ABSTRACT

Title of dissertation: EXPRESSIVE SYNDICATION ON THE WEB USING A DESCRIPTION LOGIC-BASED APPROACH

Franz Christian Halaschek-Wiener
Doctor of Philosophy, 2007

Dissertation directed by: Professor James Hendler
Department of Computer Science

Syndication on the Web has attracted a great amount of attention in recent years. However, today's state-of-the-art syndication approaches still provide relatively weak expressive power from a modeling perspective and provide very little automated reasoning support. If a more expressive approach with a formal semantics can be provided, many benefits can be achieved, including a rich semantics-based mechanism for expressing subscriptions and published content and automated reasoning for discovering subscription matches not found using traditional syntactic syndication approaches.

In this dissertation, I develop a syndication framework based on the Web Ontology Language (OWL), which is the standardized language for representing the semantics of information on the Web. One of the main advantages of the framework is its support for formal reasoning, as the semantics of subsets of OWL are founded in description logic (a decidable fragment of first-order logic). Therefore, the previously mentioned benefits can be achieved using description logic (DL) reasoning.

However, the main limitation in using OWL as the underlying representation model is related to the overhead of DL reasoning under changing data, which makes the approach impractical for many real-world domains and publication frequencies. Given this, in this dissertation, I develop incremental DL reasoning algorithms for the required reasoning services in the framework. Specifically, I present incremental consistency checking techniques, as well as algorithms to perform more efficient incremental query answering.

Lastly, to demonstrate the practicality of the syndication approach, I have implemented a prototype of the framework and performed extensive empirical evaluations using synthetic datasets, as well as real world data from the financial domain. These results show the effectiveness of the incremental reasoning services and the practicality of the syndication framework in general.

EXPRESSIVE SYNDICATION ON THE WEB USING A DESCRIPTION LOGIC-BASED APPROACH

by

Franz Christian Halaschek-Wiener

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2007

Dedication

To my family, thank you for everything.

ACKNOWLEDGMENTS

I would like to acknowledge the many people that have guided and supported me in my journey. First, James Hendler, who gave me the freedom to pursue my research and the wisdom to stay on track. Without his mentorship this would not be possible. I would also like to thank Jennifer Golbeck, who has gone above and beyond in all cases. Her dedication has pushed me in my research and helped me to attain my goals. Additionally, I would like to thank my other committee members Ashok Agrawala, Dilip Madan, Dana Nau, and Adam Porter. Their insightful comments and feedback have been extremely helpful.

The current and previous team at MINDSWAP has been invaluable in the growth of my research. With the endless collaboration of MINDSWAP members my research has flourished. Thanks to Ron and Amy Alford, Bernardo Cuenca Grau, Mike Grove, Aditya Kalyanpur, Yarden Katz, Daniel Krech, Evren Sirin, David Taowei Wang, and any others I may have forgotten. I would like to particularly thank Vladimir Kolovski, who has not only served as an intellectual peer but as a friend. Through our many long discussions, I have made it to the end. Thank you Vladimir for your hard work and all of your help. Additionally, I would like to specially thank Bijan Parsia. It is only through his honest dissection that my research has come so far. Thank you Bijan for pushing me in the right direction and always supporting me in my work.

Many thanks to the Dow Jones Newswires for their contribution and collaboration.

In particular, I would like to thank Alan Slomowitz, Jeb Dinsay, and Elizabeth Ciliotta.

To my family, words cannot express how grateful I am. My parents, Franz and Janie Halaschek-Wiener, have been endless in their love and unwavering in their support of my education. Thank you to my mother for getting me started and encouraging me throughout all of my academic endeavors. Thank you to my father for all of his wisdom and inspiration through the years. Thank you to my sister, Therie Halaschek-Wiener, for always being there when I have needed support. Without them this would not be possible.

Lastly, I would like to thank my girlfriend, Dacia Ettinger. She has supported me though the entire process and has always been there when I have needed her. Without her love, help, and support, this truly would not have been possible and life would not be complete.

Table of Contents

List of Tables viii

List of Figures x

1 Introduction 1
- 1.1 Contributions 8
- 1.2 Organization 9

2 Foundations 11
- 2.1 Syndication Systems 11
- 2.2 Semantic Web 12
- 2.3 Web Ontology Language 13
- 2.4 Description Logics 15
  - 2.4.1 Syntax and Semantics of $\mathcal{SHOIQ}(\mathcal{D})$ 18
  - 2.4.2 Description Logic Reasoning 22
  - 2.4.3 Tableau Algorithms 28
- 2.5 Belief Base Revision 35

3 Related Work 40
- 3.1 Syndication Systems 40
  - 3.1.1 Syntactic Approaches 40
  - 3.1.2 Semantic Approaches 41
  - 3.1.3 Discussion 44
- 3.2 Stream Processing Engines 45
- 3.3 Revising and Updating Logical Knowledge Bases 48
  - 3.3.1 AGM Belief Revision Theory 48
  - 3.3.2 Belief Base Revision 49
  - 3.3.3 Logical Updates 50
  - 3.3.4 Repairing Description Logic Knowledge Bases 51
- 3.4 Truth Maintenance Systems 53
- 3.5 Incremental Description Logic Reasoning 54

4 Syndication Framework 56
- 4.1 Overview 56
- 4.2 Framework Formalization 57
  - 4.2.1 Publishing 57
  - 4.2.2 Subscribing 58
  - 4.2.3 Matching 59
  - 4.2.4 Discussion 65
- 4.3 Example 66
- 4.4 Summary 69

5 Incremental Consistency Checking 70
5.1 Introduction . . . 70
5.2 ABox Additions . . . 73
5.2.1 Approach Details . . . 76
5.3 ABox Deletions . . . 77
5.3.1 Approach Details . . . 79
5.3.2 Discussion . . . 86
5.4 Incremental Consistency Checking Algorithm . . . 87
5.5 Empirical Results . . . 89
5.6 Discussion . . . 96

6 Incremental Query Answering 99
6.1 Main Observation . . . 100
6.2 Naïve Approach . . . 102
6.2.1 ABox Additions . . . 103
6.2.2 ABox Deletions . . . 105
6.2.3 Example . . . 105
6.2.4 Discussion . . . 107
6.3 Assumptions . . . 108
6.4 Concept Guide . . . 109
6.4.1 Example . . . 111
6.4.2 Approach . . . 112
6.4.3 Discussion . . . 115
6.5 Summary Completion Graph . . . 116
6.5.1 Summary Completion Graph Properties . . . 117
6.5.2 Using the Summary Completion Graph . . . 119
6.6 Supporting Complex Query Patterns . . . 128
6.6.1 Query Impact . . . 130
6.7 Finding Concept Guide Paths . . . 134
6.8 Incremental Query Answering Algorithm . . . 135
6.9 Empirical Results . . . 136
6.10 Discussion . . . 153

7 Maintaining Consistency at the Syndication Broker 155
7.1 Rejection-Based Approach . . . 156
7.2 Belief Base Revision in OWL-DL . . . 156
7.2.1 Overview . . . 157
7.2.2 Kernel Semi-Revision Operator . . . 158
7.2.3 Semi-Revision Algorithm . . . 159
7.2.4 Trust-Based Incision Function . . . 159
7.3 Discussion . . . 164

8 Implementation and Evaluation 166
8.1 System Architecture 167
8.2 Synthetic Datasets 168
8.3 Real-World Financial Dataset 173
8.3.1 News Feed Background and Overview 174
8.3.2 OWL Domain Model 175
8.3.3 Extended System Architecture 177
8.3.4 Real-World Subscriptions 177
8.3.5 Empirical Results 179
8.3.6 Discussion 184

9 Conclusions and Future Work 185
9.1 Conclusions 185
9.2 Open Issues and Future Work 187
9.2.1 Extending the Syndication Framework 188
9.2.2 Enhancing Incremental Reasoning Techniques 190
9.2.3 Evaluating Belief-Base Semi-Revision 192
9.2.4 Information Extraction 193
9.3 Summary 193

A Proofs 195
A.1 Proofs for Chapter 5 195
A.1.1 Lemma 1: Correctness of Modified Tableau Algorithm 195
A.1.2 Theorem 1: Completeness of $\mathcal{SHOIQ}$ Axiom Tracing 195
A.1.3 Theorem 2: Correctness of Algorithm 2 199
A.2 Proofs for Chapter 6 203
A.2.1 Theorem 3: Conditions for $\mathcal{SHI}$ Concept Instantiation 203
A.2.2 Theorem 4: Concept Guide Label Transfer 204
A.2.3 Theorem 5: $\mathcal{SHI}$ Concept Instantiation Overestimate 208
A.2.4 Lemma 2: Tree Containment in Summary Completion Graph . . . 211
A.2.5 Lemma 3: Summary Completion Graph Update 214
A.2.6 Lemma 4: Label Propagation of Summary Completion Graph . . 217
A.2.7 Lemma 5: Concept Guide Paths in Summary Completion Graph . 219
A.2.8 Lemma 6: Dependencies in Summary Completion Graph 222
A.2.9 Lemma 7: Clashes in Summary Completion Graph 224
A.2.10 Theorem 6: Completeness of Summary Completion Graph 228
A.2.11 Theorem 7: $\mathcal{SHI}$ Binding Requirement 229
A.2.12 Theorem 8: Completeness of Query Impact 230
A.2.13 Theorem 9: Correctness of Algorithm 4 231
A.3 Proofs for Chapter 7 232
A.3.1 Lemma 8: $\mathcal{SHOIN}$ Kernel Operator 232

Bibliography 234

List of Tables

1.1 Illustration of Expressivity in OWL-based Syndication. 5

2.1 Mnemonics for Description Logic Expressivity 17

2.2 Correspondence from OWL-DL to Description Logics 18

2.3 $\mathcal{SHOIQ}$ Tableau Expansion Rules 32

4.1 Sample Syndication Broker 67

5.1 SEMINTEC Ontology Overview 71

5.2 Consistency Checking Times for SEMINTEC Ontology 71

5.3 Sample Knowledge Base 74

5.4 Modified $\mathcal{SHOIQ}$ Tableau Expansion Rules for Axiom Tracing 82

5.5 Merge Operation for Axiom Tracing 84

5.6 Prune Operation for Axiom Tracing 85

5.7 Test-Suite Ontology TBox Overview 89

5.8 Test-Suite Ontology ABox Overview 90

5.9 Distribution of Response Times for Updates of Size 50 (time in milliseconds) 96

5.10 Memory Overhead 97

6.1 Query Answering Times for SEMINTEC Ontology 100

6.2 Sample Knowledge Base 105

6.3 Modified Tableau Expansion Rules for the Summary Completion Graph. . 121

6.4 Summary Completion Graph Initial Construction Results 139

6.5 VICODI Candidate Sizes for Different Queries and Update Sizes 141

6.6 Distribution of VICODI Response Times for Updates of Size 50 (time in milliseconds) 143

6.7 SEMINTEC Candidate Sizes for Different Queries and Update Sizes . . . 145

6.8 Distribution of SEMINTEC Response Times for Updates of Size 50 (time in milliseconds) 145

6.9 LUBM Candidate Sizes for Different Queries and Update Sizes 148

6.10 Distribution of LUBM Response Times for Updates of Size 50 (time in milliseconds) 148

6.11 UOB Candidate Sizes for Different Queries and Update Sizes 153

6.12 Distribution of UOB Response Times for Updates of Size 50 (time in milliseconds) 154

8.1 Financial Ontology Overview 176

8.2 Publication Frequency Distribution 180

8.3 Frequency of High Publication Rates in One Second 181

8.4 Average Number of OWL Assertions per Publication 181

List of Figures

1.1 Basic Syndication Architecture 2

4.1 OWL-Based Syndication Architecture 66

5.1 (a) Initial completion graph corresponding to ABox. (b) Complete clash-free completion graph for KB. 75

5.2 Complete clash-free completion graph for KB with ABox addition. 76

5.3 Addition Updates of VICODI datasets 91

5.4 Addition Updates of SEMINTEC datasets 92

5.5 Addition Updates of LUBM datasets 93

5.6 Addition Updates of UOB datasets 94

5.7 Deletion Updates of VICODI datasets 95

5.8 Deletion Updates of SEMINTEC datasets 96

5.9 Deletion Updates of LUBM datasets 97

5.10 Deletion Updates of UOB datasets 98

6.1 (a) Completion Graph 1. (b) Completion Graph 2. 106

6.2 Clash Satisfying Condition 1 107

6.3 Clashes Satisfying Condition 2: a) Clash in completion graph 1 due to negated query concept. b) Clash in completion graph 2 due to ABox addition. 107

6.4 Example Concept Guide 112

6.5 Sample Query Graph 133

6.6 (a) VICODI query 1 - additions (b) VICODI query 1 - deletions 140

6.7 (a) VICODI query 2 - additions (b) VICODI query 2 - deletions (c) VICODI query 3 - additions (d) VICODI query 3 - deletions 142

6.8 (a) SEMINTEC query 2 - additions (b) SEMINTEC query 2 - deletions (c) SEMINTEC query 3 - additions (d) SEMINTEC query 3 - deletions . 144

6.9 (a) LUBM query 1 - additions (b) LUBM query 1 - deletions (c) LUBM query 2 - additions (d) LUBM query 2 - deletions (e) LUBM query 3 - additions (f) LUBM query 3 - deletions 147

6.10 (a) LUBM query 4 - additions (b) LUBM query 4 - deletions (c) LUBM query 5 - additions (d) LUBM query 5 - deletions (e) LUBM query 6 - additions (f) LUBM query 6 - deletions 149

6.11 (a) UOB query 1 - additions (b) UOB query 1 - deletions (c) UOB query 2 - additions (d) UOB query 2 - deletions (e) UOB query 3 - additions (f) UOB query 3 - deletions 151

6.12 (a) UOB query 4 - additions (b) UOB query 4 - deletions (c) UOB query 5 - additions (d) UOB query 5 - deletions (e) LUBM UOB 6 - additions (f) UOB query 6 - deletions 152

8.1 Syndication System Architecture 167

8.2 (a) VICODI subscription 1 - additions (b) VICODI subscription 2 - additions (c) VICODI subscription 3 - additions (d) VICODI subscription 1 - deletions (e) VICODI subscription 2 - deletions (f) VICODI subscription 3 - deletions 169

8.3 (a) SEMINTEC subscription 1 - additions (b) SEMINTEC subscription 2 - additions (c) SEMINTEC subscription 1 - deletions (d) SEMINTEC subscription 2 - deletions 170

8.4 (a) LUBM subscription 1 - additions (b) LUBM subscription 2 - additions (c) LUBM subscription 3 - additions (d) LUBM subscription 1 - deletions (e) LUBM subscription 2 - deletions (f) LUBM subscription 3 - deletions 171

8.5 (a) LUBM subscription 4 - additions (b) LUBM subscription 5 - additions (c) LUBM subscription 6 - additions (d) LUBM subscription 4 - deletions (e) LUBM subscription 5 - deletions (f) LUBM subscription 6 - deletions 171

8.6 (a) UOB subscription 1 - additions (b) UOB subscription 2 - additions (c) UOB subscription 3 - additions (d) UOB subscription 1 - deletions (e) UOB subscription 2 - deletions (f) UOB subscription 3 - deletions 172

8.7 (a) UOB subscription 4 - additions (b) UOB subscription 5 - additions (c) UOB subscription 6 - additions (d) UOB subscription 4 - deletions (e) LUBM UOB 5 - deletions (f) UOB subscription 6 - deletions 173

8.8 Metadata Converter Architecture 178

8.9 Finance Subscription 1 Average Matching Time 182

8.10 Finance Subscription 2 Average Matching Time 182

8.11 Finance Subscription 3 Average Matching Time 183

8.12 Finance Subscription 4 Average Matching Time 183

Chapter 1

Introduction

Web-based syndication systems have attracted a great amount of attention in recent years as the amount of streaming content on the Web has increased at dramatic rates. In typical syndication frameworks (depicted in Figure 1.1), users register their subscription requests with syndication brokers; similarly, content publishers register their feeds with syndication brokers, and it is then the broker's task to match newly published information with registered subscriptions. As technologies have emerged, there has been a transition to more expressive syndication approaches; that is publishers (and subscribers) are provided with more expressive means for describing their published content (respectively interests), allowing more accurate dissemination. This has been enabled by the maturation of technologies for sharing information on the Web and the standardization of representation languages for Web content. In particular, through the years there has been a transition from keyword based approaches to attribute-value pairs and more recently to XML. Given the lack of expressivity of XML (and XML Schema) as a knowledge modeling language, there has been interest in using the Resource Description Framework (RDF) [100] and its accompanying schema language, RDF Schema, for syndication purposes. RDF has even been adopted as the standard representation format of RSS 1.0[1].

Today's syndication approaches still provide relatively weak expressive power from a modeling perspective (i.e., XML and RDF are comparatively inexpressive languages)

[1]RSS 1.0 Specification: http://web.resource.org/rss/1.0/spec

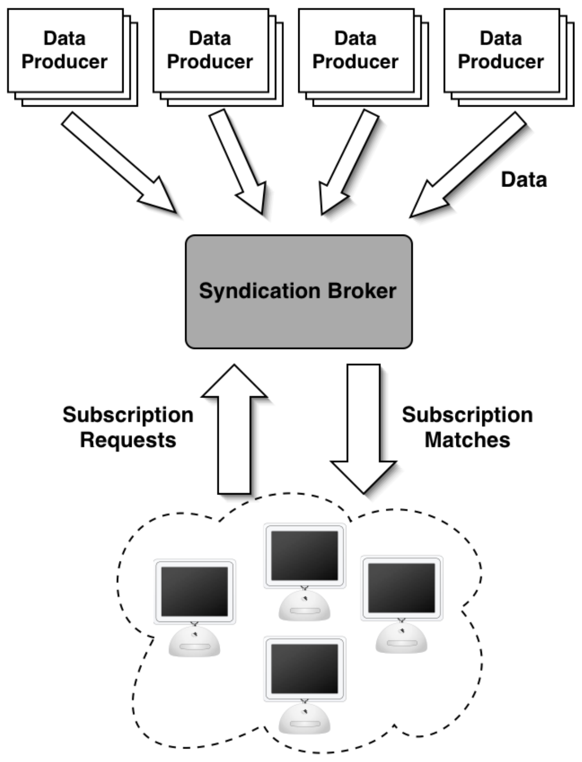


Figure 1.1: Basic Syndication Architecture

and provide very little automated reasoning support. However, if a more expressive syndication approach with a formal semantics can be provided, many benefits can be achieved; these include a rich semantics-based mechanism for expressing subscriptions and published content allowing increased selectivity and finer control for filtering, and automated reasoning for discovering subscription matches not found using traditional syntactic syndication approaches [143].

This motivation aligns with efforts over the past decade to standardize languages for formally representing the semantics of information on the Web; this work has been driven by both the academic and industrial research communities, as well as through initiatives within standardization organizations, such as the World Wide Web Consortium's (W3C) Semantic Web Activity Initiative [2]. The leading formal representation language that has resulted from this work is the W3C approved Web Ontology Language, (OWL) [118]. One of the main benefits of OWL is the support for formal reasoning in its sub-languages OWL Lite and OWL DL, whose semantics are firmly founded in description logic (a decidable fragment of First Order Logic). Therefore, description logic (DL) reasoning services can be utilized to perform various inferences over OWL ontologies which are expressed in these sub-languages. An additional benefit of OWL is its native Web embedding, in that all objects in OWL ontologies are referenced using Universal Resource Identifiers (URIs) and that it has an XML encoding (i.e., XML/RDF). Given this, OWL is a natural fit for the next generation of syndication frameworks, as it can clearly provide a much more expressive syndication approach that is fitting for the existing Web architecture.

To demonstrate how an OWL-based syndication framework might work, as well as the advantages it will provide, consider the following example: suppose we are disseminating news information in the financial domain. Also suppose that a stock trader is interested in articles that *could* discuss companies whose stocks are likely to become volatile; specifically, let us assume that the trader is interested in any *RiskyCompany* which the trader defines to be a *company* that has a *product* which *causes* an *infection* or *allergic reaction*.

Using an XML-based approach syndication brokers can provide an XML schema that contains an element *RiskyCompany* and such companies can be declared to be this type of element. A limitation of such an approach is that companies have to be explicitly declared to be a *RiskyCompany* (i.e., we only obtain explicit, syntactic matches). If we consider an RDF-based approach, then the syndication broker can model the financial domain using RDF Schema. Therefore, additional matches can be obtained as one can logically infer that a company is a *RiskyCompany*. For example, if the domain of a property *hasProductWithAdverseEffect* is declared to be of type *RiskyCompany* and we are given that *BauschAndLomb hasProductWithAdverseEffect Renu*, then we would have a (inferred) match for the subscription; such logical inference (although simplistic) is not possible with an XML-based approach. However, in an RDF based approach, more complex logical definitions (and therefore finer-grained control) of *RiskyCompany* are not expressible.

If we now consider an OWL-based approach, such functionality is clearly provided. For example, the knowledge broker can define a *RiskyCompany* as an OWL class whose necessary and sufficient conditions for inclusion are that it be a *company* that has some product which is an *AdverseEffectProduct*; similarly, an *AdverseEffectProduct* can be defined to be any *product* that causes some *infection* or *allergic reaction*. Using an OWL approach, this can easily be represented by the OWL descriptions in Table 1.1[2]. A equivalent concept can additionally be represented as the following DL concept:

$$Company \sqcap \exists hasProduct.(Product \sqcap \exists causes.(Infection \sqcup AllergicReaction))$$

Given this domain model, if we assume that it is previously known that *BauschAndLomb* is a *company* that *has product Renu*, which is known to be a *Product*, and we receive the publication that *Renu causes* some *infection*, then standard DL reasoning services can be employed to automatically infer that *BauschAndLomb* is a *RiskyCompany* and thus there is a match for the subscription.

[2]Note that this is expressed using standard turtle syntax (as opposed to RDF/XML) and can be easily generated in today's OWL ontology editors

```
:RiskyCompany  a  owl:Class;
   owl:intersectionOf  (
      [  a  owl:Restriction;  owl:onProperty  :hasProduct;
         owl:someValuesFrom  :AdverseEffectProduct ]
      :Company
   )  .
:AdverseEffectProduct  a  owl:Class;
   owl:intersectionOf  (
      [  a  owl:Restriction;  owl:onProperty  :causes;
         owl:someValuesFrom  [ owl:unionOf ( :Infection :AllergicReaction ) ] ]
      :Product
   )  .
:causes  a  owl:ObjectProperty.
:onRecommendation  a  owl:ObjectProperty.
```

Table 1.1: Illustration of Expressivity in OWL-based Syndication.

Through this discussion, it is clear that there are advantages in using the expressivity provided by OWL, and therefore syndication techniques should be extended to use it. Given this motivation, the main goal of this dissertation is to provide an expressive OWL-based syndication framework that is practical for real world use. To accomplish this, I formalize a syndication framework in which DL reasoning is used to match subscriptions with published contents[3].

While the proposed framework provides increased expressivity over an XML or RDF-based technique, the approach suffers from scalability issues due to the inherent complexity of description logic reasoning. In many domains (e.g., finance, military, etc.), response times must be minimal as critical content has to be processed and delivered in near real time (e.g., for stock trading purposes). The main scalability issue is related to reasoning in the presence changing data; this is primarily due to the static nature of existing DL reasoning techniques. In particular, the addition of information from newly published documents or data is a change in the underlying knowledge base (KB). In current DL reasoning algorithms, reasoning is performed from scratch on the updated KB; that is,

[3]Note that in this dissertation, the content creation problem (i.e., encoding published information in OWL) is not addressed, as it is out of scope of this work

the consistency of the KB must be ensured, queries must be re-evlautated, etc. While empirical results in some DL-based Web service matching [94] and publish/subscribe-like application scenarios [143] that reduce matching to concept subsumption demonstrate acceptable performance times (~20 ms) for matching new subscription requests with a fixed document base, processing incoming content is still problematic due to reasoning overhead. This problem is compounded if syndication brokers have domain knowledge with which newly published data is integrated. While there has also been recent work on DL-based publish/subscribe applications [61, 62] that adopts an approach similar to that presented in this dissertation (see Chapter 3.1.2 for a details), such work suffers from performance issues as well (response time ∼10s of seconds).

In this dissertation, I address the previously mentioned challenges related to the overhead introduced by DL reasoning that is required for the syndication framework. Specifically, in order to achieve a practical OWL-based syndication framework, the following reasoning services are addressed:

- consistency checking through updates
- query answering through updates

In order to address the first of these reasoning services, I have developed a set of incremental consistency checking techniques for addition and deletion updates (specifically DL ABox changes). The incremental techniques demonstrate orders of magnitude performance improvements, resulting in real time consistency checking of the syndication broker's knowledge base.

Similarly, to address the second reasoning task, I have developed novel techniques for continuous query answering; specifically for reducing the portion of the KB that is considered after an update. This aligns with related work on view and query maintenance in the context of relational and deductive databases, however is geared toward a different formalism (i.e., description logics). In the end, this provides an effective technique for incrementally maintaining query results as the underlying knowledge base is manipulated, and therefore, subscriptions registered with the syndication broker can be evaluated in real time for many expressive OWL ontologies.

As the syndication broker's knowledge base is updated with new publications, it is likely that logical contradictions will be encountered (e.g., due to conflicting information being published from different information sources). If this does occur, then a method for resolving these inconsistencies is necessary. Given this, I have developed a technique to regain consistency of the broker's KB. Specifically, I have developed a belief base revision algorithm for OWL DL knowledge bases, which uses the notion of trust to determine which assertions should be retracted to regain consistency.

In order to validate the practicality of the OWL-based syndication framework, I have implemented the incremental reasoning services and a prototype of the framework. Further, I have performed empirical evaluations using synthetic benchmark ontologies, simulating publications to assess the response times of the matching process; these evaluations demonstrate the practicality of the proposed syndication approach over ontologies of varying expressivity. Additionally, I have performed real-world simulations using historical news publications obtained through a collaboration with the Dow Jones Newswires[4]. Given this historical data archive, the practicality of the framework is empirically demon-

[4]Dow Jones Newswires: http://www.djnewswires.com/

strated over a high frequency publication use-case using real-world subscriptions.

1.1 Contributions

This dissertation presents a more expressive syndication framework for disseminating content over the Web; my thesis is that by using an OWL-based syndication approach, a framework can be provided that is practical in a real world setting. In order to achieve this goal, I have therefore focussed on the main performance bottlenecks of such a framework, namely incremental DL reasoning services. The specific contributions of this dissertation are as follows:

- Formalized an expressive syndication framework for the Web, which is based upon the Web Ontology Language and description logic reasoning. The framework provides a syndication approach with a rich semantics-based mechanism for expressing subscriptions and published content, allowing increased selectivity and finer control for filtering; this also provides automated reasoning for discovering subscription matches not found using traditional syntactic syndication approaches.

- Developed a set of incremental consistency checking techniques for expressive description logics. As new publications are integrated into the syndication broker's knowledge base, consistency must be guaranteed. The incremental techniques reduce the overhead introduced by performing this consistency check.

- Developed a set of incremental conjunctive (ABox) query answering techniques. After each publication, all subscriptions are evaluated to determine new matches given the new publications. In the framework, this reduces to DL conjunctive ABox query answering. Given the computational complexity of DL reasoning, this task introduces substantial overhead. The introduced techniques reduce the overhead introduced by query answering, making such a syndication framework practical.

- Developed a technique for recovering from logical inconsistencies in DL knowledge bases. Such a service is required as published information can be contradictory, leading to logical inconsistencies in the broker's knowledge base. The technique developed is a belief base semi-revision algorithm that provides a flexible mechanism to regain consistency.

- Demonstrated the practicality of the OWL-based syndication framework. This is achieved by performing a comprehensive evaluation of the OWL-based syndication framework using synthetic data, as well as real world data from the financial domain. This evaluation has investigated the utility of the incremental reasoning services for the purpose of the syndication framework and demonstrates its practicality.

1.2 Organization

This dissertation is organized as follows; Chapter 2 introduces background information related to this work. First, an overview of the Semantic Web and OWL is presented. Following this, an overview of the field of description logics is introduced, with a focus on the syntax and semantics of the description logic $\mathcal{SHOIQ}$ (which corresponds to a superset of OWL DL). Additionally, an overview of the field of belief revision is introduced.

In Chapter 3, I discuss related work. After this, I present the OWL-based syndication framework in Chapter 4. This includes a formalization of the framework, in addition to examples demonstrating its use. Chapter 5 introduces a set of incremental consistency checking techniques for the descriptions logics $\mathcal{SHIQ}$ and $\mathcal{SHOQ}$. The techniques presented support arbitrary ABox additions and deletions. Chapter 6 presents a technique for optimizing conjunctive ABox query answering in the presence of incremental ABox additions and deletions. The specific technique reduces the portion of the knowledge base that is considered as potentially new (invalidated) answers after an addition (respectively deletion). Following this, I present techniques for recovering from inconsistencies after publications in Chapter 7. This includes a belief base revision technique for the description logic $\mathcal{SHOIQ}$ (i.e., a superset of OWL DL). In Chapter 8, I present the implementation of my proposed syndication framework. This includes various details regarding the system architecture and specific implementation. Additionally, the results from empirical evaluations using synthetic data, as well as real world data from the financial domain are presented. Lastly, I conclude in Chapter 9, where a summary, general impact of the work, and outline for future work are provided.

Chapter 2

Foundations

In this chapter, I present some background information; the purpose is to familiarize the reader with the necessary concepts, terminology and definitions used throughout this dissertation.

2.1 Syndication Systems

In the context of this dissertation, a *syndication architecture* refers to an architectural paradigm comprised of three main components; publishers, subscribers, and syndication brokers, each of which are described below:

- *Publisher*: A data producer which publishes information (i.e., *publications*) to a syndication broker.
- *Subscriber*: An entity that is interested in subsets of the publications. A subscriber's interest is represented as a *subscription*.
- *Syndication Broker*: An intermediary whose primary task is to match newly published information (publications) with subscribers' interests (subscriptions).

In literature, such architectural paradigms are additionally referred to as publish-subscribe applications (e.g., [43]) and content/information dissemination systems (e.g., [150, 121]). Throughout this dissertation these terms will be used interchangeably. Note that the discussion presented above is very abstract and informal; a more precise overview of syndication frameworks investigated in literature is provided in Chapter 3.1. Additionally, a formalization of the framework developed in this dissertation is presented in detail in Chapter 4.2.

