graphs can be queried by well-designed graph patterns in *opt normal form*, containing only AND and OPTIONAL operators. As a consequence, in our encoding we are interested in distinguishing two different sets of triples of a graph. The triples involved in the evaluation of a conjunctive component and the triples involved in the evaluation of an optional component.

   For this purpose, we introduce two modalities α and β defined as follows.

- The modality α is used to identify triples involved in the evaluation of an AND component. Potentially all triples of an RDF graph G are involved in the evaluation of a conjunctive component, i.e. by definition of evaluation $[\![.]\!]_G$, the whole graph is involved in the evaluation process.

  We therefore introduce in our translation system a special node called graph node (black anonymous node in Figure 6.6), that allows to access every triple of G through modalities α. Intuitively a graph node can be seen as the starting point for the evaluation of an AND component.

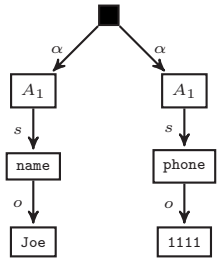


Figure 6.6: Encoding of triples in α relation

- The modality β is used to distinguish triples involved in the evaluation of an OPTIONAL component.

  Consider the simple case of an OPTIONAL component having two conjunctive components: one on the left and one on the right of the OPTIONAL operator. In the transition system, we distinguish the triples involved in the evaluation of the left member, from the triple involved in the evaluation of the right member, by introducing a modality β between two *graph nodes*, as shown in figure 6.7. The left *graph node* gives the access to the triples solutions of the mandatory part of the OPTIONAL, while the right `graph node` gives the access to the optional solution. Both *graph nodes* allow to access all triples of the graph through modalities α.

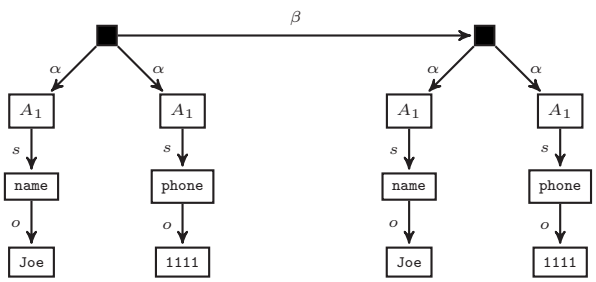


Figure 6.7: Encoding of triples in α-β relations

More generally, in well-designed SPARQL queries in *opt normal form*, the OPTIONAL operators define a tree structure of AND components. This tree representation, that we have presented as well-designed pattern tree, can be considered as a query evaluation plan [Letelier et al., 2012]. As a consequence, if we want to distinguish

triples involved in the evaluation of the several components of the query, we can take advantage of this tree representation. Intuitively, we can encode the OPTIONAL-tree structure depicted by a pattern tree, as a β-tree structure in the transition system as in Figure 6.8. We create a correspondence between the pattern tree and the transition system, such that each node of the pattern tree (conjunctive component) can be evaluated starting from a specific graph node of the transition system. Moreover, in correspondence of an edge between two nodes of the pattern tree, we have a β transition between two graph nodes in the transition system. Note, in Figure 6.8 the black anonymous nodes are all identical graph nodes that allow to access all triple as in Figure 6.6. Hence we have duplications.

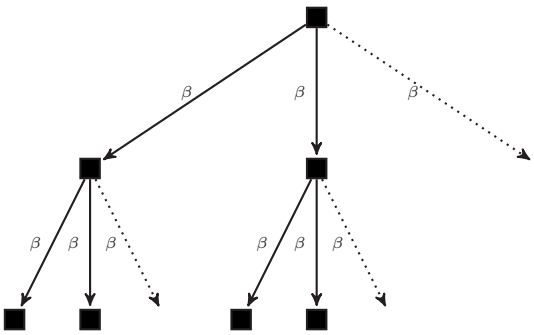


Figure 6.8: The β tree structure

We can easily notice that if we depict explicitly the β-structure in the transition system, it is not trivial encoding G. However, consider the transition system in figure 6.9. This transition system is easier to construct. Intuitively, this transition system allows one to access all triples starting from the *graph node*, as in Figure 6.6. Eventually, without depicting explicitly a β-tree as in Figure 6.7 and in Figure 6.8, but by iterating over β, it allows one to access triples involved in the OPTIONAL components. This model is more general than the previous ones, it encompasses all the previous cases while merging identical subparts of the transition system (it avoids duplication like in figure 6.7 and 6.8). It is easy to see that all the forward paths allowed in the previous transition systems are all allowed in this transition system.

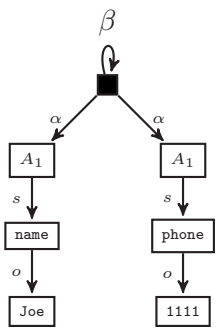


Figure 6.9: Transition system encoding the RDF graph G.

We formalize our encoding of an RDF graph as follows.

Definition 6.5.1 (Transition system associated to an RDF graph)**.** *Given a ground*[1] *RDF graph G, having only URIs, $G \subseteq U \times U \times U$, the transition system associated to G, denoted by $\sigma(G) = (W, R, V)$, over a set of atomic propositions $\mathbb{AP}_\exists = U$, is such that:*

- *$W = W' \cup W''$, with W' and W'' the smallest sets of points such that*
  - $\forall u \in U, \exists w_u \in W'$
  - $W'' = \{w_G\}$ *and* $w_G \notin W'$
- $\forall\ t = (s_t, p_t, o_t) \in G$
  - $\langle w_G, w_{s_t} \rangle \in R(\alpha)$
  - $\langle w_{s_t}, w_{p_t} \rangle \in R(s)$
  - $\langle w_{p_t}, w_{o_t} \rangle \in R(o)$
  - $\langle w_G, w_G \rangle \in R(\beta)$
- *$V : U \to \mathcal{P}(W')$; $\forall u \in U,\ V(u) = \{w_1, w_2, \ldots, w_n\}$ where $w_i \in W'$ is a point where u holds and $n = |V(u)|$ corresponds to the total number of edges incoming (indegree) and outgoing (outdegree) from u. Furthermore $\forall u, u' \in U$,*
  - $u \neq u' \Rightarrow V(u) \cap V(u') = \emptyset$

Moreover, $\mathbb{AP}_\exists$ is a set of atomic propositions such that $u \wedge u'$ is never **true** *for distinct atomic propositions u,$u' \in \mathbb{AP}_\exists$.*

The function σ associates what we call a **restricted transition system** to any RDF graph. Formally, we say that a transition system $\mathcal{M} = (W, R, V)$ is a restricted transition system iff there exists an RDF graph G such that $\mathcal{M}$=$\sigma(G)$.

A **restricted transition system** is thus a bipartite graph composed of two sets of nodes: W' those corresponding to RDF entities, and W'' those corresponding to a set of RDF triples. For example, figure 6.10 shows a restricted transition system associated to the graph of Example 2.3.1.

Example 6.5.1. (Graph as Transition System) *Let G be the graph of Figure 2.3.1 containing the following triples:*

$$\begin{array}{ll} \{(\mathtt{A}_1, \mathtt{name}, \mathtt{Joe}), & (\mathtt{A}_1, \mathtt{phone}, \mathtt{1111}), \\ \ (\mathtt{A}_2, \mathtt{name}, \mathtt{Martin}), & (\mathtt{A}_2, \mathtt{fax}, \mathtt{2222}), \\ \ (\mathtt{A}_3, \mathtt{name}, \mathtt{Louisa}), & (\mathtt{A}_3, \mathtt{phone}, \mathtt{3333}), \\ \ (\mathtt{A}_4, \mathtt{name}, \mathtt{John}), & (\mathtt{A}_4, \mathtt{web}, \mathtt{www.John.fr}), \\ \ (\mathtt{A}_4, \mathtt{email}, \mathtt{John@p.fr}), & (\mathtt{A}_4, \mathtt{phone}, \mathtt{4444}) \qquad \} \end{array}$$

[1] A ground RDF graph is an RDF graph that does not contain blank nodes.

The associated transition system is shown in figure 6.10.

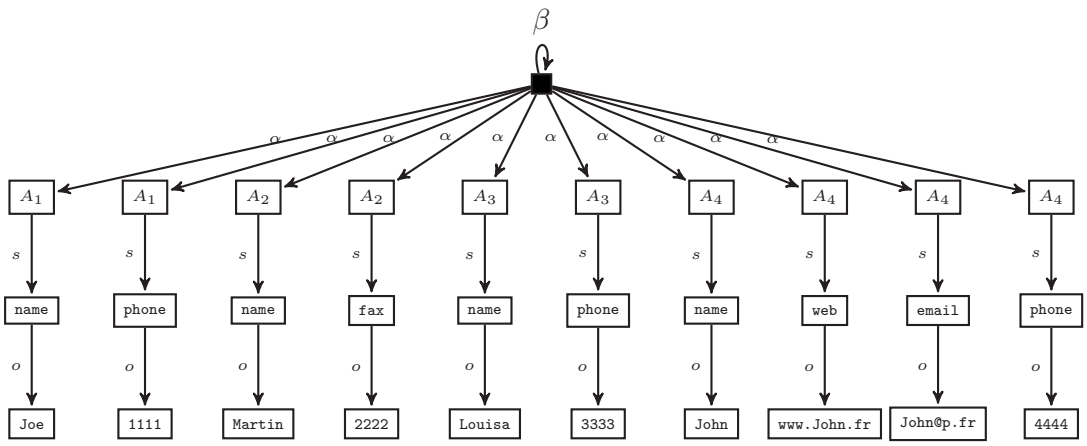


Figure 6.10: Transition system encoding the RDF graph G. The node in W'' is a black anonymous node; nodes in W' are the other nodes

Note, an `URI` *that appears in different RDF triples, it is* `true` *in multiple nodes of the associated transition system. For example, the* `URIs` A_4 *and* `name` *appear four times, hence in the transition system there are at least four nodes in which these* `URIs` *are true.*

Given that the logic chosen to determine containment is $\mathtt{K}_n$, (lacking functionality or number restrictions), one cannot impose that each subject node is connected to exactly one predicate node, and each predicate node to exactly one object node. However, we can impose a lighter restriction to achieve this by taking advantage of the technique introduced in [Genevès and Layaïda, 2006]. Since it is not possible to ensure that there is only one successor, then we restrict all the successors to bear the same constraints. They thus become interchangeable (bisimulation). To do this, we introduce a rewriting function f such that all occurrences of $\langle m\rangle\varphi$ (existential formulæ) are replaced by $\langle m\rangle\top \wedge [m]\varphi$.

As such, f is inductively defined on the structure of a $\mathtt{K}_n$ formula as follows:

$$
\begin{array}{rcl}
f(\top) & = & \top \\
f(\bot) & = & \bot \\
f(q) & = & q \quad q \in U \\
f(\neg\varphi) & = & \neg f(\varphi) \\
f(\varphi' \wedge \varphi'') & = & f(\varphi') \wedge f(\varphi'') \\
f(\varphi' \vee \varphi'') & = & f(\varphi') \vee f(\varphi'') \\
f(\langle m\rangle\varphi) & = & \langle m\rangle\top \wedge [m]f(\varphi) \quad m \in \{s, o\} \\
f(\langle m\rangle\varphi) & = & \langle m\rangle f(\varphi) \quad m \in \{\alpha, \beta\} \\
f([m]\varphi) & = & [m]f(\varphi) \quad m \in \mathbb{MOD}
\end{array}
$$

Thus, when checking for query containment, we assume that the formulæ are rewritten using function f. Along with that, we consider the following restrictions over the transition systems:

- the set of programs of the transition system is fixed, such that $\mathbb{MOD} = \{\beta, \alpha, s, o\}$;
- a model, satisfying the containment formulæ, must be a **restricted transition system**.

The last constraint can be expressed with a formula ξ that must be satisfied by each *graph node*.

$$\xi = [\alpha]\Big([s]\big([o]\texttt{isAObject} \wedge \texttt{isAPredicate}\big) \wedge \texttt{isASubject}\Big) \wedge \texttt{isAGraphNode}$$

where

$\texttt{isAGraphNode} = [s]\bot \wedge [o]\bot$

$\texttt{isASubject} = [\beta]\bot \wedge [\alpha]\bot \wedge [o]\bot$

$\texttt{isAPredicate} = [\beta]\bot \wedge [\alpha] \wedge \bot[s]\bot$

$\texttt{isAnObject} = [\beta]\bot \wedge [\alpha]\bot \wedge [s]\bot \wedge [o]\bot$

These predicates restricted the outgoing edges of a *graph node* and its descendants. In Figure 6.11 is shown that a *graph node* can have outgoing edges to subjects labeled α and outgoing edges to another *graph node* labeled β. The subjects can have outgoing edges to predicate and predicate to object. The objects have not outgoing edges.

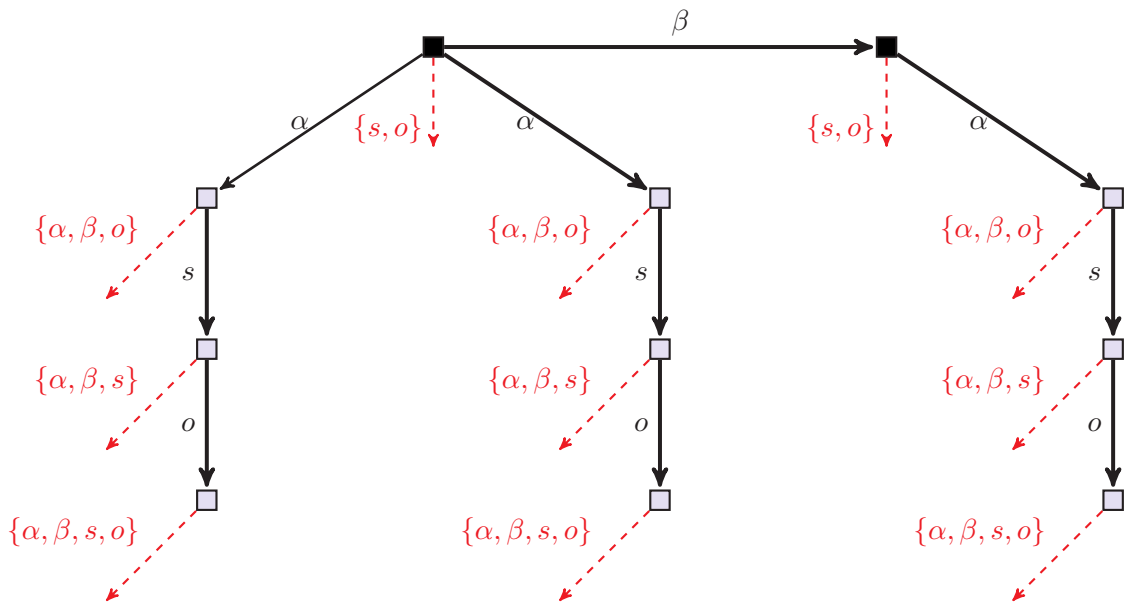


Figure 6.11: Restricted transition system

If our logic was equipped by fixpoint operators, we could have expressed this restriction using a single recursive formula à la [Chekol, 2012].

Let φ be a formula that can be stated over a restricted transition system, φ is satisfied by some restricted transition system if and only if $f(\varphi) \wedge \varphi_r$ is satisfied by some transition system over $\mathbb{MOD}$, i.e.

$$\exists \mathcal{M}_r.[\![\varphi]\!]_{\mathcal{M}_r} \Leftrightarrow \exists \mathcal{M}.[\![\varphi \wedge \varphi_r]\!]_{\mathcal{M}}$$

The formula φ_r ensures that ξ holds in every *graph node* reachable by β program. The formula ξ forces that each *graph node* must have edges to subjects and eventually to other *graph nodes*. Subjects must have edges only to predicates and predicates only to objects. The formula $f(\varphi)$ ensures that each subject node is connected to exactly one predicate, and a predicate is connected to exactly one object. Hence, the formula $\varphi \wedge \varphi_r$ avoids that a formula φ in satisfiable in a model $\mathcal{M}$ that is not a restricted transition system $\sigma(G)$,

The logic adopted in this work $\mathtt{K}_n$ does not allow fixpoint operators for recursion. However, instead of recursion, we can insert the constraint ξ each time we need it during the logical translation of well-designed patterns. Specifically, a conjunction $\xi \wedge \varphi_{P_i}$ is added for each occurrence of a formula φ_{P_i} that encodes a node of a well-designed pattern tree.

This formula duplication has no impact on computational complexity since it has no impact on the size of the lean representation introduced in Chapter 5 [Genevès and Schmitt, 2015].

In the following, we introduce the encoding of a well-designed graph patterns based on their tree representation.

6.5.2 Encoding Queries as K-Formulæ

Triples in queries can be seen as paths in the transition system, hence triple patterns of queries can be expressed by a formula which describes such paths. For instance, a query which contains the triple pattern (a, b, c) can be expressed as $a \wedge \langle s \rangle (b \wedge \langle o \rangle c)$ which states that a is satisfied if there exists a path in a transition system where starting from this node labeled with a, nodes with labels b and c can be reached by programs $\langle s \rangle$ and $\langle s \rangle \langle o \rangle$, respectively.

When encoding $q_1 \subseteq q_2$, we call q_1 left-hand side query and q_2 right-hand side query.

In this translation, variables are replaced by standard atomic propositions ($\mathbb{AP}$) or some formula which will be satisfied when they are matched in such triple relations.

We provide two different encodings for the left-hand side query and the right-hand side query. The left-hand side query variables are encoded as atomic propositions, while the right-hand side query variables are encoded as a formula (whose meaning is the following: a variable can match with itself or with a specific `URI`).

Encoding left-hand side query We use a function $\lambda : V \to \mathbb{AP}$. This function can be extended over `URI`s such that $\lambda : U \to \mathbb{AP}_\exists$, where $\mathbb{AP}_\exists = U$, is the identity function.

$$\lambda(x) = \begin{cases} u_x & \text{if } x \in V \\ x & \text{if } x \in U \end{cases}$$

We recall that $\mathbb{AP}_\exists$ is a set of atomic proposition such that $u \wedge u'$ is never `true` for distinct atomic proposition $u, u' \in \mathbb{AP}_\exists$.

Now, we are able to encode a well-designed graph pattern, and more precise its tree representation, as an unordered tree in the logic as follows.