2.2 Semantic Web

The Semantic Web is an extension of the current World Wide Web, in which information on the Web is represented in a machine processable format with a well defined meaning (semantics) [22]. Representing the knowledge on the Web in such a manner provides a variety of benefits, including ease of knowledge exchange and integration and machine-automated reasoning. The standardized Semantic Web representation languages (published as W3C recommendations) form the foundation of the Semantic Web and are structured as a layered stack. At the bottom of this stack is the Resource Description Framework (RDF) [100], which is a fairly simple assertional language that represents information in the form of triples: subject–predicate–object. Subjects in triples are required to be resources, while predicates (or properties) are attributes of resources and correspond to traditional attribute-value pairs; lastly, objects can take the form of resources or literal values. RDF is based on the successful architecture of the Web, therefore making it designed to be open, scalable and distributed. Two of its key properties are the use of the Universal Resource Identifier (URI) as the unique identifier for classes, resources, properties and that it can be serialized in an XML format. It is important to note that RDF is a simple modeling language, as it does not provide mechanisms for describing properties, support the description of relationships between properties and other resources, etc. This is, however, provided by the next level of the layered stack by the RDF vocabulary description language, RDF Schema (RDFS) [35].

RDFS provides mechanisms for defining classes and properties, in addition to declaring subclasses (classes which subsume other classes), subproperties (properties which subsume other properties), and domains and ranges (taking the form of classes and/or complex datatypes) of properties. Using these RDFS constructs, simple taxonomies can be created. Similar to RDF, all RDFS classes and properties are referenced by URIs.

2.3 Web Ontology Language

Given the relatively weak expressivety of RDFS as a knowledge modeling language, the W3C has standardized the Web Ontology Language (OWL) [37]. OWL is a far more expressive knowledge representation language than RDFS and is positioned on top of both RDF and RDFS in the layered stack. Similar to RDFS, OWL provides mechanisms to define classes and properties; however, OWL also provides constructs to define class descriptions in terms of logical combinations of other classes, cardinality restrictions on properties, transitive and inverse properties, etc. (see section 2.4 for additional details). As in RDF and RDFS, OWL is based on the architecture of the Web; therefore, all named OWL classes, properties and individuals are referenced by URIs. Additionally, OWL provides the ability to link and import other ontologies using URIs.

The OWL language comes in three different species, each of which provide increased expressivity: OWL Lite, OWL DL and OWL Full. The semantics of OWL Lite and OWL DL are aligned with a family of knowledge representation languages, namely descriptions logics, and are essentially syntactic variants of these logics (see section 2.4 for details). This implies that an OWL Lite/DL ontology is equivalent to a DL knowledge base, and therefore traditional DL reasoning techniques can be used for processing these OWL ontologies (see Table 2.2 for an overview of this translation). It is important to note that OWL Lite and DL differ from OWL Full in that they define certain constraints on the way the language constructs can be used so that this alignment exists. For example, in OWL DL a class cannot be treated as an individual or a property, and transitive properties cannot be used in cardinality restrictions (this additionally ensures decidability of these languages) [37]. In contrast, OWL Full does not impose these restrictions and therefore does not correspond to a description logic and is known to be undecidable.

It is important to note that OWL assumes an open-world semantics, making it different from traditional database schema languages which adopt a closed-world semantics. More specifically, information which is not explicitly asserted in an OWL knowledge base is assumed to be *unknown* instead of *false* (as is done under closed-world semantics). Additionally, OWL does not make the Unique Name Assumption (UNA); that is, given two individuals (i.e., instances) with different names, it is not assumed that they are distinct (i.e., different names can refer to the same individual).

Lastly, it is pointed out that there has been recent interest in developing query languages for both RDF and OWL; this includes recent work on the SPARQL [122] and RDQL [133] query languages. Details of the languages are omitted here, as in this dissertation it is assumed that queries of a specific form for description logics are used (formally discussed below); further, today's DL reasoners provide translations of (subsets) of standard RDF and OWL query languages (typically including SPARQL or RDQL) to conjunctive queries for DL KBs.

2.4 Description Logics

Description logics (DLs) [16] are a family of knowledge representation formalisms tailored for expressing knowledge about concepts and concept heirarchies. DLs are a decidable subset of First Order Logic (FOL) and are given a well-defined, model theoretic semantics [18]. At the most basic level, DLs can be used to describe the following objects [16]:

- Classes of objects – correspond to 1-place predicates in FOL. An example is the set of objects that are *Companies(x)*
- Roles between classes – correspond to 2-place predicates in FOL. An example is the set of objects have a *hasProduct(x,y)* role
- Individuals – correspond to constants in FOL. An example is *BauschAndLomb*

Using these basic building blocks, DLs provide a set of constructors for building more complex classes. Typically, the languages provide at least the standard boolean concept constructors, namely conjunction ($\sqcap$), disjunction ($\sqcup$), and negation ($\neg$); note that the intersection and disjunction notation will be further described below. DLs usually support constructors to restrict the quantification of roles, specifically *universal* ($\forall$) and *existential* ($\exists$) restrictions. Additional constructors including cardinality restrictions on roles and more expressive roles (e.g., inverse and transitive roles) are provided in some DLs and will be discussed below.

DL *knowledge bases* (KBs) are comprised of three main components, namely a *TBox*, *RBox*, and *ABox*. The TBox contains intensional knowledge (axioms about concepts) in the form of a terminology. The axioms in the TBox can be built using the previously mentioned concept constructors, as well as concept inclusion axioms ($\sqsubseteq$), which state inclusion relations between DL concepts. For example, one can state that any technology company is a company via the following axiom: *TechnologyCompany* $\sqsubseteq$ *Company*. The RBox contains intentional knowledge about the roles in the knowledge base. For example, one can state that any two individuals that satisfy the *hasCEO* role, also satisfy the *hasEmployee* role by the following axiom: *hasCEO* $\sqsubseteq$ *hasEmployee*, where *hasCEO* and *hasEmployee* are roles. Typically the role constructors provided in DLs are far less expressive than the constructor for concepts. In a similar manner to the TBox, the axioms in the RBox take the form of inclusion axioms between roles.

In contrast, the ABox contains extensional knowledge that is specific to the individuals in the domain of discourse. Assertions in the ABox take the form of concept assertions (e.g., *Company(BauschAndLomb)*), role assertions (e.g., *hasProduct(BauschAndLomb, Renu)*), and equality (*Ford = FordMotorCompnay*) and inequality (e.g., *CitiGroup* $\neq$ *CapitalCityBankGroup*) assertions. Therefore, an ABox is a set of assertions of these forms.

The DL community uses a variety of mnemonics for representing the expressivity (i.e., the supported constructs) of a given DL language [16]. This is characterized in Table 2.1[1]. These mnemonics will be used throughout this dissertation when referencing

[1]Note that A is assumed to be an atomic concept, while C, D are arbitrary concepts

Mnemonic	Expressivity
$\mathcal{AL}$	Attribute Logic (A, $\neg A$, $C \sqcap D$, $\exists R.\top$, $\forall R.C$)
$\mathcal{ALC}$	Attribute Logic with Full Complement ($\neg C$ which allows $C \sqcup D$ and $\exists R.C$)
$\mathcal{R}^+$	Transitive Roles
$\mathcal{S}$	$ALCR^+$
$\mathcal{H}$	Role Hierarchies
$\mathcal{O}$	Nominals (individuals in class expressions e.g., $\{BaushAndLomb\}$)
$\mathcal{I}$	Inverse Roles
$\mathcal{F}$	Functional Roles
$\mathcal{N}$	Unqualified Cardinality Restrictions ($\geqslant nR$, $\leqslant nR$, $= nR$)
$\mathcal{Q}$	Qualified Cardinality Restrictions ($\geqslant nR.C$, $\leqslant nR.C$, $= nR.C$)
$\mathcal{D}$	Concrete Domains

Table 2.1: Mnemonics for Description Logic Expressivity

a particular DL.

In literature, there has been substantial work on determining the computational impact of allowing various constructs in the logic (see [16, 26] for an overview). Much of this work has focussed on determining decidability and complexity results when different constructors and restrictions are supported or imposed on the particular DL. For example, one such restriction is to only allow *definitorial* TBoxes; more specifically, only inclusion axioms of the form $A \sqsubseteq C$ and $A \equiv C$ (note that $A \equiv C$ is an abbreviation for $A \sqsubseteq C$ and $C \sqsubseteq A$) are allowed, such that A is an atomic concept and the definitions are unique and acyclic (i.e., the right hand side of an axiom cannot directly or indirectly refer to the concept on its left hand side). It has been shown that this greatly simplifies reasoning complexity [16, 73]. If the TBox contains an axiom of the form $C \sqsubseteq D$ where C is a complex concept, then this axiom is referred to as a *general concept inclusion axiom* (GCI) and the TBox is referred to as a *general TBox*.

There has additionally been extensive work on developing practical reasoning procedures for DLs [73]. Further discussion regarding this topic is presented later in sections

2.4.2 & 2.4.3.

As mentioned earlier, OWL Lite and OWL DL are aligned with descriptions logics. In particular OWL Lite is a syntactic variant of $\mathcal{SHIF}$, while OWL DL is a variant of $\mathcal{SHOIN}$. An overview of a subset of this correspondence is shown below in Table 2.2. In this dissertation, incremental reasoning services are developed for large subsets of the DL $\mathcal{SHOIQ}(\mathcal{D})$, which in turn subsumes OWL Lite and OWL DL. Given this, an overview of the syntax and semantics of $\mathcal{SHOIQ}(\mathcal{D})$ is presented.

Construct	**OWL**	**DL**
Concept Subsumption	`rdfs:subClassOf` (C,D)	$C \sqsubseteq D$
Concept Equivalence	`owl:equivalentTo` (C,D)	$C \equiv D$
Negation	`owl:complementOf` (C,D)	$C \equiv \neg D$
Dijoint Concepts	`owl:disjointWith` (C,D)	$C \sqsubseteq \neg D$
Conjunction	`owl:intersectionOf` (C,D)	$C \sqcap D$
Disjunction	`owl:unionOf` (C,D)	$C \sqcup D$
Nominal Disjunction	`owl:oneOf` (a, b)	$\{a\} \sqcup \{b\}$
Existential Restriction	`owl:someValuesFrom`(R,C)	$\exists R.C$
Universal Restriction	`owl:allValuesFrom`(R,C)	$\forall R.C$
Existential Restriction (with nominals)	`owl:hasValue` (R,a)	$\exists R.\{a\}$
Number Restriction	`owl:cardinality`(S,n)	$= nS.\top$
At-most Number Restriction	`owl:minCardinality`(S,n)	$\geqslant nS.\top$
At-least Number Restriction	`owl:maxCardinality`(S,n)	$\leqslant nS.\top$

Table 2.2: Correspondence from OWL-DL to Description Logics

2.4.1 Syntax and Semantics of $\mathcal{SHOIQ}(\mathcal{D})$

Let $\mathbf{C}, \mathbf{R}, \mathbf{R_D}, \mathbf{I}, \mathbf{D}$ be non-empty and pair-wise disjoint sets of *atomic concepts*, *abstract* and *concrete atomic roles*, *individuals*, and *concrete datatypes* respectively. The set of $\mathcal{SHOIQ}(\mathcal{D})$ abstract roles is the set $\mathbf{R} \cup \{R^- \mid R \in \mathbf{R}\}$, where R^- denotes the inverse of the abstract atomic role R. In contrast, the set of $\mathcal{SHOIQ}(\mathcal{D})$ concrete roles is simply $\mathbf{R_D}$. To avoid considering the abstract role R^{--}, the function $\mathsf{Inv}(R)$ is defined such that

$\mathsf{Inv}(R) = R^-$ and $\mathsf{Inv}(R^-) = R$ for $R \in \mathbf{R}$. Inverses cannot be defined on concrete roles.

A *role inclusion axiom* is an expression of the form $R_1 \sqsubseteq R_2$ or $u_1 \sqsubseteq u_2$, where $R_1, R_2 \in \mathbf{R}$ and $u_1, u_2 \in \mathbf{R_D}$. A *transitivity axiom* is an expression of the form $\mathsf{Trans}(R)$, where $R \in \mathbf{R}$. Given this, a RBox R is a finite set of role inclusion axioms and transitivity axioms.

For ease of exposition, given an RBox R let the symbol $\mathrel{\underline{\circledast}}_{\mathsf{R}}$ denote the transitive reflexive closure of $\sqsubseteq$ on $\mathsf{R} \cup \{\mathsf{Inv}(R_1) \sqsubseteq \mathsf{Inv}(R_2) \mid R_1 \sqsubseteq R_2 \in \mathsf{R}$ and $R_1, R_2 \in \mathbf{R}\}$. Additionally, $R_1 \equiv_{\mathsf{R}} R_2$ is used as an abbreviation for $R_1 \mathrel{\underline{\circledast}}_{\mathsf{R}} R_2$ and $R_2 \mathrel{\underline{\circledast}}_{\mathsf{R}} R_1$. Lastly, the function $Tr(R_1, \mathsf{R})$ is defined to return *true* if for some R_2 with $R_1 \equiv_{\mathsf{R}} R_2$, $\mathsf{Trans}(R_2) \in \mathsf{R}$ or $\mathsf{Trans}(\mathsf{Inv}(R_2)) \in \mathsf{R}$; otherwise the function returns *false*. A role R_1 is considered *simple* with respect to R if $Tr(R_2, \mathsf{R}) =$ *false* for all $R_2 \mathrel{\underline{\circledast}}_{\mathsf{R}} R_1$. Note that $u \in \mathbf{R_D}$ is trivially simple roles as it cannot be transitive.

Before introducing the set of $\mathcal{SHOIQ}(\mathcal{D})$-concepts, the notion of a *concrete domain* is briefly presented; specifically, a concrete domain $\mathcal{D}$ is defined to be a pair $(\Delta^{\mathcal{D}}, \Phi^{\mathcal{D}})$, where $\Delta^{\mathcal{D}}$ is called the domain and $\Phi^{\mathcal{D}}$ are the set of predicate names. Further details regarding the concrete domains can be found in [19, 77]. Given this, the set of $\mathcal{SHOIQ}(\mathcal{D})$-concepts (concepts for short) is inductively defined to be the smallest set such that the following holds:

- every concept $A \in \mathbf{C}$ is a concept.
- if C and D are concepts and R is an abstract role, then $(C \sqcap D)$, $(C \sqcup D)$, $(\neg C)$, $(\exists R.C)$, and $(\forall R.C)$ are concepts. These constructors are referred to as conjunction, disjunction, negation, existential restriction, and universal restriction respectively.

- if C is a concept, S is a simple abstract role, and n is a natural number, then $(\leqslant nS.C)$ and $(\geqslant nS.C)$ are concepts. These constructors are referred to as at-most and at-least number restrictions.

- if $a \in \mathbf{I}$, then the nominal $\{a\}$ is a concept.

- if u is a concrete role, $\mathsf{P} \in \Phi^{\mathcal{D}}$ is a predicate of the concrete domain, and n is a natural number, then $(\exists u.\mathsf{P})$, $(\forall u.\mathsf{P})$, $(\leqslant nu.\mathsf{P})$ and $(\geqslant nu.\mathsf{P})$ are concepts.

For concepts C, D, a *concept inclusion axiom* is an expression of the form $C \sqsubseteq D$. A *concept equivalence axiom*, denoted by $C \equiv D$, is an abbreviation for $C \sqsubseteq D$ and $D \sqsubseteq C$. Given this, a TBox T is defined to be a finite set of concept inclusion axioms.

An ABox A is a finite set of concept assertions of the form $C(a)$ (where C can be an arbitrary $\mathcal{SHOIQ}(\mathcal{D})$-concept), abstract role assertions of the form $R(a, b)$ and inequality (equality) assertions of the form $a \neq b$ (respectively $a = b$) for $a, b \in \mathbf{I}$. Additionally, the ABox can contain concrete domain predicate assertions $\mathsf{P}(x)$ and concrete role assertions $u(a, x)$, where $\mathsf{P} \in \Phi^{\mathcal{D}}$, $a \in \mathbf{I}$, $x \in \mathbf{D}$ and $u \in \mathbf{R_D}$.

A KB $\mathsf{K} = (\mathsf{T}, \mathsf{R}, \mathsf{A})$ is triple composed of TBox T, RBox R and ABox A. When dealing with multiple KBs or ABoxes, the set of named individuals in KB K (ABox assertion α) are denoted as $\mathbf{I}_{\mathsf{K}}$ (respectively $\mathbf{I}_{\alpha}$).

Throughout this dissertation, incremental changes to a KB are performed. For ease of exposition, given some axiom (TBox, RBox, or ABox) α, the addition (resp. deletion) of α to its corresponding component is denoted by $\mathsf{K} + \alpha$ (resp. $\mathsf{K} - \alpha$); for example, if α is an added ABox assertion, then it is assumed to be added to the ABox by extending the original ABox, A, such that $\mathsf{A}' = \mathsf{A} \cup \{\alpha\}$.

The semantics of $\mathcal{SHOIQ}(\mathcal{D})$ is defined using interpretations $\mathcal{I}$, which are comprised of a non-empty set $\Delta^{\mathcal{I}}$ (i.e., the domain of the interpretation), which is assumed to be disjoint from the concrete domain $\Delta^{\mathcal{D}}$, and an interpretation function $.^{\mathcal{I}}$. More formally, an *interpretation* $\mathcal{I}$ is a pair $\mathcal{I} = (\Delta^{\mathcal{I}}, .^{\mathcal{I}})$. The interpretation function assigns to each atomic concept $A \in \mathbf{C}$ a subset of $\Delta^{\mathcal{I}}$, to each abstract atomic $R \in \mathbf{R}$ a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$, and to each $a \in \mathbf{I}$ an element of $\Delta^{\mathcal{I}}$. Additionally, the interpretation function assigns to each concrete atomic role $u \in \mathbf{R_D}$ a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{D}}$, to each predicate $\mathsf{P} \in \Phi^{\mathcal{D}}$ a subset of $\Delta^{\mathcal{D}}$, and to each $x \in \mathbf{D}$ an element of $\Delta^{\mathcal{D}}$. The interpretation function is extended to complex concept descriptions as follows, where R is an abstract role, S is a simple abstract role, u is a concrete role, and $\sharp$ denotes cardinality:

- $(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$
- $(C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}$
- $(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
- $(\exists R.C)^{\mathcal{I}} = \{a \in \Delta^{\mathcal{I}} \mid \exists b \in \Delta^{\mathcal{I}} \text{ such that } (x, y) \in R^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}}\}$
- $(\forall R.C)^{\mathcal{I}} = \{a \in \Delta^{\mathcal{I}} \mid \forall b \in \Delta^{\mathcal{I}}, \text{ if } (a, b) \in R^{\mathcal{I}} \text{ then } b \in C^{\mathcal{I}}\}$
- $(\leqslant nS.C)^{\mathcal{I}} = \{a \in \Delta^{\mathcal{I}} \mid \sharp\{b \in \Delta^{\mathcal{I}} \mid (a, b) \in S^{\mathcal{I}} \text{ and } b \in C^{\mathcal{I}}\} \leq n\}$
- $(\geqslant nS.C)^{\mathcal{I}} = \{a \in \Delta^{\mathcal{I}} \mid \sharp\{b \in \Delta^{\mathcal{I}} \mid (a, b) \in S^{\mathcal{I}} \text{ and } b \in C^{\mathcal{I}}\} \geq n\}$
- $\{a\}^{\mathcal{I}} = \{a^{\mathcal{I}}\}$
- $(\exists u.\mathsf{P})^{\mathcal{I}} = \{a \in \Delta^{\mathcal{I}} \mid \exists x \in \Delta^{\mathcal{D}} \text{ such that } (a, x) \in u^{\mathcal{I}} \text{ and } x \in \mathsf{P}^{\mathcal{I}}\}$
- $(\forall u.\mathsf{P})^{\mathcal{I}} = \{a \in \Delta^{\mathcal{I}} \mid \forall x \in \Delta^{\mathcal{D}}, \text{ if } (a, x) \in u^{\mathcal{I}} \text{ then } x \in \mathsf{P}^{\mathcal{I}}\}$
- $(\leqslant nu.\mathsf{P})^{\mathcal{I}} = \{a \in \Delta^{\mathcal{I}} \mid \sharp\{x \in \Delta^{\mathcal{D}} \mid (a, x) \in u^{\mathcal{I}} \text{ and } x \in \mathsf{P}^{\mathcal{I}}\} \leq n\}$
- $(\geqslant nu.\mathsf{P})^{\mathcal{I}} = \{a \in \Delta^{\mathcal{I}} \mid \sharp\{x \in \Delta^{\mathcal{D}} \mid (a, x) \in u^{\mathcal{I}} \text{ and } x \in \mathsf{P}^{\mathcal{I}}\} \geq n\}$

Additionally, the interpretation function is extended to complex abstract roles as follows, where R is an atomic abstract role:

$$(\mathsf{Inv}(R))^{\mathcal{I}} = \{(a, b) \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid (b, a) \in R^{\mathcal{I}}\}$$

The satisfaction of a $\mathcal{SHOIQ}(\mathcal{D})$ axiom/assertion α in an interpretation $\mathcal{I}$, denoted $\mathcal{I} \models \alpha$ is defined by the following, where R denotes an abstract role, u a concrete role, $a, b \in \mathbf{I}$ and $x \in \mathbf{D}$:

- $\mathcal{I}$ satisfies $R_1 \sqsubseteq R_2$ iff $R_1^{\mathcal{I}} \subseteq R_2^{\mathcal{I}}$
- $\mathcal{I}$ satisfies $u_1 \sqsubseteq u_2$ iff $u_1^{\mathcal{I}} \subseteq u_2^{\mathcal{I}}$
- $\mathcal{I}$ satisfies $\mathsf{Trans}(R)$ iff for every $a, b, c \in \Delta^{\mathcal{I}}$, if $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$ and $(b^{\mathcal{I}}, c^{\mathcal{I}}) \in R^{\mathcal{I}}$, then $(a^{\mathcal{I}}, c^{\mathcal{I}}) \in R^{\mathcal{I}}$
- $\mathcal{I}$ satisfies $C \sqsubseteq D$ iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$
- $\mathcal{I}$ satisfies $C(a)$ iff $a^{\mathcal{I}} \in C^{\mathcal{I}}$
- $\mathcal{I}$ satisfies $R(a, b)$ iff $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$
- $\mathcal{I}$ satisfies $u(a, x)$ iff $(a^{\mathcal{I}}, x^{\mathcal{I}}) \in u^{\mathcal{I}}$
- $\mathcal{I}$ satisfies $\mathsf{P}(x)$ iff $x^{\mathcal{I}} \in \mathsf{P}^{\mathcal{I}}$
- $\mathcal{I}$ satisfies $a \neq b$ iff $a^{\mathcal{I}} \neq b^{\mathcal{I}}$
- $\mathcal{I}$ satisfies $a = b$ iff $a^{\mathcal{I}} = b^{\mathcal{I}}$

The interpretation $\mathcal{I}$ is a model of a TBox T (resp. RBox R, ABox A) if it satisfies all the axioms in T (resp. R, A). Additionally, $\mathcal{I}$ is a model of K, denoted by $\mathcal{I} \models \mathsf{K}$, iff $\mathcal{I}$ is a model of T, R, and A.

Lastly, the following notation is additionally introduced; namely, $\top$ and $\bot$ are used to abbreviate $C \sqcup \neg C$ and $C \sqcap \neg C$ respectively.

2.4.2 Description Logic Reasoning

In description logics, there are a variety of basic reasoning tasks which are briefly outlined below.

- *Consistency Checking*: The process of ensuring that the knowledge base does not contain any contradictory facts.

- *Concept Satisfiability*: Given a concept C, checking if C is satisfiable with respect to KB K is the task of determining if there exists an interpretation $\mathcal{I}$ of K such that the interpretation of C is not equal to the empty set (i.e., $C^{\mathcal{I}} \neq \emptyset$).

- *Concept Subsumption*: Given concepts C, D, checking if C is subsumed by D relative to K, denoted $\mathsf{K} \models C \sqsubseteq D$, is the process of determining if for all interpretations $\mathcal{I}$ of K, $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$.

- *Concept Instantiation*: Given concept C and indivdiual a, checking if a instantiates C relative to K, denoted $\mathsf{K} \models C(a)$, is the process of determining if for all interpretations $\mathcal{I}$ of K, $a^{\mathcal{I}} \in C^{\mathcal{I}}$.

Given these basic reasoning tasks, the following more general standard reasoning services are typically provided in today's state of the are DL reasoners:

- *Realization*: Determining the most *specific* concepts that each individual instantiates in the KB.

- *Classification*: For all named concepts (A and B) in a KB, determine whether a *subsumption* relation holds between the concepts in either direction; that is, whether $A \sqsubseteq B$ or $B \sqsubseteq A$ or both.

- *Retrieval*: Given a concept C, retrieve all individuals which instantiate C.

It is important to note that all reasoning tasks can be reduced to ABox consistency checking [16]. This is exemplified by the following example: suppose that we want to check if an individual a instantiates a concept C with respect to a KB K; this is accomplished by checking the consistency of $\mathsf{K} \cup \{\neg C(a)\}$. If this is not consistent, then it must be the case that there does not exist an interpretation which satisfies $\neg C(a)$, therefore all interpretations must satisfy $C(a)$.

I now provide additional details regarding conjunctive ABox queries for DLs. A conjunctive query Q contains a non-empty set of concept and role atoms, $C(x)$ and $R(x, y)$

respectively, where x can be a named individual (i.e., taken from $\mathbf{I}$) or variable name and y can be a named individual, concrete datatype (i.e., taken from $\mathbf{D}$), or variable name. Further, the variable names are assumed to be typed such that each variable is either *distinguished* or *non-distinguished*; the specific difference being that the distinguished variables must be mapped to named individuals, where as the non-distinguished variables are existentially quantified. A simple example query is $(x, y) \leftarrow Company(x) \wedge hasProduct(x, y) \wedge hasComponent(y, z)$. In this example, x, y are the distinguished variables and are the answer variables; in general, the distinguished variables will be denoted as the query answer variables.

Given this brief introduction, a more formal presentation is now provided. For ease of exposition, concrete roles are disallowed as query atoms in the following discussion, however, they can easily be supported[2]. Let $\mathbf{V}$ be a countably infinite set of query *variables* that is disjoint from $\mathbf{C}$, $\mathbf{R}$, $\mathbf{R_D}$, $\mathbf{I}$, and $\mathbf{D}$. A query *atom* is defined to be an expression $C(x)$, $R(x, y)$ where C is a $\mathcal{SHOIQ}(\mathcal{D})$ concept, R an abstract role, $x, y \in \mathbf{V} \cup \mathbf{I}$. Given this, a conjunctive query is defined to be a triple $Q = (A, X, Y)$ where A is a non-empty set of query atoms, X is the set of distinguished variables and Y is the set of non-distinguished variables. The notation $Var(Q)$ will be used to denote the set of variables and individuals occurring in the query, and $DVar(Q)$ will be used to denote the set of distinguished variables. Additionally, when referring to an atom, *at*, of a given query Q, the notation $at \in Q$ will simply be used. Next we draw a simple, but important distinction between different types of queries; specifically, a *boolean* conjunctive query is a query that has no distinguished variables (i.e., $X = \emptyset$), where a *retrieval* query has at least one distinguished variable (i.e., $X \neq \emptyset$).

Next, the semantics for conjunctive queries is introduced. First, a variable substitution function $\pi : Var(Q) \rightarrow \Delta^{\mathcal{I}}$ is defined, which maps query variables and individuals to elements of the domain. Given π and an interpretation $\mathcal{I}$, the following notation is

[2]This can be accomplished by introducing concrete role query atoms, which are interpreted over the concrete domain.

introduced:

- $\mathcal{I} \models^{\pi} C(x)$ if $(\pi(x)) \in C^{\mathcal{I}}$
- $\mathcal{I} \models^{\pi} R(x, y)$ if $(\pi(x), \pi(y)) \in R^{\mathcal{I}}$

Given this, the semantics of a boolean query, Q, is presented. Specifically, an interpretation $\mathcal{I}$ satisfies Q, denoted $\mathcal{I} \models Q$, if there exists some π such that for each individual $a \in Var(Q)$, $\pi(a) = a^{\mathcal{I}}$ and for all $at \in Q$, $\mathcal{I} \models^{\pi} at$. Finally, a knowledge base K entails Q, denoted $\mathsf{K} \models Q$, if for every interpretation $\mathcal{I}$ of K, $\mathcal{I} \models Q$.

The semantics of retrieval queries (i.e., those with at least one distinguished variables) is defined in a slightly extended manner. In particular, given a retrieval query, Q, with n distinguished variables (i.e., $DVar(Q) = \{d_1, ..., d_n\}$), define the *answers* of KB K to Q to be those n-tuples $(a_1, ..., a_n) \in \mathbf{I}^n$ such that for all interpretations $\mathcal{I}$ of K, $\mathcal{I} \models^{\pi} \mathsf{K}$ for some π such that for all a_i, $\pi(d_i) = a_i$ where $1 \leq i \leq n$. In the remainder of this dissertation, when referring to the entailment of a particular substitution of named individuals $(a_1, ..., a_n) \in \mathbf{I}^n$ for a retrieval query $(x_1, ..., x_n) \leftarrow Q$, the following notation will be used:

$$\mathsf{K} \models Q[x_1/a_1, ..., x_n/a_n]$$

Lastly, a special case of these two queries is introduced which is referred to as a *ground* query. In this case, the query does not contains any variables (i.e., $X \cup Y = \emptyset$). Note however, that this does not affect the formalization previously presented.

Similar to the previously mentioned reasoning services, query answering is typically reduced to ABox consistency checking and is accomplished via a proof by refutation techniques. First, consider queries with only a single ground concept atom. As an example, consider $() \leftarrow Company(Ford)$; in order to check if $\mathsf{K} \models Company(Ford)$, K is extended with the negation of the atom (i.e., $\mathsf{K}' = \mathsf{K} \cup \{\neg Company(Ford)\}$) and the consistency of the extended KB is checked. If the KB is inconsistent then the entailment holds, as there does not exists a model in which $a^{\mathcal{I}} \in (\neg Company)^{\mathcal{I}}$ [81, 140]. Such a technique can additionally be easily extended to handle conjunctions of ground concept terms, as a conjunction is only a logical consequence if all conjuncts are logical consequences [81].

In order to extend the previous approach to role atoms, a more complicated technique is required. The general idea is to transform each role atom in the query into a concept atom, which is referred to as *rolling-up* the query [81, 140]; this rolling-up process if often enabled via the use of nominals. For example, consider the following query: $() \leftarrow hasProduct(BauchAndLomb, Renu)$. It can be shown that this role term can be transformed into the equivalent concept term $\exists hasProduct.\{Renu\}(BauchAndLomb)$ [81, 140]; this is quite intuitive because if *BauchAndLomb* instantiates this concept, it must have some *hasProduct* role to the individual *Renu*, which is semantically the same as the original query. If instead the query were $() \leftarrow hasProduct(BauchAndLomb, Renu) \wedge Product(Renu)$, the query would be further absorbed into the rolled-up concept term, resulting in $\exists hasProduct.(\{Renu\} \sqcap Product)(BauchAndLomb)$.

Unfortunately, many DLs do not support the use of nominals; for example the DL $\mathcal{SHIF}$ (i.e., OWL Lite) does not include such expressivity. However there is a well known workaround, in which the use of nominal can be simulated. The approach is to substitute each nominal in the rolled-up query concept with a new concept name that does not occur in the knowledge base [81, 140]. Additionally, an assertion is added to ensure that each individual instantiates its representative concept. For example, the previously rolled-up query concept $\exists hasProduct.\{Renu\}(BauchAndLomb)$ would be transformed to $\exists hasProduct.C_{Renu}(BauchAndLomb)$ and the assertion $C_{Renu}(Renu)$ would be added to the KB.

The previous technique can be extend to queries with variables as well. The case of boolean queries (i.e., all non-distinguished variables) can be handled in a very similar manner; however, because variables in the query can be interpreted as any element of the domain, we cannot roll-up the query as before with the use of nominals or representative concepts. In contrast, the concept $\top$ is used, as it is interpreted as any element of the domain [81]. For example, consider the query $() \leftarrow hasProduct(x, y)$; in this case, the rolling up procedure would result in $\exists hasProduct.\top$, which intuitively represents the individuals in the domain that have a *hasProduct* role to some individual in the domain. We can guarantee that the interpretation of this concept is non-empty in every model of a KB K by extending K such that $\mathsf{K}' = \mathsf{K} \cup \{\top \sqsubseteq \neg\exists hasProduct.\top\}$ and then checking for consistency [81, 140].

If the query is a retrieval query, (e.g., $(x) \leftarrow hasProduct(x, y)$), then the naïve approach is to roll up the query and then to iterate over possible substitutions of named individuals for distinguished variables; then for each of these substitutions, a consistency check is performed [81, 140]. For example, assume that $\mathbf{I} = \{Ford\}$; in this case the rolled-up query would again be $\exists hasProduct.\top$, and the consistency of $\mathsf{K}' = \mathsf{K} \cup \{\neg(\exists hasProduct.\top(Ford))\}$ would be checked.

Some comments are in order regarding the known limitations of the rolling-up technique. First, if the KB is expressed using the DL $\mathcal{SHOIQ}$ (a superset of OWL DL), then arbitrary queries cannot be supported, due to various issues (e.g., handling cycles in the query [51]); if however, there does not exist a cycle in the query involving only variables or only distinguished variables are permitted, then this issue is overcome [136]. The rolling-up techniques is also problematic for arbitrary queries for the DL $\mathcal{SHIQ}$ (a superset of OWL Lite). However, once again if the query is restricted such that there does not exist a cycle in the query involving only variables or only distinguished variables are allowed, then this is not an issue [82]. Additionally, if only simple roles are allowed as role atoms in the query, then this problem is overcome [81]; the main insight with this restriction is that due to the tree-like model property of $\mathcal{SHIQ}$, all variables in cycles in the query must be bound to named individuals (in effect they can be considered as distinguished variables).

There exists alternative query answering techniques when dealing with different query types and certain DLs [52, 53, 50, 51, 105, 136]. Details are omitted here, as the techniques presented in this dissertation are dependent on the ability to roll-up the query; it is important to note however, that after the proposed technique has been used to prune the candidate bindings (discussed in Chapter 6), any querying answering technique can be used.

It is lastly noted that many of today's DL reasoners provide translations of common OWL or RDF query languages (e.g. SPARQL [122] or RDQL [133]) to conjunctive ABox queries for DL KBs. In the remainder of this dissertation, when discussing OWL query languages we simply refer to the subsets of the languages which can be translated to conjunctive ABox queries.

2.4.3 Tableau Algorithms

There exists sound and complete decision procedures for various DLs, including $\mathcal{SHOIQ}$ (and therefore OWL DL). Current state of the art reasoning algorithms are based on the tableau calculus [17] and therefore the $\mathcal{SHOIQ}$ tableau algorithm is presented. A more comprehensive discussion, including correctness proofs can be found in [78]. For ease of presentation, the discussion presented does not address datatype support; however, the tableau algorithm can easily be extended to support datatypes (e.g., see [77]).

It is first noted that reasoning with a general TBox T and role heirachry $\mathcal{R}$ can be reduced to only reasoning with $\mathcal{R}$. This is because the TBox can be *internalized* into a single concept that is added to all individuals [78]. This process is briefly introduced[3], as the notion will be referred to later in Chapter 6.4. First, all TBox axioms in the KB are transformed into a single concept as follows:

$$C_T = \bigsqcap_{C \sqsubseteq D \in \mathsf{K}} \neg C \sqcup D$$

Following this, a transitive role U is introduced that does not occur in the KB, and the role heirarchy for the KB is extended such that U is a transitive super-role of all roles occurring in the KB; that is, the role hierarchy is extended as follows:

[3]See [78] for a more detailed discussion.

$$\mathcal{R}_U = \mathcal{R} \cup \{R \sqsubseteq U, \mathsf{Inv}(R) \sqsubseteq U \mid R \textit{ occurs in } \mathsf{K}\}$$

Given this, it has been shown that the consistency of the KB can be reduced to checking the consistency of simply the ABox w.r.t the role hierarchy $\mathcal{R}_U$ by extending the ABox with $(C_T \sqcap \forall U.C_T)(a)$ for all named individuals $a \in \mathbf{I}_{\mathsf{K}}$ [78].

In practice however, the TBox is typically partitioned into two subsets, a general and unfoldable TBox [75]. Then, only the general TBox is internalized and the unfoldable TBox is taken into account during the actual tableau algorithm (discussed further later).

In the remaining discussion and when addressing tableau algorithms in this dissertation, it is assumed that concepts are in negation normal form (NNF). This is done by performing a syntactic transformation on each concept, in which a combination of de Morgan's rules are applied to the concept, pushing negation as far inward as possible [16].

DL tableau-based algorithms decide the consistency of an ABox A w.r.t. TBox T and RBox R by trying to construct (an abstraction of) a model for A, T and R, called a *completion graph* [78]. Each node in the completion graph represents an individual, which is labeled with a set of concepts that it satisfies (in the particular model). Similarly, each edge and edge label in the completion graph represents the roles satisfied by the individuals in the model. Formally, a completion graph is a directed graph $\mathsf{G} = (\mathcal{V}, \mathcal{E}, \mathcal{L}, \dot{\neq})$, in which each node $x \in \mathcal{V}$ is labeled with a set of concepts $\mathcal{L}(x)$ and each edge $e = \langle x, y\rangle$ with a set $\mathcal{L}(e)$ of role names. The binary predicate $\dot{\neq}$ is used for recording inequalities between nodes.

Before introducing how the completion graph is constructed, a variety of terminology and notation is introduced, all of which can additionally be found in [78].

- $R \in \mathcal{L}(\langle x, y\rangle)$ is used as an abbreviation for $\langle x, y\rangle \in \mathcal{E}$ and $R \in \mathcal{L}(\langle x, y\rangle)$
- If $\langle x, y\rangle \in \mathcal{E}$, then y is called a *successor* of x and x is called a *predecessor* or y. *Ancestor* is the transitive closure of predecessor, and *descendant* is the transitive closure of successor. A node y is called a R-successor of a node x if for some R'

with $R' \sqsubseteq^{*} R$, $R' \in \mathcal{L}(\langle x, y\rangle)$. Lastly, a node y is called a *neighbor* (R-neighbor) of a node x if y is a successor (R-successor) of x or if x is a successor ($\mathsf{Inv}(R)$-successor) of y.

- A completion graph G is said to have a *clash* if one of the following holds:
  1. for some concept name $A \in \mathbf{C}$ and node x of G, $\{A, \neg A\} \subseteq \mathcal{L}(x)$
  2. for some role S and node x of G, $(\leqslant nS.C) \in \mathcal{L}(x)$ and there are $n+1$ S-neighbors $y_0, ..., y_n$ of x with $C \in \mathcal{L}(y_i)$ for each $0 \leq i \leq n$ and $y_i \dot{\neq} y_j$ for each $0 \leq i < j \leq n$
  3. for some $o \in \mathbf{I}$, there are nodes x, y s.t. $x \dot{\neq} y$ with $o \in \mathcal{L}(x) \cap \mathcal{L}(y)$
- A node x is defined to be a *nominal* node if $\mathcal{L}(x)$ contains a nominal (i.e., $o \in \mathbf{I}$ and $o \in \mathcal{L}(x)$); that is, the node corresponds to a named individual. A node that is not a nominal node is defined to be a *blockable* node. A nominal $o \in \mathbf{I}$ is said to be *new in* G if no node in G has o in its label. In the remainder of this dissertation, nominal nodes will additionally be refereed to as *root nodes*.
- A node is said to be *label blocked* if it has ancestors x', y, and y' such that
  1. x is a successor x' and y is a successor of y'
  2. y, x and all nodes on the path from y to x are blockable
  3. $\mathcal{L}(x) = \mathcal{L}(y)$ and $\mathcal{L}(x') = \mathcal{L}(y')$
  4. $\mathcal{L}(\langle x', x\rangle) = \mathcal{L}(\langle y', y\rangle)$

  In this case, we say that *y blocks x*. Further, a node is *blocked* if it is label blocked or it is blockable and its predecessor is blocked; if the predecessor of a safe (defined next) node x is blocked, then x is said to be *indirectly* blocked
- A R-neighbor y of a node x is *safe* if 1) x is blockable or 2) x is a nominal node and y is not blocked

- During the tableau algorithm some nodes are *merged* into another node. Intuitively, when a node y is merged into a node x, $\mathcal{L}(y)$ is added to $\mathcal{L}(x)$, all edges leading to y are moved so that they lead to x, and all edges leading from y to some nominal node are moved so that they lead from x to the same nominal node. Additionally, y is *pruned* from the completion graph by removing y and all blockable sub-trees below y. The merging process is denoted by $Merge(y, x)$ and the pruning process is denoted by $Prune(y)$. A formal presentation of these operations is omitted here as an extension of them is provided in Chapter 5.3, where they are discussed in detail. Additionally, the original operations can be found in [78].

The tableau algorithm starts by initializing a completion graph G with a forest of nodes and edges, each corresponding to the nominals and nominal roles in the initial ABox. Each node label initially includes its corresponding nominal name, as well as the concept assertions for the specific nominal in the ABox. Similarly, the initial edges correspond to the role assertions between the nominals and are be labeled with the role names from explicit role assertions in the ABox.

The completion graph is then further constructed by repeatedly applying a set of tableau *expansion rules*, which add new structures (nodes, edges, and labels) to the completion graph when necessary. The $\mathcal{SHOIQ}$ tableau expansion rules are provide in Table 2.3. It can be seen that the expansion rules explicate the structure implied by concepts in node labels. For example, if a conjunction $C \sqcap D$ is in the label of a node, the $\sqcap$-rule ensures that both conjuncts are in the label as well.

The expansion rules are applied until a *clash* occurs or no other rules are applicable. If a clash occurs, then the algorithm will attempt *backtrack* to a non-determinstic choice that the clash is dependent on; this non-determinsm is introduced by the a variety of the expansion rules (e.g., the disjunction rule), and searching these non-determinstic choices is the cause of intractability in tableau algorithms.

If a clash-free completion graph can be built in which no further expansion rules

$\sqcap$-rule:	if 1) $C_1 \sqcap C_2 \in \mathcal{L}(x)$, x is not indirectly blocked and
	2) $\{C_1, C_2\} \nsubseteq \mathcal{L}(x)$
	then set $\mathcal{L}(x) = \mathcal{L}(x) \cup \{C_1, C_2\}$
$\sqcup$-rule:	if 1) $C_1 \sqcup C_2 \in \mathcal{L}(x)$, x is not indirectly blocked and
	2) $\{C_1, C_2\} \cap \mathcal{L}(x) = \emptyset$
	then set $\mathcal{L}(x) = \mathcal{L}(x) \cup C$ for some $C \in \{C_1, C_2\}$
$\exists$-rule:	if 1) $\exists S.C \in \mathcal{L}(x)$, x is not blocked and
	2) x has no S-neighbor y with $C \in \mathcal{L}(y)$
	then create a new node y with $\mathcal{L}(\langle x, y\rangle) = S$ and $\mathcal{L}(y) = C$
$\forall$-rule:	if 1) $\forall S.C \in \mathcal{L}(x)$, x is not indirectly blocked and
	2) there is an S-neighbor y of x with $C \notin \mathcal{L}(y)$
	then set $\mathcal{L}(y) = \mathcal{L}(y) \cup C$
$\forall_+$-rule:	if 1) $\forall S.C \in \mathcal{L}(x)$, x is not indirectly blocked and
	2) there is some R with $\mathsf{Trans}(R)$ and $R \sqsubseteq^{*} S$,
	3) there is an R-neighbor y of x with $\forall R.C \notin \mathcal{L}(y)$
	then set $\mathcal{L}(y) = \mathcal{L}(y) \cup \{\forall R.C\}$
choose-rule:	if 1) $(\leqslant S.C) \in \mathcal{L}(x)$, x is not indirectly blocked and
	2) there is an S-neighbor y of x with $\{C, \neg C\} \cap \mathcal{L}(y) = \emptyset$
	then set $\mathcal{L}(y) = \mathcal{L}(y) \cup \{E\}$ for some $E \in \{C, \neg C\}$
$\geqslant$-rule:	if 1) $(\geqslant nS.C) \in \mathcal{L}(x)$, x is not blocked and
	2) there are not n safe S-neighbor $y_1, ..., y_n$ of x with
	$C \in \mathcal{L}(y_i)$ and $y_i \neq y_j$ for $1 \leq i < j \leq n$
	then create n new nodes $y_1, ..., y_n$ with $\mathcal{L}(\langle x, y_i\rangle) = \{S\}$,
	$\mathcal{L}(y_i) = \{C\}$, and $y_i \dot{\neq} y_j$ for $1 \leq i < j \leq n$
$\leqslant$-rule:	if 1) $(\leqslant nS.C) \in \mathcal{L}(x)$, x is not indirectly blocked and
	2) x has more than n S-neighbors and there are two S-neighbors
	y, z of x with $C \in \mathcal{L}(y) \cap \mathcal{L}(z)$ and $z \dot{\neq} y$
	then 1) if y is a nominal node, then $Merge(z, y)$
	2) else if z is a nominal node or ancestor of y, then $Merge(y, z)$
	3) else $Merge(y, z)$
$\mathcal{O}$-rule:	if for some $o \in \mathbf{I}$ there are 2 nodes x, y with $o \in \mathcal{L}(x) \cap \mathcal{L}(y)$ and not $x \dot{\neq} y$
	then then $Merge(x, y)$
NN-rule:	if 1) $(\leqslant nS.C) \in \mathcal{L}(x)$, x is a nominal node and there is a blockable
	S-neighbor y of x such that $C \in \mathcal{L}(y)$ and x is a successor of y,
	2) there is no m such that $1 \leq m \leq n$, $(\leqslant mS.C) \in \mathcal{L}(x)$, and
	there exists m nominal S-neighbors $z_1, ..., z_m$ of x with $C \in \mathcal{L}(z_i)$
	and $z_i \dot{\neq} z_j$ for all $1 \leq i < j \leq m$
	then 1) guess m with $1 \leq m \leq n$ and set $\mathcal{L}(x) = \mathcal{L}(x) \cup \{\leqslant mS.C\}$
	2) create m new nodes $y_1, ..., y_m$ with $\mathcal{L}(\langle x, y_i\rangle) = \{S\}$,
	$\mathcal{L}(y_i) = \{C, o_i\}$ for each $o_i \in \mathbf{I}$ new in G and $y_i \dot{\neq} y_j$ for $1 \leq i < j \leq m$

Table 2.3: $\mathcal{SHOIQ}$ Tableau Expansion Rules

are applicable then the algorithm returns that the KB is consistent; otherwise, the KB is inconsistent. In the remainder of this dissertation, the notation *Comp*(K) will be used to denote the set of all complete and clash free completion graphs that can be built by the tableau algorithm; therefore, *Comp*(K) corresponds to all models of K.

In order to ensure termination, a set of *blocking* conditions (introduced earlier) are used, as well an expansion rule application ordering. The blocking conditions essentially guarantee that cycles are detected, and these blocking conditions differ depending on the particular DL (e.g., see [12, 76]).

Next, an overview of the specific expansion rule application order for the $\mathcal{SHOIQ}$ tableau algorithm are as follows [78]:

1. the $\mathcal{O}$-rule is applied with highest priority
2. next, the $\leqslant$ and *NN*-rules are applied (see [78] for further ordering on the application of these rules)
3. all other rules are applied with a lower priority

However, if the KB is expressed in either $\mathcal{SHIQ}$ or $\mathcal{SHOQ}$ (or one of their sub-languages), then this expansion rule ordering is unnecessary [78] and the algorithm proceeds in a similar manner as the original tableau algorithms for $\mathcal{SHIQ}$ [80] and $\mathcal{SHOQ}$ [77].

It is important to point out that the complexity of reasoning directly depends on the expressivity of the DL. For example, in $\mathcal{ALC}$ it has been shown that worst case complexity of consistency checking is PSpace-Complete [15] . However, for more expressive DL such as $\mathcal{SHOIQ}$, the worst case complexity is 2NExpTime [141].

Given the obvious worse case performance of DL reasoning, various optimizations have been investigated [73]. These include, but are not limited to, absorption [75], lazy unfolding [13], and domain/range tableau expansion rules [142]. A brief description of these optimizations is provided below.

- *Absorption*: This process performs syntactic transformation on the TBox before the tableau algorithm is run. The intuition behind the approach is to safely transform GCIs into primitive concept definitions [75]. This can reduce the portion of the TBox that must be internalized, and therefore these primitive concept definitions can be taken into account during the actual tableau algorithm (discussed next).

- *Lazy Unfolding*: Given a definitorial TBox T and ABox A, it is possible to eliminate the TBox by recursively substituting the concept names in the ABox with primitive concept names (those on the right hand side of the definitions in the TBox) [13]; this process is referred to as *unfolding* and avoids the internalization of the TBox. *Lazy unfolding* is a slight modification to this procedure, in which the unfolding is done on the fly during the tableau algorithm. In this case, a new tableau expansion rule (*unfolding*-rule) is considered, which will unfold a concept into its definition if one exists. As noted earlier, in practice the TBox is partitioned into two components, namely a general TBox and unfoldable TBox. While the general TBox is internalized, the unfoldable TBox is typically taken into account during the tableau algorithm using lazy unfolding.

- *Domain and Range Expansion Rules*: One very common construct used in OWL ontologies is that of specifying the domain and/or range of a property. When converting these OWL axioms to DL TBox axioms, they are converted to GCIs. For example, if an OWL ontology contains a property *hasProduct* and the range of this property is defined to be a *Product*, the resulting DL TBox axiom is $\top \sqsubseteq \forall hasProduct.Product$. An alternative technique to handling domain and range constraints has been proposed in which two additional tableau expansion rules for domain and ranges are introduced [142]. Intuitively, whenever a new edge is added to the completion graph, these rules add the concept corresponding to the domain and range to the appropriate nodes.

2.5 Belief Base Revision

Belief revision is the process of modifying existing beliefs (i.e., entailments) in a knowledge base with new information in a manner such that the resulting knowledge base is still consistent. Typically, the basic structure of beliefs assumed in belief revision models is a *belief set*, which is a deductively closed (and hence in general infinite) set of formulae. In the following discussions, a set of beliefs will be denoted by B, and the set of deductively closed consequences will be denoted by $Cn(B)$.

The most influential work in belief revision theory is the AGM model, in which three main change operations on belief sets are defined; namely, *expansion* (expanding a belief set with a new belief with no guarantee of consistency after the operation), *contraction* (retracting a belief) and *revision* (expansion with consistency after the operation) [7]. Only the expansion operation is uniquely defined; specifically, the expansion of B with belief α, denoted $B + \alpha$ is defined as follows:

$$B + \alpha = Cn(B \cup \{\alpha\})$$

In contrast, the revision and contraction operations are constrained by a set of postulates. The postulates proposed in the AGM model [7] are widely accepted for deciding if a revision operation is rational. Unfortunately, various issues are encountered when attempted to apply the AGM model to description logics; details can be found in Chapter 3.3.1.

Due to the difficulty of computing with belief sets, there has been substantial work on using *belief bases* as an alternative structure (e.g., [106, 48, 66, 107, 109]). Belief bases are not closed under logical consequence and are usually interpreted as basic beliefs from which additional beliefs (the belief set) can be derived. Similar to the AGM model, the three change operations have been adapted to belief bases, and the contraction and revision operations are not uniquely defined, rather constrained by a set of postulates. However, unlike the AGM model, different constructions for proposed contraction and revision operators for belief bases are not equivalent [106]. In the AGM model, as well as most constructions for belief base operations, revision operations are defined in terms of contraction and expansion operations using the *Levy identity* [93], where $*$, $-$, and $+$ are used to denote revision, contraction, and expansion respectively:

$$B * \alpha = (B - \neg\alpha) + \alpha$$

For the purpose of revision in description logics, this is important because in many DLs (e.g., $\mathcal{SHIF}$ and $\mathcal{SHOIQ}$) negation is not supported in arbitrary expressions (e.g. role axioms). This is discussed in more detail in Chapter 3.3.2.

The notion of *semi-revision* has additionally been investigated [68, 69], which differs from the traditional belief base revision model in that the added belief may or may not be accepted (this is because it may be later retracted in order to regain consistency). In the remainder of this chapter, the topic of belief base semi-revision is more formally introduced. First, the notation of *kernel contraction* is presented. Assume that we want to retract the belief α; the general idea behind the kernel contraction operator is that if at least one element from each minimal subset of a belief base that implies α is removed from the belief base, then the base will no longer imply α [68]. Given this insight, the notion of a *kernel operator* is defined as follows (originally defined in [68]):

Definition 1 *(Kernel Operation) The kernel operation* $\perp\!\!\!\perp$ *is defined such that for any set* B *of formulas and any formula* α, $X \in B \perp\!\!\!\perp \alpha$ *iff:*

1. $X \subseteq B$
2. $X \models \alpha$, *and*
3. *for all* Y, *if* $Y \subset X$, $Y \not\models \alpha$

$B \perp\!\!\!\perp \alpha$ is a referred to as a *kernel set*, and its elements are referred to as *α-kernels*.

Note that in general, there are arbitrarily many ways to resolve inconsistencies between new information and the current knowledge base. For example, if one holds the beliefs that (i) *Every TechnologyCompany is a Company* and (ii) *Apple Inc. is a TechnologyCompany*, and the knowledge base is revised with (iii) *Apple Inc. is not a TechnologyCompany*, there are a few options: one could either retract (i), (ii), or (iii); therefore a choice must be made. Given this, an *incision function*, defined below (originally defined in [68]), determines the choice in such cases; that is, it selects the formula to be removed from every α-kernel when we contract α.

Definition 2 *(Incision Function) An incision function for B is a function σ such that for any formula α:*

1. $\sigma(B \perp\!\!\!\perp \alpha) \subseteq \bigcup(B \perp\!\!\!\perp \alpha)$*, and*
2. *If $X \in B \perp\!\!\!\perp \alpha$ and $X \neq \emptyset$, then $X \cap \sigma(B \perp\!\!\!\perp \alpha) \neq \emptyset$*

Kernel semi-revision is then defined as follows (originally defined in [69]):

Definition 3 *(Kernel Semi-Revision) The kernel semi-revision operator of B based on an incision function σ is denoted by $?_\sigma$ and defined such that for all sentences α:*

$$B?_\sigma\alpha = B \cup \{\alpha\} \setminus \sigma((B \cup \{\alpha\}) \perp\!\!\!\perp \bot)$$

This can be thought of as a two-step process: first add α to B, and second, remove inconsistencies in B if there are any. The name "semi-revision" comes from the fact that in the revision process, the formula α that we revise our knowledge with may not be accepted. In other words, α might be removed as part of the second step.

Lastly, the following postulates must be satisfied for any operator to be considered a kernel semi-revision operator (originally defined in [69]).

Proposition 1 *An operator $?$ is a kernel semi-revision operator if and only if for all set B of sentences it satisfies the following postulates:*

1. $\bot \notin Cn(B?\alpha)$ *(consistency)*

2. *$B?\alpha \subseteq B \cup \{\alpha\}$ (inclusion)*

3. *If $\beta \in B \setminus B?\alpha$, then there is some $B' \subseteq B \cup \{\alpha\}$ such that $\perp \notin Cn(B')$ and $\perp \in Cn(B' \cup \{\beta\})$ (core-retainment)*

4. *$(B + \alpha)?\alpha = B?\alpha$ (pre-expansion)*

5. *If $\alpha, \beta \in B$, then $B?\alpha = B?\beta$ (internal exchange)*

Justification for Description Logic Knowledge Bases

In this dissertation, a belief base revision algorithm for OWL-DL is developed (see Chapter 7.2). From the discussion presented above, it is clear that it is necessary to determine the set of assertions in a DL knowledge base that cause the inconsistency. In DL literature, this task has been investigated in the context of computing all extension of default theories [14]. More recently, this work has been extended for the purpose of debugging and repairing OWL ontologies [131, 85]. The set of axioms responsible for an arbitrary entailment is commonly referred to as a *justification* and can be formally defined as follows (defined in [85]):

Definition 4 *(Justification) Let $\mathsf{K} \models \alpha$ where α is a sentence. A fragment $\mathsf{K}' \subseteq \mathsf{K}$ is a justification for α in K if $\mathsf{K}' \models \alpha$, and $\mathsf{K}'' \not\models \alpha$ for every $\mathsf{K}'' \subset \mathsf{K}'$.*

It has been shown that a single justification can be provided by slightly modifying the tableau algorithm to track the propagation of structures in the tableau completion graph, which is referred to as axiom tracing (see Chapter 5 for a detailed discussion).

Further, it is possible to find all justifications [85, 130, 129] by coupling axiom tracing with the Reiter's Hitting Set Tree (HST) algorithm [123] (a similar approach is used in [147] for propositional calculus). The intuition behind the usage of the HST algorithm relies on the fact that, in order to remove an inconsistency from a KB, one needs to remove from KB at least one axiom from each justification for that inconsistency. The approach starts by adding the negation of α and finds an initial justification (subset of K) using axiom tracing [85]. Following this, a hitting set tree is initialized with the initial justification as its root; then it selects an arbitrary axiom (call it a) in the root and generates a new node with an incoming edge whose label corresponds to the removed axiom. The algorithm then tests for consistency with respect to the $\mathsf{K} \setminus \{a\}$. If it is inconsistent, then we obtain another justification for α w.r.t $\mathsf{K} \setminus \{a\}$. The algorithm repeats this process, namely removing an axiom, adding a node, checking consistency and performing axiom tracing until the consistency test turns positive. Further, details can be found in [85]. In the remainder of this dissertation, the set of all justifications for the entailment of α in a $\mathcal{SHOIQ}$ knowledge base K will be denoted as $\mathit{Just}(\mathsf{K}, \alpha)$. For purpose of this work, we are only concerned with ABox assertions, therefore it is assumed $\mathit{Just}(\mathsf{K}, \alpha)$ simply contains the ABox assertions in the justification.

Chapter 3

Related Work

This dissertation brings together the fields of knowledge representation languages and syndication systems for the Web. In this chapter, related work in both of these fields is presented.

3.1 Syndication Systems

In the following discussion, an overview of syndication frameworks previously investigated is presented. The overview demonstrates the recent trend toward more expressive syndication approaches.

3.1.1 Syntactic Approaches

Early syndication systems primarily relied on subject-based keywords in order to match user interests with published documents/data [113, 150]. In such an approach, publishers associate some number of predefined keywords with publications, and similarly, subscriptions are represented using keywords. Therefore, matching publications with subscriptions requests reduces to keyword matching. The main limitation of such an approach is its simplistic use of keyword matching.

Following this, there were efforts to capture more detailed user interests by allowing attribute-value pairs to be associated with published content [5, 43, 33, 28, 29, 43]. In such an approach users are allowed to provide values for certain attributes, and matching reduces to determining the satisfaction of attribute values with associated publications. While this provides further filtering capabilities and more accurate dissemination, the approach is still limited by the predefined attributes and the simple matching of attribute values. In order to address the scalability of both of these two approaches, distributed architectures have also been investigated (e.g., [21, 150]).

Over the past few years there has been substantial interest in utilizing XML for filtering purposes in syndication systems (e.g., see [8, 30, 60, 91, 38]). In such an approach, published documents/data are represented in XML and subscription requests are specified using an XML query/path language (e.g., XPath [30]). This approach provides a variety of benefits including the enables the ability to enforce published content validation (using XML Schema) and provides richer subscription requests (via XML path query languages), etc. The popularity of such approaches is exemplified by the number of RSS 2.0 [95] and Atom 1.0 [112, 57] feeds (both of which are XML based), which are growing at ever increasing rates. Recently, there has also been work addressing the scalability of such approaches by proposing distributed XML-based syndication architectures (e.g., see [38, 151]).

3.1.2 Semantic Approaches

Given the limited expressivity of XML (and XML Schema) as a knowledge modeling language and the fact that it does not have inferential capability, there has been interest in using formal knowledge representation languages for representing published contents. One such approach is provided in the event-based dissemination platform CREAM [33]; in this platform, the syndication model is attribute-value pair based, however attributes can be associated with semantic information described in an ontology. This allows the definition of concept hierarchies and therefore simple inferencing to determine subscription matches.

Given the recent standardization of the knowledge representation language RDF (and RDFS), there has been increased interest in using RDF [100] for syndication purposes; RDF is the underlying representation format of RSS 1.0. A variety of architectures have been proposed (e.g., [120, 121, 145]), all of which use RDF the representation language for publications. In such approaches, RDF graph-based query languages (typically triple patterns) are used to represent subscription requests and matching publications with subscriptions reduces to triple pattern matching. Additionally, in most approaches RDFS is also utilized to describe domain ontologies which the published RDF content adheres to. This allows simple semantic inferences (e.g., via subclass relationships) to be made over publications, similar to the approach adopted in CREAM.

There has also been work on addressing the scalability of an RDF-based approach by leveraging distributed syndication archictectures. For example, [32] present a distributed RDF publish/subscribe approach, in which a peer-to-peer (P2P) architecture is utilized. Specifically, a super-peer [34, 110] architecture is proposed, where super-peers in the network are responsible for transfering publications to the correct subscribers, which are regular peers in the network.

Description Logic Approaches

Over the past few years, there has additionally been work on using description logics for a variety of application scenarios which are directly or indirectly related to syndication systems. In the following discussions an overview of this work is presented.

Concept-Based Approaches. In [143, 94] the authors use a DL-based approach for Web service matching and information syndication respectively, in which DL concepts (possibly complex concepts) are used to represent both subscription requests as well as published documents/data. In such an approach, matching published content with subscription requests reduces to determining if published concepts and subscriptions are logically equivalent, subsume one another, or are not compatible (discussed below). More specifically, [94, 115] define there to be a match between a subscription S and published document D if one of the following holds (in order of match strength):

- *Exact* ($S \equiv D$): The subscription and the published concepts are equivalent concepts, and it is referred to as an *Exact* match.