Definition 6.5.2 (Left-hand `wdPT` encoding)**.** *The encoding of a `wdPT` is denoted by $\mathcal{A}(\mathcal{T})$, where $\mathcal{T} = ((V, E, r), (P_n)_{n \in V})$ is a tree rooted at r and P_n is a non-empty set of triple patterns, for every node $n \in V$. The encoding is defined recursively as follows.*

- $\mathcal{A}(\mathcal{T}_i) = \xi \ \wedge \ \mathcal{A}(P_i) \ \wedge \ [\beta]\chi_i$
- $\mathcal{A}(P_i) = \bigwedge_{t_j \in P_i} \langle\alpha\rangle\mathcal{A}(t_j)$
- $\mathcal{A}(t_j) = \mathcal{A}(s_j, p_j, o_j) = \lambda(s_j) \ \wedge \ \langle s\rangle\big(\lambda(p_j) \ \wedge \ \langle o\rangle\lambda(o_j)\big)$

where:

$$\chi_i = \begin{cases} \bigvee_{(i,j) \in E} \mathcal{A}(\mathcal{T}_j) & \textit{if } \exists n \in V, (i, n) \in E \\ \bot & \textit{otherwise} \end{cases}$$

and ξ is defined as below:

$$[\alpha]\Big([s]\big([o]([\beta]\bot \wedge [\alpha]\bot \wedge [s]\bot \wedge [o]\bot) \wedge [\beta]\bot \wedge [\alpha]\bot[s]\bot\big) \wedge [\beta]\bot \wedge [\alpha]\bot[o]\bot\Big) \wedge [s]\bot \wedge [o]\bot$$

Notice, when checking for query containment, we assume that the formulæ are rewritten using function f. Specifically, f makes sure that each subject is connected to exactly one predicate and a predicate is connected to exactly one object. Hence, recursively applied over $\mathcal{A}(\mathcal{T})$, f modifies the encoding of the triple patterns as follows. Let $t_j = (s_j, p_j, o_j)$ be a triple pattern, we have

$$f(\mathcal{A}(t_j)) = \lambda(s_j) \ \wedge \ \langle s\rangle\top \ \wedge \ [s]\big(\lambda(p_j) \ \wedge \ \langle o\rangle\top \ \wedge \ [o]\lambda(o_j)\big)$$

Example 6.5.2. *Consider the following well-designed pattern tree* $\mathcal{T} = (P_r, \mathcal{T}_1, \mathcal{T}_2)$*:*

$$\mathcal{T} : \underbrace{\{t_1 \text{ AND } t_2 \text{ AND } t_3\}}_{P_r=\{t_1,t_2,t_3\}}$$

$$\underbrace{\{t_4 \text{ AND } t_5\}}_{P_1=\{t_4,t_5\}} \qquad \underbrace{\{t_6\}}_{P_2=\{t_6\}}$$

The recursive encoding of the `wdPT` $\mathcal{A}(\mathcal{T})$ *is:*

$$\mathcal{A}(\mathcal{T}) = \xi \wedge \mathcal{A}(P_r) \wedge [\beta]\Big(\mathcal{A}(\mathcal{T}_1) \vee \mathcal{A}(\mathcal{T}_2)\Big)$$

The encoding of the root:

$$\mathcal{A}(P_r) = \langle\alpha\rangle\mathcal{A}(t_1) \wedge \langle\alpha\rangle\mathcal{A}(t_2) \wedge \langle\alpha\rangle\mathcal{A}(t_3)$$

The encoding of the first child:

$\mathcal{A}(\mathcal{T}_1) = \xi \wedge \mathcal{A}(P_1) \wedge [\beta]\bot$

$\mathcal{A}(P_1) = \langle\alpha\rangle\mathcal{A}(t_4) \wedge \langle\alpha\rangle\mathcal{A}(t_5)$

The encoding of the second child:

$\mathcal{A}(\mathcal{T}_2) = \xi \wedge \mathcal{A}(P_2) \wedge [\beta]\bot$

$\mathcal{A}(P_2) = \langle\alpha\rangle\mathcal{A}(t_6)$

The resulting formula:

$$\mathcal{A}(\mathcal{T}) = \xi \wedge \langle\alpha\rangle\mathcal{A}(t_1) \wedge \langle\alpha\rangle\mathcal{A}(t_2) \wedge \langle\alpha\rangle\mathcal{A}(t_3) \wedge [\beta]\Big(\Big(\xi \wedge \langle\alpha\rangle\mathcal{A}(t_4) \wedge \langle\alpha\rangle\mathcal{A}(t_5) \wedge [\beta]\bot\Big) \vee \Big(\xi \wedge \langle\alpha\rangle\mathcal{A}(t_6) \wedge [\beta]\bot\Big)\Big)$$

Encoding right-hand side query We define a set of mappings $m = \{m_1, \ldots, m_n\}$ such that each m_i contains formula assignments to the variables of the right-hand side query. We consider the following sets:

- $sub(q)$ all subject of the triple patterns in q
- $pre(q)$ all predicate of the triple patterns in q
- $obj(q)$ all object of the triple patterns in q
- $const(q)$ all constants of the triple patterns in q
- $var(q)$ all variables of the triple patterns in q

Let q_1 be the left-hand side query and q_2 be the right-hand side query, the set of mappings is defined as

$$m = \bigtimes_{i=0}^{k} \{x_i \mapsto \begin{cases} t \in term(q_1, q_2, x_i) \cap const(q_1) & \text{if } term(q_1, q_2, x_i) \cap const(q_1) \neq \emptyset \\ & \text{and } \mid x_i \mid > 1 \text{ in } q_2 \\ \top & \text{otherwise} \end{cases} \}$$

where x_i is a variable of q_2, K is the number of variables in q_2, $\mid x_i \mid$ denotes the number of occurrences of x_i in q_2 and t is an `URI` that appears in the left-hand side query q_1. The set of `URIs` (constants) that can match with a variables is identified by the function $term(q_1, q_2, x_i)$ as follows.

$$term(q_1, q_2, x_1) = \begin{cases} sub(q_1) & \text{if } x_i \in sub(q_2) \\ pre(q_1) & \text{if } x_i \in pre(q_2) \\ obj(q_1) & \text{if } x_i \in obj(q_2) \\ sub(q_1) \cup pre(q_1) & \text{if } x_i \in sub(q_2),\ x_i \in pre(q_2) \\ sub(q_1) \cup obj(q_1) & \text{if } x_i \in sub(q_2),\ x_i \in obj(q_2) \\ sub(q_1) \cup pre(q_1) \cup obj(q_1) & \text{if } x_i \in sub(q_2),\ x_i \in pre(q_2),\ x_i \in obj(q_2) \\ pre(q_1) \cup obj(q_1) & \text{if } x_i \in pre(q_2),\ x_i \in obj(q_2) \end{cases}$$

The maximum size of the encoding of the right-hand side query is $\mid const(q_1) \mid^{|var(q_2)|}$, where $\mid const(q_1) \mid$ is the number of constants of q_1 and $\mid var(q_2) \mid$ is the number of variables of q_2. Let a mapping m_i, we extend the function λ previously defined for the left-hand side query variables as follows:

$$\lambda(x, m_i) = \begin{cases} u_x \vee m_i(x) & \text{if } x \in V \\ x & \text{if } x \in U \end{cases}$$

Definition 6.5.3 (Right-hand `wdPT` encoding)**.** *The encoding of a righ-hand side `wdPT` is denoted by $\mathcal{A}(\mathcal{T}, m)$, where $\mathcal{T} = ((V, E, r), (P_n)_{n \in V})$ is a tree rooted at r, P_n is a non-empty set of triple patterns, for every node $n \in V$, and m is a set of mapping.*

$$\begin{aligned} \mathcal{A}(\mathcal{T}, m) &= \bigvee_{k=1}^{|m|} \mathcal{A}(\mathcal{T}, m_k) \\ \mathcal{A}(\mathcal{T}_i, m_k) &= \xi \ \wedge \ \mathcal{A}(P_i, m_k) \ \wedge \ [\beta](\chi_i, m_k) \\ \mathcal{A}(P_i, m_k) &= \bigwedge_{t_j \in P_i} \langle \alpha \rangle \mathcal{A}(t_j, m_k) \\ \mathcal{A}(t_j, m_k) = \mathcal{A}((\mathtt{s_j}, \mathtt{p_j}, \mathtt{o_j}), m_k) &= \lambda(s_j, m_k) \ \wedge \ \langle s \rangle (\lambda(p_j, m_k) \ \wedge \ \langle o \rangle \lambda(o_j, m_k)) \end{aligned}$$

where:

$$(\chi_i, m_k) = \begin{cases} \bigvee_{(i,j) \in E} \mathcal{A}(\mathcal{T}_j, m_k) & \textit{if } \exists n \in V \textit{ and } (i, n) \in E \\ \bot & \textit{otherwise} \end{cases}$$

6.5.3 Containment Problem

Once correct encodings of queries have been produced, the next step requires reducing containment to the validity problem in the K-logic. Intuitively, $q_1 \sqsubseteq q_2$ is reduced to the unsatisfiability test of $\mathcal{A}(q_1) \wedge \neg\mathcal{A}(q_2, m)$.

Example 6.5.3 (Containment example)**.** *In the following, we want to check if $\mathcal{T}_A \sqsubseteq \mathcal{T}_B$. Let $\mathcal{T}_A$ and $\mathcal{T}_B$ be the following pattern trees:*

$$\mathcal{T}_A = (\mathtt{?a}, \mathtt{name}, \mathtt{Joe})\text{AND}(\mathtt{?a}, \mathtt{phone}, \mathtt{1111})$$

$$\mathcal{T}_B = ((\mathtt{?a}, \mathtt{name}, ?\mathtt{n})\text{AND}(\mathtt{?a}, \mathtt{phone}, ?\mathtt{n}))\text{OPTIONAL}(\mathtt{?a}, \mathtt{web}, \mathtt{?w})$$

The encoding of the left-hand side query is:

$$-\mathcal{A}(\mathcal{T}_A) = \underbrace{\xi \wedge \mathcal{A}(\{(\mathtt{?a}, \mathtt{name}, \mathtt{Joe}), (\mathtt{?a}, \mathtt{phone}, \mathtt{1111})\})}_{\varphi_A} \wedge [\beta]\bot$$

The encoding of the right-hand side query is:

$$-\mathcal{A}(\mathcal{T}_B, m) = \xi \wedge \mathcal{A}(\{(\mathtt{?a}, \mathtt{name}, \mathtt{?n}), (\mathtt{?a}, \mathtt{phone}, \mathtt{?n})\}, m) \wedge [\beta]\mathcal{A}(\{(\mathtt{?a}, \mathtt{web}, \mathtt{?w})\}, m)$$

where:

$$m = \{\underbrace{\{\mathtt{?a} \to \mathtt{true},\ \mathtt{?n} \to Joe,\ \mathtt{?w} \to \mathtt{true}\}}_{m_1}, \underbrace{\{\mathtt{?a} \to \mathtt{true},\ \mathtt{?n} \to 1111,\ \mathtt{?w} \to \mathtt{true}\}}_{m_2}\}$$

We have two mappings that associate to the variable $\mathtt{?n}$ *the value* Joe *and* 1111*, respectively. Abusing notation, we have the following two encodings:*

$$-\mathcal{A}(\mathcal{T}_B, m_1) = \underbrace{\xi \wedge \mathcal{A}(\{(\mathtt{?a}, \mathtt{name}, \mathtt{Joe}), (\mathtt{?a}, \mathtt{phone}, \mathtt{Joe})\})}_{\varphi_{B_1}} \wedge [\beta]\underbrace{\mathcal{A}(\{(\mathtt{?a}, \mathtt{web}, \mathtt{?w})\})}_{\psi}$$

$$-\mathcal{A}(\mathcal{T}_B, m_2) = \underbrace{\xi \wedge \mathcal{A}(\{(\mathtt{?a}, \mathtt{name}, \mathtt{1111}), (\mathtt{?a}, \mathtt{phone}, \mathtt{1111})\})}_{\varphi_{B_2}} \wedge [\beta]\underbrace{\mathcal{A}(\{(\mathtt{?a}, \mathtt{web}, \mathtt{?w})\})}_{\psi}$$

The containment problem can be reduced to the validity problem of the following formula in the logic:

$$\psi : \mathcal{A}(\mathcal{T}_A) \Rightarrow \mathcal{A}(\mathcal{T}_B, m)$$

$$\shortparallel$$

$$\psi : \neg\mathcal{A}(\mathcal{T}_A) \vee (\neg\mathcal{A}(\mathcal{T}_B, m))$$

or reduced to the unsatisfiability test of $\neg\psi$*:*

$$\texttt{unsat}\Big(\mathcal{A}(\mathcal{T}_A) \wedge \neg\mathcal{A}(\mathcal{T}_B, m)\Big) = \texttt{unsat}\Big(\mathcal{A}(\mathcal{T}_A) \wedge \neg(\mathcal{A}(\mathcal{T}_B, m_1) \vee \mathcal{A}(\mathcal{T}_B, m_2))\Big)$$

that is equal to:

$$\texttt{unsat}\Big(\mathcal{A}(\mathcal{T}_A) \wedge \neg\mathcal{A}(\mathcal{T}_B, m_1) \wedge \neg\mathcal{A}(\mathcal{T}_B, m_2))\Big)$$

$$\shortparallel$$

$$\texttt{unsat}\Bigg(\Big(\varphi_A \wedge [\beta]\bot\Big) \wedge \neg\Big(\varphi_{B_1} \wedge [\beta]\psi\Big) \wedge \neg\Big(\varphi_{B_2} \wedge [\beta]\psi\Big)\Bigg)$$

We can easily check that it this formula is satisfied, that implies $\mathcal{T}_A \not\sqsubseteq \mathcal{T}_B$

Note that we can split the check in two different independent tests:

1. $\texttt{unsat}\Bigg(\Big(\varphi_A \wedge [\beta]\bot\Big) \wedge \neg\Big(\varphi_{B_1} \wedge [\beta]\psi\Big)$

2. $\texttt{unsat}\Bigg(\Big(\varphi_A \wedge [\beta]\bot\Big) \wedge \neg\Big(\varphi_{B_2} \wedge [\beta]\psi\Big)\Bigg)$

If one of this test concludes the unsatisfiability of the formula, hence the query containment holds $\mathcal{T}_A \sqsubseteq \mathcal{T}_B$*.*

6.6 Reducing Query Containment to Unsatisfiability

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

Lemma 6.6.1 (Canonical graph[Hayes, 2004])**.** *Given a query q, there exists an* `RDF` *graph G, such that* $[\![q]\!]_G \neq \emptyset$*.*

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

□

Lemma 6.6.2. *For any graph G and* `wdPT` $\mathcal{T}$*:*

$$[\![\mathcal{T}]\!]_G \neq \emptyset \quad \Leftrightarrow \quad \sigma(G) \models \mathcal{A}(\mathcal{T})$$

Proof.

($\Rightarrow$) Assume $[\![\mathcal{T}]\!]_G \neq \emptyset$, by lemma 6.3.1, there exists a subtree $\mathcal{T}_\rho$ of $\mathcal{T}$ such that:

1. $\mathtt{dom}(\rho) = \mathtt{vars}(\mathcal{T}_\rho)$
2. $\mathcal{T}_\rho$ is the maximal subtree of $\mathcal{T}$, s.t. $\rho \sqsubseteq [\![\mathtt{and}(\mathcal{T}_\rho)]\!]_G$.

Consider that $\mathcal{T}_\rho$ is the smallest subtree of $\mathcal{T}$ consisting only of the root node and consider G is a canonical instance of $\mathtt{and}(\mathcal{T}_\rho)$. (cf. Lemma 6.6.1). Using G, we construct a restricted transition system $\sigma(G) = (W, R, V)$ in the same way as it is done in Definition 6.5.1. To prove that $\sigma(G)$ is a model of $\mathcal{A}(\mathcal{T})$, we need only to encode the variables of $\mathcal{T}$ with $\top$.

($\Leftarrow$) Assume $\sigma(G) \models \mathcal{A}(\mathcal{T})$, by Definition 6.5.2, $\sigma(G) \models \xi \wedge \mathcal{A}(P_r) \wedge [\beta]\chi_r$.

$\forall w_s, w_p, w_o \in W'$ and $W'' = \{w_G\}$ and $\langle w_G, w_s\rangle \in R(\alpha)$ and $\langle w_s, w_p\rangle \in R(s)$ and $\langle w_p, w_o\rangle \in R(o)$ and for each triple $t_i = (\mathtt{s_i}, \mathtt{p_i}, \mathtt{o_i}) \in P_r$, if $\mathtt{s_i} \in V(w_s)$ and $\mathtt{p_i} \in V(w_p)$ and $\mathtt{o_i} \in V(w_o)$ then $(\mathtt{s_i}, \mathtt{p_i}, \mathtt{o_i}) \in G$. If $\mathtt{s_i}$ or $\mathtt{p_i}$ or $\mathtt{o_i}$ is a variable, in G we replace it, with a fresh blank node. Note here that if $\mathtt{s_i}$ or $\mathtt{p_i}$ or $\mathtt{o_i}$ appear in another triple $t_j = (\mathtt{s_j}, \mathtt{p_j}, \mathtt{o_j}) \in P_r$, then the equivalent item in t_j is replaced with the value of the corresponding entry in t_i.

Since G is a technical construction obtained from a restricted transition system $\sigma(G)$ and G is a canonical instance of $\mathtt{and}(P_r)$, then $[\![\mathtt{and}(P_r)]\!]_G \neq \emptyset$ implies by Definition 6.3.1, $[\![\mathcal{T}]\!]_G \neq \emptyset$ □

This lemma holds even in presence of a mapping m_i. Let $\mathcal{A}(\mathcal{T})$ and $\mathcal{A}(\mathcal{T}, m_i)$ the two differs in the encoding of variables. Let a variable $x \in \mathcal{T}$, in absence of mapping we have $\lambda(x) = u_x$, where u_x is an atomic proposition; while in presence of a mapping m_i, we have $\lambda(x, m_i) = u_x \vee m(x_i)$. By the semantics of $\vee$ both formula u_x and $u_x \vee m(x_i)$ can be satisfied by the same models.

Lemma 6.6.3. *For any model $\mathcal{M}$ and* `wdPT` $\mathcal{T}$*:*

$$\mathcal{M} \models \mathcal{A}(\mathcal{T}) \Leftrightarrow \mathcal{M} = \sigma(G) \text{ and } [\![\mathcal{T}]\!]_G \neq \emptyset$$

Proof (Sketch).

($\Leftarrow$) this case follows immediately from Lemma 6.6.2.

($\Rightarrow$) Assume $\mathcal{M} \models \mathcal{A}(\mathcal{T})$, by Definition 6.5.2, we have $\mathcal{M}, w \models \xi \wedge \mathcal{A}(P_r) \wedge [\beta]\chi_r$. This formula is satisfiable in two cases.

CASE A w has no outgoing edge β. In this case, we have to prove that

$$\mathcal{M}, w \models \xi \wedge \mathcal{A}(P_r) \Rightarrow \mathcal{M} = \sigma(G) \text{ and } [\![\mathcal{T}]\!]_{\sigma(G)} \neq \emptyset$$

We assume $\mathcal{M}, w \models \xi \wedge \mathcal{A}(P_r) \Leftrightarrow \mathcal{M}, w \models \xi \wedge \Big(\bigwedge_{i \in P_r} \langle \alpha \rangle \mathcal{A}(t_i) \Big)$.

$\mathcal{M}, w \models \xi$ implies that $\mathcal{M}$ is a transition system that allows only paths $\langle \alpha \rangle \langle s \rangle \langle o \rangle$ from w. It is easy to see w as a *graph node* of a restricted transition system and each path $\langle \alpha \rangle \langle s \rangle \langle o \rangle$ as the encoding of a triple of a graph RDF G. ξ when satisfied by a modal ensures that this model is an encoding of a graph. Hence, this condition implies that there exists a graph G, such that $\mathcal{M} = \sigma(G)$.

$\mathcal{M}, w \models \bigwedge_{i \in P_r} \langle \alpha \rangle \mathcal{A}(t_i)$ means that for each triple patterns $t_j = (s_j, p_j, o_j) \in P_r$, its encoding formula $\mathcal{A}(t_j) = \langle \alpha \rangle (\lambda(s_j) \wedge \langle s \rangle (\lambda(p_j) \wedge \langle o \rangle \lambda(o_j)))$ is satisfied in $\mathcal{M}$. Then, starting from w there exists at least a path $\langle \alpha \rangle \langle s \rangle \langle o \rangle$ that reaches nodes labeled $\lambda(s_j), \lambda(p_j), \lambda(o_j)$. Due to the satisfiability of ξ in $\mathcal{M}$, $(\lambda(s_i), \lambda(p_j), \lambda(p_j))$ can be seen as a triple in the graph G, s.t. $\mathcal{M} = \sigma(G)$. Hence, G contains triples such that $[\![and(P_r)]\!]_{\neq} \emptyset$ that implies $[\![\mathcal{T}]\!]_{\neq} \emptyset$.

CASE B w has outgoing edges β. In this case, we have to prove that

$$\mathcal{M}, w \models \xi \wedge \mathcal{A}(P_r) \wedge \langle \beta \rangle \chi_r \Rightarrow \mathcal{M} = \sigma(G) \text{ and } [\![\mathcal{T}]\!]_{\sigma(G)} \neq \emptyset$$

.

Let $R(\beta) = \langle w, w' \rangle$, we assume $\mathcal{M}, w \models \xi \wedge \mathcal{A}(P_r)$ and $\mathcal{M}, w' \models \chi_r \Leftrightarrow \mathcal{M}, w' \models \bigvee_{(r,i) \in E} \mathcal{A}(\mathcal{T}_i)$.

Analogously, to the previous case, $\mathcal{M}, w \models \xi \wedge \mathcal{A}(P_r)$ implies that w can be seen as a *graph node* of a restricted transition system $\sigma(G)$ and G is such that $[\![P_r]\!]_G \neq \emptyset$, as P_r identifies the mandatory component of a well-designed graph pattern, we have $[\![\mathcal{T}]\!]_{\sigma(G)} \neq \emptyset$.