- *PlugIn* ($S \sqsubset D$): The subscription is a sub-concept of the published concept, and it is referred to as a *PlugIn* match.

- *Subsume* ($D \sqsubseteq S$): The subscription is a super-concept of the published concept, and it is referred to as a *Subsume* match.

- *Intersection* ($\neg(S \sqcap D \sqsubseteq \bot)$): The intersection of the subscription and the published concept is satisfiable, and it is referred to as an *Intersection* match.

Matching subscription requests is therefore accomplished by two reasoning tasks; namely classifying the KB when either subscription requests or new data is received and by performing concept satisfiability tests for the intersection of each subscription and published concept. While empirical results demonstrate accepted performance times (~20 ms) for matching new subscription requests with a fixed set of published documents, processing a large amount of incoming published content is still problematic (~10s of seconds) [143, 94]. This is due to the cost of repeatedly reclassifying the published concepts; this problem is compounded if syndication brokers have domain knowledge (potentially very large) with which newly published data must be integrated.

Query-Based Approach. [61, 62] presents an agent-based document retrieval system (which is essentially a publish/subscribe application) in which published contents are represented as ABox assertions and subscription requests as a DL concept (viewed as an instance retrieval query). Therefore, matching is reduced to instance retrieval of the subscription concept. Given the performance issues of using such an approach (i.e., response times in the 10s of seconds), the authors introduce two optimizations for more effective incremental instance retrieval, both of which are discussed below:

- *Query Ordering*: In query ordering, a partial ordering is induced upon all registered subscription concepts based on their subsumption relations; these subsumption relations are determined by classifying the atomic subscription concepts. In the approach more general subscriptions are answered first, thereby reducing the number of individuals that must be considered for more specific queries (due to the subsumption relation).

- *Candidate Individual Reduction*: Candidate individual reduction is the process of not considering previous individuals which satisfied registered queries; that is, once an individual has satisfied the subscription concept, it does not need to be reconsiders when new data is published. This holds, due to the monotonicity of the DLs considered in the work and the fact that deletions are not supported.

3.1.3 Discussion

In this dissertation, a more expressive means is investigated for representing published contents. This allows the use of automated reasoning procedures to infer matches not found using syntactic approaches (keyword, attribute-value pair, and/or XML) and simpler semantic approaches (e.g., RDF/S). While there has been work on application scenarios which are related to DL-based publish-subscribe systems, the related techniques either take a different approach for representing published contents and subscriptions requests (e.g., [143, 94]) or assume a simpler subscription format with only atomic concepts (e.g., [61, 62]). Further, in the syndication framework proposed in this dissertation, it is assumed the publications are encoded in OWL, which provides additional benefits (discussed in Chapter 1).

Another general distinction between the syndication framework developed in this dissertation and a majority of those presented above is that the developed approach allows for publications to persist at the syndication broker for varying time frames; this in turn allows composite subscription matches, in which the information contained in multiple publications comprises a publication match (see Chapter 4 for details).

It is lastly noted that there has been additional work on determining optimal subscription evaluation orders for syndication systems. For example, in [98] a generic publish/subscribe architecture is proposed, in which expensive filters (i.e, subscriptions) can be used to filter published contents; an example of these expensive filters are pattern recognition operations over streaming video feeds. One of the main contributions of [98] is an approximation algorithm for finding the (near) optimal subscription evaluation ordering, which exploits the overlap of registered subscriptions. In contrast, in this dissertation the scalability of an OWL-based syndication framework is addressed by developing optimized incremental DL reasoning services. This focus was taken as even a single subscription in an OWL-based syndication can take 10s of seconds to evaluate.

3.2 Stream Processing Engines

In the following discussion, related work on stream processing engines is presented. This discussion additionally addresses work on continuous query answering for various data models/representation formats.

Relational Model

In the relational database field, there has been substantial work on incremental maintenance of database views and queries. Much of this work initially focussed on developing more efficient techniques for incrementally updating materialized database views and integrity constraints (e.g., [23, 138]). There has additionally been substantial work on providing more effective continuous query answering techniques, in which queries are assumed to persist for long periods of time [139, 132, 11, 97, 99]. Continuous queries in this context are reminiscent of registered subscriptions at the syndication broker, in that they persist for varying time periods. Given the overhead of reissuing such queries after each update to the database, the focus of this work has been on optimizing this process. For example, in [99] an approach for sharing the necessary processing of relational operations across multiple continuous queries is presented.

More recently, there has been increased interest in stream processing engines, in which streams are composed of relational tuples and relational operators can be placed at various points of the stream to filter/join tuples [27, 3, 20, 148]. Further, various types of relational stream operators have been introduced, including sliding windows that allow operations to be performed on varying sized subsets of the current stream contents.

While similar in some aspects, continuous query answering and stream processing in the context of relational databases/streams is inherently different from continuous query answering in the syndication framework proposed in this dissertation; this is because a more expressive formalism is assumed for representing published information. This, in turn allows the use of logical inferences when determining subscription matches in the developed syndication systems, which are not possible by directly using a relational model. Importantly, this impacts reasoning complexity; specifically, reasoning algorithms for OWL Lite and OWL DL are not polynomial, which is generally the complexity of relational database query languages.

XML

There has additionally been substantial work on XML stream processing and continuous query answering (e.g., [31, 111]). Similar to the work done in the context of the relational model, the focus of this work is also on developing more effective techniques for evaluating continuous queries as the underlying XML database or stream changes. For example, [31] proposes various approaches for grouping continuous XML queries, thereby reducing the overhead of re-evaluating them as the underling data changes. As in the relational context, the work done in this dissertation addresses a different knowledge representation formalism.

Deductive Databases

There has been substantial work on incremental view maintenance in deductive database systems (e.g., [40, 41, 9, 89, 59, 71, 144, 137]). Much of this work has proposed approaches in which only the ground facts in the database can change, whereas the rules are assumed to be fixed [9, 89, 59, 71]. The techniques investigated have ranged from maintaining dependencies for derived facts [9], to computing the difference between consecutive database states [89]. Further techniques have been investigated in which the propagation of changed (added/removed) facts is determined via a delete, re-derive and insert approach [59, 144, 137, 71]. The underlying idea is to first select an overestimate of the intentional facts that should be deleted, as they are dependent on a deleted fact. This is an over-estimate as these facts can potentially be re-derived by additional facts in the database, and therefore the second step takes this into account. Lastly, the insertion step adds new facts to the database.

There has been recent work on extending incremental view maintenance to support changes to the rules in the database as well [144]. This work can be seen as an extension of the delete, re-derive and insert approach, in which the changes to the rules are taken into account; this is accomplished by updating the materialization of predicates that must be maintained in the view and by updating the original maintenance programs used to update the views (see [144] for additional details).

The techniques proposed in this dissertation are similar to some of the approaches discussed above; for example, the incremental consistency checking approach (discussed in Chapter 5) tracks the dependencies of structures in tableau completion graphs which intuitively correspond to inferred facts. In general, however, the approaches developed in this dissertation are in applicable to a different knowledge representation formalism.

Generalized Approaches

I lastly note that there has also been recent work on developing more generalized stream processing frameworks, which do not impose a particular representation framework on stream elements (e.g., [1]). In contrast, the focus of this dissertation is scoped to addressing the challenges in achieving a practical syndication framework with an explicit OWL representation format. However, the contributions in the dissertation will be clearly usable within the context of a more generalized syndication framework, if one chooses to utilize an OWL (or DL) based representation.

3.3 Revising and Updating Logical Knowledge Bases

There have been numerous approaches investigated in literature for revising and updating logical knowledge bases. Recently, there have been attempts to apply these approaches to description logic knowledge bases, and an overview of some key approaches is now presented.

3.3.1 AGM Belief Revision Theory

Belief revision is the process of modifying existing beliefs in a knowledge base to take into account new pieces of information. This revision process is typically only necessary when the new information is inconsistent with existing knowledge. The most influential work in belief revision is the AGM model [7], in which the authors introduce three main change operations on belief states, which are represented by logically closed sets of sentences (referred to as *belief sets*). As discussed in Chapter 2.5, the three operations of change are *expansion* (expanding a belief set with a new belief with no guarantee of consistency after the operation), *contraction* (retracting a belief) and *revision* (expansion with consistency after the operation). Additionally, the authors define a set of postulates for both contraction and revision, which specify the properties a contraction/revision operator must meet in order to be rational; these postulates are widely accepted and assumed in many belief revision approaches.

There has recently been interest in applying traditional belief revision approaches based on the AGM postulates to description logic knowledge bases [47, 46, 45] (including those corresponding to OWL Lite and OWL DL). Specifically, in [45, 46] the authors define a class of AGM-compliant logics, which can satisfy the AGM postulates for contraction. Additionally, the authors show that the description logics $\mathcal{SHIF}$ and $\mathcal{SHOIN}$ do not fall in the AGM-compliant class of logics; this negative result implies that the traditional AGM belief revision theory cannot be applied to OWL Lite or OWL DL. Given this result, additional sets of postulates have been introduced to replace the AGM-postulates, such that the approach can be applied to a larger class of logics (including OWL Lite and OWL DL) [47, 126].

3.3.2 Belief Base Revision

It is documented in literature that one of the main problems with the AGM model of belief revision is related to the difficulty of computing with belief sets. Given this, there has been substantial work on using *belief bases* as an alternative structure [106, 48, 66, 107, 109]. Belief bases are not closed under logical consequence and are usually interpreted as basic beliefs from which additional beliefs (the belief set) can be derived. In [48, 67, 70] the traditional AGM change operators are defined in terms of belief bases.

There has been recent work on applying traditional belief base revision algorithms to description logics. In [44], the author follows the base revision approach presented in [48] and again finds negative results, in that $\mathcal{SHIF}$ and $\mathcal{SHOIN}$ cannot satisfy the belief base revision postulates originally presented in [48].

Given these negative results, there has been recent interest in applying belief base *semi-revision* [69] to DL KBs. Semi-revision differs with the traditional belief base revision model in that the added belief may or may not be accepted (this is because it may be later retracted in order to regain consistency). In literature, semi-revision has successfully been applied to propositional logic. For example, [147] presents an algorithm for belief base semi-revision for propositional logic based on the construction presented in [69], and the author shows how the diagnosis problem as described by [123] can be used to provide a semi-revision algorithm for propositional logic.

As stated above, there has been recent work on applying semi-revision techniques to description logics. In fact, the algorithm presented in Chapter 7.2 is such an approach and is an extension of the work presented in [147]. Additionally, [125] presents two different constructions based on semi-revision, both of which aim to ensure that the new belief is entailed by the revised KB. In the first approach, it is guaranteed that the new belief will be entailed by the knowledge base, unless that new belief is inconsistent. In the second construction, the new belief will always be entailed after revision, however if the new belief is inconsistent then the revised knowledge base is inconsistent as well. The main distinction with the approach presented in this dissertation is that while it is ensured that the KB is consistent after the revision, the new belief may not be entailed by the revised KB. It is argued in this dissertation that this is in fact a desirable effect for syndication systems (see Chapter 7.2 for a more detailed discussion).

3.3.3 Logical Updates

In literature, there is a distinction between belief update and belief revision; specifically, belief revision is the task of changing ones beliefs about a static world in light of new information, where as belief update is the task of incorporating new information into a changing work [119]. Historically, determining the choice of which semantics to adopt is largely application dependent and often debatable.

Over the past two decades, there have been a variety of update semantics proposed in the context of logical databases [149, 124, 88, 4] and in first order logic (for example, situation calculus [102] can be adapted to first order logic). Recently, there has been work in addressing updates in expressive DLs [96, 127, 49]. In [96], the authors provide an approach for updating DL ABoxes under simple ABox updates, which are restricted to assertions of the form $A(a)$ or $R(a, b)$, where A is an *atomic* concept. Further, the authors adopt the standard model based update semantics, as described in literature [149, 124, 128], in which models of the KB are minimally changed.

One very interesting finding in [96] is that in DLs which are less expressive then $\mathcal{ALCO}^{@}$, updates are not representable; more specifically the authors show that in order to respresent updates in a variety of DLs, nominals and the "@" operator from hybrid logic [10] must be included in the logic. This implies that both $\mathcal{SHIF}$ and $\mathcal{SHOIN}$ (and therefore OWL Lite and OWL DL) cannot represent minimal model change updates without the "@" operator. Additionally, the authors show that an exponential increase in the size of input (original ABox and the update) cannot be avoided. It is also shown that under these update semantics, even in propositional logic, an exponential increase in size of the whole input cannot be avoided. I do note that the authors show that if additional concepts can be introduced into an acyclic TBox, the updates are polynomial in size of the original ABox.

[49] additionally investigates the minimal model change update semantics, however considers a less expressive DL, referred to DL-Lite [25]. Due to the limited expressivity of the language (e.g., disjunctions are not allowed), reasoning can be shown to be tractable. [49] is able to show that in this DL, the minimal model change update semantics can in fact be represented. However, in the syndication framework presented in this dissertation, I aim to support more expressive DLs corresponding to larger portions of OWL.

3.3.4 Repairing Description Logic Knowledge Bases

There has been recent work on debugging and repairing description logic knowledge bases. In [131], the author presents an approach to identify the minimal set of (base) axioms (i.e., a justifcation) that cause an unsatisfiablie concept in KBs expressed in $\mathcal{ALC}$. The authors continue this work and utilize Reiter's HST algorithm to compute repair plans, given the conflicting sets [129, 130].

Recently, this work has been extended to more expressive DL KBs, effectively covering all of OWL DL [85]. In order to detect the set of axioms responsible for unsatisfiable concepts (and arbitrary entailments), [85] extends work on axiom tracing [14], in which dependencies for inferences made during tableau algorithms are effectively traced. Leveraging this technique, [85] employs Reiter's HST algorithm to find all justifications for the entailment (see Chapter 2.5 for a more detailed discussion). [85, 86] additionally develops techniques for ranking axioms in order to develop repair plans which are found using a slightly modified HST algorithm. In order to repair unsatisfiable concepts (or inconsistencies in general), [86] introduces the following ranking metrics which contribute to the generated repair plans:

- Axioms frequency in justifications
- Impact on the ontology when the axiom is removed (i.e., number of entailments which are lost)
- User specified entailments which should be less likely to be lost
- Axiom relevance to the ontology in terms of usage
- Provenance information about the axiom (e.g., author)

There are a variety of similarities between this work and the belief base revision algorithm presented in Chapter 7.2. Specifically, the technique presented in this dissertation leverages the technique for finding all justifications presented in [85]. However, in this dissertation the problem is formalized in the context of belief base revision; additionally, the notion of trust is exploited to select which assertions should be removed from the set of justifications which lead to the inconsistency.

3.4 Truth Maintenance Systems

Over the past few decades, there has been extensive work in Truth Maintenance Systems (TMSs) for logical theories (see [101] for a survey). As the name implies, TMSs are used to more efficiently maintain beliefs through changes caused by the introduction of new information. Generally, TMSs are geared toward propositional logic and approaches for both monotonic and non-monotonic constructs have been investigated.

Many TMSs are justification-based [42], and in such an approach, the dependencies for beliefs are maintained in the system. Then, in the event of a deletion, the invalidated beliefs can be determined using the sets of justifications (by checking if the justification includes a deleted sentence). In order to overcome some of the performance issues with justification-based TMSs, there has additionally been substantial work on assumption-based TMSs (ATMS) [36]. One of the main differences with ATMSs is that rather than maintaining the entire justification, only the assumptions which an inferred statement is dependent on is stored. An additional distinction is that ATMSs typically provide support for multiple contexts, meaning the consistency of the knowledge base is considered with respect to only a subset of the fact maintained (i.e., the context).

When comparing the contributions of this dissertation with TMSs, it can be seen that there are various similarities. In particular, the notion of justifications is used for incrementally maintaining tableau completion graphs (see Chapter 5). One important distinction is that traditionally TMSs only support propositional logic; however, in this dissertation a more expressive formalism is supported. Additionally, in the approach presented in this dissertation, all justifications are not maintained, rather only the dependencies for occurrences of structures in a single model (which the completion graph corresponds to). It is noted that only monotonic DLs are addressed in this dissertation.

3.5 Incremental Description Logic Reasoning

While there has been substantial work on optimizing reasoning services for description logics (see [73] for an overview), the topic of reasoning through evolving DL knowledge bases remains relatively unaddressed. There are a few notable exceptions which are introduced here.

As mentioned earlier, [62] presents a DL-based publish/subscribe system in which the subscriber registers queries (restricted to single, named concepts) that model their interests and published data is modeled as ABox assertions. [62] also presents two optimizations, namely inducing a partial ordering upon all registered queries (i.e., atomic concepts) and disregarding previous individuals that satisfy registered queries when data is published. This is directly related to the techniques presented in this dissertation. However, I additionally address incremental consistency checking and present novel techniques to prune the individuals in the KB that must be considered for queries after updates are developed; further, the approach supports conjunctive queries.

There has been recent work on optimizing classification of DL KBs in the presence of arbitrary TBox and ABox changes [56, 116]. In [116], the authors present set of techniques for avoiding subsumption tests when re-classifiying concepts in DL KBs expressed in $\mathcal{SHOIN}$ (i.e., OWL DL). For example, in the presence of additions, due to monotonicy of the DL considered, previous sumbsumption tests can be avoided. Additionally, the authors propose caching portions of tableau completion graphs (in literature referred to as pseudo models) built during the previous subsumption checks; after an addition, if these pseudo models have not changed, then subsumption checks can be avoided.

[56] exploits the notion of ontology modularity to localize the portions of the KB which must be considered for re-classiciation after changes. Specifically, modules for concepts in the KB are maintained through updates, and the authors are able to show that it suffices to only reclassify the axioms contained in the affected modules, which generally are very small in comparison to the entire KB (see [56] for additional details).

The techniques presented in this dissertation address different reasoning services, as they are necessary to make the developed syndication framework practical.

Chapter 4

Syndication Framework

4.1 Overview

In this chapter, I present a novel OWL-based syndication framework. As in typical syndication systems, I assume that a collection of information producers publish content to a syndication broker, and similarly a collection of subscribers registers their interests with the broker. Therefore, it is the syndication broker's task to disseminate relevant information to the appropriate subscribers, based on their subscription requests.

In the framework, all publication are encoded in the OWL representation language, specifically as OWL individual assertions (i.e., type, property, equality, and inequality assertions). Further, the syndication broker maintains an OWL ontology with which newly published content is integrated. The broker has a fixed schema and initial set of instance assertions that encode background domain knowledge and can therefore be used to further process publications (e.g., via inferences enabled by the intensional information defined in the class and property axioms). In order to represent subscribers' interests in published content (i.e., OWL instance assertions), subscription requests are represented as conjunctive instance queries (specified using standard OWL query languages, such as SPARQL [122] or RDQL [133]), which are registered with the syndication broker. This is intuitive as subscriptions are intended to ask for information matching a set of parameters. Given this, matching subscription requests with newly published information reduces to OWL query answering.

In order to allow automated inferences using decidable OWL reasoners during the matching process, the usage of OWL constructs within the framework (i.e., in the syndication brokers local KB and published information) is assumed to fall within one of the

DL sub-languages of OWL: OWL Lite, OWL DL, or some subset of these languages. In order to enable the use of DL reasoners for the actual matching process via OWL query answering, it is assumed that the registered subscriptions can be translated into conjunctive ABox queries (as described in Chapter 2.4.2). This assumption is realistic as today's OWL reasoners perform this translation automatically.

Intuitively, this OWL-based syndication framework can be reduced to a DL-based syndication framework, in which conjunctive query answering over DL KBs (specifically the brokers local KB) is the means for determining subscription matches; given this, the formalization of the framework is presented in terms of DLs. It is important to note that due to the alignment of OWL with DLs, this can analogously be viewed as an OWL-based syndication framework.

4.2 Framework Formalization

4.2.1 Publishing

In the framework, a *publisher* is defined to be identified by a unique identifier:

Definition 5 *(Publisher) A publisher* Pub *is defined to be composed of and identified by a unique identifier i.*

As discussed above, a *publication* is a set of ABox assertions (which correspond to a set of OWL instance assertions); as discussed in Chapter 2.4, ABox assertions can take the form of individual type, property, equality and inequality. In the framework, a publication is also associated with a number of time units in which the publication is valid; after the specified time units have passed, the publication is discarded from the syndication brokers KB (discussed below). Additionally, a boolean value is associated with a publication, denoting if the assertions should be added (or retracted) to (respectively from) the brokers KB. Retractions are supported as in many realistic syndication applications, revisions to previous publications are sometimes necessary; such a revision can be viewed as a deletion followed by an addition. Lastly, a publication is associated with an identifier of the publisher that produced the information. Given this, a publication is formally defined as follows:

Definition 6 *(Publication) A publication* P *is defined as a tuple* (β, t, v, p)*, where* β *is a set of DL ABox assertions that expire after t time units (*$t > 0$*), v is a boolean value that is true in the event of an addition and false for retractions, and p is the identifier of the publisher that produced the publication.*

Given a publication P, we denote the set of ABox assertions as $\mathsf{P}(\beta)$, the expiration time as $\mathsf{P}(t)$, the boolean addition/retraction value as $\mathsf{P}(v)$, and the publisher that produced the publication as $\mathsf{P}(p)$.

4.2.2 Subscribing

Within this framework, the main component of a subscription is a conjunctive ABox query, which represents the subscribers interests. In the framework, subscriptions are represented as retrieval queries (see Chapter 2.4.2 for a distinction between retrieval and boolean queries); this assumption is made for a variety of reasons, which are discussed in section 4.2.4. Additionally, a subscription is composed of the number of time units that the subscription is valid. Intuitively, the subscription query can be thought of as a *continuous conjunctive query* (defined below) that should be evaluated for the specified number of time units. Therefore, the query is issued once over a changing KB and the answer set of the query is continuously updated as the ABox changes.

A continuous conjunctive query is denoted by Q_c and is syntactically equivalent to conjunctive ABox retrieval queries (introduced in Chapter 2.4.2). The answer set of a continuous conjunctive retrieval query at time t is the set of all distinguished variable substitutions entailed by the KB at time t:

Definition 7 *(Continuous Conjunctive Retrieval Query) Given a continuous conjunction ABox retrieval query Q_c with n distinguished variables (i.e., $DVar(Q_c) = \{d_1, ..., d_n\}$), define the answers of* K *at time t, denoted K_t, to Q_c to be those n-tuples $(a_1, ..., a_n) \in \mathbf{I}^n_{\mathsf{K}_t}$ such that the following holds:*

$$\mathsf{K}_t \models Q_c[d_1/a_1, ..., d_n/a_n]$$

In the remainder of this dissertation, when referring to a continuous conjunctive retrieval query, "continuous" or "incremental" query will be used. Given a continuous query, denote the set of answer tuples at time t by $Q_c(t)$.

A *subscription* is assumed to be composed of a continuous query, in addition to a number of time units which the query should be evaluated:

Definition 8 *(Subscription) A subscription* S *is defined as a pair (Q_c, t), where Q_c is a continuous query that is evaluated for t time units ($t > 0$).*

The continuous query of a subscription is denoted as $\mathsf{S}(Q_c)$ and the expiration time is denoted as $\mathsf{S}(t)$. Next, a *subscriber* is be introduced and intuitively is composed of a set of subscriptions and a unique identifier:

Definition 9 *(Subscriber) A subscriber* Sub *is defined to be a pair (s, i), where s is a set of subscriptions and i is a unique identifier.*

Similar to publishers and subscriptions, $\mathsf{Sub}(s)$ and $\mathsf{Sub}(i)$ denotes a subscriber's set of subscriptions and identifier respectively.

4.2.3 Matching

A *syndication broker* maintains a local DL KB, in which newly published information is integrated. In the framework this KB can initially contain a fixed TBox and ABox providing background domain information. Additionally, the syndication broker maintains the currently registered subscribers, which have associated subscriptions, and publishers. This is formally defined as follows:

Definition 10 *(Syndication Broker) A syndication broker* B *is defined as a triple* (S, P, K)*, where* S *is a set of subscribers,* P *is a set of publishers, and* K *is the broker's local DL KB.*

A syndication broker's subscribers, publishers, and KB are denoted as $\mathsf{B}(S)$, $\mathsf{B}(P)$, and $\mathsf{B}(\mathsf{K})$ respectively. If it is clear from the context of the discussion, the broker's KB (resp. KB at time t) will simply be referred to as K (resp. K_t). Additionally, the notation $\mathsf{B}(\mathsf{K}_\mathsf{P})$ will be used to denote the set of ABox assertions present in the broker's KB that are included in a non-expired publication.

After a new publication is received, it is the broker's task to determine the subscribers for which this new information is relevant. Prior to doing this, the new publications must be integrated in the broker's KB. As discussed in Chapter 3.3, there have been a variety of semantics investigated related to how this new information can be taken into account. Further, as pointed out previously, there have been various negative results regarding the application of the most common theories for belief revision and update semantics to OWL knowledge bases (and the DLs that they correspond to).

In this framework a slightly different approach is utilized; specifically, a syntactic change/update of ABox assertions is adopted, referred to as *Syntactic Updates*. By *syntactic*, we refer to the explicitly asserted ABox facts; this is similiar to the distiction between belief bases and belief sets in belief revision literature [108]. Intuitively, *Syntactic Updates* can be described as an update in which all new ABox assertions are directly added (or removed) to the asserted (base) axioms; therefore the only changes that occur are those explicitly stated in the ABox update. Using this approach to add new assertions can lead to an inconsistency; this issue is not addressed here, however Chapter 7 presents a variety of approaches to regain consistency. Further, removing an assertion from the

ABox under these semantics does not guarantee that the removed assertion will not be entailed anymore. Formally, this type of update is defined as follows:

Definition 11 *(Syntactic Updates) Let* A *be the ABox of DL KB* K*. Then under syntactic updates, updating* K *with an ABox addition (resp. deletion)* β*, written as* $\mathsf{K}{+}\beta$ *(resp.* $\mathsf{K}{-}\beta$*), results in an updated set of ABox axioms* A' *such that* $\mathsf{A}' = \mathsf{A} \cup \{\beta\}$ *(resp.* $\mathsf{A}' = \mathsf{A} \setminus \{\beta\}$*).*

If the update type (i.e., addition or deletion) is clear from the context of the discussion, $\mathsf{K} \oplus \beta$ will simply be used to denote the syntactic update of K with β. This type of change to the broker's KB is adopted in this dissertation for various reasons. First, it is fitting for the syndication application. Additionally, it is representable in the representation languages considered in this dissertation, while many other leading theories of change have been shown to be inapplicable to OWL Lite and OWL DL. Further, as will be shown the remainder of this dissertation, adopting this type of semantics leads to a practical syndication framework which is empirically shown to be scalable.

Due to the fact that the syndication broker can have initial ABox assertions in its knowledge base prior to any publications, under these update semantics an ambiguity can arise if a retraction publication is received by the broker which contains an assertion that is present in the initial ABox; namely, one could choose to remove this assertion or decide to leave it as it is assumed to be fixed background domain knowledge. In the remainder of this dissertation, it is assumed that this retraction will not remove this assertion as it is considered background information. In general this decision is application dependent and should be decided depending on the syndication use case.

Let us now consider subscription matches. As information is published from multiple publishers and can remain valid in the broker's local KB for varying time periods, a match for a subscription can be a composition of the information from multiple publications; that is, the information provided in multiple publications collectively forms a match for the query. Further, the broker's local knowledge base could additionally contain background knowledge which can attribute to subscription matches as well. Recent approaches have not investigated such functionality; rather, only information from individually published documents form a match for a given subscription. Such a capability is beneficial, as information can be considered collectively and form matches not found otherwise (examples are discussed later).

A distinction between two types of subscription matches is made in this formalization, namely *information* and *publication* matches. Intuitively, an *information match* refers to the individuals bound to the distinguished variables of a continuous query representing a subscription; that is, the result returned to the subscriber is actually the query answer rather than the publication(s) responsible for the answer. This type of match aligns with recent work in XML-based syndication literature, in which the actual information is filtered and the query answers are returned to the user (e.g., [91]).

In contrast, a *publication match* refers to the collection of publications that satisfy a subscription; that is, given an information match for a registered subscription, it is all minimal sets of publications that cause this match (i.e., entailment) to occur. This aligns with the task of selective content-based filtering of publications (e.g., [38]). Given a information match, there is a corresponding set of publication matches.

The distinction between these two match types is made as the type of match required is application dependent; for example, in OWL-based syndication of news feeds, publication matches are needed. In contrast, in the financial domain, analysts are generally interested with the actual information rather than the documents themselves. If we consider our previous example involving the concept *RiskyCompany*, one can observe that analysts are likely to be more interested in the actual instances of *RiskyCompany*, rather than the publications that discuss them; this is intuitive, as the actual query answer is the actionable information for their purposes (e.g., stock trading).

Given this distinction, an *information match* is defined as follows:

Definition 12 *(Information Match) Define a n-tuple of individuals* $(a_1, ..., a_n) \in \mathbf{I}^n_{\mathsf{K}_t}$ *to be an information match, denoted* $\mathcal{M}_I$*, at broker* B *for subscription* S *at time t, if the* *following condition holds:*

$$\mathsf{K}_t \models \mathsf{S}(Q_c[x_1/a_1, ..., x_n/a_n])$$

Due to the fact that publications can persist at the syndication broker for varying time periods, answer tuples may remain valid for varying time periods as well. Given this, there are various ways in which the broker could maintain these answers and notify subscribers. For example, the broker could maintain a list of all current bindings and only forward new information matches. However, this will have some ramifications with respect to the space that it takes to store the answer sets. In contrast, in some applications it may be better to pass all current bindings to the subscriber; however, yet again, there are performance impacts due to such an approach related to the transmission cost of transferring all bindings (including those already transfered) to a subscriber. Given the fact that this is application dependent, in the formalization it is not dictated how an actual instantiation of this framework should proceed with respect to this issue; rather it is left to the individuals deploying such a framework.

Related to this issue is that a previous information match may be invalidated in the event of a retraction publication (or the expiration of a publication). Once again, there are various notification strategies that can be adopted. Specifically, a subscriber could be notified if a previous information match is invalidated due to a retraction publication; in contrast, such a notification may not be necessary in some scenarios. Therefore, such a decision is not imposed here. When discussing examples, specific decisions regarding these issues will be made if it is not clear from the context of the discussion.

As discussed previously, a publication match is the collection of publications that satisfy a subscription. It is important to note that given an information match, additional computation is needed to derive *all* the minimal sets of publications responsible for an entailment. Clearly, in the event of a new information match for a subscription, the most recent (addition) publication which is received at the broker contributes to a publication match. However, we must determine the other publications which contribute to the match. For this purpose, the notion of *minimal justifications* for an entailment in DLs is utilized. As discussed in Chapter 2.5, this topic has been formally investigated in literature [85, 130, 129] and techniques have been developed to solve this problem. As noted in Chapter 2.5, the set of all minimal ABox justifications for the entailment of an axiom α by KB K is denoted by $Just(\mathsf{K}, \alpha)$. Given this discussion, the definition of a *publication match* is provided; note that α denotes an ABox assertion.