As a consequence, to conclude our proof we have to demonstrate that w' is a *graph node.*

We assume $\mathcal{M}, w' \models \bigvee_{(r,i) \in E} \mathcal{A}(\mathcal{T}_i)$ that, by definition, is equal to

$$\mathcal{M}, w' \models \bigvee_{(r,i) \in E} \xi \wedge \mathcal{A}(P_i) \wedge \langle \beta \rangle \chi_i \Leftrightarrow \mathcal{M}, w' \models \xi \wedge \bigvee_{(r,i) \in E} \mathcal{A}(P_i) \wedge \langle \beta \rangle \chi_i$$

$\mathcal{M}, w' \models \xi$ implies that w' can be seen a *graph node.*

□

Theorem 6.6.4. *Given two* `wdPTs` $\mathcal{T}_1$ *and* $\mathcal{T}_2$

$$\mathcal{T}_1 \sqsubseteq \mathcal{T}_2 \Leftrightarrow \mathcal{A}(\mathcal{T}_1) \wedge \neg\mathcal{A}(\mathcal{T}_2, m) \text{ unsatisfiable}$$

Proof. $\mathcal{T}_1 \sqsubseteq \mathcal{T}_2$

$\Leftrightarrow \forall G. [\![\mathcal{T}_1]\!]_G \sqsubseteq [\![\mathcal{T}_2]\!]_G$

by Theorem 6.6.2, $[\![\mathcal{T}_1]\!]_G \Leftrightarrow \sigma(G) \models \mathcal{A}(\mathcal{T}_1)$ *and* $[\![\mathcal{T}_2]\!]_G \Leftrightarrow \sigma(G) \models \mathcal{A}(\mathcal{T}_2, m)$

$\Leftrightarrow \forall G. \Big(\sigma(G) \models \mathcal{A}(\mathcal{T}_1) \;\Rightarrow\; \sigma(G) \models \mathcal{A}(\mathcal{T}_2, m)\Big)$

$\Leftrightarrow \forall G.\ \sigma(G) \models \mathcal{A}(\mathcal{T}_1) \;\Rightarrow\; \mathcal{A}(\mathcal{T}_2, m)$

$\Leftrightarrow \forall G.\ \sigma(G) \models \neg\mathcal{A}(\mathcal{T}_1) \;\vee\; \mathcal{A}(\mathcal{T}_2, m)$

$\Leftrightarrow \forall G.\ \sigma(G) \not\models \mathcal{A}(\mathcal{T}_1) \;\wedge\; \neg\mathcal{A}(\mathcal{T}_2, m)$

$\Leftrightarrow \forall \mathcal{M}.\ \mathcal{M} \not\models \mathcal{A}(\mathcal{T}_1) \;\wedge\; \neg\mathcal{A}(\mathcal{T}_2, m)$

$\mathcal{A}(\mathcal{T}_1) \wedge \neg\mathcal{A}(\mathcal{T}_2, m)$ is unsatisfiable. □

6.7 Experimentation

In order to experiment with the proposed approach, the satisfiability solvers from Chapter 5, the theorem prover from [Hustadt and Schmidt, 2000] and an *ad hoc* query containment solver from [Letelier et al., 2012] are used to test containment among different queries. These three different nature implementation are all able to solve the query containment problem for SPARQL. In the following, we call our implementation KSolver, the one from [Hustadt and Schmidt, 2000] `MSPASS` and the one from [Letelier et al., 2012] `SPARQL`-Algebra. A set of queries, shown in Appendix B, are tested for their containment.

6.7.1 Benchmark

In this section, we present the query containment setup, the structure of the benchmark and the benchmark description. We first present the design of the containment benchmark suite. The test suite is made of elementary test cases asking for the containment of one query into another. We then introduce the principle and software used for evaluating the containment solver. The benchmark is available on-line at

`http://tyrex.inria.fr/benchmark/index.html`

6.7.1.1 Query Containment Setup

The test case for containment comprises two queries q and q'. The query containement Solver produces *yes*/*no* answers, i.e., yes if q is contained in q' and no otherwise. A general workflow diagram designed for this purpose is illustrated in figure 6.12

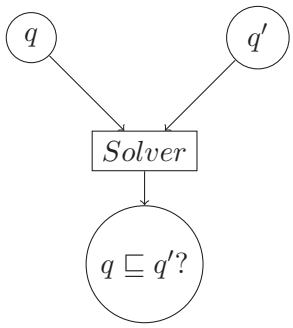


Figure 6.12: General workflow of query containment tests

6.7.1.2 Structure of the Benchmark

A benchmark should fulfill the following requirements:

- understandability , the queries should be understandable
- scalability
- portability, it shoud be able to run on different platforms
- relevance, testing typical operations such as joins and left-joins.

Thus we have designed the benchmark following these principles.

For our purpose we have adapted queries taken either from third-party benchmarks such as the Berlin Benchmark or that we have automatically generated. Queries in the Berlin Benchmark are based on the specific requirements of a real world use case. The queries in Appendix B emulate the search and navigation patterns of a consumer looking for a product. In a real world setting, such a query sequence could for instance be executed by a shopping portal which is used by consumers to find products and sales offers [Bizer and Schultz, 2009].

For our purpose, we have identified a qualitative dimension along which tests can be designed: the type of graph pattern connectors (AND, OPTIONAL). In addition to this dimension, quantitative measures are:

- the number of triple patterns;
- the number of OPTIONAL constructors;

- the number of existential (diamond) and universal (box) modalities in the obtained logical formula;
- and the number of atomic propositions of the formula.

These tests are part of a test suite designed to model increasing expected difficulties by using more constructors. Most of the tests are used for conformance testing, i.e. testing that the solver returns the correct answer, but we also report on some stress tests trying to evaluate the solvers at its limits.

Test	Problem	TRIPLES	# OPT	# DIA	# BOX	# AP
1	Q1a⊑Q1b	3	0	10	7	7
2	Q1b⊑Q1a	3	0	7	10	7
3	Q2a⊑Q2b	10	0	19	19	13
4	Q2b⊑Q2a	10	0	19	19	13
5	Q2c⊑Q2a	9	0	16	19	13
6	Q2c⊑Q2b	9	0	16	19	13
7	Q3a⊑Q3b	9	1	20	17	15
8	Q3b⊑Q3a	9	1	17	20	15
9	Q3a⊑Q3c	9	1	20	17	15
10	Q3c⊑Q3a	9	1	17	20	15
11	Q4a⊑Q4b	10	3	20	20	23
12	Q4b⊑Q43	10	3	20	20	23
13	Q5a⊑Q5b	11	4	23	20	25
14	Q5b⊑Q5a	11	3	20	23	25
15	Q5a⊑Q5c	11	4	24	21	25
16	Q5c⊑Q5a	11	4	21	24	25

Table 6.2: The test suite

6.7.1.3 Benchmarking Software Architecture

For testing containment solvers we designed an experimental setup which comprises several software components. This setup is illustrated in Figure 6.13. It simply consider a containment checker as a software module taking as input two `SPARQL` queries (q and q') and returns `true` or `false` depending if q is contained in q'.

The two queries are firstly parsed using the Jena framework and the ARQ query engine [2]. Then, queries are translated into K-formulæ for working on the KSolver and the `MSPASS` implementations, while this step is not required for `SPARQL`-Algebra that works directly on ARQ representation. Note that, the KSolver and *MSPASS* having two different syntax,

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

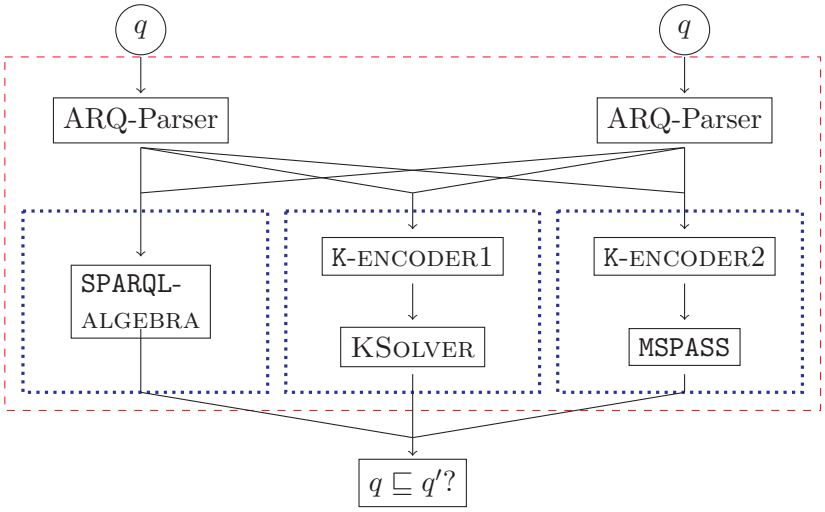


Figure 6.13: Experimental setup for our query containment test. The tester (dashed red rectangle) parses queries and passes them to a solver wrapper (dotted blue rectangle)

hence two different encoding procedures are provided. The K-formulæ are then parsed and transformed in each solver's internal representation.

6.8 Query Containment Solvers

In following we briefly present the MSPASS and SPARQL-ALGEBRA state-of-the art query containment solvers, while a detailed description of the KSOLVER is reported in Chapter 5

6.8.1 MSPASS

The MSPASS solver [Hustadt and Schmidt, 2000] decides $\mathtt{K}_n$ using a completely different approach from the table method of Chapter 5. MSPASS is an enhancement of the first-order theorem prover SPASS [Weidenbach et al., 2009]. MSPASS is a resolution-based solver which translates modal formualae into first-order logic formulæ. Specifically, reasoning in MSPASS is performed in three stages [Hustadt and Schmidt, 2000]:

1. translation of a given set of modal formulæ into a set of first-order formulæ;
2. transformation into clausal form,
3. saturation based resolution.

Useful links related to MSPASS are available here:

http://www.spass-prover.org/links/

6.8.2 SPARQL-Algebra

SPARQL-Algebra is an implementation of SPARQL query subsumption and equivalence based on the theoretical result of [Letelier et al., 2012]. This implementation supports AND and OPTIONAL queries with no projection.

6.8.3 Experimentation

We evaluated the three identified query containment solvers with the test suits. We run experiments on Ubuntu Linux machine with an Intel Core i7 2.7GHz, 16 GB of RAM.

Reported figure are the average of 5 runs (we run the test 7 times and ruled out each time the best and worse performance). For each test, we measured the total time spent in solving the query containment problem.

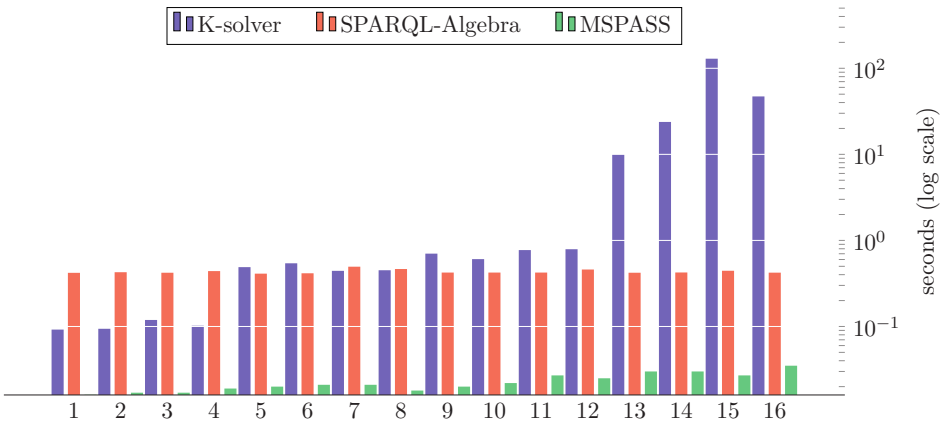


Figure 6.14: Results for test suite (logarithmic scale).

We report the system time elapsed between the start of the execution and the time when the containment is either checked or disproved.

Our results indicate that the KSolver, that is not optimized, suffers for modally-heavy formulæ (tests 13, 14, 15 and 16) while SPARQL-Algebra and MSPASS are less affected.

We observe that MSPASS outperforms the other approaches for all tests. Specifically, the running times reported in Figure 6.14 show that the MSPASS[3] solver is up to 10 times faster than SPARQL-Algebra.

Experiments show that the approach is not only feasible but effectively leverages on the availability of K_n solvers (and first-logic solvers) for carrying the static analysis of real-world SPARQL queries. P

[3]We used SPASS 3.7 with options -EMLTranslation=-2, -EMLFuncNary=1, -PProblem=0, -Sorts=0, -Select=2, -CNFOptSkolem=0, -CNFStrSkolem=0, -RInput=0, -TimeLimit=1000. Compiled with gcc-4.9.1.

One advantage of our encoding procedures of SPARQL Queries into K formulæ is that it opens the doors for implementations based on different kinds of solvers for the well-studied $\mathtt{K}_n$ logic. We can replace our own satisfiability solver by any third-party solver capable of solving the satisfiability problem for $\mathtt{K}_n$. Thus we can benefit from years of research in optimization of these solvers.

6.9 Conclusion

We propose a way to solve the query containment problem for a fragment of SPARQL queries with the optional operator. Our approach consists in a linear translation of these queries in terms of formulæ expressed in the modal logic $\mathtt{K}_n$. This approach naturally supports the optional operator. One advantage of this approach is to open the way for implementations using off-the-shelf satisfiability solvers for $\mathtt{K}_n$. This makes it possible to benefit from years of research in optimization of modal logic satisfiability solvers in the context of SPARQL static analysis. Here, we would like to emphasize that no implementation, using modal logic without fixpoints has been reported in previous works in literature.

7

Towards Query-Update Independence Analysis for SPARQL

This chapter reports on preliminary investigations concerning a novel topic: the query-update independence problem for `SPARQL`*. A query is independent of an update when the execution of the update does not affect the result of the query. Determining independence is especially useful in the context of huge* `RDF` *repositories, where it permits to avoid expensive yet useless re-evaluation of queries. While this problem has been intensively studied for fragments of relational calculus, to the best of our knowledge, no works exist for the standard query language for the semantic web. We report on our investigations on how a notion of independence can be defined in the new setting brought by* `SPARQL` *and* `RDF`*.*

Contents

7.1 Introduction **108**

7.2 Requirements for Independence Analyses **108**

7.3 Notion of Independence **110**

7.3.1 Definition of Independence 111

7.3.2 A Condition for Independence 112

7.3.3 Direction and Approaches. 113

7.4 Syntactic checking **113**

7.5 Containment Approach **117**

7.5.1 From Updates to Select Queries 117

7.5.2 From Select Queries to Graphs 119

7.5.3 Homomorphism Notions 119

7.5.4 Containment between Queries and Updates 120

7.6 Conclusion **123**

7.1 Introduction

The release of an update language for SPARQL [Schenk et al., 2010] introduces a setting where one might now study the problem of determining independence between SPARQL queries and SPARQL updates.

Detecting query-update independence is of crucial importance in many contexts, for example to ensure isolation when queries and updates are executed concurrently [Goldstein and Larson, 2001, Chaudhuri et al., 1995]. It can be used in view maintenance to identify that some views are independent of certain updates. We can provide greater flexibility by identifying that one query is independent of updates made by another program or person. Finally, we can use independence in query optimization by ignoring parts of the RDF dataset for which updates do not affect a specific query (to ensure isolation).

In all these contexts, benefits are amplified when query-update independence can be statically detected, that is to say by analyzing only the structure of the query with respect to the structure of the update.

Nevertheless, and despite the importance of static query analysis, and in particular of query-update independence analysis, research on the static analysis of SPARQL queries has only received little attention so far. The study of static analysis considering the query-update independence problem for SPARQL is the main focus of this chapter.

In the following, we define a first notion of query-update independence for SPARQL (Section 7.3) and discuss several approaches for detecting such independence. We start with a simple condition for independence, that can be implemented via a syntactic check, and we prove its soundness and completeness (Section 7.4). Then, we provide a method reasoning over the graph patterns that can avoid a full re-evaluation of the query over the update graphs when a containment relation between the query and update holds (Section 7.5).

7.2 Requirements for Independence Analyses

To the best of our knowledge, our work is the first to formalize and investigate the query-update independence problem for the SPARQL query language. As presented in Chapter 3, several techniques have been developed for the static analysis of SPARQL, mainly focusing on query containment [Serfiotis et al., 2005, Chekol et al., 2011, Letelier et al., 2012], possibly in presence of schemas [Chekol et al., 2011, Chekol et al., 2012b]. Such techniques aim at detecting relations (typically inclusions) between two queries, both evaluated using the same semantics. This common semantics is a key ingredient that these techniques exploit. These techniques hardly extend to the present context where the semantics of an update is significantly different from the semantics of the query. We cannot take advantages of these techniques directly, due to a lack of a common semantics for the query and the update.

Moreover, in the presence of schema constraints, as pointed out in [Ahmeti and Polleres, 2013] several semantics can be provided for SPARQL UPDATE and, there is no "one-size-fits-all" update semantics.

This work is partly inspired by the works on SPARQL query caching [Martin et al., 2010], where a first notion of query solution invariance for caching is defined. Outside the SPARQL context, the query-update independence problem has been intensively studied, in particular in the relational context [Blakeley et al., 1989, Levy and Sagiv, 1993] and in the setting of XML structures [Benedikt and Cheney, 2010, Junedi et al., 2012]. However, due to the RDF's open world assumption, these results can not directly transfer to the SPARQL context.

When dealing with the query-update independence problem, we have to consider the following two requirements.

Costs

We can verify at runtime whether an update impacts a query: we simply run the update after a first query evaluation, then re-run the query, and finally compare the results. The overall cost c of such a check is dominated by the cost of evaluating the query (twice) on the whole RDF dataset and the update (once). Thus, a method for testing independence makes sense only if its cost is lower than c. A first class of interesting methods regroups all *purely static* analysis methods, whose cost depends only on the size of the query and on the size of the update, and not on the size of the RDF dataset. A second class of interesting methods regroups hybrid static/dynamic methods that might involve evaluation or partial evaluation of the query on a fraction of the RDF dataset, and whose cost lower than c nevertheless depends on the RDF dataset size.

Dealing with the Open-World Assumption

While similar query-update independence problems have been intensively studied for other query languages (in particular for fragments of the relational calculus), no work exists for SPARQL. We believe that one reason for this is due to RDF's underlying *open world assumption* (a.k.a. OWA). We recall that OWA applies to a system that has incomplete information. The web, for instance, is traditionally considered as a system with incomplete information. The absence of some information on the web does not mean that this information is false (as for system under *close world assumption* CWA), but simply that this information has not been made explicit: it is unknown (see example 7.2.1).

Example 7.2.1. *Consider the following information:*

array

Name
Joe
Martin
Louisa
John

Student

Name
Joe
Martin

If we ask for a person that is not a student, we have the following two answers:
Answer in CWA= {*Louisa,John*}
Answer in OWA= {}
Under the OWA, we do not know if Louisa and John are Student or not.

RDF constitutes an open-world framework that allows anyone to make statements about any resource.

Consider the following SPARQL query and SPARQL update over an arbitrary graph G:

q : SELECT ?a WHERE $\{(\texttt{?a}, \texttt{type}, \texttt{Person})\}$

u : DELETE $(\texttt{?x}, \texttt{type}, \texttt{City})$ WHERE $\{(\texttt{?x}, \texttt{type}, \texttt{City})\}$

The variable ?a in q selects all the entities of type Person, while variable ?x in u selects entities of type City. Intuitively, one might consider that the domains of variables ?a and ?x are disjoint. This is the case when this disjointness constraint is explicitly specified, i.e. using OWL assertions. However, in SPARQL there is no such constraints on the RDF by default. Instead, OWA applies and this notion of disjointness is unknown.

In the rest of the chapter, we consider SPARQL in the absence of OWL constraints on the graphs. Considering SPARQL with OWL constraints is beyond the scope of this chapter and briefly commented in the perspectives of this thesis.

Developing independence analysis methods for SPARQL without constraints is challenging in several aspects. This is due to the nature of SPARQL based on variables and the notion of graph pattern matching. For this reason, previous methods developed for query languages such as SQL and XPath cannot be directly used for SPARQL. Instead, new methods must be developed. The present chapter reports on my preliminary investigations toward this goal.

7.3 Notion of Independence

In this section we introduce and formalize the query-update independence problem for SPARQL. Intuitively, a query q is independent of an update u, if evaluating q after or before u returns the same result. In other terms, independence holds whenever solutions of query patterns remain unchanged during the addition or deletion of triples from the RDF graph.

We first recall the closest notion that we have found in the literature:

Proposition 7.3.1 (Query Solution Invariance [Martin et al., 2010])**.** *If Ω is the set of all solutions for the query pattern q, with respect to an RDF graph G and for a triple t, there exists no mapping ρ from query variables to RDF terms such that $t \in \rho(q)$, then Ω is also the set of all solutions for $G_+ = G \cup \{t\}$ and $G_- = G \setminus \{t\}$.*

We generalize this proposition below, taking into consideration an update u that defines a set of triples to add or to delete from an RDF graph G.