Definition 13 *(Publication Match) Let a n-tuple of individuals* $(a_1, ..., a_n) \in \mathbf{I}^n_{\mathsf{K}_t}$ *be an information match,* $\mathcal{M}_I$*, at broker* B *at time t for subscription* S*, where* $\mathsf{S}(Q_c)$ *has n distinguished variables (i.e.,* $DVar(\mathsf{S}(Q_c)) = \{d_1, ..., d_n\}$*). Additionally, let J be the set of minimal justifications for* $\mathcal{M}_I$ *such that:*

$$J = Just(\mathsf{K}_t, \mathsf{S}(Q_c[d_1/a_1, ..., d_n/a_n]))$$

Define a set of publications P to be a publication match, denoted $\mathcal{M}_P$*, at broker* B *for subscription* S *at time t if there exists* $j \in J$ *such that the following holds:*

- *for all* $\mathsf{P} \in P$*, there exists some* $\alpha \in j$ *such that* $\alpha \in \mathsf{P}(\beta)$ *and*
- *for all* $\alpha \in j$ *one of the following holds:*
  - *there exists some* $\mathsf{P} \in P$ *and* $\alpha \in \mathsf{P}(\beta))$ *or*
  - $\alpha \notin \mathsf{B}(\mathsf{K_P})$

The last two conditions of Definition 13 state that for a publication match to occur, there must be some justification for the entailment such that there is at least one assertion from each publication in the publication match that is in the justification; further, each assertion in the justification must be present in at least one of the publications or the assertion is background information in the KB.

As in the case for information matches, there are various ways to proceed related to the manner in which subscribers should be notified about publications matches (e.g., in the event of a retraction publication, subscribers could be notified about invalidated publication matches). Again, due to the application dependence of such a decision, a choice is not imposed here.

Lastly, Figure 4.1 presents an overview of the framework. The figure is an extension of the basic syndication architecture presented in Chapter 1. To reiterate the main points of the framework, note that all publications are represented as OWL instances (ABox assertions) and are integrated into the brokers (evolving) ABox. Additionally, subscriptions are assumed to be represented as conjunctive queries which are evaluated over the broker's knowledge base.

4.2.4 Discussion

In the formalization of the OWL-based syndication framework, conjunctive queries representing subscriber's interests are retrieval queries (queries that have at least one distinguished variable and return some tuple of named individuals). As stated in Chapter 2.4.2, queries which do not contain at least one distinguished variable are referred to as boolean queries. Boolean queries are not supported in the framework as there is no notion of an information match when subscription interests are represented as boolean queries (as there is not an answer tuple for the query). Additionally, in many real world applications investigated in OWL and DL literature, including publish/subscribe applications [62], queries typically have some number of distinguished variables. In general, it is easy to extend the framework to support boolean queries and it should be possible to extend the techniques to support boolean queries as well (see Chapter 9.2 for a discussion). However, given the previously mentioned points, in the remainder of this dissertation only retrieval queries are addressed.

It is important to note that this framework could easily be extended to support dif-

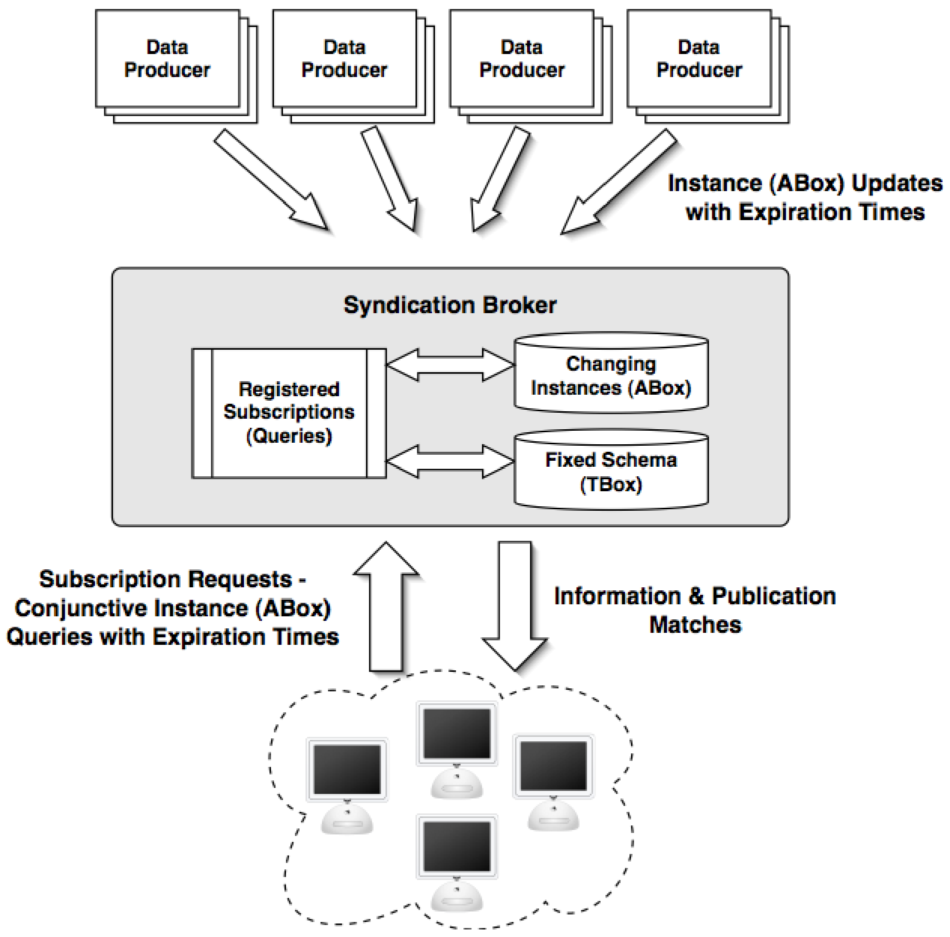

Data Producer
Data Producer
Data Producer
Data Producer
Instance (ABox) Updates with Expiration Times
Syndication Broker
Registered Subscriptions (Queries)
Changing Instances (ABox)
Fixed Schema (TBox)
Subscription Requests - Conjunctive Instance (ABox) Queries with Expiration Times
Information & Publication Matches

Figure 4.1: OWL-Based Syndication Architecture

ferent types of update semantics. However, given the discussion presented earlier related to this topic, investigating alternative semantics is out of the scope of this dissertation and syntactic updates are assumed.

4.3 Example

This concluding example demonstrates a composite match (both information and publication matches) and the framework in general. Let us assume that a syndication broker B is aware of two subscribers, S_1 and S_2, and two publishers, P_1 and P_2. Also, assume that the broker contains existing background information in its knowledge base; specifically, let the KB contain the axioms defined previously in Table 1.1, in addition to the type assertions that *BauschAndLomb* is a *Company* and *FusariumEyeInfection* is an *Infection*. A summary of the components of the syndication broker are shown in Table 4.1 (denoted in DL notation for ease of exposition).

Component	**Value**
Subscribers	$\{ S_1,\ S_2 \}$
Publishers	$\{ P_1,\ P_2 \}$
Knowledge Base	$\{ \mathit{RiskyCompany} \equiv \mathit{Company} \sqcap \exists \mathit{hasProduct}.\mathit{AdverseEffectProduct},$ $\mathit{AdverseEffectProduct} \equiv \mathit{Product} \sqcap \exists \mathit{causes}.(\mathit{Infection} \sqcup \mathit{ImparedState}),$ $\mathit{Company}(\mathit{BauschAndLomb}),\ \mathit{Infection}(\mathit{FusariumEyeInfection}) \}$

Table 4.1: Sample Syndication Broker

Additionally, assume that subscriber S_1 is interested in information about risky companies. Given this interest, S_1 registers a non-expiring subscription with the broker that is composed of a continuous query for all individuals of type *RiskyCompany*; formally the subscription is represented as follows:

$$((x) \leftarrow \mathit{RiskyCompany}(x), \infty) \in S_1(s)$$

where ∞ indicates that the subscription does not expire. On the other hand, let us assume that subscriber S_2 is interested in products that have some adverse effect. Therefore, S_2 registers the following subscription:

$$((x) \leftarrow \mathit{AdverseEffectProduct}(x), \infty) \in S_2(s)$$

Next, suppose that P_1 publishes an addition publication which expires in 24 hours which contains the assertions that *BauschAndLomb* has a product named *Renu* and that *Renu* is a *Product*. Assume that P_2 publishes an addition publication that also expires in 24 hours, however it contains the assertion that *Renu* causes the *FusariumEyeInfection*. These publications can be formally represented as:

$$\mathsf{P}_{P_1} = (\{ \mathit{hasProduct}(\mathit{BauschAndLomb}, \mathit{Renu}),\ \mathit{Product}(\mathit{Renu})\}, 24h, \mathit{true}, P_1)$$

$$\mathsf{P}_{P_2} = (\{ \mathit{causes}(\mathit{Renu}, \mathit{FusariumEyeInfection})\ \}, 24h, \mathit{true}, P_2)$$

where 24*h* indicates that the publications expire in 24 hours. For ease of exposition, assume that P_{P_1} and P_{P_2} arrive at the broker at time 1 and 2 respectively.

When P_{P_1} arrives at the broker, $\mathsf{P}_{P_1}(\beta)$ is integrated into $\mathsf{B}(\mathsf{K})$, resulting in a updated broker KB K'. At this time the individual *BauschAndLomb* will not satisfy the subscription, as it cannot be inferred that *Renu* is an *AdverseEffectProduct*; therefore, there will not be a match for S_1 or S_2 at time 1.

However, when P_{P_2} is published at time 2 and integrated into K', there will be new information matches for both registered subscriptions, as the broker's KB now entails that *Renu* is an *AdverseEffectProduct* and that *BauschAndLomb* is a *RiskyCompany*. This is fairly straightforward given the domain model and assertions in the syndication broker's KB; specifically, *Renu* is inferred to be a *AdverseEffectProduct* because it was known to be a product and P_{P_2} has provided the information that *Renu* causes a particular infection (i.e., the fusarium eye infection). Given this, it is inferred that *BauschAndLomb* is a *RiskyCompany* because it is a company that has a product that is an *AdverseEffectProduct* (i.e., *Renu*).

There is additionally a composite publication match $\{\mathsf{P}_{P_1}, \mathsf{P}_{P_2}\}$ for both subscriptions. Let us consider the subscription from S_1; first, we must determine the justifications for the entailment that *BauschAndLomb* is a *RiskyCompany*. In this case, there is only one justification:

$$Company(BauschAndLomb)$$
$$Product(Renu)$$
$$hasProduct(BauschAndLomb, Renu)$$
$$causes(Renu, FusariumEyeInfection)$$

Collectively P_{P_1} and P_{P_2} form a publication match, as P_{P_1} contains the assertions *hasProduct(BauschAndLomb, Renu)* and *Product(Renu)*, while P_{P_2} contains *causes(Renu, FusariumEyeInfection)*. Note that *Company*(*BauschAndLomb*) was originally contained in the brokers KB independent of any publications (i.e., it is background information); therefore, $\{\mathsf{P}_{P_1}, \mathsf{P}_{P_2}\}$ satisfies the definition of a publication match.

In a similar manner, there is one justification for the entailment that *Renu* is a *AdverseEffectProduct*:

$$Product(Renu)$$
$$hasProduct(BauschAndLomb, Renu)$$
$$causes(Renu, FusariumEyeInfection)$$

Therefore, $\{\mathsf{P}_{P_1}, \mathsf{P}_{P_2}\}$ constitutes a publication match for S_2's subscription.

4.4 Summary

In this chapter, an OWL-based syndication framework has been formalized in which matching newly published information with subscription requests is reduced to DL query answering. This framework provides a rich semantics-based mechanism for expressing published content allowing finer control for filtering. Additionally, given OWL's alignment with DL's, automated reasoning is provided for discovering subscription matches not found using traditional syntactic syndication approaches.

Given the performance overhead of DL reasoning, the next two chapters in this dissertation provide more efficient techniques for the reasoning services required in this syndication framework. In particular, the next chapter introduces an algorithm for efficient incremental consistency checking.

Chapter 5

Incremental Consistency Checking

5.1 Introduction

In the OWL-based syndication framework presented in the previous chapter, two reasoning services are required in order to incorporate new publications into the syndication broker's KB and match them with registered subscriptions. The first of these reasoning tasks is checking the consistency of the broker's knowledge base after a new publication has been integrated into it. As noted in Chapter 2.4.2, checking the consistency of a DL KB (or OWL ontology) is the task of ensuring that the KB does not contain any contradictory facts. An example of an inconsistency is demonstrated as follows: assume that in our domain model, it is stated that *TechnologyCompanies* and *AutomobileCompanies* are disjoint classes. This can be accomplished via the DL TBox axiom[1]:

$$\mathit{TechnologyCompany} \sqsubseteq \neg\, \mathit{AutomobileCompany}$$

If our KB additionally includes the ABox assertions $TechnologyCompnay(AppleInc)$ and *AutomobileCompany*(*AppleInc*), stating that *AppleInc* is both a technology and automobile company, then the KB would be inconsistent, as this is a logical contradiction. Unfortunately, if the KB is inconsistent, reasoning cannot be performed because everything is trivially entailed. Therefore, upon the integration of a new publication into the broker's KB, consistency must be ensured.

As discussed in Chapter 2.4.3, DL reasoners typically use tableau algorithms for checking the consistency of the KB. Unfortunately, if one attempts to use off-the-shelf DL reasoners today, scalability issues are immediately encountered related to this reasoning task for the syndication framework. To illustrate this issue further, let us consider

[1]Note that this can similarly be accomplished in OWL using the *disjointWith* constructor.

current consistency checking response times using today's tableau-based reasoners for a publicly available OWL ontology from the financial domain developed within the SEMINTEC project[2]. The ontology[3] is expressed using a subset of OWL Lite, namely the DL $\mathcal{ALCIF}$ and specifically includes subclass axioms, disjoint classes, functional and inverse properties, property restrictions, and domain/range constraints. Table 5.1 presents additional details related to the ontology, including the number of classes, properties, individuals and triples (which corresponds to the number of ABox assertions).

Ontology	**Expressivity**	**$\sharp$ Classes**	**$\sharp$ Properties**	**$\sharp$ Individuals**	**$\sharp$ Triples**
SEMINTEC	$\mathcal{ALCIF}$	59	16	17,941	65,723

Table 5.1: SEMINTEC Ontology Overview

Let us assume that this ontology represents the broker's KB, which consists of persistent publications and background information. When a new publication is received at the broker, it is integrated into the KB and then consistency is checked. Table 5.2 presents the consistency checking times using two of today's state-of-the-art tableau-based OWL reasoners, Pellet[4] and RacerPro[5]. The experiments were run on a Linux machine with 2Gb of RAM and a 3.06GHz Intel Xeon CPU; consistency checking times were averaged over 50 iterations and Pellet v1.5 and RacerPro v1.9.0 were used. It can be observed that response times using these reasoners will not be practical for the syndication framework.

Reasoner	**Response Time (sec)**
Pellet	2.137
Racer	8.16

Table 5.2: Consistency Checking Times for SEMINTEC Ontology

Tableau algorithms check the consistency of a KB by trying to construct an abstraction of a model, called a completion graph, for the knowledge base. With respect to

[2]SEMINTEC project homepage: http://www.cs.put.poznan.pl/alawrynowicz/semintec.htm
[3]Available at http://www.cs.put.poznan.pl/alawrynowicz/financial.owl
[4]Pellet Project Homepage: http://pellet.owldl.com/
[5]RacerPro is commercially supported by Racer Systems GmbH & Co. KG: http://www.racer-systems.com/index.phtml

the syndication framework, one of the main performance issues related to this reasoning service in today's tableau-based OWL reasoners is that the entire completion graph is rebuilt from scratch in the event of an update to the KB. If the KB was consistent prior to the update, there was a compete, clash-free completion graph corresponding to a model of the KB; when an update is received, this completion graph is discarded entirely and then rebuilt. In the presence of reasonably sized updates to the KB, their impact on the completion graph will be very small. Therefore, one would expect that a more effective technique can be developed in which the completion graph from a previous consistency check is not discarded.

In this chapter, I present a more effective incremental consistency checking approach, in which a previously constructed completion graph is updated instead of building a new completion graph after each update. This exploits the intuition that incremental changes to the KB will only affect a small portion of the previous completion graph, and therefore, maintaining it through changes will be much more effective. It is noted that in today's reasoners, the completion graph built during the initial consistency check is not discarded immediately after the reasoning service; this is because the structure is used for additional optimization techniques (e.g., pseudo model merging [72, 63]) during other reasoning services (assuming there has not been a change to the KB). Therefore, as we will see, maintaining the completion graph introduces manageable memory overhead.

Within the syndication framework, publications are sets of ABox assertions; given this, the techniques developed are scoped to ABox changes to the KB. Additionally, as both addition and deletion publications are supported in the syndication framework, approaches for incremental checking under both ABox additions and deletions are provided.

The approach developed is applicable to the DLs $\mathcal{SHIQ}$ and $\mathcal{SHOQ}$. OWL Lite is aligned with the DL $\mathcal{SHIF}$ (which is a subset of $\mathcal{SHIQ}$), while OWL DL is aligned with $\mathcal{SHOIN}$ (not subsumed by $\mathcal{SHIQ}$ or $\mathcal{SHOQ}$). Therefore, the technique presented is applicable to ontologies expressed in OWL Lite; OWL DL ontologies are supported, with the restriction that nominals and inverse cannot both be used. While the approach is applicable to $\mathcal{SHIQ}$ and $\mathcal{SHOQ}$, in the remainder of this chapter, the tableau algorithm for $\mathcal{SHOIQ}$ will simply be assumed, as it is applicable to these DLs as well.

Lastly, it is noted that the original definition of the $\mathcal{SHOIQ}$ tableau algorithm [78] does not include an ABox . This is because in the presence of nominals, ABox assertions can be transformed into semantically equivalent TBox axioms. However, in the remainder of this chapter, it is assumed this transformation is not performed and therefore ABoxes are considered. The $\mathcal{SHOIQ}$ algorithm with an ABox is not formally presented as it is a straightforward extension of the $\mathcal{SHOIQ}$ algorithm, which can be accomplished in a similar manner as the modification of $\mathcal{SHIQ}$ tableau algorithm [74] to include ABoxes [80].

5.2 ABox Additions

The overall goal of the approach is to incrementally update a completion graph from a previous consistency check (i.e., to take into account the assertions contained in the update). The main insight in the approach for addition updates follows from the observation that the expansion rules can be applied to the completion graph in an arbitrary order in the $\mathcal{SHOIQ}$ tableau algorithm when the TBox contains either nominals or inverses, but not both (i.e., it is expressed in $\mathcal{SHIQ}$, $\mathcal{SHOQ}$, or one of their sub-languages).

Lemma 1 *Let* K *be a* $\mathcal{SHIQ}$ *or* $\mathcal{SHOQ}$ *KB. Then performing the* $\mathcal{SHOIQ}$ *tableau algorithm for* K *without imposing the tableau expansion rule application ordering is sound, complete, and terminating.*

Proof See Appendix A.1.1 for the proof of this lemma. □

This means that the tableau algorithm has incremental properties, as any expansion rule ordering can be assumed when performing the tableau algorithm. This critical observation implies that given a complete and clash-free completion graph for the KB prior to the update, new ABox assertions can be added to the KB even after previous expansion rules have been fired for existing nodes and edges in the graph. This is accomplished via a two step process, assuming a previously consistent KB; first, the structures (nodes, edges, labels, and inequality relations) induced from the addition are added to the completion graph. Then, the necessary tableau expansion rules are applied for the newly added nodes, edges and labels.

To illustrate this, consider the KB shown in Table 5.3. When checking the consistency of this KB, an initial completion graph corresponding to the ABox is created. In particular, nodes corresponding to the individuals *BauschAndLomb* and *Renu* are created, as well as an edge between these two nodes; additionally, the concept name *RiskyCompany* is added to the label of the node corresponding to *BauschAndLomb* and the role name *hasProduct* is added to the edge label between the two nodes in the completion graph. This initial completion graph is depicted in Figure 5.1 (a).

Knowledge Base
$RiskyCompany \equiv Company \sqcap \exists hasProduct.AdverseEffectProduct$,
$EyeCareProduct \equiv Product \sqcap \exists cures.EyeProblem$,
$RiskyCompany(BauschAndLomb),\ hasProduct(BauschAndLomb, Renu)$

Table 5.3: Sample Knowledge Base

The tableau algorithm then proceeds to repeatedly apply the expansion rules, until a complete, clash-free completion graph is constructed. For illustration purposes, lazy unfolding is used (see Chapter 2.4.3 for a discussion). Due to the first axiom in the KB, the label *RiskyCompany* will be unfolded into the concept name $Company \sqcap \exists hasProduct.AdverseEffectProduct$. Following this, the $\sqcap$-rule will apply to this newly added label, thereby adding both *Company* and $\exists hasProduct.AdverseEffectProduct$ to the label of this node. Lastly, the $\exists$-rule will create a new node and edge to be added to the completion graph, followed by setting the appropriate labels for these new structures. The

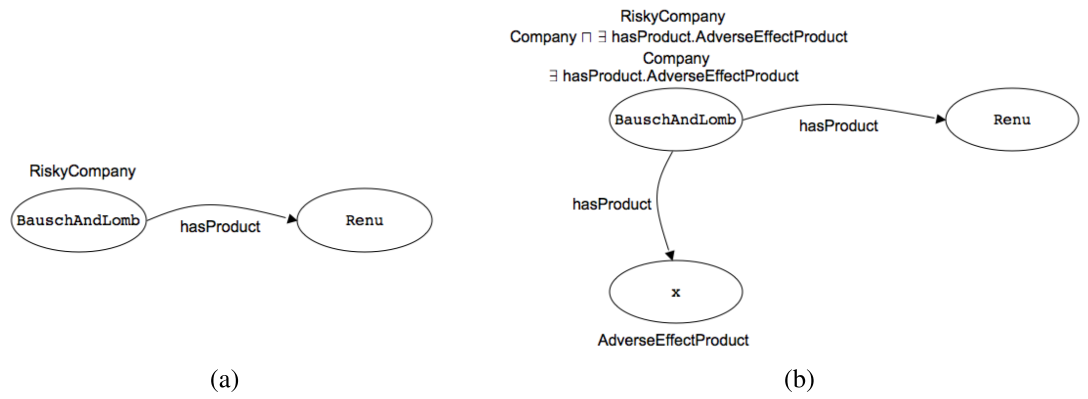


Figure 5.1: (a) Initial completion graph corresponding to ABox. (b) Complete clash-free completion graph for KB.

resulting completion graph depicted in Figure 5.1 (b)[6].

Let us now consider adding an ABox assertion to this KB stating that *Renu* is an *EyeCareProduct*; therefore, we need to check the consistency of the original KB K extended with the assertion *EyeCareProduct(Renu)* (i.e., $\mathsf{K} \cup \{$*EyeCareProduct*(*Renu*)$\}$). Rather than discarding the completion graph depicted in Figure 5.1 (b), the label of the node corresponding to *Renu* can simply be extended with the concept name *EyeCare-Product* and then the necessary expansion rules can be applied to the completion graph (note the rule applications proceed in a similar manner as previously for the concept name *RiskyCompany*); this will result in the complete and clash-free completion graph depicted in Figure 5.2.

In this case, this is precisely the completion graph which would have been constructed if the tableau algorithm were run from scratch for $\mathsf{K} \cup \{$*EyeCareProduct*(*Renu*)$\}$ and the unnecessary reconstruction of the structure of the completion graph has been avoided.

[6]Note that there are no non-determinstic choices when applying the expansion rules, therefore there is only one complete and clash-free completion graph.

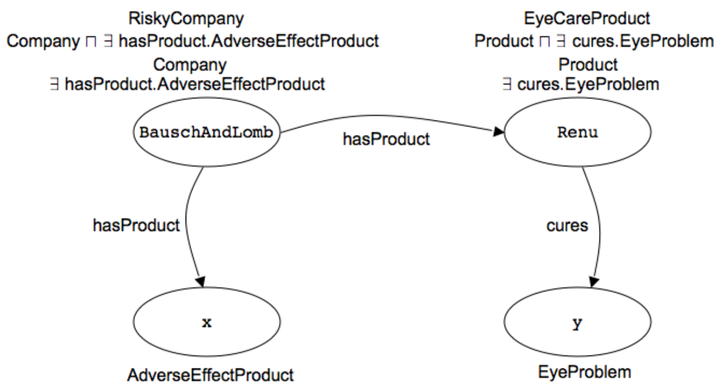


Figure 5.2: Complete clash-free completion graph for KB with ABox addition.

5.2.1 Approach Details

Various details related to this approach are now presented. As just illustrated, the general idea of the approach for additions to the KB is to first add the structure from the update to a complete, clash-free completion graph for the KB prior to the update, and then to apply the necessary expansion rules and let the tableau algorithm run as usual.

The actions that must be performed for a given addition are now presented via a case by case analysis of the form of the ABox addition. The notation x_a is used to denote the node in the completion graph corresponding to the named individual a (i.e., a will be mapped to node x_a if $a \in \mathcal{L}(x_a)$).

- Type assertion $C(a)$: if a node corresponding to the individual a does not exist in the completion graph, then it is added (i.e., if $x \notin \mathcal{V}$ such that $a \in \mathcal{L}(x)$, then $\mathcal{V} = \mathcal{V} \cup \{x\}$). Then, if C is not in $\mathcal{L}(x_a)$, it is added.

- Role assertion $R(a, b)$: if a node corresponding to the individuals a or b does not exist in the completion graph, then it is added. If there is not an edge between the nodes x_a and x_b, then the edge is added; formally, if $\langle x_a, x_b \rangle \notin \mathcal{E}$ then $\mathcal{E} = \mathcal{E} \cup \{\langle x_a, x_b \rangle\}$. Lastly, the edge label is added if necessary (i.e., if $R \notin \mathcal{L}(\langle x_a, x_b \rangle)$ then $\mathcal{L}(\langle x_a, x_b \rangle) = \mathcal{L}(\langle x_a, x_b \rangle) \cup \{R\}$).

- Inequality relation $a \neq b$: similar to the case for role assertions, if nodes corresponding to the individuals a, b do not exist in the completion graph, then they are added. Additionally, if not $x_a \dot{\neq} x_b$, then it is added.

- Equality relation $a = b$: similar to the case for inequality assertions, if nodes corresponding to the individuals a, b do not exist in the completion graph, then they are added. Additionally, b is added to $\mathcal{L}(x_a)$ and a is added to $\mathcal{L}(x_b)$.

After the structure introduced from the addition has been added to the previous completion graph, the expansion rules are reapplied to the completion graph and the algorithm runs as usual; this is necessary as the update may cause additional expansion rules to be fired. If the initial structure introduced from the update introduces any clashes or if a clash is encountered when re-applying the expansion rules, standard back-jumping techniques must be used to revert the completion graph to the non-deterministic choice points (e.g., a disjunction) that the clash is dependent on.

5.3 ABox Deletions

Supporting ABox deletion updates proceeds in a similar manner as additions. However, rather than adding the structure for the update to a previous completion graph, all structures dependent on the deleted assertions need to be removed. Following this, the expansion rules are re-applied to the resulting completion graph and the tableau algorithm proceeds as usual.

When performing ABox deletion updates, structures (nodes, edges, labels, etc.) in the previous completion graph that correspond to the removed assertion cannot be simply removed. This is because as expansion rules are applied, newly added portions of the graph are dependent on the presence of the original assertions in the KB. For example, consider the sample KB depicted earlier in Table 5.3 and the deletion of the assertion *RiskyCompany(BauschAndLomb)*; in this case, one cannot simply remove the concept name *RiskyCompany* from the label of the node corresponding to *BauschAndLomb*; the additional concept names in the label of this node, in addition to the node x and its incoming edge and labels, were added as a result of the deleted assertion.

Therefore, the components of the completion graph that are dependent on the deleted assertion need to be updated as well. In order to account for this, I leverage and extend previous work on axiom tracing [14, 85, 87]. In such approaches, the dependencies of completion graph structures (and the events that manipulate them) on original source axioms from the KB are tracked through the tableau expansion rule application process. Further details regarding the tracing approach will be discussed in detail later, however the general idea is to utilize axiom tracing in order to track the dependencies of parts of the completion graph, so that the effects of deleted ABox assertions can be *rolled-back*.

In general, the update algorithm for deletion works as follows: when an ABox assertion is removed, the algorithm determines all of the change events in the previous completion graph whose axiom traces include the deleted axiom and these events are *rolled-back*. By *roll-back*, we refer to simply undoing the event (e.g., rolling back an event that adds the concept name C to the label of node x would be the process of removing C from $\mathcal{L}(x)$). Following this, the tableau expansion rules are re-applied to the resulting completion graph and the tableau algorithm runs as usual. The expansion rules must be re-applied as the deleted structures may be re-added due to subsequent rule applications. For example, if the sample KB depicted in Table 5.3 additionally contained the assertion $Company \sqcap RiskyCompany(BauschAndLomb)$, then even if the ABox assertion *RiskyCompany(BauschAndLomb)* were removed, the *RiskyCompany* label would need to be re-added to the label of the node corresponding to *BauschAndLomb* due to the assertion $Company \sqcap RiskyCompany(BauschAndLomb)$.

It is also important to note that the previous exploration of a non-determinstic choice in the tableau algorithm that resulted in a clash dependent on a removed assertion must be re-explored; this is because the clash may have been invalidated due to the deletion.

In contrast to the approach for additions, the approach for deletions is applicable to a completion graph corresponding to the previous KB that contains a clash.

5.3.1 Approach Details

As stated previously, axiom tracing [14, 85, 87] is extended in order to account for the dependency of completion graph structures on deleted assertions. More specifically, [85] presents an algorithm in which the application of the tableau expansion rules triggers a set of *events*, denoted EV, that change the state of the completion graph. A summary of the change events presented in [85] is as follows:

- $Add(C, \mathcal{L}(x))$ represents the action of adding a concept C to $\mathcal{L}(x)$
- $Add(R, \mathcal{L}(\langle x, y\rangle))$ represents the addition of a role R to $\mathcal{L}(\langle x, y\rangle)$
- $E(x, y)$ is the action of merging the nodes x, y.
- $NE(x, y)$ is the action of adding of an inequality relation $x \dot{\neq} y$.

In order to record the changes to the completion graph, [85] introduces a tracing function, τ, which keeps track of the asserted axioms responsible for changes events; more specifically, τ maps each event $e \in EV$ to a set containing a fragment of the KB that cause the event to occur, and this tracing function is maintained throughout the application of tableau expansion rules.