7.3.1 Definition of Independence

Proposition 7.3.2. *Let q be a query and u be an update. We say that q and u are independent if and only if*

$$\forall G.\ [\![q]\!]_G = [\![q]\!]_{u(G)}$$

Independence between queries and updates is thus expressed as the equivalence of two evaluations: one evaluation that computes the answer to the query before the update, and a second evaluation that computes the answer after the update. Consider the following examples.

Example 7.3.1 (Query-Update Dependences)**.** *Consider the following graph G:*

$$G = \{(\mathtt{A}_1, \mathtt{name}, \mathtt{Joe}), (\mathtt{A}_1, \mathtt{phone}, \mathtt{1111}), (\mathtt{A}_4, \mathtt{name}, \mathtt{John}), (\mathtt{A}_4, \mathtt{phone}, \mathtt{4444}), (\mathtt{A}_4, \mathtt{fax}, \mathtt{5555})\}$$

and the following query q:

$$\begin{array}{lll} q: & \textit{SELECT} & \textit{?a ?n ?p} \\ & \textit{WHERE} & \{(\textit{?a}, \mathtt{name}, \textit{?n})\text{AND}(\textit{?a}, \mathtt{phone}, \textit{?p})\} \end{array}$$

then the evaluation of q over G, $[\![q]\!]_G = \Omega_G$ is the following:

$$\Omega_G = \{\varrho_1 = \{\textit{?a} \to \textit{A}_1, \textit{?n} \to \textit{Joe}, \textit{?p} \to \textit{1111}\}, \varrho_2 = \{\textit{?a} \to \textit{A}_4, \textit{?name} \to \textit{John}, \textit{?p} \to 4444\}\}$$

Now, consider u be the following update:

$$\begin{array}{lll} u: & \textit{DELETE} & (\textit{?a}, \mathtt{name}, \textit{?n}) \\ & \textit{WHERE} & \{((\textit{?a}, \mathtt{name}, \textit{?n})\text{AND}(\textit{?a}, \mathtt{fax}, \textit{?f})\} \end{array}$$

the result of the execution of u over G is updated graph:

$$u(G) = \{(\mathtt{A}_1, \mathtt{name}, \mathtt{Joe}), (\mathtt{A}_1, \mathtt{phone}, \mathtt{1111}), (\mathtt{A}_4, \mathtt{phone}, \mathtt{4444}), (\mathtt{A}_4, \mathtt{fax}, \mathtt{5555})\}$$

and the answers of the query q over the graph updated $u(G)$ is:

$$[\![q]\!]_{u(G)} = \Omega_{u(G)} = \{\varrho = \{\textit{?a} \to \textit{A}_1, \textit{?n} \to \textit{Joe}, \textit{?p} \to \textit{1111}\}\}$$

It is easy to see that $[\![q]\!]_G = \Omega_G \neq \Omega_{u(G)} = [\![q]\!]_{u_G}$

Otherwise, consider the following update

$$u': \quad \begin{array}{ll} \textit{DELETE} & (\texttt{?a}, \texttt{fax}, \texttt{?n}) \\ \textit{WHERE} & \{((\texttt{?a}, \texttt{name}, \texttt{?n}) \text{AND} (\texttt{?a}, \texttt{fax}, \texttt{?f})\} \end{array}$$

the result of executing u' over G is the update graph:

$$u'(G) = \{(\texttt{A}_1, \texttt{name}, \texttt{Joe}), (\texttt{A}_1, \texttt{phone}, \texttt{1111}), (\texttt{A}_4, \texttt{name}, \texttt{John}), (\texttt{A}_4, \texttt{phone}, \texttt{4444})\}$$

and the answers of the query q over the graph update $u'(G)$ is $[\![q]\!]_{u'(G)} = [\![q]\!]_G$, so the query and the update are independent.

Intuitively, the triple patterns that appear in the `DeleteClause` play an important role in detecting the query update independence.

7.3.2 A Condition for Independence

We define a condition for checking query-update independence, as follows:

Theorem 7.3.3 (Condition for independence)**.** *Let $\mathcal{Q}$ be the set of all triple patterns that appear in the query q and $\mathcal{U}$ be the set of all quad patterns that appear in the `DeleteClause` and `InsertClause` of an update u:*

$$\Big(\forall G.\ [\![q]\!]_G = [\![q]\!]_{u(G)} \Big) \Leftrightarrow \ \mathcal{Q} \dot{\cap} \mathcal{U} = \emptyset$$

where $\mathcal{Q} \dot{\cap} \mathcal{U} = \emptyset$ is defined as follows:

$$\forall\ t_q \in \mathcal{Q},\ t_u \in \mathcal{U},\ t_q \not\equiv t_u$$

and

$$t_q \not\equiv t_u \Leftrightarrow \forall \varrho_q \in [\![t_q]\!]_G, \forall \varrho_u \in [\![t_u]\!]_G, \varrho_q(t_q) \neq \varrho_u(t_u).$$

Proof. (Sketch)

($\Rightarrow$) By contradiction.

Suppose $\exists G'.[\![q]\!]_{G'} \neq [\![q]\!]_{u(G')}$, it means that there exists at least a triple t that is added or deleted from G' such that t is a triple matching a triple pattern of the query q. We can easily check that $t \in \mathcal{Q} \cap \mathcal{U} \neq \emptyset$. In fact, $t \in \mathcal{U}$ holds because t is a triple added to or deleted from G', so $\exists t_u \in \mathcal{U}.t \equiv t_u$ and $t \in \mathcal{Q}$ holds because the result of q, before and after u, changes, so $\exists t_q \in \mathcal{Q}.t \equiv t_q$.

($\Leftarrow$) By contradiction.

Suppose $\mathcal{Q} \cap \mathcal{U} \neq \emptyset$, by definition, $\exists t_u \in \mathcal{U}, t_q \in \mathcal{Q}.t_u \equiv t_q$. Let $t_u \equiv t_q$, there always exists a ground triple t that matches both t_q and t_u (i.e. replacing variables by blank nodes). Starting from t we can easily construct a graph G ([Hayes, 2004]), such that $[\![q]\!]_G \neq \emptyset$ and $[\![q]\!]_G \neq [\![q]\!]_{G \setminus \{t\}}$ or $[\![q]\!]_G \neq [\![q]\!]_{G \cup \{t\}}$. □

Our condition above can be statically checked over the query and the update, independently from any particular RDF dataset. Such a test fits in the category of purely static analysis methods. This condition will be the base of our investigation in techniques to solve the query-update independence problem for SPARQL.

7.3.3 Direction and Approaches.

In the next sections, we report two propositions for detecting static properties between queries and updates in SPARQL.

To the best of our knowledge no works report about the query update independence problem and how updates can be efficiently performed when dependences are detected. The results obtained and described in the following are still in an early stage.

The first technique proposed is based on a syntactic check for the condition of Theorem 7.3.3. This technique is a merely implementation of the condition of independence, but presents limitations due to RDF'sOWA that we comment in the following (see Example 7.4.4) .

To cope with these limitations, we report a notion of containment between the graph patterns of queries and updates to qualify their mutual dependence.

7.4 Syntactic checking

We investigate under which types of updates the results of SPARQL query patterns change. Intuitively speaking the solution of a query pattern stays the same at least until a triple, which matches any of the triple patterns being part of the graph pattern, is added to or deleted from the RDF dataset through an update operation [Martin et al., 2010]. In Theorem 7.3.3, we have presented a condition for independence. This condition can be checked through a syntactic check between triples patterns.

In the following, we denote by T the set of all possible *terms* of a triple pattern, such that $T = UBLV$.

Definition 7.4.1 (Function `Equiv`). *The function* $\mathit{Equiv}: T \times T \rightarrow \{\texttt{true}, \texttt{false}\}$ *returns* `true` *if the pair of terms in input matches,* `false` *otherwise.*

$$\mathit{Equiv}(t_1, t_2) = \begin{cases} \texttt{true} & \textit{if } t_1 \in BV \quad \textit{or} \quad t_2 \in BV \quad \textit{or} \quad t_1 = t_2 \\ \texttt{false} & \textit{otherwise} \end{cases}$$

This function can be extended to triple patterns, such that $\mathit{Equiv} : tp \times tp \rightarrow \{\texttt{true}, \texttt{false}\}$ *is defined as follows*

$$\mathit{Equiv}\Big((s_1, p_1, o_1), (s_2, p_2, o_2)\Big) = \mathit{Equiv}(s_1, s_2) \ \&\& \ \mathit{Equiv}(p_1, p_2) \ \&\& \ \mathit{Equiv}(o_1, o_2)$$

Note that blank nodes in graph patterns act as variables [Garlik and Seaborne, 2013].

Using the function "`Equiv`", we are now able to define when two triple patterns tp_1, tp_2 do not match on the same portion of a graph (denoted by $tp_1 \not\equiv tp_2$).

Proposition 7.4.1. *Given two triple patterns,* tp_1 *and* tp_2*,*

$$tp_1 \not\equiv tp_2 \quad \Leftrightarrow \quad \mathit{Equiv}(tp_1, tp_2) = \texttt{false}$$

Example 7.4.1. *Consider the following triple patterns*

$tp_1 = (\texttt{?x}, \texttt{phone}, \texttt{1111})$

$tp_2 = (\texttt{A}_4, \texttt{phone}, \texttt{?p})$

$tp_3 = (\texttt{A}_4, \texttt{fax}, \texttt{?f})$

- $tp_1 \equiv tp_2 \Leftrightarrow \mathit{Equiv}(tp_1, tp_2) = \texttt{true}$

$$\mathit{Equiv}(tp_1, tp_2) = \underbrace{\mathit{Equiv}(\texttt{?x}, \texttt{A}_4)}_{\mathit{true}} \ \&\& \ \underbrace{\mathit{Equiv}(\texttt{phone}, \texttt{phone})}_{\mathit{true}} \ \&\& \ \underbrace{\mathit{Equiv}(\texttt{1111}, \texttt{?p})}_{\mathit{true}}$$

- $tp_1 \not\equiv tp_3 \Leftrightarrow \mathit{Equiv}(tp_1, tp_3) = \texttt{false}$

$$Equiv(tp_1, tp_3) = \underbrace{\mathit{Equiv}(\texttt{?x}, \texttt{A}_4)}_{\mathit{true}} \ \&\& \ \underbrace{\mathit{Equiv}(\texttt{phone}, \texttt{fax})}_{\mathit{false}} \ \&\& \ \underbrace{\mathit{Equiv}(\texttt{1111}, \texttt{?f})}_{\mathit{true}}$$

- $tp_2 \not\equiv tp_3 \Leftrightarrow \mathit{Equiv}(tp_1, tp_2) = \texttt{false}$

$$Equiv(tp_2, tp_3) = \underbrace{\mathit{Equiv}(\texttt{A}_4, \texttt{A}_4)}_{\mathit{true}} \ \&\& \ \underbrace{\mathit{Equiv}(\texttt{phone}, \texttt{fax})}_{\mathit{false}} \ \&\& \ \underbrace{\mathit{Equiv}(\texttt{?p}, \texttt{?f})}_{\mathit{true}}$$

We can rewrite the condition for independence of Theorem 7.3.3 using the `Equiv` function, as follows.

Proposition 7.4.2 (Syntactic check test)**.** *Let q be a query and $u(q_d, q_i, q_w)$ be an update operation, we say that q-u are independent if and only if*

$$\forall t_q \in \mathcal{Q},\ \forall t_u \in \mathcal{U}. \quad \mathit{Equiv}(t_q, t_u) = \texttt{false}$$

where $\mathcal{Q} = q$ and $\mathcal{U} = q_d$ UNION q_i.

Example 7.4.2 (Independence through syntactic check)**.** *Consider the query and update of the Example 7.3.1.*

$$\begin{array}{lll} q: & \mathit{SELECT} & ?a\ ?n\ ?p \\ & \mathit{WHERE} & \{(?a, \texttt{name}, ?n)\text{AND}(?a, \texttt{phone}, ?p)\} \\ u: & \mathit{DELETE} & (?a, \texttt{name}, ?n) \\ & \mathit{WHERE} & \{(?a, \texttt{name}, ?n)\text{AND}(?a, \texttt{fax}, ?f)\} \end{array}$$

then, the sets $\mathcal{Q}$ and $\mathcal{U}$ are the following:

$$\mathcal{Q} = \{(?a, \texttt{name}, ?n), (?a, \texttt{phone}, ?p)\}$$

$$\mathcal{U} = \{(?a, \texttt{name}, ?n)\}$$

For Proposition 7.4.2, we have that q is independent from u, if and only if the following condition is `false`*.*

$$\mathit{Equiv}((?a, \texttt{name}, ?n), (?a, \texttt{name}, ?n)) \ \ || \ \ \mathit{Equiv}((?a, \texttt{name}, ?n), (?a, \texttt{phone}, ?p)) = \mathit{false}$$

To be false, $\mathit{Equiv}((?a, \texttt{name}, ?n), (?a, \texttt{name}, ?n))$ *and* $\mathit{Equiv}((?a, \texttt{name}, ?n), (?a, \texttt{phone}, ?p))$ *must be false both. But the first condition is trivially true. Hence q and u are dependent.*

The function `Equiv` allows to check query-update independece for every input `RDF` graphs. The complexity of the approach is $\Theta(n \times m)$, where n is the number of triple patterns in the `WhereClause` of the query, and m is the number of quad patterns in the `DeleteClause` and `InsertClause` of the update. This approach presents some limitations strictly related to the `SPARQL`'s `OWA`. Consider the example 7.4.3.

Example 7.4.3. *Given q and u defined as follows:*

$$\begin{array}{lll} q: & \mathit{SELECT} & ?a\ ?n \\ & \mathit{WHERE} & \{(?a, \texttt{name}, ?n)\} \\ u: & \mathit{DELETE} & (?x, ?y, ?z) \\ & \mathit{WHERE} & \{(?x, \texttt{phone}, \texttt{1111})\text{AND}(?x, ?y, ?z)\} \end{array}$$

It is easy to see that $q - u$ are dependent, due the truth of $\mathit{Equiv}((?x, ?y, ?z), (?a, \texttt{name}, ?n))$*.*

We can observe that in presence of quad pattern with three variables in the `InsertClause` or `DeleteClause` of an update, every query is dependent of u. This is a direct consequence of the `RDF`'s `OWA` and the possibility of adding any kind of information into an `RDF` graph.

Proposition 7.4.3. *Let $u(q_d, q_i, q_w)$ be an update, t be a quad pattern of u, such that $t \in \mathcal{U}$ and t contains only variables, then for every query q, there exists a graph G such that q and u are dependent.*

$$\forall q, u.\ t \in \mathcal{U},\ t : V \times V \times V \ \Rightarrow\ \exists G. [\![q]\!]_G \neq [\![q]\!]_{u(G)}$$

Proof. We assume that $t \in \mathcal{U}$ is a quad pattern with three variables. Hence, we have to prove that $\exists G. [\![q]\!]_G \neq [\![q]\!]_{u(G)}$ holds. By Theorem 6.6.1, we have:

$$\exists G. [\![q]\!]_G \neq [\![q]\!]_{u(G)} \Leftrightarrow \mathcal{U} \dot{\cap} \mathcal{Q} \neq \emptyset$$

$\mathcal{Q} \dot{\cap} \mathcal{U} \neq \emptyset$ holds by definition, when $t \in \mathcal{U}$ and t contains three variables, since $t \equiv t_q$, for every t_q in $\mathcal{Q}$ □

Example 7.4.4. *Consider the query q and the update u of the previous example 7.4.3, where $u(t, \_, q_W)$ has a quad pattern t with three variable in the* `DeleteClause`*. In the following, we show the existence of a graph G, such that the evaluation of q over G, before and after u, changes. Hence q-u are dependent.*

Let G_q and G_u be two graph obtained by replacing variables with blank nodes in the triple patterns of q and in the quad patterns of u, respectively (see [Hayes, 2004]).

The graph $G_q = \{(\_:a, name, \_:n)\}$ is s.t. $[\![q]\!]_{G_q} \neq \emptyset$.

The graph $G_u = \{(\_:x, \_:y, \_:z), (\_:x, \texttt{phone}, \texttt{1111})\}$ is s.t. $G_u \neq u(G_u)$. Note, in according with the `SPARQL UPDATE` *Semantics (see Chapter 2), we have $[\![t \text{ AND } q_w]\!]_{G_u} \neq \emptyset$.*

By renaming blank nodes of G_u in the following way

$$\{\_:x \rightarrow \_:a,\ \_:y \rightarrow name,\ \_:z \rightarrow n\}$$

,

we obtain, by merging G_q and G_u renamed, the following graph G

$$G = \{(\_:a, \texttt{name}, \_:n), (\_:a, \texttt{phone}, \texttt{1111})\}$$

such that the result of evaluating q over G is

$$\Omega_G = \{\varrho = \{?a \rightarrow \_:a, ?n \rightarrow \_:n\}\}$$

The execution of u over G is such that $u(G) = \{\}$.

Hence, $[\![q]\!]_G \neq [\![q]\!]_{u(G)}$

We have found a graph G such that the evaluation of q over G before and after u changes.

Due the limitation of this approach, that more often detect dependences between queries and updates, in the following we introduce an approach that qualifies these dependences using a notion of containment. We study the structure of a query and an update in terms of their graph patterns. Then we define the containment among graph patterns, that might allow more precise analyses.

7.5 Containment Approach

In the previous section, we have presented an approach to syntactically check the condition for independence of Theorem 7.3.3. The approach reviewed in this section aims at qualifying the dependence relation between the query and the update, in terms of query patterns containment. When containment is detected between the query and the update, more efficient execution of updates should be put in place.

The basic idea behind the approach is to consider the query and update graph patterns as graphs (intuitively by replacing variables with blank nodes) and verifies the existence of an homomorphism between these graphs. In the following, we restrict our attention to query patterns that contain only the constructors AND (`CQs` fragment).

This procedure aims at qualifying the dependence between a query and an update, using a notion of graph patterns containment. In order to check this containment condition, a two-steps procedure is proposed.

First, we convert the graph patterns of the query and the update into graphs; then we check the existence of a graph homomorphism between the query and the update graphs. The existence of an homomorphism between graphs implies a containment condition (see Definition 7.5.3).

In the following, in order to implement the first step of the procedure, we introduce two translation functions: the first takes as input an update and returns a query, and the second takes a query and returns an `RDF` graph.

Once we have two `RDF` graphs, we go to the second step of the procedure and a containment condition between `RDF` graphs is detected using the notion of homomorphism between `RDF` graphs.

7.5.1 From Updates to Select Queries

We present a procedure for rewriting updates into queries. For our purpose, we split the procedure into two subprocedures:

1. a subprocedure for rewriting delete operations (`DeleteClause`)
2. a subprocedure for rewriting insert operations (`InsertClause`)

The interest in splitting the procedure in two is strictly related to the different nature of the insert/delete operations and how a notion of containment can have sense in presence of delete operations, but not in presence of inserts (and vice versa).

We introduce two similar functions, Δ_{del} and Δ_{ins}, that respectively take as input the `DeleteClause` and the `InsertClause` of an update and return two different queries

Definition 7.5.1 (Δ_{del} (resp. Δ_{ins})). *Given an update $u = u(q_d, q_i, q_w)$, the function $\Delta_{del}(u)$ (respectively $\Delta_{ins}(u)$) returns a query, q_{del} (resp. q_{ins}), containing all triple patterns appearing in the* `DeleteClause` *q_d (resp.* `InsertClause` *q_i), and the* `WhereClause`*, q_w, of u. Hence the resulting query q_{del} (resp. q_{ins}) is such that:*

- *all the variables in u will be distinguished variables in q_{del} (resp. q_{ins})*
- *the query pattern q_{del} (resp. q_{ins}) is the conjunction of the quad patterns q_d (resp. q_i) and q_w, such that:*

$$q_{del} = \Delta_{del}\Big(u(q_d, q_i, q_w)\Big) = q_d \text{ AND } q_w$$

$$q_{ins} = \Delta_{ins}\Big(u(q_d, q_i, q_w)\Big) = q_i \text{ AND } q_w$$

Example 7.5.1 (Δ_{del}). *Given the following update:*

$$\begin{array}{lll} u: & \textit{DELETE} & (\texttt{?a}, \texttt{name}, \texttt{?n}) \\ & \textit{WHERE} & \{(\texttt{?a}, \texttt{name}, \texttt{?n}) \text{AND} (\texttt{?a}, \texttt{fax}, \texttt{?f})\} \end{array}$$

the corresponding query $q_{del} = \Delta_{del}(u)$ is

$$\begin{array}{lll} q_{del}: & \textit{SELECT} & \texttt{?a ?n ?f} \\ & \textit{WHERE} & \{(\texttt{?a}, \texttt{name}, \texttt{?n}) \text{AND} (\texttt{?a}, \texttt{fax}, \texttt{?f})\} \end{array}$$

Note that the translation procedures loose information concerning the update. More precisely, we are not able to identify, inside the resulting select query, which are the triple patterns that appeared in the `DeleteClause` or `InsertClause` of the update. These information, as we have seen in Theorem 7.3.3, are vital for detecting a condition for independence. We have identified such triple patterns as $\mathcal{U}$. In this approach, we already know that the query and the update are dependent, hence we are interested only in qualifying this dependence relation. For this purpose, the knowledge of the set of the whole triple patterns of an update is sufficient.

7.5.2 From Select Queries to Graphs

For every query q it is possible to build a canonical RDF graph [Hayes, 2004], by replacing variables with blanks (Theorem 6.6.1). We denote by G_q the RDF graph obtained by replacing variables in q with blank nodes.

Definition 7.5.2. *Let q be a query, his canonical RDF graph G_q is such that:*

$$\forall(\mathtt{s},\mathtt{p},\mathtt{o}) \in q. \quad (\lambda(\mathtt{s}), \lambda(\mathtt{p}), \lambda(\mathtt{o})) \in G_q$$

where $\lambda : UBLV \rightarrow UBL$ is defined as follows

$$\lambda(x) = \begin{cases} \mathit{newBlank}(x) & \text{if } x \in V \\ x & \text{otherwise} \end{cases}$$

The function $\mathit{newBlank}(.) : V \rightarrow B$ returns a fresh blank nodes given a variable in input.

Example 7.5.2 (From Select to Graph)**.** *Given the select query q of the Example 7.5.1:*

$$\begin{array}{lll} q: & \mathit{SELECT} & \mathit{?a}\ \mathit{?n}\ \mathit{?f} \\ & \mathit{WHERE} & \{(\mathit{?a}, \mathtt{name}, \mathit{?n}) \text{AND} (\mathit{?a}, \mathtt{fax}, \mathit{?f})\} \end{array}$$

the corresponding graph is:

$$G_q = \{(\_\mathit{:a}, \mathtt{name}, \_\mathit{:n}), (\_\mathit{:a}, \mathtt{fax}, \_\mathit{:f})\}$$

where the variable ?a is replaced by $\lambda(\mathtt{?a}) = \mathtt{newBlank}(\mathtt{?a}) = \mathtt{\_:a}$ and the variable ?f by $\mathtt{newBlank}(\mathtt{?f}) = \mathtt{\_:f}$.

7.5.3 Homomorphism Notions

We have shown how to translate updates and queries into RDF canonical graphs. Now, we report the notion of graph homomorphism between RDF graphs [Arenas et al., 2009].

Definition 7.5.3 (RDF Graph Homomorphism)**.** *A homomorphism ϕ between two RDF graphs G_1 and G_2 (written as $\phi : G_1 \rightarrow G_2$) is a blank node mapping $\varphi : B \rightarrow UBL$ such that $\phi(G_1) \subseteq G_2$, where $\phi(G_1)$ denotes the graph obtained by replacing in G_1 every blank node $\_\mathit{:b} \in B$ with $\varphi(\_\mathit{:b})$.*

Note that the homomorphism definition is formulated in terms of graph "containment". For our purpose, where the graphs are canonical instances of graph patterns, the expression $G_1 \subseteq G_2$ is read G_1 is less constrained than G_2. In fact, G_1 and G_2 can be seen as sets of constraints (each triple pattern is a constraint) and the fact $G_1 \subseteq G_2$ means G_2 is more constrained than G_1.

7.5.4 Containment between Queries and Updates

Let a query q and an update u, we are now able to reason over the structures of q and u represented by their associated canonical graphs, G_q and G_u respectively.

To abstract the difficulties of the analysis, we consider a finite set of possible triple patterns $G = \{t_1, t_2, t_3\}$, such that only these triple patterns can appear in the query and the update (hence, $G_q \subseteq G$ and $G_u \subseteq G$).

In order to facilitate the understanding of our discussion, we consider also the graph in Figure 7.1, as the default graph of our examples, such that t_1, t_2, t_3 can match (hence are homomorphic [Arenas et al., 2009]) with the following triples:

- $t_1 \equiv (\mathtt{A}_1, \mathtt{name}, \mathtt{Joe})$, $t_1 \equiv (\mathtt{A}_4, \mathtt{name}, \mathtt{John})$
- $t_2 \equiv (\mathtt{A}_1, \mathtt{phone}, \mathtt{1111})$, $t_2 \equiv (\mathtt{A}_4, \mathtt{phone}, \mathtt{4444})$
- $t_3 \equiv (\mathtt{A}_4, \mathtt{fax}, \mathtt{5555})$

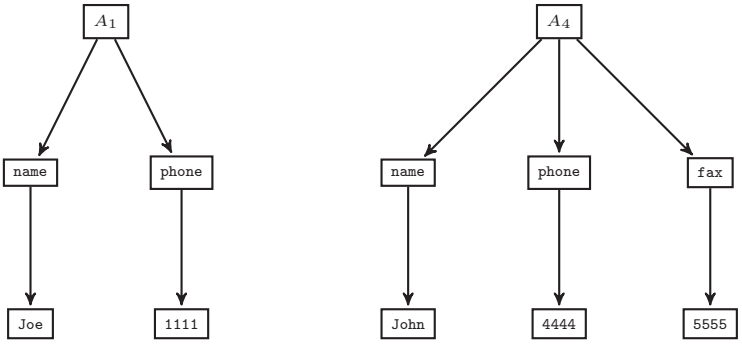


Figure 7.1: RDF graph of Example 7.3.1

We can now discuss which kinds of homomorphism relations (hence containment relations) between the query and update structures are interesting in order to detect static properties, that might allow more precise analyses. We recall that we want to analyse the delete operations separately from the insert operations. Therefore, we have three possible graphs G_q, G_{del} and G_{ins} (subset of G) and 4 possible cases of containment. In the following triples in *red* identify triples to delete, while triples in *blue* identify the triples to add.

SCENARIO A $[\phi(\mathtt{G_{del}}) \subseteq \mathtt{G_q}]$. The query is more constrained than the delete update.

Suppose we have $G_{del} = \{\overline{t_1}, t_2\}$ and $G_q = \{t_1, t_2, t_3\}$. In this scenario we can directly update the query result (or view). Every constraint expresses in the delete operation is also expressed in the query, hence if the result of the query is stored, we can directly update it. Consider the following example.

Example 7.5.3 (Scenario A)**.** *Figure 7.2 shows the query results (or view) enclosed in the black rectangle. The red rectangles enclose the set of triples that are involved in the update. As we can see, a red rectangle is entirely included in the black one. It means we can directly*

update the view because all the constraints required for deleting $(\mathtt{A}_4, \mathtt{name}, \mathtt{John})$ *are required also in the view.*

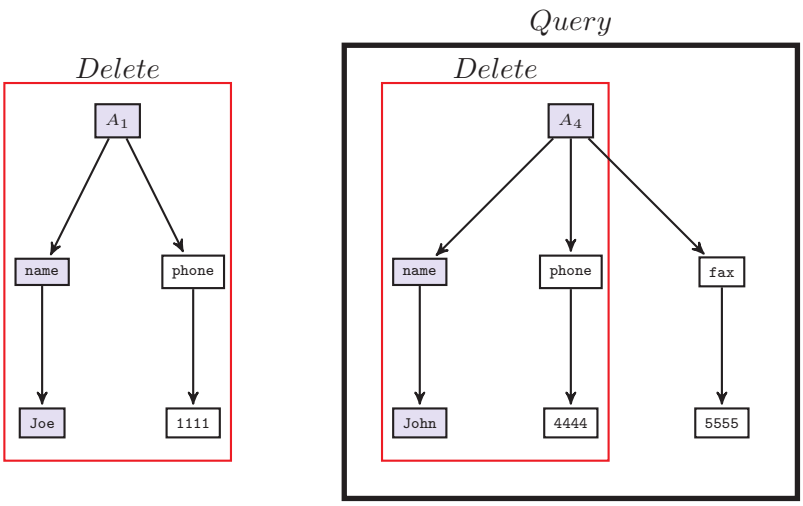


Figure 7.2: SCENARIO A. $\phi(\mathbf{G_{del}}) \subseteq \mathbf{G_q}$

SCENARIO B $[\phi(\mathbf{G_q}) \subset \mathbf{G_{del}}]$. The delete update is more constrained than the query.

Suppose we have $G_q = \{t_1, t_2\}$ and $G_{del} = \{t_1, t_2, t_3\}$. In this scenario we cannot update directly the query view. Not all the constraints of the update are constraints of the query, hence the update may ask information that are not stored. Consider the following example.

Example 7.5.4 (Scenario B)**.** *Figure 7.3 shows the query view enclosed in two black rectangle while the red rectangle encloses the set of triples that are involved in the update. As we can see, one of the black rectangles is strictly contained in the red one. It means that the query is less constrained than the delete update, i.e in the query the constraint* t_3 *is unknown. As a consequence, if we try to execute the update directly on the query, no solutions are found for the constraint* t_3*, so no triples are deleted from the view.*

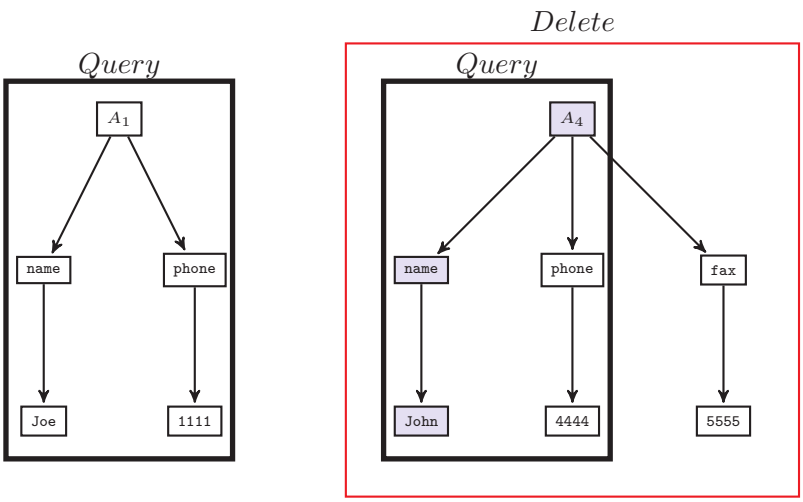


Figure 7.3: SCENARIO B. $\phi(\mathtt{G_q}) \subset \mathtt{G_{del}}$.

SCENARIO C $[\phi(\mathbf{G_{ins}}) \subseteq \mathbf{G_q}]$. The query is more constrained than the insert update.

Suppose we have $G_{ins} = \{\underline{t_1}, t_2\}$, $G_q = \{t_1, t_2, t_3\}$. In this scenario, we cannot directly update the query view. In contrast with the scenario A, in presence of insert operation the fact that q is more constrained than the update avoids the possibility of re-evaluating the query on the update results. In general, q requires more constraints that the update, hence these new triples may not satisfy all the constraints of q. Consider the following example.

Example 7.5.5 (Scenario C)**.** *Figure 7.4 shows the query result enclosed in the black rectangle, as usual, while the blue rectangle enclose the set of triples that are involved in the update. This blue rectangle contains triples that may be added to the query view. Unfortunately, t_3 is unknown in the blue rectangle, hence it prevents the possibility of directly adding the triples in the query view.*

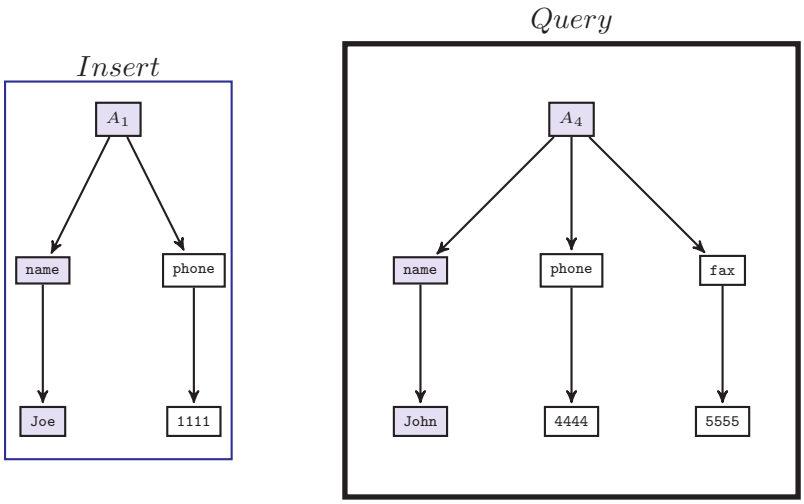


Figure 7.4: SCENARIO C. $\phi(\mathtt{G_{ins}}) \subset \mathtt{G_q}$.

SCENARIO D $[\phi(\mathbf{G_q}) \subseteq \mathbf{G_{ins}}]$. The insert operation is more constrained than the query.

Suppose we have:

$$G_q = \{t_1, t_2\} \qquad\qquad G_{ins} = \{\underline{t_1}, t_2, t_3\}$$

In this scenario we can directly update the query result. Every constraint of the query is also expressed in the insert update, hence every new triples can directly be inserted into the view. Consider the following example.

Example 7.5.6. *Figure 7.5 shows the query result enclosed in the black rectangle on the left, while the triples involved in the insert operations are enclosed in a blue rectangle on the right. We can draw a black dotted rectangle inside the blue one. This dotted rectangle potentially contains triples that can be added to the query view. As we can see, the dotted rectangle is entirely included in the blue one. It means we can directly re-evaluate the query over the update.*

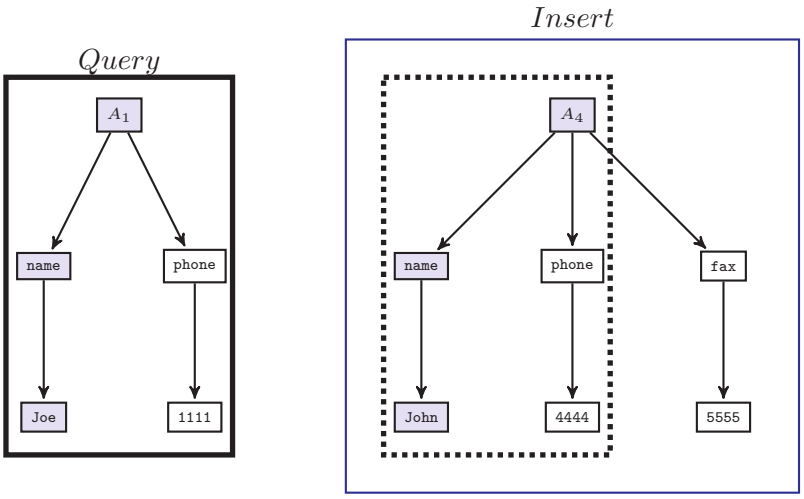


Figure 7.5: SCENARIO D. $\phi(\mathtt{G_q}) \subseteq \mathtt{G_{ins}}$

Synthesis

When a condition of dependence between a query and an update is detected, we have discussed a procedure that aims at qualifying such dependence relation. The purpose of this procedure is to identify containment relations between the query and the update that can allow an efficient execution of the update. We have identified two cases in which a notion of containment relation allows a more precise analysis.

1. In the presence of delete operations, let $q_{del} = \Delta_{del}(u)$, there exists a homomorphism $\phi : G_{q_{del}} \rightarrow G_q$ such that $\phi(G_{q_{del}}) \subseteq G_q$.

2. In the presence of insert operations, let $q_{ins} = \Delta_{ins}(u)$, there exists a homomorphism $\phi : G_q \rightarrow G_{q_{ins}}$ such that $\phi(G_q) \subseteq G_{q_{ins}}$.

The main advantage of this characterization of the query-update dependence condition is the possibility of avoiding full re-computation of a query over an updated graph. This characterization is still in a early stage, but we believe it might open a way to take advantages of well-known techniques of the containment problem (see Chapter 3) for dealing with the query-update independence problem.

7.6 Conclusion

To conclude, we have presented a first formalization of the `SPARQL` query-update independence problem. We motivate the interest of searching for methods that solve this problem. We also explain difficulties and discuss characteristics desired for algorithms that effectively check for independence. We have introduced a simple condition for detecting independence, which admits an efficient implementation. We have discussed other possible methods.

The first proposed approach provides an efficient implementation of the condition of independence. We are interested in checking when an update potentially can affect the result of a query. This condition can be easily satisfied when at least an added/removed triple matches with a triple pattern of the query, imposing a re-evaluation of the query itself.

The second proposed approach aims at qualifying the dependency relation between the query and the update. We are interested in checking when the structure of the query is contained into the structure of the update, and vice versa, in order to improve the execution of the update.

In the perspectives section of the next chapter, we discuss how to detect the query-update independence in presence of constraints.

8

Conclusion and Perspectives for Research

Contents

8.1 Conclusion **126**

8.2 Summary of the Publications **127**

8.3 Perspectives **127**

8.3.1 Containment under Schema 128

8.3.2 Query-Update Independence under Schema 128

8.3.3 Relational Algebra Based Approach 129

8.4 Summary **132**

8.1 Conclusion

We now summarize the main contributions of this work, and propose further research directions.

In this thesis, we have faced three aspects of research:

1. contribution in a well-studied domain: the query containment problem for SPARQL;
2. validation of the theoretical results with practical experiments that involved the development of implementation techniques;
3. preliminary investigations in a novel topic: the query-update independence problem for SPARQL.

We addressed static analysis for SPARQL. Static analysis refers to the study of queries without any knowledge of the dataset they will be evaluated over. Problems in this area include the idea of query containment and query-update independence detection.

Based on the theoretical studies of [Letelier et al., 2012, Pichler and Skritek, 2014], we propose a solution through a reduction to satisfiability in the logic, a technique which is inspired by [Calvanese et al., 2000, Genevès et al., 2007, Chekol, 2012]. We summarize below the results achieved in the development of this work.

With the finite model tree property (see Chapter 4) and the implementations that have been put to practice, the modal logic K with multiple modalities, denoted by $\mathtt{K}_n$, has been chosen for the task of static analysis of SPARQL queries.

In Chapter 6, we have shown how to encode RDF graphs into transition systems and well-designed SPARQL queries into $\mathtt{K}_n$-formulas, thus the formulas can be interpreted over transition systems. The algorithm proposed for translating a well-designed query q into $\mathtt{K}_n$-formula $\mathcal{A}(q)$ works on the tree representations of the query, called *pattern trees*. The containment of queries can be reduced to a satisfiability test by encoding the set inclusion $q_1 \subseteq q_2$ as an implication and the queries as formulas $\mathcal{A}(q_1) \rightarrow \mathcal{A}(q_2)$.

In order to complement our analysis, we have carried out several experiments. In this line, we proposed a compliance benchmark for containment, equivalence and satisfiability of semantic web queries with the OPTIONAL operator. The benchmark is used to test our K solver prototype (see Chapter 5) against the current-state-of-the-art tools. A comparison of these tools based on running times is discussed.

As its importance increases in the semantic web, SPARQL is being extended as a query language for modifying RDF graphs [Schenk et al., 2010]. SPARQL UPDATE introduces a setting where one might now study the problem of determining independence between SPARQL queries and SPARQL updates.

In this direction, we started to investigate the query-update problem for SPARQL. We have identified a condition for independence between a query and an update [Guido et al., 2015]. A query q and an update u are independent, if evaluating q after or before u returns the

same result. This condition for independence can be statically checked by analyzing the graph patterns in the `InsertClause` and `DeleteClause` of an update and the graph patterns of a query. We only take in consideration the graph patterns in the `InsertClause` and `DeleteClause`, because these graph patterns are the only ones that contain information concerning the triples to add to and to delete from a graph. Due to the `RDF`'s `OWA` and the possibility of adding any kind of triples to a graph, the information inside the `WhereClause` of the update are useless. As a consequence, more often a dependence condition between the query and the update is detected. To cope with this limitation, we introduced an approach that aims at qualifying the query-update dependence when detected. We proposed a method, still in an early stage, that given a query dependent from an update, checks if the set of graph patterns of the query is entirely contained in the set of the graph patterns of the update (and vice versa). Checking these types of containment can avoid a full re-evaluation of the query in case of dependence from an update.

8.2 Summary of the Publications

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

All of these publications focus on the area of statically analysing semantic web queries.