In order to provide axiom tracing for the purpose of ABox deletions, the original set of change events is now extended to include all possible events that can occur during the application of expansion rules; this is necessary as in the presence of a deletion, all of the effects of the removed assertions need to be rolled-back. The new change events are as follows:

- $Add(x, \mathcal{V})$ represents the action of adding a node x to $\mathcal{V}$

- $Add(\langle x, y\rangle, \mathcal{E})$ represents the action of adding a edge $\langle x, y\rangle$ to $\mathcal{E}$
- $Remove(x, \mathcal{V})$ represents the action of removing a node x from $\mathcal{V}$
- $Remove(\langle x, y\rangle, \mathcal{E})$ represents the action of removing an edge $\langle x, y\rangle$ from $\mathcal{E}$
- $Remove(C, \mathcal{L}(x))$ represents the removal of a concept C from $\mathcal{L}(x)$
- $Remove(R, \mathcal{L}(\langle x, y\rangle))$ represents the removal of a role R from $\mathcal{L}(\langle x, y\rangle)$

Collectively, these change events account for all possible changes that can occur during the application of $\mathcal{SHOIQ}$ tableau expansion rules [78].

For purpose of ABox deletions, the change events necessary for creating the initial completion graph corresponding to the initial ABox need to be taken into account as well. This is because the structures that are dependent on these concept and role assertions may need to be retracted due to a deletion. The initialized tracing function τ for a completion graph corresponding to an initial ABox is shown in Algorithm 1[7]. It can be observed that these events directly align with those that occur when the initial completion graph is constructed from the ABox. However, because edges between nodes corresponding to named individuals can be added due to any role assertion involving the two indvidiauls, a special condition in the definition of τ is also introduced; specifically, assuming that $R(a, b) \in \mathsf{A}$, then $R(a, b) \in \tau(Add(\langle x_a, x_b\rangle, \mathcal{E}))$ if and only if there does not exist some role S, where $S \neq R$, such that $S(a, b) \in \mathsf{A}$. This states that an edge involving named individuals is only dependent on an ABox assertion if there does not exist another role assertion involving those individuals. Such a condition is not necessary for nodes corresponding to named individuals, as a node is always created for each named individual (i.e., it is independent of an ABox assertion).

The $\mathcal{SHOIQ}$ tableau expansion rules extended with axiom tracing are introduced in Table 5.4. For simplicity, $\tau(C, x)$, $\tau(R, \langle x, y\rangle)$, $\tau(x, \mathcal{V})$, and $\tau(\langle x, y\rangle, \mathcal{E})$ are used as

[7]Again given a named individual a, the notation x_a is used to denote the node in G corresponding to a.

Algorithm 1 *initialize_tracing*(A, G)

```
Input:
    $\mathsf{A}$: ABox
    $\mathsf{G}$: Initial completion graph corresponding to $\mathsf{A}$
Output:
    $\tau$: Initialized tracing function
 1: for all $\alpha \in \mathsf{A}$ do
 2:     if $\alpha$ of the form $C(a)$ then
 3:         $\tau(Add(C, \mathcal{L}(x_a))) \leftarrow \{C(a)\}$
 4:     else if $\alpha$ of the form $R(a, b)$ then
 5:         $\tau(Add(R, \mathcal{L}(\langle x_a, x_b \rangle))) \leftarrow \{R(a, b)\}$
 6:     else if $\alpha$ of the form $a = b$ then
 7:         $\tau(Add(b, \mathcal{L}(x_a))) \leftarrow \{a = b\}$
 8:         $\tau(Add(a, \mathcal{L}(x_b))) \leftarrow \{a = b\}$
 9:     else if $\alpha$ of the form $a \neq b$ then
10:         $\tau(NE(x_a, x_b)) \leftarrow \{a \neq b\}$
11:     end if
12: end for
13: return $\tau$
```

abbreviations for $\tau(Add(C, \mathcal{L}(x))$, $\tau(Add(R, \mathcal{L}\langle x, y \rangle))$, $\tau(Add(x, \mathcal{V}))$, and $\tau(Add(\langle x, y \rangle, \mathcal{E}))$ respectively. Additionally, given a role S, concept C, and node x in completion graph G, $S^{\mathsf{G}}(x, C)$ is defined such that:

$$S^{\mathsf{G}}(x, C) = \{y \mid y \text{ is a } S\text{-neighbor of } x \text{ and } C \in \mathcal{L}(x)\}$$

which represents all S-neighbors of x that satisfy the concept C. For ease of exposition, an abbreviation for the edges and edge labels that satisfy the S-neighbor relation is also used. Specifically, if x has S-neighbors $\{y_1, ...y_m\}$, denote the actual edge and edge label between x and y_i in the completion graph as $S_e(x, y_i)$ and $S_l(x, y_i)$ respectively. This notation will also be used when referring to a single S-neighbor (e.g., for the $\forall$-rule). Lastly, given $S^{\mathsf{G}}(x, C) = \{y_1, ..., y_m\}$, $\tau(S^{\mathsf{G}}(x, C))$ is used an abbreviation for the union of all axiom traces for the node x, edges $S_e(x, y_i)$, edge labels $S_l(x, y_i)$, nodes y_i, and concept labels $C \in \mathcal{L}(y_i)$; that is $\tau(S^{\mathsf{G}}(x, C))$ is a abbreviation for the following:

$$\tau(x, \mathcal{V}) \cup \tau(S_e(x, y_1), \mathcal{E}) \cup ... \cup \tau(S_e(x, y_m), \mathcal{E}) \cup \tau(S_l(x, y_m), \mathcal{L}(S_e(x, y_1))) \cup ... \cup$$

$$\tau(S_l(x, y_m), \mathcal{L}(S_e(x, y_m))) \cup \tau(C, y_1) \cup ... \cup \tau(C, y_m) \cup \tau(y_1, \mathcal{V}) \cup ... \cup \tau(y_m, \mathcal{V})$$

In Table 5.4, the merge operation has not been addressed. The updated *Merge* function is provided in Table 5.5. It is noted that the merge function is called from two

$\sqcap$-rule:	if 1) $C_1 \sqcap C_2 \in \mathcal{L}(x)$, x is not indirectly blocked and 2) $\{C_1, C_2\} \not\subseteq \mathcal{L}(x)$ then set $\mathcal{L}(x) = \mathcal{L}(x) \cup \{C_1, C_2\}$ and $\tau(C_1, x) \leftarrow \tau((C_1 \sqcap C_2), x) \cup \tau(x, \mathcal{V})$ and $\tau(C_2, x) \leftarrow \tau((C_1 \sqcap C_2), x) \cup \tau(x, \mathcal{V})$
$\sqcup$-rule:	if 1) $C_1 \sqcup C_2 \in \mathcal{L}(x)$, x is not indirectly blocked and 2) $\{C_1, C_2\} \cap \mathcal{L}(x) = \emptyset$ then set $\mathcal{L}(x) = \mathcal{L}(x) \cup C$ for some $C \in \{C_1, C_2\}$ and $\tau(C, x) \leftarrow \tau((C_1 \sqcup C_2), x) \cup \tau(x, \mathcal{V})$
$\exists$-rule:	if 1) $\exists S.C \in \mathcal{L}(x)$, x is not blocked and 2) x has no S-neighbor y with $C \in \mathcal{L}(y)$ then create a new node y with $\mathcal{L}(\langle x, y\rangle) = \{S\}$ and $\mathcal{L}(y) = \{C\}$ and $\tau(y, \mathcal{V}) \leftarrow \tau((\exists S.C), x) \cup \tau(x, \mathcal{V})$ and $\tau(\langle x, y\rangle, \mathcal{E}) \leftarrow \tau((\exists S.C), x) \cup \tau(x, \mathcal{V})$ and $\tau(C, y) \leftarrow \tau((\exists S.C), x) \cup \tau(x, \mathcal{V})$ and $\tau(S, \langle x, y\rangle) \leftarrow \tau((\exists S.C), x) \cup \tau(x, \mathcal{V})$
$\forall$-rule:	if 1) $\forall S.C \in \mathcal{L}(x)$, x is not indirectly blocked and 2) there is an S-neighbor y of x with $C \notin \mathcal{L}(y)$ then set $\mathcal{L}(y) = \mathcal{L}(y) \cup \{C\}$ and $\tau(C, y) \leftarrow \tau((\forall S.C), x) \cup \tau(x, \mathcal{V}) \cup \tau(y, \mathcal{V}) \cup \tau(S_l(x, y), \mathcal{L}(S_e(x, y))) \cup \tau(S_e(x, y), \mathcal{E})$
$\forall_+$-rule:	if 1) $\forall S.C \in \mathcal{L}(x)$, x is not indirectly blocked and 2) there is some R with $\mathsf{Trans}(R)$ and $R \mathrel{\underline{\sqsubseteq}} S$, 3) there is an R-neighbor y of x with $\forall R.C \notin \mathcal{L}(y)$ then set $\mathcal{L}(y) = \mathcal{L}(y) \cup \{\forall R.C\}$ and $\tau(\forall R.C, y) \leftarrow \tau((\forall S.C), x) \cup \tau(x, \mathcal{V}) \cup \tau(y, \mathcal{V}) \cup \tau(R_l(x, y), \mathcal{L}(R_e(x, y))) \cup \tau(R_e(x, y), \mathcal{E})$
choose-rule:	if 1) $(\leqslant nS.C) \in \mathcal{L}(x)$, x is not indirectly blocked and 2) there is an S-neighbor y of x with $\{C, \neg C\} \cap \mathcal{L}(y) = \emptyset$ then set $\mathcal{L}(y) = \mathcal{L}(y) \cup \{E\}$ for some $E \in \{C, \neg C\}$ and $\tau(E, y) \leftarrow \tau((\leqslant S.C), x) \cup \tau(x, \mathcal{V}) \cup \tau(y, \mathcal{V}) \cup \tau(S_l(x, y), \mathcal{L}(S_e(x, y))) \cup \tau(S_e(x, y), \mathcal{E})$
$\geqslant$-rule:	if 1) $(\geqslant nS.C) \in \mathcal{L}(x)$, x is not blocked and 2) there are not n S-neighbor $y_1, ..., y_n$ of x with $C \in \mathcal{L}(y_i)$ and $y_i \dot{\neq} y_j$ for $1 \leq i < j \leq n$ then create n new nodes $y_1, ..., y_n$ with $\mathcal{L}(\langle x, y_i\rangle) = \{S\}$, $\mathcal{L}(y_i) = \{C\}$, and $y_i \dot{\neq} y_j$ for $1 \leq i < j \leq n$ and $\tau(y_i, \mathcal{V}) \leftarrow \tau((\geqslant nS.C), x) \cup \tau(x, \mathcal{V})$ and $\tau(\langle x, y_i\rangle, \mathcal{E}) \leftarrow \tau((\geqslant nS.C), x) \cup \tau(x, \mathcal{V})$ and $\tau(C, y_i) \leftarrow \tau((\geqslant nS.C), x) \cup \tau(x, \mathcal{V})$ and $\tau(S, \langle x, y\rangle) \leftarrow \tau((\geqslant nS.C), x) \cup \tau(x, \mathcal{V})$ and $\tau(NE(y_i, y_j)) \leftarrow \tau((\geqslant nS.C), x) \cup \tau(x, \mathcal{V})$
$\leqslant$-rule:	if 1) $(\leqslant nS.C) \in \mathcal{L}(z)$, z is not indirectly blocked and 2) $S^{\mathsf{G}}(z, C) = \{y_1, ..., y_m\}$ such that $m > n$, and there are two S-neighbors x, y of z with not $x \dot{\neq} y$ and $C \in \mathcal{L}(x) \cap \mathcal{L}(y)$ then 1) if x is a nominal node, then $Merge(y, x)$ 2) else if y is a nominal node or an ancestor of x, then $Merge(x, y)$ 3) else $Merge(y, x)$
O-rule:	if for some $o \in \mathbf{I}$ there are 2 nodes x, y with $o \in \mathcal{L}(x) \cap \mathcal{L}(y)$ and not $x \dot{\neq} y$ then $Merge(x, y)$
NN-rule:	if 1) $(\leqslant nS.C) \in \mathcal{L}(x)$, x is a nominal node and there is a blockable S-neighbor y of x such that $C \in \mathcal{L}(y)$ and x is a successor of y, 2) there is no m such that $1 \leq m \leq n$, $(\leqslant mS.C) \in \mathcal{L}(x)$, and there exists m nominal S-neighbors $z_1, ..., z_m$ of x with $C \in \mathcal{L}(z_i)$ and $z_i \dot{\neq} z_j$ for all $1 \leq i < j \leq m$ then 1) guess m with $1 \leq m \leq n$ and set $\mathcal{L}(x) = \mathcal{L}(x) \cup \{\leqslant mS.C\}$ and $\tau((\leq mS.C), x) \leftarrow \tau((\leq nS.C), x) \cup \tau(x, \mathcal{V}) \cup \tau(S_l(x, y), \mathcal{L}(S_e(x, y))) \cup \tau(S_e(x, y), \mathcal{E})$ 2) create m new nodes $y_1, ..., y_m$ with $\mathcal{L}(\langle x, y_i\rangle) = \{S\}$ and $\tau(y_i, \mathcal{V}) \leftarrow \tau((\leq nS.C), x) \cup \tau(x, \mathcal{V}) \cup \tau(S_l(x, y), \mathcal{L}(S_e(x, y))) \cup \tau(S_e(x, y), \mathcal{E})$ and $\tau(\langle x, y_i\rangle, \mathcal{E}) \leftarrow \tau((\leq nS.C), x) \cup \tau(x, \mathcal{V}) \cup \tau(S_l(x, y), \mathcal{L}(S_e(x, y))) \cup \tau(S_e(x, y), \mathcal{E})$ and $\tau(S, \langle x, y_i\rangle) \leftarrow \tau((\leq nS.C), x) \cup \tau(x, \mathcal{V}) \cup \tau(S_l(x, y), \mathcal{L}(S_e(x, y))) \cup \tau(S_e(x, y), \mathcal{E})$ and $\mathcal{L}(y_i) = \{C, o_i\}$ for each $o_i \in \mathbf{I}$ new in G and $\tau(C, y_i) \leftarrow \tau((\leq nS.C), x) \cup \tau(x, \mathcal{V}) \cup \tau(S_l(x, y), \mathcal{L}(S_e(x, y))) \cup \tau(S_e(x, y), \mathcal{E})$ and $\tau(o_i, y_i) \leftarrow \tau((\leq nS.C), x) \cup \tau(x, \mathcal{V}) \cup \tau(S_l(x, y), \mathcal{L}(S_e(x, y))) \cup \tau(S_e(x, y), \mathcal{E})$ and $y_i \dot{\neq} y_j$ and $\tau(NE(y_i, y_j)) = \tau(x, \mathcal{V}) \cup \tau(S_l(x, y), \mathcal{L}(S_e(x, y))) \cup \tau(S_e(x, y), \mathcal{E})$ for $1 \leq i < j \leq m$

Table 5.4: Modified $\mathcal{SHOIQ}$ Tableau Expansion Rules for Axiom Tracing

different expansion rules: the $\leqslant$-rule and the O-rule. The dependencies differ depending on which rule calls the function. For example, if the $\leqslant$-rule calls the merge function, then all events are dependent on the existence of more than n S-neighbors; this is not the case for the O-rule. In order to avoid presentation of two different versions of the merge function, the following notation is used: let M represented the following set of assertions depending on the expansion rule that calls the merge function:

$$M = \begin{cases} \leqslant\text{-rule:} & \tau(S^{\mathsf{G}}(x, C)) \cup \tau((\leq nS.C), \mathcal{L}(x)) \\ O\text{-rule:} & \tau(x, \mathcal{V}) \cup \tau(y, \mathcal{V}) \cup \tau(o, x) \cup \tau(o, y) \end{cases}$$

Some problematic cases related to rolling back change events through node merges can occur when a label (or edge) exists in both of the nodes being merged. For example, if y is merged into x and both nodes are labeled with D, then the tracing function for the dependency of $D \in \mathcal{L}(x)$ will only include the dependency of the events which added D to the label of x. Therefore, if an assertion is removed that is included in this dependency, D would be removed from the label of x even though it should still exist because y was merged into x.

To overcome this and similar cases, all events which occur during the merge operation are dependent on all edges (incoming and outgoing), labels (both edge and node), and inequality relations for both x and y. This ensures that if one of these assertions is removed, the entire merge operation will in turn be rolled-back; then, when the expansion rules are re-applied to the completion graph after the roll-back, the merge can be re-performed correctly. Clearly, this is an overestimate for the dependencies of the events that occur during the merge; however, it is still complete (shown in Theorem 1). Further, this approach allows us to effectively recover from the merge operation even in problematic cases. The variable T is introduced to denote this dependency set. First, let P denote the set of incoming and outgoing edges of both x and y (which are the nodes to be merged), let N_x, N_y denote the set of node labels for x, y, and E_x E_y denote the set of inequality relations for x, y. Then T can be defined as follows:

$$T = \Big(\bigcup_{\langle m,n\rangle \in P} \tau(m, \mathcal{V}) \cup \tau(n, \mathcal{V}) \cup \tau(\langle m, n\rangle, \mathcal{E}) \cup \tau(l, \mathcal{L}(\langle m, n\rangle)) \text{ for all}$$

$$l \in \mathcal{L}(\langle m,n\rangle)\Big) \cup \Big(\bigcup_{l\in N_x} \tau(l,x)\Big) \cup \Big(\bigcup_{l\in N_y} \tau(l,y)\Big) \cup \Big(\bigcup_{e\in E_x} \tau(NE(e))\Big) \cup \Big(\bigcup_{e\in E_y} \tau(NE(e))\Big)$$

Finally, the prune function that is called from the merge function is addressed in Table 5.6.

```
$Merge(y,x)$ :
    1. for all nodes $z$ such that $\langle z,y\rangle \in \mathcal{E}$
      (a) if $\{\langle x,z\rangle, \langle z,x\rangle\} \cap \mathcal{E} = \emptyset$, then add $\langle z,x\rangle$ to $\mathcal{E}$ and set $\mathcal{L}(\langle z,x\rangle) = \mathcal{L}(\langle z,y\rangle)$, and
          $\tau(\langle z,x\rangle, \mathcal{E}) \leftarrow M \cup T$ and
          $\tau(l, \langle z,x\rangle) \leftarrow M \cup T$ for each $l \in \mathcal{L}(\langle z,y\rangle)$

      (b) if $\langle z,x\rangle \in \mathcal{E}$, then
          $\tau(l, \langle z,x\rangle) \leftarrow M \cup T$ for each $l \in \mathcal{L}(\langle z,y\rangle)$ and $l \notin \mathcal{L}(\langle z,x\rangle)$ and
          set $\mathcal{L}(\langle z,x\rangle) = \mathcal{L}(\langle z,x\rangle) \cup \mathcal{L}(\langle z,y\rangle)$

      (c) if $\langle x,z\rangle \in \mathcal{E}$, then
          $\tau(\mathsf{Inv}(S), \langle x,z\rangle) \leftarrow M \cup T$ for each $S \in \mathcal{L}(\langle z,y\rangle)$ and $\mathsf{Inv}(S) \notin \mathcal{L}(\langle x,z\rangle)$ and
          set $\mathcal{L}(\langle x,z\rangle) = \mathcal{L}(\langle x,z\rangle) \cup \{\mathsf{Inv}(S) \mid S \in \mathcal{L}(\langle z,y\rangle)\}$

      (d) remove $\langle z,y\rangle$ from $\mathcal{E}$
          $\tau(Remove(\langle z,y\rangle, \mathcal{E})) \leftarrow M \cup T$
          $\tau(Remove(l, \mathcal{L}(\langle z,y\rangle))) \leftarrow M \cup T$ for all $l \in \mathcal{L}(\langle z,y\rangle)$

    2. for all nominal nodes $z$ such that $\langle y,z\rangle \in \mathcal{E}$
      (a) if $\{\langle x,z\rangle, \langle z,x\rangle\} \cap \mathcal{E} = \emptyset$, then add $\langle x,z\rangle$ to $\mathcal{E}$ and set $\mathcal{L}(\langle x,z\rangle) = \mathcal{L}(\langle y,z\rangle)$, and
          $\tau(\langle x,z\rangle, \mathcal{E}) \leftarrow M \cup T$ and
          $\tau(l, \langle x,z\rangle) \leftarrow M \cup T$ for each $l \in \mathcal{L}(\langle y,z\rangle)$

      (b) if $\langle x,z\rangle \in \mathcal{E}$, then
          $\tau(l, \langle x,z\rangle) \leftarrow M \cup T$ for each $l \in \mathcal{L}(\langle y,z\rangle)$ and $l \notin \mathcal{L}(\langle x,z\rangle)$ and
          set $\mathcal{L}(\langle x,z\rangle) = \mathcal{L}(\langle x,z\rangle) \cup \mathcal{L}(\langle y,z\rangle)$

      (c) if $\langle z,x\rangle \in \mathcal{E}$, then
          $\tau(\mathsf{Inv}(S), \langle z,x\rangle) \leftarrow M \cup T$ for each $S \in \mathcal{L}(\langle y,z\rangle)$ and $\mathsf{Inv}(S) \notin \mathcal{L}(\langle z,x\rangle)$ and
          set $\mathcal{L}(\langle z,x\rangle) = \mathcal{L}(\langle z,x\rangle) \cup \{\mathsf{Inv}(S) \mid S \in \mathcal{L}(\langle y,z\rangle)\}$

      (d) remove $\langle y,z\rangle$ from $\mathcal{E}$
          $\tau(Remove(\langle y,z\rangle, \mathcal{E})) \leftarrow M \cup T$
          $\tau(Remove(l, \mathcal{L}(\langle y,z\rangle))) \leftarrow M \cup T$ for all $l \in \mathcal{L}(\langle y,z\rangle)$

    3. $\tau(l,x)) \leftarrow M \cup T$ for each $l \in \mathcal{L}(y)$ and $l \notin \mathcal{L}(x)$ and
       set $\mathcal{L}(x) = \mathcal{L}(x) \cup \mathcal{L}(y)$

    4. $\tau(NE(x,z)) \leftarrow M \cup T$ for each $z \in y \dot{\neq} z$ and not $x \dot{\neq} z$ and
       add $x \dot{\neq} z$ for all $z$ such that $y \dot{\neq} z$

    5. $Prune(y)$.
```

Table 5.5: Merge Operation for Axiom Tracing

The modifications of the expansion rules do not impact the correctness of the original $\mathcal{SHOIQ}$ tableau algorithm; this is a direct consequence of the fact that the original events caused by expansion rule applications are not modified. Importantly, it can also be shown that if an event is caused during the application of the expansion rules that is dependent on an ABox assertion, then the tracing function captures the dependency of

```
Prune(y) :
    1. for all successors $z$ of $y$
       $remove\langle y, z\rangle$ from $\mathcal{E}$ and
       $\tau(Remove(l, \mathcal{L}(\langle y, z\rangle))) \leftarrow M \cup T$ for all $l \in \mathcal{L}(\langle y, z\rangle)$ and
       $\tau(Remove(\langle y, z\rangle, \mathcal{E})) \leftarrow M \cup T$ and
       and if $z$ blockable, $Prune(z)$
    2. remove $y$ from $\mathcal{V}$ and
       $\tau(Remove(l, \mathcal{L}(y)) \leftarrow M \cup T$ for all $l \in \mathcal{L}(y)$ and
       $\tau(Remove(y, \mathcal{V})) \leftarrow M \cup T$
```

Table 5.6: Prune Operation for Axiom Tracing

this event; this implies that the tracing function is complete.

Let ψ be the *dependency function* that is constructed from τ, which maps an ABox assertion α to a set of events E, where E contains all e such that $\alpha \in \tau(e)$.

Theorem 1 *Let* K *be a* $\mathcal{SHOIQ}$ *KB,* G *be a completion graph for* K *resulting from some sequence of application of tableau expansion rules defined in Table 5.4. Also let E be the sequence of events that occur during the tableau algorithm and* ψ *be the dependency function constructed from* τ *for* G. *If an event* $e \in E$ *is dependent on ABox assertion* $\alpha \in \mathsf{K}$*, then* $e \in \psi(\alpha)$.

Proof See Appendix A.1.2 for the proof of this theorem. □

It is a consequence of this theorem that the tracing function can be used to roll-back all change events that are dependent on a removed assertion. Therefore, the update algorithm for deletions proceeds as follows: when an ABox assertion is removed, the algorithm performs a lookup in the previous completion graph for all *change events* whose axiom traces include the deleted assertion. These events are *rolled-back* if and only if their axiom trace contains the deleted assertion. Following this, the tableau expansion rules are re-applied to the resulting completion graph and the tableau algorithm proceeds as usual. In the event of a deletion update, previously explored branches which had a clash must be re-explored because the deletion could have removed the assertion which contributed the clash. Given this, in the event of a deletion, all non-determinstic choice points are simply reconsidered if back-jumping occurs when the expansion rules are reapplied.

To illustrate the approach for deletions, assume that K is the KB shown in Table 5.3 with the additional ABox assertion *EyeCareProduct(Renu)*. In this case, the initial complete, clash-free completion graph will be identical to the one shown in Figure 5.2. Next, consider the deletion of the assertion *EyeCareProduct(Renu)*; the events that are dependent on this assertion are as follows:

$$\begin{gathered}
\text{Add}(\mathit{EyeCareProduct},\, x_{\mathit{Renu}}) \\
\text{Add}(\mathit{Product} \sqcap \exists \mathit{cures}.\mathit{EyeProblem},\, x_{\mathit{Renu}}) \\
\text{Add}(\mathit{Product},\, x_{\mathit{Renu}}) \\
\text{Add}(\exists \mathit{cures}.\mathit{EyeProblem},\, x_{\mathit{Renu}}) \\
\text{Add}(y,\, \mathcal{V}) \\
\text{Add}(\mathit{EyeProblem},\, \mathcal{L}(y)) \\
\text{Add}(\langle x_{\mathit{Renu}}, y\rangle,\, \mathcal{E}) \\
\text{Add}(\mathit{cures},\langle x_{\mathit{Renu}}, y\rangle)
\end{gathered}$$

Given this axiom trace, if *EyeCareProduct(Renu)* is deleted, then all of these events would be rolled-back. Therefore, all of these structures would be removed from the completion graph. After this, the expansion rules would not be applicable to any more labels and the algorithm would result in the same completion graph as depicted in Figure 5.1 (b).

Lastly, it is pointed out that axiom tracing requires a small modification to the update approach for ABox additions in order to maintain axiom traces; specifically, the tracing function must be updated to take into account the new structures added to the completion graph. This is addressed in the next chapter, in which the pseudo-code for the incremental consistency checking algorithm is provided.

5.3.2 Discussion

As only ABox updates are allowed, only dependencies on ABox assertions are maintained in the approach just presented. Additionally, only the standard tableau expansion rules have been addressed. It is important to note that the technique described in the previous section can be easily extended to trace dependencies for TBox axioms and support known optimizations which introduce additional tableau expansion rules (e.g., the lazy-unfolding rule). The interested readers is referred to [65] for a discussion of these topics.

5.4 Incremental Consistency Checking Algorithm

Algorithm 2 presents the incremental consistency checking algorithm for ABox additions and deletions. The update algorithm takes as input a $\mathcal{SHIQ}$ or $\mathcal{SHOQ}$ KB K, a completion graph for K, the axiom tracing and dependency functions for the completion graph, a set of ABox assertions, a boolean value indicating if the assertions should be added or removed, and it returns a completion graph. As noted earlier, if the update is an addition, the previous KB must be consistent, implying that G is complete and clash-free. Given a named individual a, x_a is used to denote the node in the completion graph corresponding to a.

Next, the correctness of the algorithm is shown for checking ABox consistency under syntactic updates.

Theorem 2 *Let K be a $\mathcal{SHIQ}$ or $\mathcal{SHOQ}$ KB, G be a completion graph for K, τ and ψ be the tracing and dependency functions respectively for G, β be a set of ABox assertions, and υ be some boolean value. Then Algorithm 2 terminates and returns a complete and clash-free completion graph if and only if a complete and clash-free completion graph can be constructed by the tableau algorithm for $\mathsf{K} \oplus \beta$.*

Proof See Appendix A.1.3 for the proof of this theorem. □

The complexity of Algorithm 2 is now addressed. As stated in Chapter 2.4.3, the known complexity of the $\mathcal{SHOIQ}$ tableau algorithm is 2NExpTime [141]. The axiom tracing modifications to the tableau expansion rules occur in either constant or linear time. Additionally, lines 1–37 of Algorithm 2 run in linear time. Thus, the worst case complexity of Algorithm 2 is 2NExpTime as well. However, in the next section, empirical results

Algorithm 2 *Inc_Consistency*$(\mathsf{K}, \mathsf{G}, \tau, \psi, \beta, \upsilon)$

Input:

K: $\mathcal{SHIQ}$ KB
G: Completion graph for K
τ: Axiom tracing function for G
ψ: Axiom dependency function for G
β: Set of ABox assertions
υ: Boolean value indicating if β is an addition or deletion

Output:

G: If $\mathsf{K} \oplus \beta$ is consistent, then a complete and clash-free completion graph for $\mathsf{K} \oplus \beta$, otherwise a completion graph containing a clash

```
1:  if there exists $a \in \mathbf{I}_\beta$ s.t. $a \notin \mathbf{I}_\mathsf{K}$ then
2:      Add new node $x_a$ to $\mathcal{V}$ that corresponds to $a$ and set $\mathcal{L}(x_a) = \{a\}$
3:  end if
4:  for all $\alpha \in \beta$ do
5:      if $\upsilon = true$ then
6:          if $\alpha \in \mathsf{K}$ then
7:              continue
8:          end if
9:          $\mathsf{K} = \mathsf{K} + \alpha$
10:         if $\alpha$ of the form $C(a)$ then
11:             Set $\mathcal{L}(x_a) \leftarrow \mathcal{L}(x_a) \cup \{C\}$
12:             Set $\tau(Add(C, \mathcal{L}(x_a))) \leftarrow \{C(a)\}$
13:             Set $\psi(C(a)) \leftarrow \{Add(C, \mathcal{L}(x_a))\}$
14:         else if $\alpha$ of the form $R(a, b)$ then
15:             if $\langle x_a, x_b \rangle \notin \mathcal{E}$ then
16:                 Add $\langle x_a, x_b \rangle$ to $\mathcal{E}$
17:             end if
18:             Set $\mathcal{L}(\langle x_a, x_b \rangle) \leftarrow \mathcal{L}(\langle x_a, x_b \rangle) \cup \{R\}$
19:             Set $\tau(Add(R, \mathcal{L}(\langle x_a, x_b \rangle))) \leftarrow \{R(a, b)\}$
20:             Set $\psi(R(a, b)) \leftarrow \{Add(R, \mathcal{L}(\langle x_a, x_b \rangle))\}$
21:         else if $\alpha$ of the form $a = b$ then
22:             Add $a$ to $\mathcal{L}(x_b)$ and $b$ to $\mathcal{L}(x_a)$
23:             Set $\tau(Add(a, \mathcal{L}(x_b))) \leftarrow \{a = b\}$
24:             Set $\tau(Add(b, \mathcal{L}(x_a))) \leftarrow \{a = b\}$
25:             Set $\psi(a = b) \leftarrow \{Add(a, \mathcal{L}(x_b)), Add(b, \mathcal{L}(x_a))\}$
26:         else if $\alpha$ of the form $a \neq b$ then
27:             Add $x_a \dot{\neq} x_b$
28:             Set $\tau(NE(x_a, x_b)) \leftarrow \{a \neq b\}$
29:             Set $\psi(a \neq b) \leftarrow \{NE(x_a, x_b)\}$
30:         end if
31:     else
32:         $\mathsf{K} = \mathsf{K} - \alpha$
33:         $E \leftarrow \psi(\alpha)$
34:         $\psi(\alpha) \leftarrow \emptyset$
35:         for all $e \in E$ do
36:             $\tau(e) \leftarrow \emptyset$
37:             Roll-back $e$
38:         end for
39:     end if
40: end for
41: If $\mathsf{G}$ contains a clash perform standard back-jumping
42: Apply expansion rules to nodes in $\mathsf{G}$ and proceed with tableau algorithm
43: return $\mathsf{G}$
```

will demonstrate dramatic performance improvements over re-checking consistency from scratch.