The paper [Guido et al., 2015] studies for the first time the query-update independence problem for `SPARQL`. In the paper a condition for Independence in this new setting is done (see Chapter 7)

An article addressing the query containment problem for `SPARQL` with modal logic has been submitted to a journal.

We have not yet published our main results obtained for the containment problem of the well-designed queries with OPTIONAL operators. A paper is now ready to be published and it has been submitted to a major conference in the field.

A resumé of this thesis [Guido, 2015] has been presented at the `IJCAI 2015 - Doctoral Consortium`.

8.3 Perspectives

Further improvements can be performed on the developed static analysis techniques, both for the query containment problem and the query-update independence problem. In the following four plans are highlighted.

8.3.1 Containment under Schema

In this thesis the query containment problem for the well-designed `SPARQL` queries under schema has not been addressed.

It is not clear if schema information can be treated with the modal logic `K`. Due the absence of fixpoint operators, it is not trivial to find a solution for the propagation of the schema constraints over the worlds of the transition system. The use of the universal modalities might be a solution. Nevertheless, we can always efficiently extend the set of the supported features of the modal logic `K`. As we have seen in Chapter 4, if we add the great and least fixpoints to the logic `K`, we obtain the modal μ-calculus.

We have some example in literature, i.e. [Chekol, 2012], such that the use of the μ-calculus is power enough to address the query containment problem under schema. A schema can be encoded as a set of conjunctive formulae, that must be propagated over all worlds of the transition system using great fixpoints ν.

Following this direction, we aim at solving the query containment problem for well designed `SPARQL` queries under schema by encoding the schema axioms $\mathcal{C} = \{c_1, \ldots, c_n\}$ using a function η. The function η encodes each axiom into an equivalent formula, such that:

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

Hence, let q_1 and q_2 be two well-designed queries and C be a set of axioms, we should have

$$q_1 \sqsubseteq_{\mathcal{C}} q_2 \Leftrightarrow \eta(\mathcal{C}) \wedge \mathcal{A}(q_1) \wedge \neg\mathcal{A}(q_2) \text{ is unsatisfiable}$$

where the function $\mathcal{A}$ is the same introduced in Chapter 6, while η should be defined in function of the modal logic and the description logic adopted to express a schema.

A prove of the correctness of reducing query containment under schema to unsatisfiability test must be provided.

8.3.2 Query-Update Independence under Schema

In Chapter 7, we have identified two methods: one for detecting the query-update independence, the other for qualifying the dependence condition when the first method does not hold.

In the following, we identify two main research directions for the further development of methods aimed at detecting query-update independence:

Hybrid static/dynamic methods. The condition that we presented in Chapter 7 involves reasoning over all graphs, and is thus purely static. It would be very interesting to develop hybrid (static and dynamic) methods that would also take into account a specific dataset as input. Such methods should satisfy our cost requirements in order to remain relevant (c.f. Section 7.2) but would unlock the potential for many more detections of independence cases.

Methods supporting schema constraints. One perspective is the development of methods that analyse the structures of the query and of the update in the presence of constraints on the dataset, when such constraints are available. Typical constraints on RDF datasets include OWL2 constraints that can express, for example, the fact that two classes are disjoint. Such constraints – and more generally constraints expressed with any description logic supporting negation – are game-changers for the independence problem. The potential contradictions that they introduce can be leveraged for detecting more cases of independence.

In order to understand the impact of schema constraints, let us consider the following example.

Example 8.3.1 (Schema example)**.** *Let q be a query and u be an update defined as follows:*

$$q(\{?x\}) = \{(?x, fax, 2222)\}$$

$$u\Big(\{(?x, ?y, ?z)\}, \{\}, \{(\mathtt{A}_4, \mathtt{phone}, ?x) \text{ AND } (?x, ?y, ?z)\}\Big)$$

q and u are dependent under OWA. For example, if we add the triple $t = (\_:a, \mathtt{phone}, \mathtt{A}_2)$ in the graph of Figure 2.2, the resulting graph G is such that $[\![q]\!]_G \neq [\![q]\!]_{u(G)}$.

However, the situation changes completely if we add schema constraints stating that phone has a domain Person and a range Number and in addition that Person is disjoint from Number (which can be easily formulated in OWL2 for instance). In this case, we can see that the variable ?x of q belongs to "Person', while the ?x of u belongs to "Number", so the update does not affect the query result. In the presence of such constraints q and u are independent.

8.3.3 Relational Algebra Based Approach

For leveraging existing techniques in the search for algorithms for independence, an important and non-trivial step consists in finding a common formal ground that unifies the semantics of SPARQL queries and updates. In Chapter 2, we have presented the mapping based and the relational semantics of SPARQL and the naïve semantics of SPARQL UPDATE. In the following, we introduce the relational semantics of SPARQL UPDATE and we present an idea for detecting the query-update independence based on an unify relational semantics of queries and updates. The relational algebra of SPARQL was firstly introduced in [Cyganiak, 2005]. In [Chebotko et al., 2009], authors show the equivalence of the relational and mapping based semantics. To the best of our knowledge, this relational semantics has not yet been extend to SPARQL UPDATE.

8.3.3.1 Relational Algebra Based Semantics of SPARQL UPDATE

In the following, we extend the relational semantics to the SPARQL UPDATE. We first define the relational representation of an update and then we provide a set of premise-conclusion rules for the delete/insert operations.

Definition 8.3.1 (Relational representation of SPARQL UPDATE)**.** *The relational representation of a SPARQL UPDATE query is a set R of tuples r or simply a relation R containing all the triples that will be added to or deleted from a graph.*

Example 8.3.2 (Relational representation of SPARQL UPDATE solution)**.** *Consider the following query:*

DELETE (?a, phone, ?p)
WHERE {(?a, name, Joe)}

its solution over the RDF graph of Figure 2.2 is represented as follows.

$R_{where} =$ $\xi(R_{where})$:

?a	*name*	*Joe*
A_1	*name*	*Joe*

r_{where1} :

$R_{delete} =$ $\xi(R_{delete})$:

?a	*phone*	?p
A_1	*phone*	1111
A_3	*phone*	3333
A_4	*phone*	4444

$r_{delete1}$:
$r_{delete2}$:
$r_{delete3}$:

Now, we calculate $R = R_{where} \bowtie_{[R_{where}.\texttt{?a}=R_{delete}.\texttt{?a}]} R_{delete}$ and the result is:

$R =$ $\xi(R)$

?a	*name*	*Joe*	*phone*	?p
A_1	*name*	*Joe*	*phone*	1111

r_1

$R_{\pi(\texttt{?a,phone,?p})} =$ $\xi(R)$

?a	phone	?p
A_1	phone	1111

r_1

Figure 8.1: Relational representation of a delete update

The premise-conclusion rules for the SPARQL UPDATE queries are shown in Figure 8.1.

$$\frac{eval(tp, G) = R_1(tp.sp, tp.pp, tp.op) \qquad eval(gp) = R_2 \qquad R = R_1 \bowtie R_2}{eval(\texttt{DELETE}(tp)\texttt{WHERE}(gp), G) = \pi_{tp.sp,tp.pp,tp.op} R} \tag{6}$$

$$\frac{eval(tp, G) = R_1(tp.sp, tp.pp, tp.op) \qquad eval(gp) = R_2 \qquad R = R_1 \bowtie R_2}{eval(\texttt{INSERT}(tp)\texttt{WHERE}(gp), G) = \pi_{tp.sp,tp.pp,tp.op} R} \tag{7}$$

$$\frac{\begin{array}{c} eval(\texttt{DELETE}(tp)\texttt{WHERE}(gp), G) = R_1 \qquad eval(\texttt{INSERT}(tp_1)\texttt{WHERE}(gp), G) = R_2 \\ R = R_1 \cup R_2 \end{array}}{eval(\texttt{DELETE}(tp)\texttt{INSERT}(tp_1)\texttt{WHERE}(gp), G) = R} \tag{8}$$

Figure 8.2: Relational algebra based semantics of `SPARQL UPDATE`

8.3.3.2 Disjointness Analysis

We have introduced the relational representation of select queries and updates. We are now able to detect the query-update independence by checking the disjointness of a query and an update over their relational representation.

Example 8.3.3. *Take the relational representation of the query select example shown in Figure 2.4 and the update representation shown in Figure 8.1, we are able to say they are dependent over the graph in Figure 2.2.*

$R_q =$

$\xi(R)$:	?a	phone	?p	email	?e
r_1:	A_1	phone	1111	NULL	NULL
r_2:	A_2	phone	3333	NULL	NULL
r_3	A_4	phone	4444	email	john@p.fr

$R_u = R_{\pi(?a,\text{phone},?p)} =$

$\xi(R)$	?a	phone	?p
r_1	A_1	phone	1111

The triple `t=(`A_1 `phone 1111)` is part of the result of the select query and also is a triple to be deleted. If we execute the query after the update, we will have the following resulting relation:

R =

$\xi(R)$:	?a	phone	?p	email	?e
~~r_1:~~	~~A_1~~	~~phone~~	~~1111~~	~~NULL~~	~~NULL~~
r_2:	A_2	phone	3333	NULL	NULL
r_3	A_4	phone	4444	email	john@p.fr

The result is clearly changed.

Proposition 8.3.1. *For every query q and update u, q and u are independent if and only if for each triple pattern $t = (s, p, o) \in q$*

$$\pi_{t.s,t.p,t.o} R_q \cap R_u = \emptyset$$

where the intersection operator is a classical relational operator defined as follows

$$R_1 \cap R_2 = \{\ t \mid (t \in R_1)\ \textit{and}\ (t \in R_2)\ \textit{and}\ (\xi(R_1) \equiv \xi(R_2))\}$$

There are several works that study transformations from SPARQL into SQL [Cyganiak, 2005, Chebotko, 2008, Chebotko et al., 2009, Polleres, 2007, Elliott et al., 2009]. To the best of our knowledge, this could be the first work that attempts to define the relational semantics of SPARQL UPDATE. One of the objective of this study is the identification of a common formal ground that unifies the semantics of the queries and the updates. We have chosen the relational semantics because a lot of RDF stores use DBMS technologies, where RDF triples can be stored into relations with three-column schema. This work is still in an early stage. We move from the RDF's OWA to the SQL relational representation under CWA. Hence, we have to consider all the consequences related to this transition.

8.4 Summary

We have proposed a way to solve the query containment problem for a fragment of SPARQL queries with the OPTIONAL operator. Our approach consists in a neat linear translation of these queries in terms of formulas expressed in the modal logic $\mathtt{K}_n$. One advantage of this approach is to open the way for implementations using off-the-shelf satisfiability solvers for $\mathtt{K}_n$. This makes it possible to benefit from years of research in optimization of modal logic satisfiability solvers in the context of SPARQL static analysis. One further advantage of using a generic and well-characterized logical language such as $\mathtt{K}_n$, as opposed to ad-hoc algorithms, is extensiblity. $\mathtt{K}_n$ is subsumed by decidable logics that open perspectives for further extensions of the approach, as the possibility of reasoning about containment in presence of schema constraints.

We presented a first formalization of the SPARQL query-update independence problem. To the best of our knowledge, this is the first work that discuss the query-update independence problem in the SPARQL-RDF settings. We motivated the interest of searching for methods that solve this problem. We also explained difficulties and discussed characteristics desired for algorithms that effectively check for independence. We introduced a simple condition for detecting independence, which admits an efficient implementation. In presence of query-update dependence, we introduced a method for qualifying such dependence that might allow more precise analyses. We identified more general classes of interesting methods and drawn perspectives for further research.

Bibliography

[Abiteboul et al., 1994] Abiteboul, S., Hull, R., and Vianu, V. (1994). Foundations of Databases. Addison Wesley, facsimile edition.

[Ahmeti and Polleres, 2013] Ahmeti, A. and Polleres, A. (2013). SPARQL Update under RDFS Entailment in Fully Materialized and Redundancy-Free Triple Stores. In Proceedings of the 2nd International Workshop on Ordering and Reasoning, OrdRing 2013, Co-located with the 12th International Semantic Web Conference (ISWC 2013), Sydney, Australia, October 22nd, 2013, pages 21–32.

[Aho et al., 1979a] Aho, A. V., Sagiv, Y., and Ullman, J. D. (1979a). Efficient Optimization of a Class of Relational Expressions. ACM Trans. Database Syst., 4(4):435–454.

[Aho et al., 1979b] Aho, A. V., Sagiv, Y., and Ullman, J. D. (1979b). Equivalences Among Relational Expressions. SIAM J. Comput., 8(2):218–246.

[Areces et al., 2000] Areces, C., Gennari, R., Heguiabehere, J., and de Rijke, M. (2000). Tree-based Heuristics in Modal Theorem Proving. In Horn, W., editor, Proceedings of ECAI'2000, pages 199–203, Berlin, Germany.

[Arenas et al., 2009] Arenas, M., Gutierrez, C., and Pérez, J. (2009). Foundations of RDF Databases. In Tessaris, S., Franconi, E., Eiter, T., Gutierrez, C., Handschuh, S., Rousset, M.-C., and Schmidt, R. A., editors, Reasoning Web. Semantic Technologies for Information Systems, 5th International Summer School 2009, Brixen-Bressanone, Italy, August 30 - September 4, 2009, Tutorial Lectures, Lecture Notes in Computer Science, pages 158–204. Springer.

[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.

[Auer et al., 2007] Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., and Ives, Z. G. (2007). DBpedia: A Nucleus for a Web of Open Data. In The Semantic Web, 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC 2007 + ASWC 2007, Busan, Korea, November 11-15, 2007., ISWC '07, pages 722–735.

[Baader and Tobies, 2001] Baader, F. and Tobies, S. (2001). The Inverse Method Implements the Automata Approach for Modal Satisfiability. In Gorè, R., Leitsch, A., and Nipkow, T., editors, Automated Reasoning, volume 2083 of Lecture Notes in Computer Science, pages 92–106. Springer Berlin Heidelberg.

[Baget, 2005] Baget, J.-F. (2005). RDF Entailment As a Graph Homomorphism. In Proceedings of the 4th International Conference on The Semantic Web, ISWC'05, pages 82–96, Berlin, Heidelberg. Springer-Verlag.

[Benedikt and Cheney, 2010] Benedikt, M. and Cheney, J. (2010). Destabilizers and Independence of XML Updates. Proc. VLDB Endow., 3(1-2):906–917.

[Berners-Lee and Fischetti, 1999] Berners-Lee, T. and Fischetti, M. (1999). Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web by Its Inventor. Harper San Francisco, 1st edition.

[Berners-Lee et al., 2001] Berners-Lee, T., Hendler, J., and Lassila, O. (2001). The Semantic Web. Scientific American, 284(5):34–43.

[Bizer et al., 2009] Bizer, C., Heath, T., and Berners-Lee, T. (2009). Linked Data - The Story So Far. Int. J. Semantic Web Inf. Syst., 5(3):1–22.

[Bizer and Schultz, 2009] Bizer, C. and Schultz, A. (2009). The Berlin SPARQL Benchmark. International Journal on Semantic Web and Information Systems (IJSWIS), 5(2):1–24.

[Blackburn et al., 2001] Blackburn, P., de Rijke, M., and Venema, Y. (2001). Modal Logic. Cambridge University Press, New York, NY, USA.

[Blackburn et al., 2006] Blackburn, P., van Benthem, J. F. A. K., and Wolter, F. (2006). Handbook of Modal Logic, Volume 3 (Studies in Logic and Practical Reasoning). Elsevier Science Inc., New York, NY, USA.

[Blakeley et al., 1989] Blakeley, J. A., Coburn, N., and Larson, P.-V. (1989). Updating Derived Relations: Detecting Irrelevant and Autonomously Computable Updates. ACM Trans. Database Syst., 14(3):369–400.

[Bonatti et al., 2006] Bonatti, P. A., Lutz, C., Murano, A., and Vardi, M. Y. (2006). The Complexity of Enriched μ-calculi. In Proceedings of the 33rd International Conference on Automata, Languages and Programming - Volume Part II, ICALP'06, pages 540–551, Berlin, Heidelberg. Springer-Verlag.

[Bonatti and Peron, 2004] Bonatti, P. A. and Peron, A. (2004). On the Undecidability of Logics with Converse, Nominals, Recursion and Counting. Artif. Intell., 158(1):75–96.

[Bryant, 1986] Bryant, R. E. (1986). Graph-Based Algorithms for Boolean Function Manipulation. IEEE Trans. Comput., 35(8):677–691.

[Bull and Segerberg, 1984] Bull, R. and Segerberg, K. (1984). Basic Modal Logic. In Gabbay, D. and Guenthner, F., editors, Handbook of Philosophical Logic, volume 165 of Synthese Library, pages 1–88. Springer Netherlands.

[Calvanese et al., 2008] Calvanese, D., De Giacomo, G., and Lenzerini, M. (2008). Conjunctive query containment and answering under description logic constraints. ACM Trans. Comput. Log., 9(3).

[Calvanese et al., 2000] Calvanese, D., De Giacomo, G., Lenzerini, M., and Vardi, M. Y. (2000). Containment of Conjunctive Regular Path Queries with Inverse. In KR 2000, Principles of Knowledge Representation and Reasoning Proceedings of the Seventh International Conference, Breckenridge, Colorado, USA, April 11-15, 2000., KR '00, pages 176–185. Morgan Kaufmann.

[Calvanese et al., 1998] Calvanese, Diego andDe Giacomo, G., Lenzerini, M., Nardi, D., and Rosati, R. (1998). Description Logic Framework for Information Integration. In Proceedings of the Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR'98), Trento, Italy, June 2-5, 1998., KR '98, pages 2–13.

[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 9th Annual ACM Symposium on Theory of Computing, May 4-6, 1977, Boulder, Colorado, USA, STOC9 '77, pages 77–90.

[Chaudhuri et al., 1995] Chaudhuri, S., Krishnamurthy, R., Potamianos, S., and Shim, K. (1995). Optimizing Queries with Materialized Views. In Proceedings of the Eleventh International Conference on Data Engineering, March 6-10, 1995, Taipei, Taiwan, ICDE '95, pages 190–200.

[Chaudhuri and Vardi, 1992] Chaudhuri, S. and Vardi, M. Y. (1992). On the Equivalence of Recursive and Nonrecursive Datalog Programs. In Proceedings of the Eleventh ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 2-4, 1992, San Diego, California, USA, PODS '92, pages 55–66.

[Chebotko, 2008] Chebotko, A. (2008). Querying and Managing Semantic Web Data and Scientific Workflow Provenance Using Relational Databases. PhD thesis, Department of Computer Science, Wayne State University.

[Chebotko et al., 2009] Chebotko, A., Lu, S., and Fotouhi, F. (2009). Semantics preserving SPARQL-to-SQL translation. Data Knowl. Eng., 68(10):973–1000.

[Chekol, 2012] Chekol, M. W. (2012). Static Analysis of Semantic Web Queries. PhD thesis, University of Grenoble.

[Chekol et al., 2011] Chekol, M. W., Euzenat, J., Genevès, P., and Layaïda, N. (2011). PSPARQL query containment. In Database Programming Languages - DBPL 201, 13th International Symposium, Seattle, Washington, USA, August 29, 2011. Proceedings, DBPL '11.

[Chekol et al., 2012a] Chekol, M. W., Euzenat, J., Genevès, P., and Layaïda, N. (2012a). SPARQL Query Containment under RDFS Entailment Regime. In Automated Reasoning - 6th International Joint Conference, IJCAR 2012, Manchester, UK, June 26-29, 2012. Proceedings, IJCAR '12, pages 134–148.

[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, July 22-26, 2012, Toronto, Ontario, Canada., aaai '12.

[Chekol et al., 2013] Chekol, M. W., Euzenat, J., Genevès, P., and Layaïda, N. (2013). Evaluating and Benchmarking SPARQL Query Containment Solvers. In The Semantic Web - ISWC 2013 - 12th International Semantic Web Conference, Sydney, NSW, Australia, October 21-25, 2013, Proceedings, Part II, SEMWEB '13, pages 408–423.

[Clarke et al., 1996] Clarke, E. M., Grumberg, O., and Long, D. E. (1996). Model checking. In Proceedings of the NATO Advanced Study Institute on Deductive Program Design, Marktoberdorf, Germany, NATO '96, pages 305–349.

[Cyganiak, 2005] Cyganiak, R. (2005). A relational algebra for SPARQL.

[Deutsch and Tannen, 2001] Deutsch, A. and Tannen, V. (2001). Optimization Properties for Classes of Conjunctive Regular Path Queries. In Database Programming Languages, 8th International Workshop, DBPL 2001, Frascati, Italy, September 8-10, 2001, Revised Papers, DBLP '01, pages 21–39.