5.5 Empirical Results

A prototype of the incremental consistency checking approach as been implemented as an extension to an open source OWL-DL reasoner, Pellet [117]. Pellet is a highly optimized tableau-based DL reasoner that implements the $\mathcal{SHOIQ}$ tableau algorithm [78]. As assumed in the previous discussions, Pellet does not transform ABox assertions in the TBox axioms, therefore the correctness of the algorithm holds.

In order to evaluate the algorithm for the purpose of OWL-based syndication, an empirical evaluation has been performed using various OWL KBs with large ABoxes. This allows the simulation of background information and publications which persist in the broker's KB for a long period of time. Table 5.7 presents an overview of the ontologies have been used in the experiments. They have been selected as a test suite because they are expressed in the DLs that the algorithm supports and provide a range of expressivity. The constructs used in these ontologies align with common usage of OWL constructs that have been observed in a recent comprehensive survey of publicly available OWL ontologies on the Web [146].

Ontology	Expressivity	♯ Classes	♯ Properties
VICODI	$\mathcal{ALHI}$	194	29
SEMINTEC	$\mathcal{ALCIF}$	59	16
LUBM	$\mathcal{SHI}$	43	63
OUBM	$\mathcal{SHIF}$	51	77

Table 5.7: Test-Suite Ontology TBox Overview

The VICODI ontology covers the domain of European history and was created within the VICODI project[8]. The ontology is one of the least expressive ontologies in the test suite and is expressed in the DL $\mathcal{ALHI}$. In particular, it has class and role inclusion

[8]VICODI project homepage: http://www.vicodi.org/

axioms, as well as domain and range constraints. The TBox is quite large, including 194 concepts and 29 roles.

The SEMINTEC ontology models the financial services domain and has been created within a project called SEMINTEC[9]. The SEMINTEC ontology is more expressive than VICODI and is expressed in $\mathcal{ALCIF}$; it additionally utilizes functional roles, negation and universal property restrictions.

LUBM is a benchmark ontology developed at Lehigh University [58]. LUBM models the university domain and is commonly used in literature for performing scalability testing of DL reasoning systems. The ontology is expressed in $\mathcal{SHI}$ and the TBox is of moderate size.

UOB is an extension of the LUBM benchmark ontology; the extension has been developed by IBM and is available through their Integrated Ontology Development Toolkit (IODT)[10]. UOB extends LUBM with more complex modeling and includes functional roles; its expressivity is $\mathcal{SHIF}$.

Ontology	**♯ Individuals**	**♯ Triples**
VICODI1	16,942	54,081
VICODI2	33,884	107,734
SEMINTEC1	17,941	65,560
SEMINTEC2	17,941	65,723
LUBM1	35,882	130,800
LUBM2	37,450	225,095
UOB1	25,272	246,266
UOB2	51,762	423,031

Table 5.8: Test-Suite Ontology ABox Overview

As mentioned earlier, each of these ontologies has large ABoxes, which have been manually created[11] or are automatically generated using dataset generators accompanying the ontologies. In the experiments, different sized ABoxes have been used to investigate

[9]SEMINTEC project homepage: http://www.cs.put.poznan.pl/alawrynowicz/semintec.htm

[10]IODT project homepage: http://www.alphaworks.ibm.com/tech/semanticstk

[11]We would like to acknowledge Boris Motik for his creation of the larger VICODI and SEMITEC datasets (described in [104]).

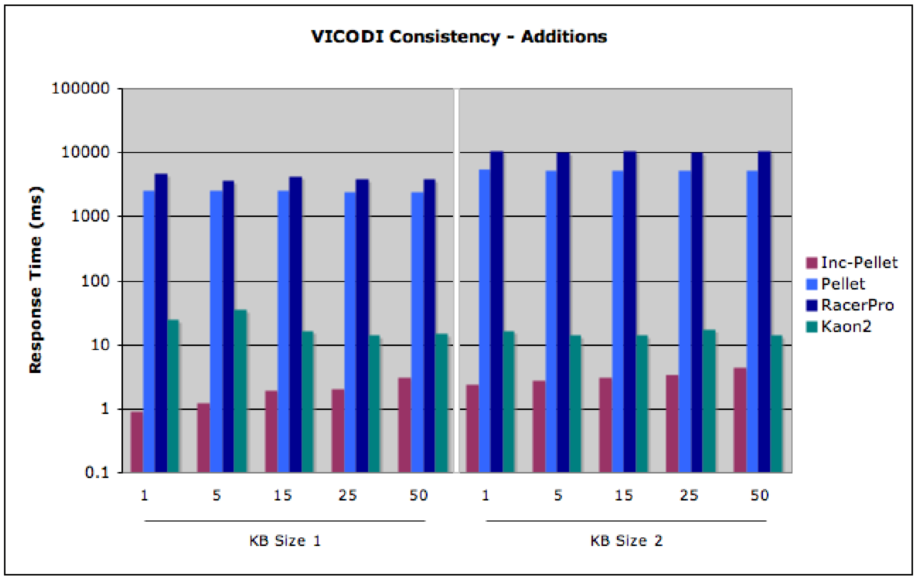


Figure 5.3: Addition Updates of VICODI datasets

the way in which the incremental consistency algorithm scales. Statistics related to these datasets are presented in Table 5.8, which includes the number of individuals in each dataset, as well as the total number of triples in the ABox (which corresponds to the number of ABox assertions).

In order to assess the effectiveness of the algorithm developed in this chapter and its utility for the incremental consistency checking of the syndication broker's KB, the following evaluation has been performed: for each KB size, varying sized ABox additions and deletions were randomly selected from the dataset. Update sizes include 1, 5, 15, 25, and 50 ABox assertions; these sizes were selected as they align with publication sizes expected in realistic syndication systems. These randomly selected assertions where then added (or removed) to the KB and consistency was checked. The evaluation simulates new publications arriving at the syndication broker.

In the evaluations, two versions of the DL reasoner Pellet were used; a regular version of the reasoner and a version that has been extended with the incremental consistency checking algorithm. The DL reasoner RacerPro was also used in the evaluation; similar to Pellet, RacerPro is a highly optimized tableau-based DL reasoner, however RacerPro

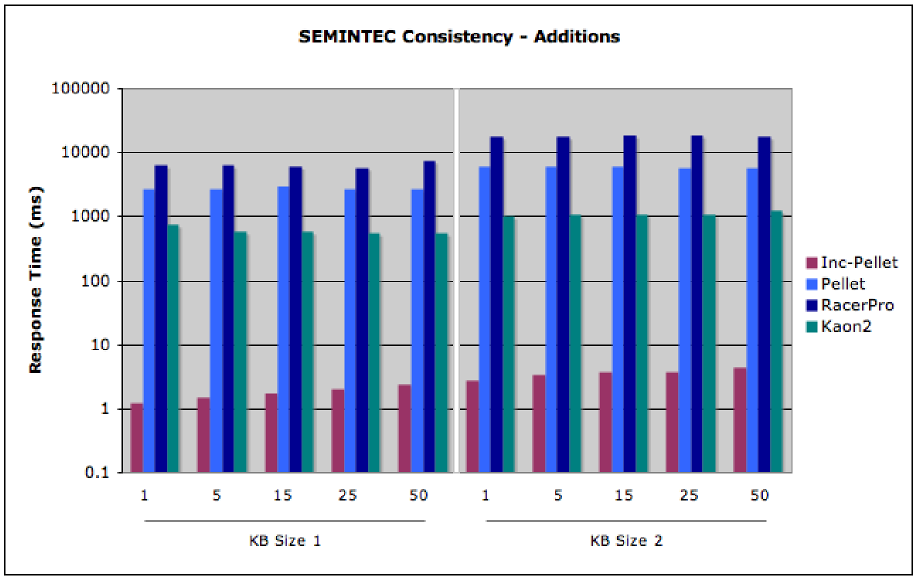


Figure 5.4: Addition Updates of SEMINTEC datasets

is only sound and complete for the DL $\mathcal{SHIQ}$.

The KAON2[12] OWL reasoner was also used in the evaluation; similar to RacerPro, KAON2 only supports reasoning for the DL $\mathcal{SHIQ}$. Interestingly, KAON2 is not a tableau-based reasoner; rather, it reduces OWL KBs to disjunctive datalog and is highly optimized for ABox reasoning [104]. Our aim in using KAON2 in evaluation was to gain insights into tableau-based algorithms for syndication purposes when compared to other possible approaches.

The experiments were run on a Linux machine with 2Gb of RAM and a 3.06GHz Intel Xeon CPU. Pellet v1.5, RacerPro v1.9.0, and KAON2 release 2007-09-07 were used in the experiments. Additionally, all results were averaged over 75 iterations. Note that in all of the figures, the X-axis corresponds to the update size, while the Y-axis is the response time for consistency checking in milliseconds; the scale is logarithmic. Lastly, it is noted that a maximum response time of 100 seconds was imposed in the tests, as any response time above 100 seconds will clearly cause scalability issues for the syndication framework.

[12]KAON2 project homepage: http://kaon2.semanticweb.org/

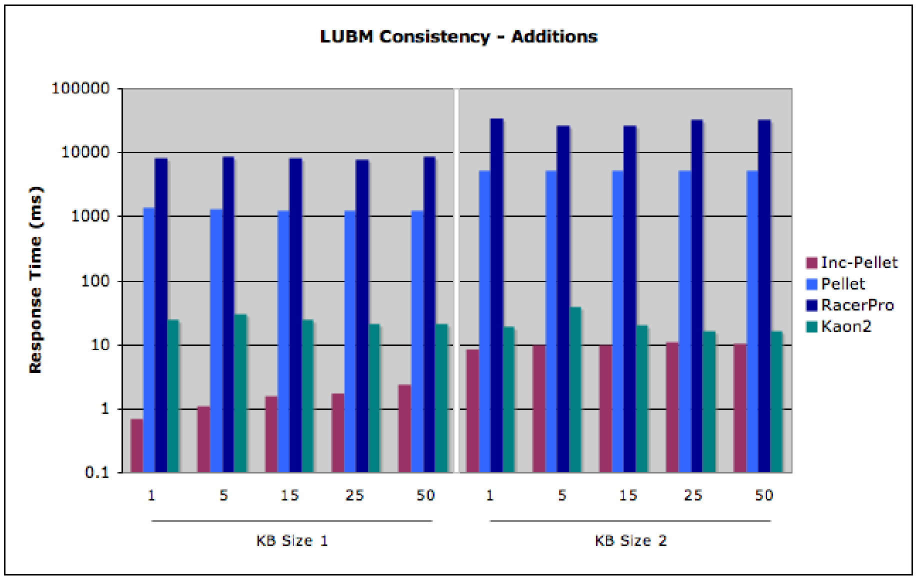


Figure 5.5: Addition Updates of LUBM datasets

Figure 5.3 present the consistency checking times for addition updates for both VICODI datasets. First observe that the consistency checking time for the regular version of Pellet are comparable through update sizes and is between 5 to 8 seconds, depending on the dataset size. This is expected as consistency is rechecked from scratch and the update sizes are small relative to the overall KB size. Response times for RacerPro exhibits similar properties. The incremental consistency checking approach demonstrates substantial performance improvements over both the regular version of Pellet and RacerPro. For both datasets, approximately 3 orders of magnitude performance improvements are exhibited and the response time is always under 10 milliseconds. This is clearly due to the avoidance of reconstructing the entire completion graph. As the update size is increased, the performance of the update approach scales well. KAON2 clearly outperforms the regular tableau-based reasoners, and response times are comparable through the update sizes. However, the incremental version of Pellet performs better than KAON2 in this experiment.

Figure 5.4 presents the addition results for SEMINTEC; note that the response time for Pellet, RacerPro, and KAON2 are again comparable through update sizes. In this

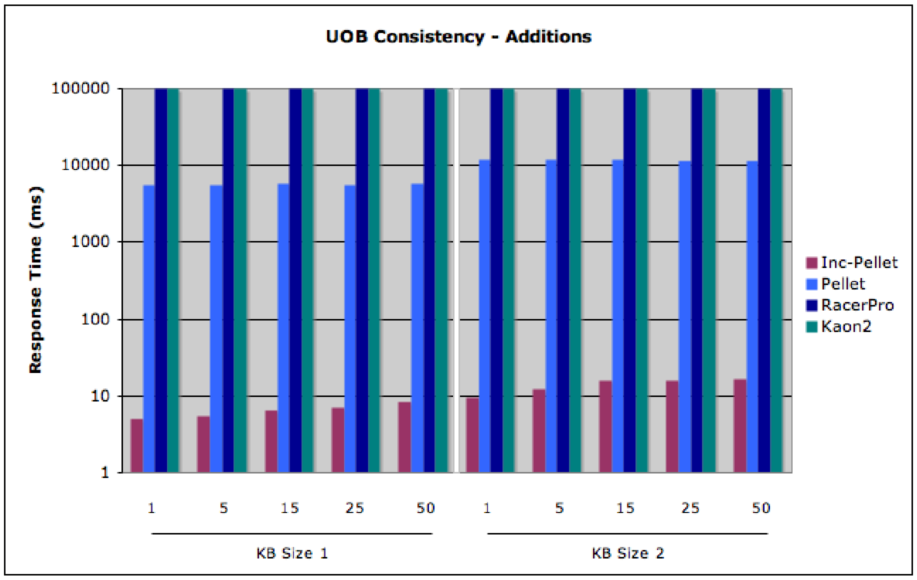


Figure 5.6: Addition Updates of UOB datasets

case KAON2 does not perform as well as in the VICODI datasets. It can be observed that the incremental consistency checking approach exhibits 2 to 3 orders of magnitude performance over the other reasoners.

The results from the LUBM and UOB datasets are presented in Figures 5.5 and 5.6. In the experiments using LUBM, the performance results are similar to those exhibited in the cases for VICODI and SEMINTEC. In contrast for UOB, both KAON2 and RacerPro did not terminate within 100 seconds. However, the incremental algorithm demonstrates substantial performance improvements over Pellet and responses times generally under 10 milliseconds.

Figures 5.7 & 5.8 present the deletion results for VICODI and SEMINTEC. The results for the regular version of Pellet, RacerPro, and KAON2 are essentially the same as the results from the addition experiments with these ontologies. As in the results for additions, the incremental consistency checking algorithm demonstrates substantial performance improvements.

Figure 5.9 presents the deletion results for LUBM; again the results are similar for Pellet, RacerPro, and KAON2. Similarly, substantial performance improvements are ob-

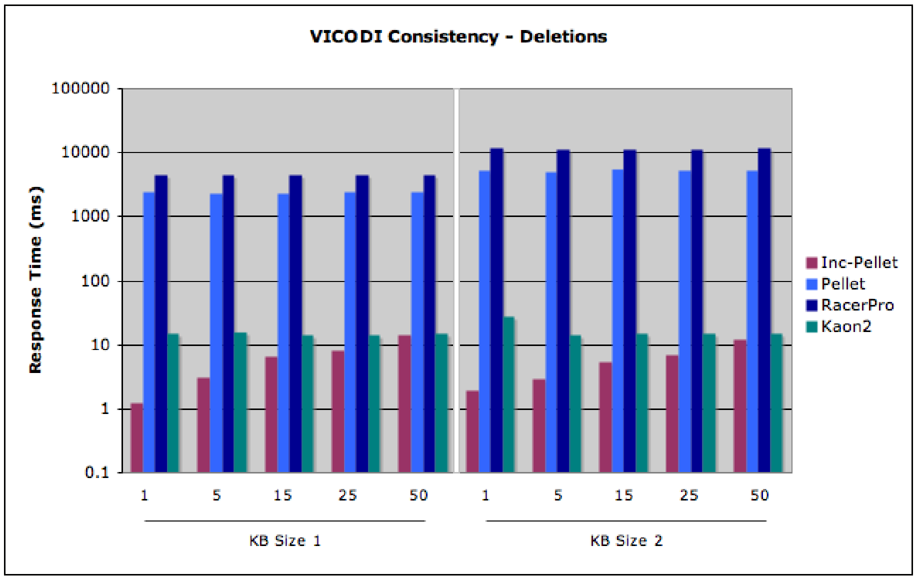


Figure 5.7: Deletion Updates of VICODI datasets

served using the incremental reasoning technique over both the regular version of Pellet and RacerPro. It can be observed that the technique also outperforms KAON2, with the exception of larger sized updates for the LUBM2 dataset. However, in this case reasoning is still in the 10s of milliseconds using the algorithm presented in this chapter and substantially outperforms the regular tableau-based algorithms. Figure 5.10 presents the deletion results for UOB, and the results exhibit similar characteristics as the previous cases.

Table 5.9 presents a summary of the distribution of response times for publications of size 50 observed in the evaluation. In the table, the notation “VIC-1” and “VIC-2” is used to denote the VICODI KB of size 1 and 2 respectively (the same notation is used for the other ontologies as well); additionally, the table presents the minimum, median, maximum, and average response times observed, as well as the standard deviation. This provides additional insights into the results of the algorithm, as it demonstrates that the response times observed for the ontologies are generally very close to the average.

One issue with the algorithm developed in this chapter is related to the impact on memory due to axiom tracing. I have investigated this and the memory overhead is pre-

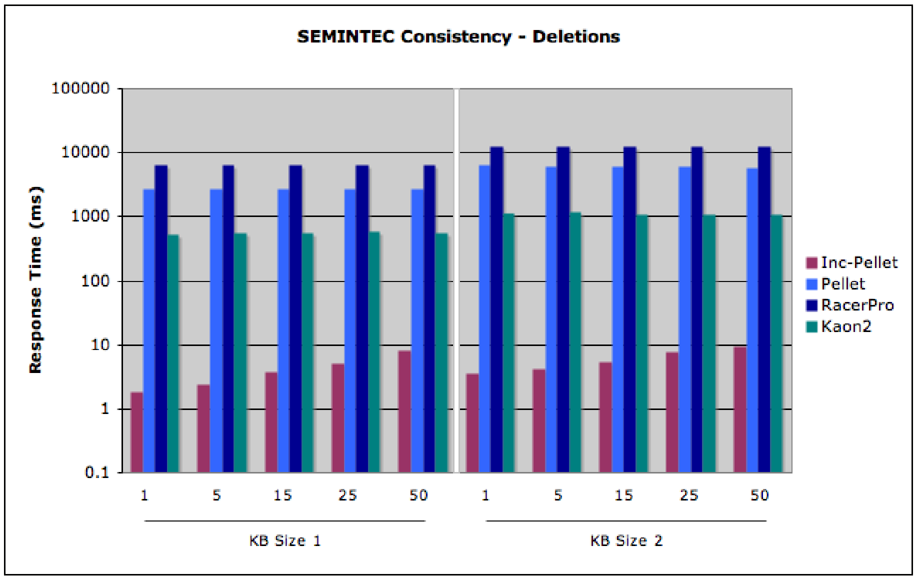


Figure 5.8: Deletion Updates of SEMINTEC datasets

KB	Additions					Deletions				
	Min	Med	Max	Avg	Stdv	Min	Med	Max	Avg	Stdv
VIC-1	2	3	4	3	.5	10	14	23	14	2.6
VIC-2	3	4	6	4.4	.7	8	12	17	12	1.5
SEM-1	2	2	3	2.4	.4	6	8	27	8.1	2.6
SEM-2	3	4	6	4.3	.5	7	9	50	9.6	4.9
LUBM-1	2	2	4	2.4	.5	4	6	13	5.9	1.3
LUBM-2	8	10	15	10.6	1.06	33	75	141	78.6	21.7
UOB-1	6	9	12	8.4	1.1	15	30	45	28.7	6.3
UOB-2	14	16	29	16.4	2	31	57	91	57.6	13.5

Table 5.9: Distribution of Response Times for Updates of Size 50 (time in milliseconds)

sented in Table 5.10. This ranges from approximately 50 to 240 mb. While there is overhead introduced, the approach provides dramatic performance improvements over checking consistency from scratch.

5.6 Discussion

The empirical results presented in the previous section demonstrate that incremental consistency can be performed in a practical manner for realistic KBs. In the remainder of

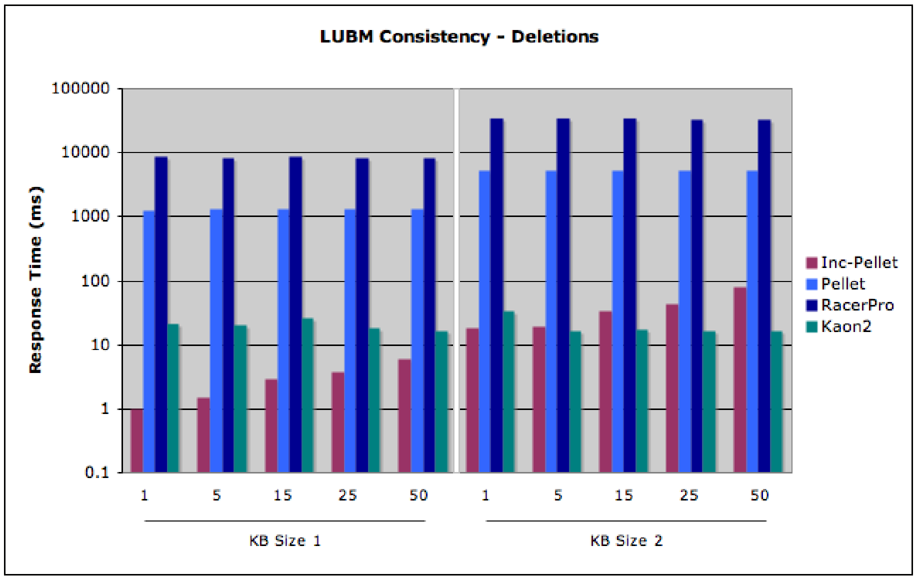


Figure 5.9: Deletion Updates of LUBM datasets

Ontology	Tracing Memory (mb)
VICODI1	81
VICODI2	121
SEMINTEC1	51
SEMINTEC2	105
LUBM1	64
LUBM2	185
OUBM1	109
OUBM2	240

Table 5.10: Memory Overhead

this chapter a few open issues will be discussed.

First, the approach presented in this chapter is only applicable to the DLs $\mathcal{SHOQ}$ and $\mathcal{SHIQ}$; this is primarily due to fact that there is no expansion rule ordering imposed by the tableau algorithm when these logics are considered. Extending the technique to support $\mathcal{SHOIQ}$ is not addressed in this dissertation and is left as future work. The limitation to $\mathcal{SHOQ}$ and $\mathcal{SHIQ}$ is reasonable because a very large subset of OWL has been covered using the algorithm just described and it demonstrates our ability to handle more expressive syndication. As pointed out earlier, recent surveys of publicly available

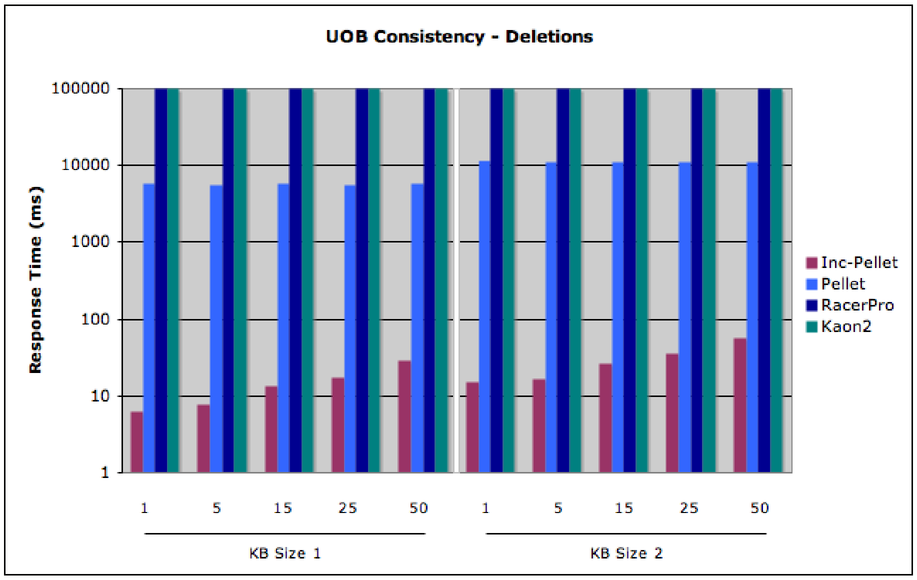


Figure 5.10: Deletion Updates of UOB datasets

ontologies indicate that a majority of ontologies are expressed in $\mathcal{SHIQ}$, $\mathcal{SHOQ}$, or one of their sublanguages [146]. In particular, over 88% of the surveyed ontologies are expressed in $\mathcal{SHIF}$ or one of its sub-languges.

Lastly, if the KB is inconsistent after the update, nothing is done to resolve the inconsistency. This issue is the focus of Chapter 7.2.

Chapter 6

Incremental Query Answering

As discussed in Chapter 4, registered subscriptions in the syndication framework are represented as continuous conjunctive retrieval queries. Therefore, matching newly published information in the framework reduces to DL query answering. Unfortunately, as in the case for consistency checking (discussed in the previous chapter), if standard DL reasoning algorithms and off-the-shelf reasoners are used within this syndication framework, scalability issues with respect to query answering are immediately encountered. Let us consider current query answering response times using today's tableau-based reasoners for the publicly available OWL ontology from the financial domain discussed in Chapter 5.1[1]; assume that a subscriber is interested in information matches about insurance payments. Given this, the subscriber registers a subscription using the concept *InsurancePayments* from the ontology. The subscription is represented by the following:

$$((x) \leftarrow \mathit{InsurancePayment}(x), \infty)$$

Then, when a new publication is received at the broker, the query is re-evaulated over the updated broker's KB. Table 6.1 presents the query answering times using the two tableau-based OWL reasoners as used in Chapter 5.1, Pellet and RacerPro. The times shown are only for actual query answering and do not include consistency checking or query preparation time. It is clear that the response times using these reasoners demonstrate that scalability issues will be encountered if they are used for matching in a high-demand (i.e., high publication rate) syndication domain.

Therefore, in this chapter, a technique is developed for more efficient incremental query answering. One of the main issues with current query answering algorithms is that

[1]Available at http://www.cs.put.poznan.pl/alawrynowicz/financial.owl

Reasoner	Response Time (sec)
Pellet	1.69
Racer	14.4

Table 6.1: Query Answering Times for SEMINTEC Ontology

after an update the entire knowledge base is considered when re-evaluating the query. Given this, the technique developed in this chapter aims to reduce the portion of the KB that must be considered as candidate answers after an update; therefore, the query only needs to be re-evaulated over a subset of the KB. This aligns with work in relational and deductive database query and view maintenance (see Chapters 3.2 & 3.2 for a discussion), in which the entire query result or view is not reconstructed from scratch given an update to the database.

6.1 Main Observation

In this section, the main theorem underlying the approach developed in this chapter is introduced. Let us assume that we are given a retrieval query $(x) \leftarrow C(x)$ for some concept C and named individual a. Additionally, say that an ABox update β is integrated into the KB under syntactic updates, such that the resulting KB is consistent. The main insight underlying the approach is that the dependencies of clashes in completion graphs for the KB caused by β and $\neg C(a)$ can be exploited to provide an overestimate of the individuals that instantiate C after the update. Before formally presenting the necessary conditions for the entailment, the definition of the dependency of a node label in a completion graph is presented.

Definition 14 *(Label Dependence) Define a node label $l \in \mathcal{L}(x)$ to be dependent on a node label $C \in \mathcal{L}(y)$ (or node $y \in \mathcal{V}$, edge $\langle y,z\rangle \in \mathcal{E}$, or edge label $R \in \mathcal{L}(\langle y,z\rangle)$) if during the application of expansion rules to construct completion graph G, l is added to $\mathcal{L}(x)$ due to the existence of $C \in \mathcal{L}(y)$ (respectively $y \in \mathcal{V}$, $\langle y,z\rangle \in \mathcal{E}$, $R \in \mathcal{L}(\langle y,z\rangle)$).*

The notion of *clash* dependence is a straightforward extension of this definition; that is, if a clash[2] $c = (x, \neg C, C)$ is observed and either $\neg C$ or C is dependent on $l \in \mathcal{L}(y)$ (or node $y \in \mathcal{V}$, edge $\langle y, z\rangle \in \mathcal{E}$, or edge label $R \in \mathcal{L}(\langle y, z\rangle)$), then the clash is said to be dependent on $l \in \mathcal{L}(y)$ (respectively $y \in \mathcal{V}$, $\langle y, z\rangle \in \mathcal{E}$, $R \in \mathcal{L}(\langle y, z\rangle)$). Finally, we say that a label $l \in \mathcal{L}(x)$ is dependent on an update β if β causes the addition of some node label $C \in \mathcal{L}(y)$ (or node $y \in \mathcal{V}$, edge $\langle y, z\rangle \in \mathcal{E}$, or edge label $R \in \mathcal{L}(\langle y, z\rangle)$) s.t. l is dependent on $C \in \mathcal{L}(y)$ (respectively $y \in \mathcal{V}$, $\langle y, z\rangle \in \mathcal{E}$, $R \in \mathcal{L}(\langle y, z\rangle)$); note that a clash dependency on an update can be defined in a similar way.

Given this, the main theorem underlying the approach developed in this chapter is introduced[3]; the theorem presents two conditions, one of which must be satisfied in order for the new entailment to occur (or be invalidated). For ease of exposition, when referring to the addition of the structure of a set of ABox assertions β to a completion graph G and then applying any sequence of the necessary expansion rules (as discussed in Chapter 5[4]), the terminology "*adding* β to G", denoted $\mathsf{G} \uplus \beta$, will be used.