[Domingue et al., 2011] Domingue, J., Fensel, D., and Hendler, J. A., editors (2011). Handbook of Semantic Web Technologies. Springer.

[Elliott et al., 2009] Elliott, B., Cheng, E., Thomas-Ogbuji, C., and Özsoyoglu, Z. M. (2009). A complete translation from SPARQL into efficient SQL. In International Database Engineering and Applications Symposium (IDEAS 2009), September 16-18, 2009, Cetraro, Calabria, Italy, IDEAS '09, pages 31–42.

[Fischer and Ladner, 1979] Fischer, M. J. and Ladner, R. E. (1979). Propositional Dynamic Logic of Regular Programs. J. Comput. Syst. Sci., 18(2):194–211.

[Fitting and Mendelsohn, 1988] Fitting, M. and Mendelsohn, R. L. (1988). First-order Modal Logic, volume 277 of Synthese Library. Kluwer Academic Publishers, Norwell, MA, USA.

[Florescu et al., 1998] Florescu, D., Levy, A. Y., and Suciu, D. (1998). Query Containment for Conjunctive Queries with Regular Expressions. In Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 1-3, 1998, Seattle, Washington, USA, PODS '98, pages 139–148.

[Garlik and Seaborne, 2013] Garlik, S. H. and Seaborne, A. (2013). SPARQL 1.1 Query Language. World Wide Web Consortium, Recommendation REC-sparql11-query-20130321.

[Genevès, 2008] Genevès, P. (2008). Logics for XML. CoRR, abs/0810.4460.

[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 et al., 2007] Genevès, P., Layaïda, N., and Schmitt, A. (2007). Efficient static analysis of XML paths and types. In Proceedings of the ACM SIGPLAN 2007 Conference on Programming Language Design and Implementation, San Diego, California, USA, June 10-13, 2007, PLDI '07, pages 342–351.

[Genevès et al., 2015] Genevès, P., Layaïda, N., Schmitt, A., and Gesbert, N. (2015). Efficiently Deciding μ-Calculus with Converse over Finite Trees. ACM Trans. Comput. Log., 16(2):16.

[Genevès and Schmitt, 2015] Genevès, P. and Schmitt, A. (2015). Expressive Logical Combinators for Free. In Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015, IJCAI '15, pages 311–317. Springer.

[Gerber et al., 2008] Gerber, A., van der Merwe, A., and Barnard, A. (2008). A Functional Semantic Web Architecture. In The Semantic Web: Research and Applications, 5th European Semantic Web Conference, ESWC 2008, Tenerife, Canary Islands, Spain, June 1-5, 2008, Proceedings, volume 5021 of ESWS '08, pages 273–287. Springer.

[Goldstein and Larson, 2001] Goldstein, J. and Larson, P. (2001). Optimizing Queries Using Materialized Views: A practical, scalable solution. In Proceedings of the 2001 ACM SIGMOD international conference on Management of data, Santa Barbara, CA, USA, May 21-24, 2001, SIGMOD '01, pages 331–342.

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

[Guido, 2015] Guido, N. (2015). On the Static Analysis for SPARQL Queries Using Modal Logic. In Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015, IJCAI '15, pages 4367–4368.

[Guido et al., 2015] Guido, N., Genevès, P., Layaïda, N., and Roisin, C. (2015). On Query-Update Independence for SPARQL. In Proceedings of the 24th ACM International on Conference on Information and Knowledge Management, CIKM 2015, Melbourne, VIC, Australia, October 19 - 23, 2015, CIKM '15, pages 1675–1678.

[Gutierrez et al., 2011] Gutierrez, C., Hurtado, C. A., Mendelzon, A. O., and Pérez, J. (2011). Foundations of Semantic Web databases. In Proceedings of the Twenty-Third Symposium on Principles of Database Systems (PODS), June 14-16, 2004, Paris, France, volume 77 of PODS '11, pages 520–541.

[Gutierrez et al., 2012] Gutierrez, J., Klaedtke, F., and Lange, M. (2012). The μ-Calculus Alternation Hierarchy Collapses over Structures with Restricted Connectivity. In Faella, M. and Murano, A., editors, Proceedings Third International Symposium on Games, Automata, Logics and Formal Verification, Napoli, Italy, September 6-8, 2012, volume 96 of Electronic Proceedings in Theoretical Computer Science, pages 113–126. Open Publishing Association.

[Halpern and Moses, 1992] Halpern, J. Y. and Moses, Y. (1992). A Guide to Completeness and Complexity for Modal Logics of Knowledge and Belief. Artificial Intelligence, 54(2):319–379.

[Hayes, 2004] Hayes, P. (2004). RDF Semantics. World Wide Web Consortium, Recommendation REC-rdf-mt-20040210.

[Heflin, 2004] Heflin, J. (2004). Web Ontology Language (OWL): Use Cases and Requirements. World Wide Web Consortium, Recommendation REC-webont-req-20040210.

[Hintikka, 1957] Hintikka, J. (1957). Quantifiers in Deontic Logic, volume 24. Societas Scientiarum Fennica, Commentationes humanarum litterarum.

[Horrocks et al., 2005] Horrocks, I., Parsia, B., Patel-Schneider, P. F., and Hendler, J. A. (2005). Semantic Web Architecture: Stack or Two Towers? In Principles and Practice of Semantic Web Reasoning, Third International Workshop, PPSWR 2005, Dagstuhl Castle, Germany, September 11-16, 2005, Proceedings, PPSWR '05, pages 37–41.

[Hustadt and Schmidt, 1998] Hustadt, U. and Schmidt, R. A. (1998). Issues of Decidability for Description Logics in the Framework of Resolution. In Automated Deduction in Classical and Non-Classical Logics, Selected Papers, FTP '98, pages 191–205.

[Hustadt and Schmidt, 2000] Hustadt, U. and Schmidt, R. A. (2000). MSPASS: Modal Reasoning by Translation and First-Order Resolution. In Automated Reasoning with Analytic Tableaux and Related Methods, International Conference, TABLEAUX 2000, St Andrews, Scotland, UK, July 3-7, 2000, Proceedings, TABLEAUX '00, pages 67–71.

[Ioannidis and Ramakrishnan, 1995] Ioannidis, Y. E. and Ramakrishnan, R. (1995). Containment of Conjunctive Queries: Beyond Relations as Sets. ACM Trans. Database Syst., 20(3):288–324.

[Jónsson and Tarski, 1951] Jónsson, B. and Tarski, A. (1951). Boolean algebras with operators I. American Journal of mathematics., 73:891–939.

[Jónsson and Tarski, 1952] Jónsson, B. and Tarski, A. (1952). Boolean algebras with operators II, volume 74. American journal of mathematics.

[Junedi et al., 2012] Junedi, M., Genevès, P., and Layaïda, N. (2012). XML query-update independence analysis revisited. In ACM Symposium on Document Engineering, DocEng '12, Paris, France, September 4-7, 2012, DOCENG '12, pages 95–98.

[Klug, 1988] Klug, A. C. (1988). On conjunctive queries containing inequalities. J. ACM, 35(1):146–160.

[Kneale and Kneale, 1962] Kneale, W. and Kneale, M. (1962). The Development of Logic. Clarendon Press: Oxford University Press.

[Kozen, 1983] Kozen, D. (1983). Results on the Propositional mu-Calculus. Theor. Comput. Sci., 27:333–354.

[Kripke, 1959] Kripke, S. A. (1959). A Completeness Theorem in Modal Logic. J. Symb. Log., 24(1):1–14.

[Kripke, 1963] Kripke, S. A. (1963). Semantic Analysis of Modal Logic I. Zeitschrift für Mathematische Logik und Grundlagen der Mathematik, 9(56):67–96.

[Ladner, 1977] Ladner, R. E. (1977). The Computational Complexity of Provability in Systems of Modal Propositional Logic. SIAM J. Comput., 6(3):467–480.

[Lassila and Swick, 1999] Lassila, O. and Swick, R. (1999). Resource Description Framework (RDF) Model and Syntax. World Wide Web Consortium, Recommendation REC-rdf-syntax-19990222.

[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-SIGART Symposium on Principles of Database Systems, PODS 2012, Scottsdale, AZ, USA, May 20-24, 2012, PODS '12, pages 89–100.

[Levy and Rousset, 1996] Levy, A. Y. and Rousset, M. (1996). Verification of Knowledge Bases Based on Containment Checking. In Proceedings of the Thirteenth National Conference on Artificial Intelligence and Eighth Innovative Applications of Artificial Intelligence Conference, AAAI 96, IAAI 96, Portland, Oregon, August 4-8, 1996, Volume 1., AAAI '96, pages 585–591.

[Levy and Sagiv, 1993] Levy, A. Y. and Sagiv, Y. (1993). Queries Independent of Updates. In 19th International Conference on Very Large Data Bases, August 24-27, 1993, Dublin, Ireland, Proceedings., VLDB '93, pages 171–181.

[Levy and Sagiv, 1994] Levy, A. Y. and Sagiv, Y. (1994). Semantic Query Optimization in Datalog Programs. In ILPS 1994, Workshop 2: Constraints and Databases, Ithaca, New York, USA, November 17, 1994, SLP '94.

[Lewis, 1912] Lewis, C. I. (1912). Implication and the Algebra of Logic, volume 21. Oxford University Press.

[Lewis, 1914] Lewis, C. I. (1914). The Calculus of Strict Implication, volume 23. Oxford University Press.

[Łukasiewiez, 1953] Łukasiewiez, J. (1953). A System of Modal Logic, volume 1. The Journal of Computing Systems.

[Martin et al., 2010] Martin, M., Unbehauen, J., and Auer, S. (2010). Improving the Performance of Semantic Web Applications with SPARQL Query Caching. In The Semantic Web: Research and Applications, 7th Extended Semantic Web Conference, ESWC 2010, Heraklion, Crete, Greece, May 30 - June 3, 2010, Proceedings, Part II, ESWS '10, pages 304–318.

[Mateescu et al., 2009] Mateescu, R., Meriot, S., and Rampacek, S. (2009). Extending SPARQL with Temporal Logic.

[Millstein et al., 2003] Millstein, T. D., Halevy, A. Y., and Friedman, M. (2003). Query containment for data integration systems. J. Comput. Syst. Sci., 66(1):20–39.

[Minsky, 1974] Minsky, M. (1974). A Framework for Representing Knowledge. Technical Report 306, MIT-AI Laboratory.

[Nielson et al., 1999] Nielson, F., Nielson, H. R., and Hankin, C. (1999). Principles of Program Analysis. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

[Niwiǹski and Walukiewicz, 1996] Niwiǹski, D. and Walukiewicz, I. (1996). Games for the μ-Calculus. Theoretical Computer Science, 163(1&2):99–116.

[Pan et al., 2006] Pan, G., Sattler, U., and Vardi, M. Y. (2006). Bdd-based decision procedures for the modal logic k. Journal of Applied Non-Classical Logics, 16:169–208.

[Papadimitriou, 1994] Papadimitriou, C. H. (1994). Computational complexity. Addison-Wesley.

[Patel-Schneider and Horrocks, 1999] Patel-Schneider, P. F. and Horrocks, I. (1999). DLP and FaCT. In Automated Reasoning with Analytic Tableaux and Related Methods, International Conference, TABLEAUX '99, Saratoga Springs, NY, USA, June 7-11, 1999, Proceedings, TABLEAUX '99, pages 19–23.

[Pérez et al., 2006] Pérez, J., Arenas, M., and Gutierrez, C. (2006). Semantics and Complexity of SPARQL. In The Semantic Web - ISWC 2006, 5th International Semantic Web Conference, ISWC 2006, Athens, GA, USA, November 5-9, 2006, Proceedings, DBLP:conf/semweb/2006, pages 30–43.

[Pérez et al., 2006] Pérez, J., Arenas, M., and Gutierrez, C. (2006). Semantics of SPARQL. Technical report, Universidad de Chile TR/DCC-2006-17.

[Pérez et al., 2009] Pérez, J., Arenas, M., and Gutierrez, C. (2009). Semantics and complexity of SPARQL. ACM Trans. Database Syst., 34(3).

[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, Snowbird, UT, USA, June 22-27, 2014, PODS '14, pages 39–50.

[Polleres, 2007] Polleres, A. (2007). From SPARQL to rules (and back). In Proceedings of the 16th International Conference on World Wide Web, WWW 2007, Banff, Alberta, Canada, May 8-12, 2007, WWW '07, pages 787–796.

[Prior, 1957] Prior, A. N. (1957). Time and Modality. John Locke Lectures. Clarendon Press.

[Prud'hommeaux and Seaborne, 2008] Prud'hommeaux, E. and Seaborne, A. (2008). SPARQL Query Language for RDF. World Wide Web Consortium, Recommendation REC-rdf-sparql-query-20080115.

[Ramakrishnan et al., 1989] Ramakrishnan, R., Sagiv, Y., Ullman, J. D., and Vardi, M. Y. (1989). Proof-Tree Transformation Theorems and Their Applications. In Proceedings of the Eighth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, March 29-31, 1989, Philadelphia, Pennsylvania, USA, PODS '89, pages 172–181.

[Sagiv and Yannakakis, 1980] Sagiv, Y. and Yannakakis, M. (1980). Equivalences Among Relational Expressions with the Union and Difference Operators. J. ACM, 27(4):633–655.

[Savitch, 1969] Savitch, W. J. (1969). Deterministic Simulation of Non-Deterministic Turing Machines (Detailed Abstract). In Proceedings of the 1st Annual ACM Symposium on Theory of Computing, May 5-7, 1969, Marina del Rey, CA, USA, STOC '69, pages 247–248.

[Schenk et al., 2010] Schenk, S., Gearon, P., and Passant, A. (2010). SPARQL 1.1 update. W3C Recommendation.

[Schmidt et al., 2010] Schmidt, M., Meier, M., and Lausen, G. (2010). Foundations of SPARQL query optimization. In Database Theory - ICDT 2010, 13th International Conference, Lausanne, Switzerland, March 23-25, 2010, Proceedings, ICDT '10, pages 4–33.

[Schmidt, 1999] Schmidt, R. A. (1999). Decidability by resolution for propositional modal logics. J. Autom. Reasoning, 22(4):379–396.

[Serfiotis et al., 2005] Serfiotis, G., Koffina, I., Christophides, V., and Tannen, V. (2005). Containment and minimization of RDF/S query patterns. In The Semantic Web - ISWC 2005, 4th International Semantic Web Conference, ISWC 2005, Galway, Ireland, November 6-10, 2005, Proceedings, semweb '05, pages 607–623.

[Shadbolt et al., 2006] Shadbolt, N., Berners-Lee, T., and Hall, W. (2006). The Semantic Web Revisited. IEEE Intelligent Systems, 21(3):96–101.

[Somenzi, 1998] Somenzi, F. (1998). CUDD: CU Decision Diagram Package.

[Stocker et al., 2008] Stocker, M., Seaborne, A., Bernstein, A., Kiefer, C., and Reynolds, D. (2008). SPARQL basic graph pattern optimization using selectivity estimation. In Proceedings of the 17th International Conference on World Wide Web, WWW 2008, Beijing, China, April 21-25, 2008, WWW '08, pages 595–604.

[Stockmeyer, 1976] Stockmeyer, L. J. (1976). The polynomial-time hierarchy . Theoretical Computer Science, 3(1):1 – 22.

[Tanabe et al., 2008] Tanabe, Y., Takahashi, K., and Hagiya, M. (2008). A decision procedure for alternation-free modal μ- calculi. In Advances in Modal Logic 7, papers from the seventh conference on "Advances in Modal Logic," held in Nancy, France, 9-12 September 2008, AIML '08, pages 341–362.

[Tanabe et al., 2005] Tanabe, Y., Takahashi, K., Yamamoto, M., Tozawa, A., and Hagiya, M. (2005). A Decision Procedure for the Alternation-Free Two-Way Modal μ- Calculus. In Automated Reasoning with Analytic Tableaux and Related Methods, International Conference, TABLEAUX 2005, Koblenz, Germany, September 14-17, 2005, Proceedings, TABLEAUX '05, pages 277–291.

[Trahktenbrot, 1950] Trahktenbrot, B. A. (1950). Impossibility of an algorithm for the decision problem in finite classes. Dokl. Akad. Nauk. SSSR, 70(4):569–572.

[Ullman, 1988] Ullman, J. D. (1988). Principles of Database and Knowledge-Base Systems, Volume I. Computer Science Press.

[Ullman, 1989] Ullman, J. D. (1989). Principles of Database and Knowledge-Base Systems, Volume II. Computer Science Press.

[Ullman, 1997] Ullman, J. D. (1997). Information Integration Using Logical Views. In Database Theory - ICDT '97, 6th International Conference, Delphi, Greece, January 8-10, 1997, Proceedings, ICDT '97, pages 19–40.

[van der Meyden, 1992] van der Meyden, R. (1992). The Complexity of Querying Indefinite Information: Defined Relations, Recursion and Linear Order. PhD thesis, New Brunswick, NJ, USA. UMI Order No. GAX93-21333.

[Vardi, 1982] Vardi, M. Y. (1982). The Complexity of Relational Query Languages (Extended Abstract). In Proceedings of the 14th Annual ACM Symposium on Theory of Computing, May 5-7, 1982, San Francisco, California, USA, STOC '82, pages 137–146.

[Vardi, 1996] Vardi, M. Y. (1996). Why is modal logic so robustly decidable? In Descriptive Complexity and Finite Models, pages 149–184.

[von Wright, 1951] von Wright, G. H. (1951). Deontic logic. Mind, 60(237):1–15.

[Weidenbach et al., 2009] Weidenbach, C., Dimova, D., Fietzke, A., Kumar, R., Suda, M., and Wischnewski, P. (2009). SPASS Version 3.5. In Automated Deduction - CADE-22, 22nd International Conference on Automated Deduction, Montreal, Canada, August 2-7, 2009. Proceedings, CADE '09, pages 140–145.

[Whaley, 2007] Whaley, J. (2003-2007). JavaBDD. `http://javabdd.sourceforge.net/`.

[Yang and Larson, 1987] Yang, H. Z. and Larson, P. (1987). Query Transformation for PSJ-Queries. In VLDB'87, Proceedings of 13th International Conference on Very Large Data Bases, September 1-4, 1987, Brighton, England, VLDB '87, pages 245–254.

A

Appendix A

A.1 Conjunctive Queries (CQs)

Conjunctive queries are the most common SQL queries used in the commercial database management systems, and have been therefore extensively studied [Chandra and Merlin, 1977, Aho et al., 1979b, Klug, 1988, Ullman, 1988, Ullman, 1989].

A conjunctive query, under relational algebra theory, is a query that only uses Selection, Projection and Cartesian product operations. Using a deductive databases notation, a conjunctive query is a safe, nonrecursive rule with the predicates of the body defined over EDB (Extensional Databases) predicates [Ullman, 1988, Ullman, 1989].

Conjunctive queries can be represented using different notations that are equivalent. Among these notations, the most common are Datalog rules (deductive databases notation), SQL queries, or Relational Algebra expressions.

In the following we shortly present the logical rule notation of Datalog and we show how to convert a SQL query into its equivalent Datalog rule.

A.1.1 Datalog Notation

To begin, we use the logical rule notation from [Ullman, 1988].

Datalog programs are collections of *rules*, which are Horn clauses or *if-then* expressions.

The rules have the following form:

$$\texttt{Head}: -\texttt{Body}$$

where the symbol $: -$ is read "if". The Head and the Body of a rule are made by *Atoms*. An *Atom* is a predicate applied to arguments. In general, there is one atom, the Head of rule, and there is the logical AND of zero or more atoms, the Body. The atoms in the Body of the rule are called *subgoals*.

Example A.1.1 ([Ullman, 1997]). *The following rule:*

$$\mathtt{p(X,Y)} \ :- \ \mathtt{a(X,Y), a(Y,Z)}$$

is a rule that talks about a *an* EDB *predicate (Extensional DataBase or stored relation), and* p *an* IDB *predicate (Intensional DataBase or predicate whose relation is constructed by rules).*

In this and several other examples, it is useful to think of a *as an "arc" predicate defining a graph, while other predicates define certain structures that might exist in the graph.*

That is, a(X, Y) *means there is an arc from node* X *to node* Y*. In this case, the rule says* p(X, Z) *is* true*, if there are two arcs labelled with* a*, one from node* X *to node* Y *and one from* Y *to* Z*. That is,* p *represents* a*−paths of length 2 in the graph.*

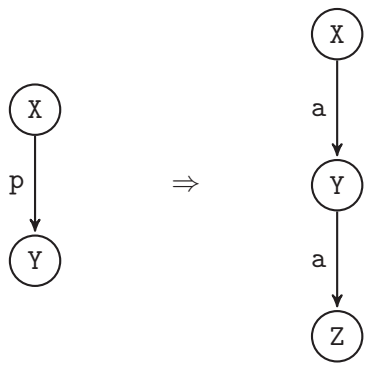


Figure A.1: Graph representation of the rule

A Datalog program is a collection of rules. One IDB predicate is distinguished and represents the result of the program. In the following, we assume that each variable appearing in the head also appears somewhere in the body. This "safety requirement" insures that when we use a rule, we are not left with undefined variables in the head, when we try to infer a fact about the head's predicate. We also assume that atoms consist of a predicate and zero or more arguments. An argument can be either a variable or a constant.

A.1.1.1 Datalog Notation for Conjunctive Queries

A *conjunctive query* (CQ) is a rule with subgoals that are assumed to have EDB predicates. A CQ is applied to the EDB relations by considering all possible substitutions of values for the variables in the body. If a substitution makes all the subgoals true, then the same substitution, applied to the head, is an inferred fact about the head's predicate.

A.1.2 From SQL to Datalog Rule

Algorithm 1. Transform an SQL conjunctive query into its equivalent Datalog program.

Input: An SQL query of the form

SELECT DISTINCT $A_1, \ldots, A_n$

FROM $table_1, \ldots, table_t$

WHERE $\langle condition_1 \rangle$ and ... and $\langle condition_c \rangle$

Output: The same query written as a Datalog rule.

Procedure:

- There will be a different variable for each attribute in the relation schema of all tables in the FROM clause.
- For each of the relation in the FROM clause, add a predicate to the body of the rule, with the same number of attributes and in the same order as in the relational schema.
- For each of the equalities or inequalities in the WHERE clause, add a built-in predicate to the body of the rule that establishes the (in)equality between two variables or one variable and one constant.
- The variables in the head of the rule are those variables that appear in the SELECT clause.

Consider the following example.

Example A.1.2. *Let D be a database with the following scheme:*

stud(StudNo, StudName, StudFacNo, ExamNum)

fac(FacNo, FacName)

The query "Obtain the names of the students of the faculty of Computer Science that have done 5 exams" can be represented as the following SQL query:

SELECT DISTINCT StudName

FROM stud, fac

WHERE stud.StudFacNo = fac.FacNo AND fac.Name = "ComputerScience" AND

stud.ExamNum = 5

Following the previous algorithm, the equivalent Datalog rule is built:

- *There are the following variables:*

  StudNo, StudName, StudFacNo, ExamNum, FacNo, FacName

- *The predicates in the body of the rule are:*

  stud(StudNo, StudName, StudFacNo, ExamNum)

  fac(FacNo, FacName)

- *The following built-in predicates are added*

  StudFacNo = FacNo, FacName = "ComputerScience", ExamNum = 5

- *The variable in the head of the rule is:*

  StudName

Therefore, the **Datalog** *rule that represents this query is:*

```
result(StudName) :- stud(studNo, StudName, StudFacNo, ExamNum),
                    fac(FacNo, FacName),
                    StudFacNo = FacNo,  FacName = "ComputerScience",
                    ExamNum = 5
```

We have the equality FacName = "ComputerScience"*, hence* FacName *can be replaced by* ComputerScience *in the query. Additionally, the variable* StudFacNo *and* FacNo *are in the same equivalence class. Considering* FacNo *as the representative of this class, the query can be rewritten in a compressed form as:*

```
result(StudName) :- stud(studNo, StudName, FacNo, ExamNum),
                    fac(FacNo, "ComputerScience"),
                    ExamNum = 5
```

B

Appendix B

B.1 Benchmark

In the following, we present the benchmark queries. The benchmark is designed for well-designed SPARQL queries. Alternatively, the benchmark is also available online at

http://tyrex.inria.fr/benchmark/index.html.

The benchmark queries use the namespace prefixes shown below:

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX bsbm: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/>
PREFIX rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
```

In the following, we report the queries used in our tests.

$Q1_a$

```
SELECT *

WHERE {
  ?product <rdfs:label> ?label.
  ?product <rdfs:type>  ?type.
}
```

$Q1_b$

```
SELECT *

WHERE {
  ?product <rdfs:label> ?label .
}
```

$Q2_a$

```
SELECT *

WHERE {
  ?product <rdfs:label> ?label .
  ?product <rdf:a> <foaf:ProductType> .
  ?product <bsbm:productFeature> <foaf:ProductFeature1> .
  ?product <bsbm:productFeature> <foaf:ProductFeature2> .
  ?product <bsbm:productPropertyNumeric1> ?value1 .
}
```

$Q2_b$

```
SELECT *

WHERE {
   ?product <rdfs:label> ?label .
  ?product <rdf:a> <foaf:ProductType> .
  ?product <bsbm:productFeature> <foaf:ProductFeature1> .
  ?product <bsbm:productFeature> <foaf:ProductFeature2> .
  ?product <bsbm:productPropertyNumeric1> <foaf:Value1>  .
}
```

$Q2_c$

```
SELECT *

WHERE {
  ?product <rdfs:label> ?label .
  ?product <rdf:a> <foaf:ProductType> .
  ?product <bsbm:productFeature> <foaf:ProductFeature1> .
  ?product <bsbm:productFeature> <foaf:ProductFeature2> .
}
```

$Q3_a$

```
SELECT *

WHERE {
  ?product <rdfs:label> ?label .
  ?product <rdf:a> <foaf:ProductType> .
  ?product <bsbm:productFeature> <foaf:ProductFeature1> .
  OPTIONAL {
    ?product <bsbm:productFeature> <foaf:ProductFeature2> .
    ?product <rdfs:label> ?testVar.
  }
}
```

$Q3_b$

```
SELECT *

WHERE {
  ?product <rdfs:label> ?label .
  ?product <rdf:a> <foaf:ProductType> .

  OPTIONAL {
    ?product <bsbm:productFeature> <foaf:ProductFeature2> .
    ?product <rdfs:label> ?testVar.
  }
}
```

$Q3_c$

```
SELECT *

WHERE {
  ?product <rdfs:label> ?label .
  ?product <rdf:a> <foaf:ProductType> .
  ?product <bsbm:productFeature> <foaf:ProductFeature1> .
  OPTIONAL {
    ?product <bsbm:productFeature> <foaf:ProductFeature2> .
  }
}
```

$Q4_a$

```
SELECT *

WHERE {
  <foaf:ProductXYZ> <rdfs:label> ?label .
  <foaf:ProductXYZ> <rdfs:comment> ?comment .
  OPTIONAL { <foaf:ProductXYZ> <bsbm:productPropertyTextual4> ?propertyTextual4 .}
  OPTIONAL { <foaf:ProductXYZ> <bsbm:productPropertyTextual5> ?propertyTextual5 .}
  OPTIONAL { <foaf:ProductXYZ> <bsbm:productPropertyNumeric4> <foaf:Value1> . }
}
```

$Q4_b$

```
SELECT *

WHERE {
  <foaf:ProductXYZ> <rdfs:label> ?label .
  <foaf:ProductXYZ> <rdfs:comment> ?comment .
  OPTIONAL { <foaf:ProductXYZ> <bsbm:productPropertyTextual4> ?propertyTextual4 .}
  OPTIONAL { <foaf:ProductXYZ> <bsbm:productPropertyTextual5> ?propertyTextual5 .}
  OPTIONAL { <foaf:ProductXYZ> <bsbm:productPropertyNumeric4> ?propertyNumeric6 .}
}
```

$Q5_a$

```
SELECT *

WHERE {
  <foaf:ProductXYZ> <rdfs:label> ?label .
  OPTIONAL {
    ?offer  <bsbm:product>  <foaf:ProductXYZ> .
    ?offer  <bsbm:price> ?price .
  }
  OPTIONAL {
    ?review  <bsbm:reviewFor>  <foaf:ProductXYZ> .
    OPTIONAL { ?review  <bsbm:rating1> ?rating1 . }
    OPTIONAL { ?review  <bsbm:rating2> ?rating2 . }
  }
}
```

$Q5_b$

```
SELECT *

WHERE {
  <foaf:ProductXYZ> <rdfs:label> ?label .
  OPTIONAL {
    ?offer  <bsbm:product>  <foaf:ProductXYZ> .
    ?offer  <bsbm:price> ?price .
  }
  OPTIONAL {
    ?review  <bsbm:reviewFor>  <foaf:ProductXYZ> .
    OPTIONAL { ?review  <bsbm:rating1> ?rating1 . }
 }
}
```

$Q5_c$

```
SELECT *

WHERE {
  <foaf:ProductXYZ> <rdfs:label> ?label .
  OPTIONAL {
    ?review  <bsbm:reviewFor>  <foaf:ProductXYZ> .
    OPTIONAL { ?review  <bsbm:rating1> ?rating1 . }
    OPTIONAL {
      ?review  <bsbm:rating2> ?rating2 .
      OPTIONAL {?offer  <bsbm:price> ?price .}
    }
  }
}
```

C

Résumé étendu

C.1 Motivation et objectifs

L'objectif de la thèse et de définir des techniques d'analyse statique pour requêtes SPARQL sur le Web Sémantique utilisant la logique modale.

C.1.1 Cadre des travaux

Le terme "Web sémantique" se réfère à un Web des données qui peuvent être lisibles et traitées par des machines et pas seulement par les humains. Le terme a été inventé par Tim Berners Lee. En 1999, le *Resource Description Framework* (`RDF`) a été publié en tant que `W3C` Recommandation, conçu comme un modèle de données pour représenter des informations des ressources Web [Lassila and Swick, 1999].

Conjointement avec cette première version, le problème de l'interrogation et de la modification des données `RDF` a été soulevée, et donc la question de savoir comment gérer les données `RDF` a été au centre de la communauté du Web sémantique. Une première réponse a été donnée avec la sortie de `SPARQL` comme `W3C` Recommandation [Prud'hommeaux and Seaborne, 2008]. Avec le temps, `SPARQL` est devenu le langage de requêtes standard pour `RDF`. Depuis lors, la quantité de données `RDF` publiées sur le Web n'a cessé de croître, comme en témoigne la popularité des initiatives comme "Linked Open Data project" [Bizer et al., 2009] et "DBpedia" [Auer et al., 2007].

L'utilisation de `RDF` et `SPARQL` nécessite de revisiter les problèmes classiques des langages relationnels de requêtes, comme `SQL`. Ainsi, plusieurs travaux de recherche étudient les propriétés fondamentales du langage `SPARQL`, tandis que d'autres s'intéressent au développement de nouvelles techniques pour traiter les problèmes classiques, comme l'analyse statique et l'optimisation des requêtes.

L'analyse statique se réfère à l'étude des requêtes sans connaissance *a priori* de l'ensemble de données sur lesquelles elles seront évaluées. Les problèmes dans ce domaine comprennent:

- l'inclusion des requêtes, qui signifie décider si la réponse d'une requête sera toujours contenue dans la réponse d'une autre requête, lorsqu'elle est évaluée sur le même ensemble de données;

- l'équivalence, qui signifie décider si deux requêtes différentes auront toujours les mêmes résultats lorsqu'elles sont évaluées sur le même ensemble de données;
- l'indépendance entre requête et mise à jour, qui signifie détecter lorsque le résultat de la requête ne change pas après une mise à jour de l'ensemble de données.

D'autre part, l'optimisation des requêtes se réfère à l'étude de la façon d'évaluer plus efficacement une requête donnée. Lors de l'évaluation d'une requête écrite dans un langage comme SQL, la requête est d'abord *analysée*, c'est-à-dire transformée en un arbre syntaxique représentant la structure de la requête. L'arbre syntaxique est ensuite transformé en une expression de l'algèbre relationnelle, qui est appelée *plan d'exécution de la requête*. Dans cette algèbre, on a la possibilité d'appliquer de nombreuses opérations algébriques, avec l'objectif de produire le meilleur plan d'exécution [Levy and Sagiv, 1994]. Ceci est un problème qui a été étudié pendant plusieurs décennies, et les techniques d'optimisation basées sur les plans d'exécution pour les requêtes de la logique relationnelle sont très répandues [Levy and Sagiv, 1994].

Dans cette dernière décennie, l'analyse statique et l'optimisation de requêtes SPARQL ont reçu une attention accrue [Serfiotis et al., 2005, Stocker et al., 2008, Schmidt et al., 2010, Letelier et al., 2012, Chekol et al., 2012b, Pichler and Skritek, 2014]. La plupart de ces travaux portent sur le fragment SPARQL contenant les opérateurs UNION et AND. Cependant, SPARQL est capable de travailler avec des informations partielles: puisque les données sur le Web forment un ensemble intrinsèquement incomplet, il est essentiel que les utilisateurs soient en mesure de demander des informations facultatives. Par exemple, si l'on doit demander les noms, numéros de téléphone et adresses e-mail d'un groupe de personnes en utilisant des requêtes conjonctives, et si l'adresse électronique d'une personne est inconnue, alors il n'y aura aucune information concernant cette personne. L'opérateur OPTIONAL de SPARQL permet aux utilisateurs d'obtenir des informations facultatives. Dans l'exemple précédent, si un utilisateur souhaite les noms des personnes et leurs numéros de téléphone, et éventuellement leurs adresses e-mail, alors la réponse à la requête inclura tous les noms et numéros de téléphone connus dans l'ensemble de données, ainsi que les adresses e-mail chaque fois qu'elles sont disponibles.

Malheureusement, quand on va au-delà du fragment UNION et AND de SPARQL, les problèmes deviennent beaucoup plus compliqués [Pérez et al., 2009, Arenas and Pérez, 2011]. L'optionalité est avérée particulièrement complexe à prendre en compte dans l'évaluation des requêtes. Pourtant, son utilisation dans la pratique est importante, ce qui rend son étude essentielle. [Pérez et al., 2009] a montré comment la complexité de ce problème d'évaluation pour SPARQL augmente de façon spectaculaire, à partir de la classe P, pour le fragment le plus élémentaire, jusqu'à PSPACE-complété quand il comprend l'optionalité. La raison de cette accroissement de complexité est liée à l'utilisation sans restriction de variables internes et externes dans les expressions OPTIONAL. Néanmoins, [Pérez et al., 2009] ont également défini un fragment incluant l'opérateur OPTIONAL: la classe de requêtes SPARQL appelée *well-designed* qui fait diminuer la complexité d'évaluation à coNP. Ce fragment fondamental de SPARQL, très utilisé dans la pratique, impose la restriction suivante: si une variable se produit à la fois sur le côté droit de l'expression OPTIONAL et partout ailleurs dans la requête, alors il doit se produire aussi sur le côté gauche de l'expression OPTIONAL.

En ce qui concerne les implémentations disponibles pour résoudre le problème de l'inclusion de requêtes, nous avons aujourd'hui deux solutions.

Dans [Chekol et al., 2012b], les auteurs ont abordé le problème de l'analyse statique de requête `SPARQL` avec les opérateurs AND et UNION. Ils ont proposé une approche basée sur la logique, très facilement extensible en présence de contraintes sur les graphes `RDF`.

Dans [Letelier et al., 2012], les auteurs ont abordé le problème de l'analyse statique de requêtes `SPARQL` *well-designed*, mettant principalement l'accent sur le problème d'inclusion et de l'équivalence des requêtes. Pour étudier ces problèmes, les auteurs introduisent une nouvelle représentation des requêtes `SPARQL`, appelée *pattern tree*, qui peut être considérée comme des plans d'exécution des requêtes en présence de l'opérateur `OPTIONAL`. Ils proposent une implémentation *ad hoc* pour résoudre le problème de l'inclusion, difficilement extensible en présence de contraintes sur les graphes.

C.1.2 Démarche

Dans cette thèse, nous avons poursuivi ces travaux pour obtenir des résultats plus complets sur l'analyse statique. Concernant le problème d'inclusion de requêtes, sur la base des résultats de la complexité théorique de [Letelier et al., 2012], nous introduisons une nouvelle procédure pour tester l'inclusion des requêtes SPARQL "well-designed". À ce jour, l'inclusion de requêtes a été testée à l'aide de différentes techniques: homomorphisme de graphes, bases de données canoniques, techniques de la théorie des automates et réduction au problème de la validité d'une logique. Dans cette thèse, nous utilisons la dernière technique pour tester l'inclusion des requêtes `SPARQL` avec `OPTIONAL` utilisant une logique expressive appelée "logique `K`". En utilisant cette technique, il est possible de résoudre le problème d'inclusion des requêtes pour plusieurs fragments de SPARQL, même en présence de contraintes sur les graphes. Cette extensibilité n'est pas garantie par les autres méthodes.

Concernant le problème de l'indépendance entre requêtes et mise à jour, nous présentons une formulation du problème et un résultat préliminaire. Une requête est indépendante de la mise à jour lorsque l'exécution de la mise à jour ne modifie pas le résultat de la requête. Bien que ce problème ait été intensivement étudié pour des fragments de calcul relationnel, il n'existe pas de travaux pour le langage de requêtes standard pour le web sémantique. Nous proposons une définition de la notion de l'indépendance dans le contexte de `SPARQL` et nous établissons des premières pistes pour son analyse statique dans certaines situations d'inclusion entre une requête et une mise à jour.

C.2 Principales contributions

Dans cette thèse, nous avons effectué une recherche bibliographique dans différents domaines, en présentant les principes du web sémantique (chapitre 2), la technique de l'analyse statique pour le langage d'interrogation de bases de données (chapitre 3) et en présentant un aperçu sur

les logiques modales et leurs problèmes de satisfaisabilité (chapitre 4 et 5). Les contributions de cette thèse (chapitre 6, 7 et 8) sont les suivantes :

- Nous fournissons une procédure de codage pour déterminer l'inclusion de l'optionalité de requêtes `SPARQL` conjonctives. Cette procédure se compose de trois sous-procédures:

Graphe `RDF` G	Requête `SPARQL` q	q_1 est inclus dans q_2
$\downarrow \sigma$	$\downarrow \mathcal{A}$	$\downarrow$
Système de transition $\sigma(G)$	Formula $\mathcal{A}(q)$	$\mathcal{A}(q_1) \rightarrow \mathcal{A}(q_2)$

  1. nous définissons le processus de traduction d'un graphe G dans un système de transition $\sigma(G)$, qui sera le modèle sémantique pour la satisfaisabilité des formules dans la logique;
  2. nous définissons le processus de traduction d'une requête `SPARQL` *well-designed* dans une formule de la logique;
  3. nous montrons comment résoudre le problème de l'inclusion en le réduisant au problème de satisfaisabilité d'une formule modale.

- Nous présentons une mise en œuvre pour résoudre le problème de satisfaisabilité dans la logique modale `K` basée sur la procédure décisionnelle de [Pan et al., 2006]. Nous montrons avec succès que notre codage du problème d'inclusion peut être résolu avec la logique modale en pratique, validant ainsi notre approche.
- Nous montrons comment notre formule modale représentant l'inclusion de requêtes peut être résolue avec des outils existants, profitant de toutes les optimisations de ces outils. Nous nous sommes intéressés à `MSPASS` [Hustadt and Schmidt, 2000] et nous avons obtenu une réduction du temps remarquable (gain de temps d'un facteur 10) par rapport à SPAM [Letelier et al., 2012], la seule solution existante à ce jour pour tester l'inclusion de requêtes *well-designed*.
- Enfin, nous présentons un aperçu préliminaire du problème de l'indépendance entre requêtes et mise à jour dans le contexte de `SPARQL` et nous donnons une formulation précise du probleme.

C.3 Conclusion et perspectives

Avec cette thèse, nous avons ouvert une voie pour résoudre le problème de l'inclusion de requêtes `SPARQL` en présence de l'opérateur OPTIONAL en utilisant la logique modale `K`.

Différentes perspectives pour des recherches futures sont envisagés à partir des travaux de cette thèse. Par exemple, le problème d'inclusion de requêtes `SPARQL` n'a pas été encore traité en présence de contraintes sur les graphes `RDF` pour le fragment de requêtes *well-designed*. Il serait intéressant d'étendre notre approche logique en présence de ces contraintes, ce qui est envisageable à partir de [Chekol, 2012] qui a déjà proposé une approche logique similaire, mais pour le fragment sans l'opérateur OPTIONAL.

Notre travail préliminaire sur le problème de l'indépendance entre requête et modification de graphes a été pionnier dans le contexte `SPARQL` [Guido et al., 2015], mais les techniques et implémentations doivent encore être étudiées pour détecter l'indépendance entre requêtes et mises à jour dans la pratique.