Theorem 3 *Let* K *be a consistent* $\mathcal{SHI}$ *KB,* β *an ABox addition (or deletion), and* C *some* $\mathcal{SHI}$ *concept. If* $\mathsf{K} \not\models C(a)$ *(respectively* $\mathsf{K} \models C(a)$*) for some* $a \in \mathbf{I}_{\mathsf{K}} \cup \mathbf{I}_{\beta}$, $\mathsf{K} + \beta \not\models \bot$, *and* $\mathsf{K} + \beta \models C(a)$ *(respectively* $\mathsf{K} - \beta \not\models C(a)$*), then one of the following conditions is satisfied when adding* $\beta \cup \{\neg C(a)\}$ *to every* $\mathsf{G} \in \mathit{Comp}(\mathsf{K})$*:*

1. *there exists* $\mathsf{G} \in \mathit{Comp}(\mathsf{K})$ *(respectively* $\mathsf{G} \in \mathit{Comp}(\mathsf{K} - \beta)$*) s.t.* $\mathsf{G} \uplus (\beta \cup \{\neg C(a)\})$ *results in a clash* c *that is dependent on both* $\neg C(a)$ *and* β

2. *there exists the same node* x *with* $D_1 \sqcup D_2 \in \mathcal{L}(x)$ *in* $\{\mathsf{G}_1, \mathsf{G}_2\} \subseteq \mathit{Comp}(\mathsf{K})$ *(respectively* $\{\mathsf{G}_1, \mathsf{G}_2\} \subseteq \mathit{Comp}(\mathsf{K} - \beta)$*) such that:*

[2]Note that due to the fact that number restrictions are not allowed in $\mathcal{SHI}$, clashes only occur if $\{\neg C, C\} \subseteq \mathcal{L}(x)$ for some node x; when referring to a clash in the label of node x, the clash will be denoted as a triple $(x, \neg C, C)$.

[3]As stated in Chapter 2.4.3, the notation $\mathit{Comp}(\mathsf{K})$ will be used to denote the set of complete, clash-free completion graphs that can be constructed for K.

[4]In this discussion, it is assumed that back-jumping is not used, as all completion graphs are maintained.

- $\mathsf{G}_1 \uplus (\beta \cup \{\neg C(a)\})$ *results in a clash that is independent of* $\neg C(a)$ *and is dependent on both* β *and* $D_1 \sqcup D_2 \in \mathcal{L}(x)$
- $\mathsf{G}_2 \uplus (\beta \cup \{\neg C(a)\})$ *results in a clash that is independent of* β *and is dependent on both* $\neg C(a)$ *and* $D_1 \sqcup D_2 \in \mathcal{L}(x)$

Proof See Appendix A.2.1 for the proof of this theorem. □

It is important to reiterate that in condition 2 of the theorem, the node x in G_1 and G_2 corresponds to the same node (which corresponds to either a named or existential individual). Note that the case in which x corresponds to an existential is not problematic to maintain, as nodes are not merged (i.e., number restrictions are not supported in $\mathcal{SHI}$) and all completion graphs are maintained; therefore, when a disjunction is encountered on an existential and various new completion graphs are constructed, the correspondences between the existential node x in each of the completion graphs can trivially be determined.

Intuitively, the first condition of Theorem 3 states that for a named individual a to instantiate a concept after an addition, then some clash observed when incrementally updating a completion graph for the original KB with β and $\neg C(a)$ will be dependent on structures from both β and $\neg C(a)$. On the other hand, the second condition states that $\neg C(a)$ and β will cause clashes in different completion graphs that are dependent on the same non-determinstic choice (i.e., a disjunction label). Analogous statements can be made for deletions.

6.2 Naïve Approach

Given a retrieval query composed of a single DL concept, Theorem 3 implies that if all completion graphs for the KB are maintained through updates and the two conditions are checked, then the detection of new candidate bindings (respectively invalidated bindings for deletions) for a given query can be accomplished; then, only this subset of the individuals would have to actually be checked for the entailment. The main insight is that if this technique (or an overestimate of it) can be accomplished in a practical manner, then this set of candidates may be a small subset of the original KB, thereby decreasing the overhead of re-evaluting queries given an ABox update. In the remainder of this section, a naïve approach is outlined which exploits Theorem 3 for this task. Then, in the remainder of this chapter, I extend the technique to make it practical.

6.2.1 ABox Additions

It can be observed that in order to take into account the first condition of Theorem 3 for additions, one can update all completion graphs for the initial KB with β and track the label dependencies for β. Following this, one can update the resulting complete and clash-free completion graphs with $\neg C(a)$ and determine which clashes are dependent on both β and $\neg C(a)$. This is sufficient due to the fact that there is not an expansion rule application ordering imposed for $\mathcal{SHI}$ and condition 1 of Theorem 3 states the clash must be dependent on both β and $\neg C(a)$ when applying any sequence of the necessary expansion rules[5]. An additional observation related to this is also made; in particular, for a clash to be dependent on both β and $\neg C(a)$, then after adding β to all $\mathsf{G} \in \mathit{Comp}(\mathsf{K})$ resulting in the set of complete and clash-free completion graphs $\mathsf{G}_{\mathsf{K}+\beta}$, it must be the case that when adding $\neg C(a)$ to some $\mathsf{G}' \in \mathsf{G}_{\mathsf{K}+\beta}$, a root node that had a node label, edge, or edge label added due to β must have a label added due $\neg C(a)$. This is a consequence of the previous observation and the tree-like model property of $\mathcal{SHI}$, which intuitively states that the completion graph will be a forest of trees rooted at nodes corresponding to named individuals. Therefore, one can easily determine an over-estimate of the individuals that could satisfy condition 1 by first updating all $\mathsf{G} \in \mathit{Comp}(\mathsf{K})$ with β, while tracking the root nodes N with a node label, edge, or edge label change. Then, the following can be

[5]Note, however, there could be a case in which there is an immediate clash as a result of adding the structures of β and $\neg C(a)$ to some $\mathsf{G} \in \mathit{Comp}(\mathsf{K})$ prior to applying any expansion rules. Using this approach, such a case would not be detected; however this case can trivially be covered by inspecting β and $\neg C(a)$.

performed for each individual a to determine if a is in fact a candidate: the negated query concept, $\neg C$, can be to $\mathcal{L}(x_a)$[6] in each updated completion graph, and for the condition to be satisfied for a, a label must be added to some $n \in N$ due to the addition of $\neg C$ to $\mathcal{L}(x_a)$.

A straightforward observation is made regarding the second claim for Theorem 3 as well. In particular, when updating each $\mathsf{G} \in \mathit{Comp}(\mathsf{K})$ with β, all clashes dependent on β that are independent of $\neg C$ will be observed. Therefore, if the dependencies of labels on disjunctions are tracked during the tableau algorithm (addressed in more detail shortly), one can easily determine the individuals that satisfy the second condition by adding $\neg C$ to each individual and checking if this causes a clash that is dependent on some disjunction that contributed to a clash when updating each $\mathsf{G} \in \mathit{Comp}(\mathsf{K})$ with β. Similar to the discussion for the first condition, an over-estimate of this approach can be provided as well; in particular, using a disjunction label dependency function (addressed shortly), one can easily determine the root nodes N that have a label or non-root descendent with a label that is dependent on some disjunction that contributed to a clash observed when adding β to the completion graph. Then, an overestimate of the individuals which satisfy the second condition can be provided by determining the individuals a such that adding $\neg C$ to $\mathcal{L}(x_a)$ causes some label to be added to some $n \in N$. This is sufficient because for there to be a clash dependent on the disjunction, it must be the case that the expansion rules add a label to one of these root nodes.

Such a disjunction dependency tracking function can easily be maintained in a similar manner to the axiom tracing technique discussed in Chapter 5. In particular, when a disjunction is added to a node in a completion graph, the function will be updated with the label addition events that are a result of the disjunction. As this can be accomplished in an analogous manner as axiom tracing, this function is simply assumed.

[6]Note that as in the previous chapter, given a named individuals a, denote by x_a the root node corresponding to a.

6.2.2 ABox Deletions

By additionally leveraging the axiom tracing function introduced in the previous chapter, this theorem can be exploited to support incremental deletions in a similar to the approach just discussed for addition updates; specifically, due to the completeness of the axiom tracing function (shown in Theorem 1), if there was a change event in some $\mathsf{G} \in \mathit{Comp}(\mathsf{K})$ that was caused due to the existence of some $\alpha \in \beta$, then it must be the case that there is a corresponding change event in $\psi(\alpha)$ reflecting this change. Thus, the complete and clash-free completion graphs for $\mathsf{K} - \beta$ can easily be obtained by first rolling-back the change events in each $\mathsf{G} \in \mathit{Comp}(\mathsf{K})$ that are dependent on some $\alpha \in \beta$ and then applying the necessary expansion rules as in Chapter 5[7]. Thus, the deletions can be supported in the same manner as additions.

6.2.3 Example

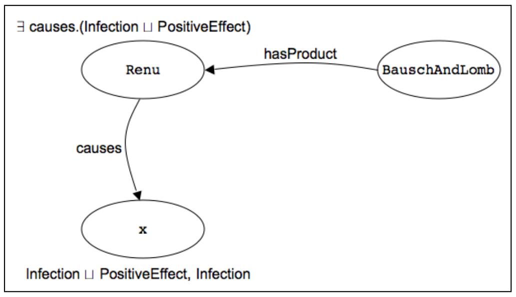
To further illustrate the general approach, consider the KB presented in Table 6.2, which consists of two ABox assertions stating that *BauschAndLomb* has product *Renu* and that *Renu* causes some thing that is either an *Infection* or a *PositiveEffect* (for simplicity, there are not any TBox axioms in the KB).

Knowledge Base
hasProduct(BauschAndLomb,Renu),
$\exists \mathit{causes}.(\mathit{Infection} \sqcup \mathit{PositiveEffect})(\mathit{Renu})$

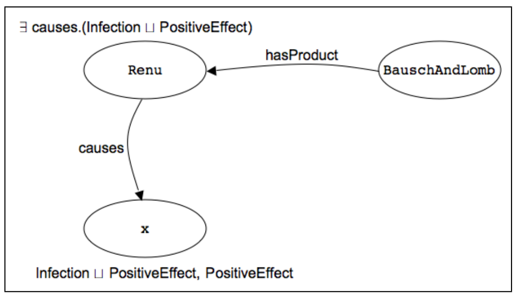
Table 6.2: Sample Knowledge Base

When performing the tableau algorithm for this KB, the complete and clash-free completion graphs shown in Figure 6.1 would be constructed. Note that there are two completion graphs due to the type assertion for *BaushAndLomb* involving a disjunction.

[7]Note that all non-determinstic choices must be explored when applying the expansion rules and back-jumping does not need to be used as all completion graphs are maintained; additionally, it is assumed that the disjunction and clash dependency functions are updated to reflect the deletion by simply retracting entries that involve some structure removed during the roll-back.


∃ causes.(Infection ⊔ PositiveEffect)
hasProduct
Renu
BauschAndLomb
causes
x
Infection ⊔ PositiveEffect, Infection

(a)


∃ causes.(Infection ⊔ PositiveEffect)
hasProduct
Renu
BauschAndLomb
causes
x
Infection ⊔ PositiveEffect, PositiveEffect

(b)

Figure 6.1: (a) Completion Graph 1. (b) Completion Graph 2.

Next, assume that a query is issued over the KB for all things that have some product which causes an infection; more formally the query is represented by $(x) \leftarrow (\exists hasProduct.(\exists causes.Infection))(x)$. Given an update, for an individual to now instantiate the query concept, then one of the two conditions presented in Theorem 3 must be satisfied. To demonstrate the first condition, let us assume that an ABox addition is received stating that *Renu* only causes infections, denoted by ($\forall$*causes*.*Infection*)(*Renu*). If we now consider whether *BauschAndLomb* satisfies condition 1, it can be seen that when adding the negation of the query concept (i.e., $\forall hasProduct.(\forall cause.(\neg Infection))$, to $\mathcal{L}$(*BauschAndLomb*) and $\forall causes.Infection$ to $\mathcal{L}$(*Renu*) (corresponding to the structure of the update) in the second completion graph, the clash $(x, Infection, \neg Infection)$ would be observed which is dependent on both of these labels; this is depicted in Figure 6.2. Therefore, *BaushAndLomb* must be considered as a candidate for the query concept; note, however, that in this case, it does not instantiate the query.

Next, let us consider the second condition of Theorem 3; in this case, assume that an ABox addition is received which states that *Renu* does not cause *PositiveEffects*; formally ($\forall causes.\neg$*PositiveEffect*)(*Renu*). If we consider *BauschAndLomb* again, there would be a clash involving the labels ¬*Infection* and *Infection* in the first completion graph that is dependent on the negated query concept and the disjunct *Infection* from the disjunction *Infection* ⊔ *PositiveEffect*. Further, the addition will in turn cause a clash with the other disjunct (i.e. *PositiveEffect*) of the disjunction in $\mathcal{L}(x)$. Therefore, the second condition

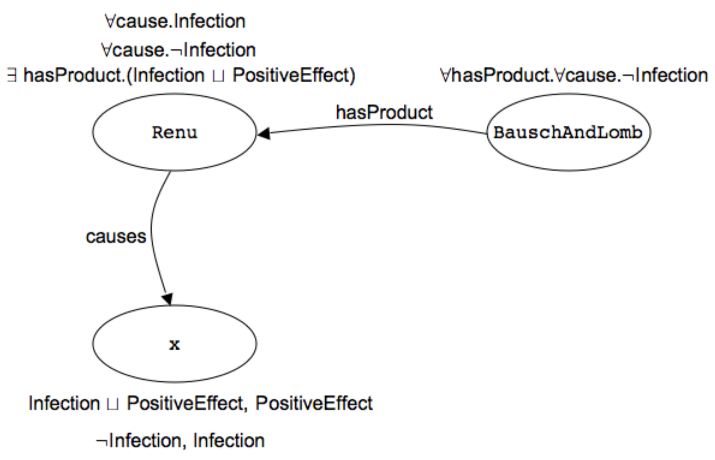
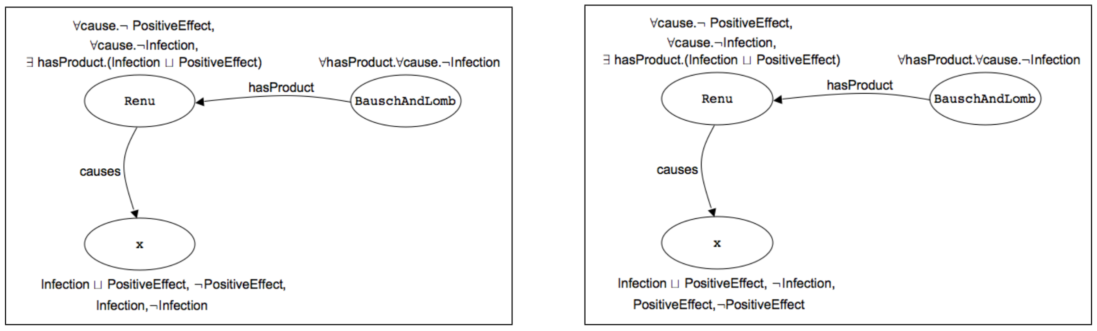

∀cause.Infection
∀cause.¬Infection
∃ hasProduct.(Infection ⊔ PositiveEffect)
∀hasProduct.∀cause.¬Infection
hasProduct
Renu
BauschAndLomb
causes
x
Infection ⊔ PositiveEffect, PositiveEffect
¬Infection, Infection

Figure 6.2: Clash Satisfying Condition 1


∀cause.¬ PositiveEffect,
∀cause.¬Infection,
∃ hasProduct.(Infection ⊔ PositiveEffect)
∀hasProduct.∀cause.¬Infection
hasProduct
Renu
BauschAndLomb
causes
x
Infection ⊔ PositiveEffect, ¬PositiveEffect,
Infection,¬Infection
∀cause.¬ PositiveEffect,
∀cause.¬Infection,
∃ hasProduct.(Infection ⊔ PositiveEffect)
∀hasProduct.∀cause.¬Infection
hasProduct
Renu
BauschAndLomb
causes
x
Infection ⊔ PositiveEffect, ¬Infection,
PositiveEffect,¬PositiveEffect

(a) (b)

Figure 6.3: Clashes Satisfying Condition 2: a) Clash in completion graph 1 due to negated query concept. b) Clash in completion graph 2 due to ABox addition.

will be satisfied, and in this case, the updated KB would in fact entail that *BauschAndLomb* is a new answer for the query. The resulting completion graphs containing the clashes are depicted in Figure 6.3

6.2.4 Discussion

It is clear that adopting such a naïve approach to determine the candidate individuals will impose substantial overhead, and likely be worse than simply running the query from scratch. This is because one would have to perform such a check for each named individual in all completion graphs, which would clearly be ineffective. Further, maintaining all completion graphs for the KB is itself impractical due to the potential exponential number of completion graphs.

Given this, in the remainder of this chapter, a more practical approach to exploit Theorem 3 is presented. First, the issue of adding the negated query concept to all named individuals is addressed in section 6.4; the main idea of the technique presented is to build a structure that can be used to perform a structural search in each completion graph after it has been incrementally updated with an ABox update; then, only the nodes corresponding to named individuals reached during this search need to be considered. In using such an approach, applying the expansion rules to the negated query concept and attempting to detect clashes is almost entirely avoided.

Secondly, in section 6.5, I develop a technique to avoid maintaining all completion graphs for the KB; the main insight is to construct a single completion graph structure which acts as a summary for all completion graphs, therefore allowing only the maintenance of this one (summary) completion graph. However, until section 6.5, it is assumed that all complete and clash free completion graphs for the KB are maintained through updates. Further, in the remainder of this chapter, when referring to the update of some completion graph, it is assumed that backtracking does not occur during the reapplication of expansion rules; this is because it is assumed all completion graphs are maintained.

6.3 Assumptions

In the techniques developed in the remaining sections of this chapter some restrictions and assumptions are imposed on the queries supported. First, it must be the case that the query can be rolled-up into a distinguished variable (see Chapter 2.4.2 for a discussion about the rolling-up procedure). This implies that the rolling-up technique must be applicable to the query and that it must contain at least one distinguished variable (implying it is a retrieval query). While this is a restriction of the approach in general, it is noted that the requirement that the query is a retrieval query does not directly impact the utility of the technique for the purpose of the syndication framework; this is because subscriptions are in fact retrieval queries. The second restriction is that transitive roles or sub-roles of transitive roles are not allowed as query atoms (i.e., all roles are simple); this is necessary to ensure completeness of the proposed techniques. However, as stated in Chapter 2.4.2, rolling-up arbitrarily shaped queries in the presence of transitive roles is known to be problematic [140, 52, 51]; therefore, this restriction implies that the query can in fact be rolled-up (as required in the first restriction). It is noted that an additional syntactic restriction is imposed on the queries supported, however this will be discussed later in section 6.4.

Lastly, without loss of generality, in the remainder of this chapter we assume that the query is connected [50].

6.4 Concept Guide

In this section, an approach is presented for avoiding the addition of the negated query concept to all named individuals. The main goal behind the approach is to build a structure that can be used to determine the propagation of labels to root nodes in a completion graph due to the addition of the negated query concept. Therefore, this structure can be used to detect the individuals such that if the query concept were added to their node label, the application of expansion rules could cause a label to propagate to a node with some other label, (incoming/outoing) edge, or edge label dependent on the update β or some disjunction. As discussed earlier, determining this propagation is necessary in order to check if the conditions of Theorem 3 are satisfied. In the remainder of this section, only retrieval queries involving a single DL concept are assumed (i.e., concept retrieval queries); however, the technique introduced here is extended to complex query patterns in section 6.6.

In order to support general TBoxes, a restriction is imposed on the queries supported in the approach. Before explaining this restriction, the following notation is introduced:

given KB K, let C_T denote the concept constructed as a result of internalizing[8] the TBox for K into a single concept (in NNF). Additionally, given a concept D, let $clos(D)$ denote the smallest set of concepts containing D that is closed under sub-concepts (in NNF); further, given KB K, let $clos(\mathsf{K})$ denote the union of $clos(C_T)$ and all $clos(D)$ for each concept assertion $D(a) \in \mathsf{K}$; that is $clos(\mathsf{K}) = clos(C_T) \cup \bigcup_{D(a)\in\mathsf{K}} clos(D)$.

Then, given retrieval query for concept C (in NNF), it is assumed that if $\forall R.D \in clos(\mathsf{K}) \cup clos(\neg C)$, then it must be the case that $\exists P.E \notin clos(\neg C)$, where $\mathsf{Inv}(P) \sqsubseteq R$. Intuitively, given a complete and clash-free completion graph G and some named individual a, this restriction ensures that if a new edge is added to G as a result of extending G with $\neg C(a)$, then no concepts will be transfered back up this newly added edge. This effectively isolates the propagation of labels due to expansion rule applications from the addition of the negated query concept to G (see section 6.10 for a discussion regarding the impact of the restriction in practice). Given a KB K and concept C, we will say that C is *safe* with respect to K if C satisfies the restriction just introduced.

I now introduce the structure that is leveraged for determine the effects of adding a concept name to the label of a node in a completion graph; this is referred to as a *concept guide*. Intuitively, a concept guide is a labeled, directed graph, which is formally defined in Definition 15.

Definition 15 *(Concept Guide) Define a concept guide $\mathcal{G}$ to be a labeled directed graph $\mathcal{G} = (N_{\mathcal{G}}, E_{\mathcal{G}}, L_{\mathcal{G}})$. Each node $n \in N_{\mathcal{G}}$ is labeled with a non-empty set of $\mathcal{SHI}$ concepts, and each edge $(n, m) \in E_{\mathcal{G}}$ is labeled with a non-empty set of role names.*

Note that for ease of readability later in this chapter, edges in a concept guide are denoted via (n, m), whereas edges in a completion graph are denoted by $\langle x, y\rangle$. The concept guide is built by repeatedly inspecting the form of the concept names in node labels in the concept guide.

[8]See Chapter 2.4.3 for a discussion of the process.

Definition 16 *(Concept Guide Construction) Let* K *be a* $\mathcal{SHI}$ *KB,* C *a* $\mathcal{SHI}$ *concept, and* $\mathcal{G} = (\emptyset, \emptyset, L)$*. Define the concept guide for* C*, denoted* $guide(C)$*, to be initialized with* $n \in N_{\mathcal{G}}$ *and* $L_{\mathcal{G}}(n) \leftarrow \{C\}$*. Then define the following rules to be repeatedly applied to the concept guide node labels until no further modifications can be made to* $\mathcal{G}$*:*

1. *if* $\forall R.D \in L_{\mathcal{G}}(n)$ *and* $(n, m) \notin E_{\mathcal{G}}$ *s.t.* $R \in L_{\mathcal{G}}((n, m))$ *and* $D \in L_{\mathcal{G}}(m)$ *then*
   (a) *add a new node* m *to* $N_{\mathcal{G}}$ *and set* $L_{\mathcal{G}}(m) \leftarrow \{D\}$
   (b) *set* $E_{\mathcal{G}} \leftarrow E_{\mathcal{G}} \cup \{(n, m)\}$
   (c) *set* $L_{\mathcal{G}}((n, m)) \leftarrow L_{\mathcal{G}}((n, m)) \cup \{R\}$
2. *if* $\forall R.D \in L_{\mathcal{G}}(n)$*, there is some* S *with* $\mathsf{Trans}(S)$ *and* $S \sqsubseteq^* R$*, and* $\{(n, m), (m, m)\} \not\subseteq E_{\mathcal{G}}$ *s.t.* $R \in L_{\mathcal{G}}((n, m))$*,* $D \in L_{\mathcal{G}}(m)$*, and* $S \in L_{\mathcal{G}}((n, m))$ *and* $S \in L_{\mathcal{G}}((m, m))$ *for all* S *s.t.* $\mathsf{Trans}(S)$ *and* $S \sqsubseteq^* R$ *then*
   (a) *add a new node* m *to* $N_{\mathcal{G}}$ *and set* $L_{\mathcal{G}}(m) \leftarrow \{D\}$
   (b) *set* $E_{\mathcal{G}} \leftarrow E_{\mathcal{G}} \cup \{(n, m)\} \cup \{(m, m)\}$
   (c) *set* $L_{\mathcal{G}}((n, m)) \leftarrow L_{\mathcal{G}}((n, m)) \cup \{R\}$
   (d) *set* $L_{\mathcal{G}}((n, m)) \leftarrow L_{\mathcal{G}}((n, m)) \cup \{S\}$ *and* $L_{\mathcal{G}}((m, m)) \leftarrow L_{\mathcal{G}}((m, m)) \cup \{S\}$ *for all* S *s.t.* $\mathsf{Trans}(S)$ *and* $S \sqsubseteq^* R$
3. *if* $C_1 \sqcap C_2 \in L_{\mathcal{G}}(n)$ *or* $C_1 \sqcup C_2 \in L_{\mathcal{G}}(n)$ *and* $\{C_1, C_2\} \not\subseteq L_{\mathcal{G}}(n)$ *then set* $L_{\mathcal{G}}(n) \leftarrow L_{\mathcal{G}}(n) \cup \{C_1, C_2\}$

It is a straightforward consequence of the definition that the construction of the concept guide will terminate; this is implied by the conditions checked prior to performing the operations in the definition and the fact that $clos(C)$ is of finite size.

6.4.1 Example

To demonstrate the construction of a concept guide, let us consider the previous query from section 6.1, $(x) \leftarrow (\exists hasProduct.(\exists causes.Infection))(x)$. Additionally, assume that *hasProduct* is a transitive role; observe that the negation of the query concept is $\forall hasProduct.(\forall causes.\neg Infection)$. Let us consider the concept guide for the negated query concept; when constructing the concept guide, a new node will first be added and labeled with the negated query concept. Following this, the second condition of Definition

15 will be applicable to this node label. Therefore, a new node and edge will be created with labels *hasProduct* and $\forall causes.\neg Infection$ respectively. Further, because the *hasProduct* role is transitive, a self-looping edge labeled with this role will be added to the most recently added node. Lastly, due to the newly added $\forall causes.\neg Infection$ label, a new edge and node will be created and labeled with *causes* and ¬*Infection* respectively. The resulting concept guide is depicted in Figure 6.4.

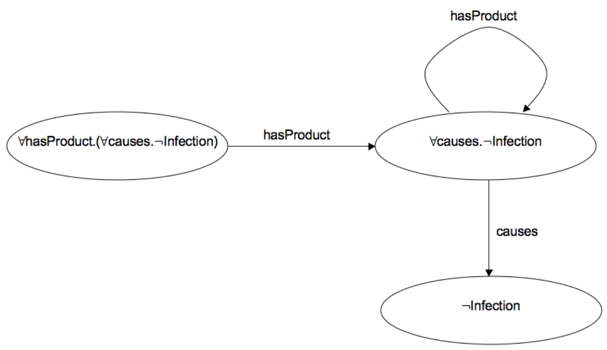


Figure 6.4: Example Concept Guide

6.4.2 Approach

In this section, I discuss how the concept guide is used to determine the propagation of labels due to an update in a completion graph. First, the notion of a *concept guide path* between two nodes in a completion graph in defined.

Definition 17 *(Concept Guide Path): Let* K *be a* $\mathcal{SHI}$ *KB, C some concept,* $\mathcal{G} = guide(C)$, $G \in Comp(\mathsf{K})$, *and* $n, m \in N_{\mathcal{G}}$. *Define there to be an n-m-concept guide path between two nodes* $x, y \in \mathcal{V}$, *denoted* $path(n, m, x, y, \mathcal{G}, G)$, *if there is a sequence of edge traversals,* $\langle x_1, y_1\rangle, ..., \langle x_k, y_k\rangle \in \mathcal{E}$ *and* $(n_1, m_1), ..., (n_k, m_k) \in E_{\mathcal{G}}$, *such that the following holds:*

1. $x_1 = x$ *and* $n_1 = n$ *(i.e., the path starts at* $n \in N_{\mathcal{G}}$ *and* $x \in \mathcal{V}$*)*
2. *for each edge traversal,* $\langle x_i, y_i\rangle \in \mathcal{E}$, *and corresponding edge traversal,* $(n_i, m_i) \in E_{\mathcal{G}}$, *it is the case that for some* $R \in L_{\mathcal{G}}((n_i, m_i))$, x_i *has* R-neighbor y_i

(a) *if some node $z \in \mathcal{V}$ is blocked by $w \in \mathcal{V}$, then an edge $\langle v, w\rangle$ with $\mathcal{L}(\langle v, w\rangle) \leftarrow \mathcal{L}(\langle v, z\rangle)$, where v is the predecessor of z, is also considered for traversal*

3. *$y_k = y$ and $m_k = m$ (i.e., the path ends at $m \in N_{\mathcal{G}}$ and $y \in \mathcal{V}$)*

Condition 2a is necessary due to blocking conditions utilized during the creation of $\mathcal{SHI}$ completion graphs, as cyclic models can occur. Condition 2a overcomes the problematic case where there *would* be a path in the completion graph that satisfies the constraints imposed by the concept guide, yet due to blocking, the path does not explicitly exist.

Assuming the restrictions imposed on the form of C, it can be shown that if adding a concept $\neg C$ to root node x_a in a complete and clash-free completion graph causes a label to be added to a root node x_b, then there is a concept guide path that starts at the concept guide node labeled with $\neg C$ between the two nodes.

Theorem 4 *Let K be a $\mathcal{SHI}$ KB, $\mathsf{G} \in Comp(\mathsf{K})$, C be a $\mathcal{SHI}$ concept that is safe with respect to K, and $\mathcal{G} = guide(\neg C)$. If adding $\neg C(a)$, $a \in \mathbf{I}_{\mathsf{K}}$, to G causes a concept name to be added to $\mathcal{L}(x_b)$, $b \in \mathbf{I}_{\mathsf{K}}$, then there is a concept guide path $path(n, m, x_a, x_b, \mathcal{G}, \mathsf{G})$ for some $n, m \in N_{\mathcal{G}}$ s.t. $\neg C \in \mathcal{L}_{\mathcal{G}}(n)$.*

Proof See Appendix A.2.2 for the proof of this theorem. □

Theorem 4 implies that the concept guide can be used to determine all root nodes that could be reached by label propagations as a result of adding the concept name $\neg C$ to the node corresponding to a named individual in the KB. This in turn means that we can avoid adding $\neg C$ to all individuals in the KB to take into account the first condition of Theorem 3 for additions. This is because, one can maintain the root nodes x with a node label, edge, or edge label change due to β, and then search for all nodes that satisfy the concept guide path relation involving x. This set of nodes therefore constitutes a superset of individuals which, if $\neg C$ were added to their label, would cause a clash that is dependent on $\neg C$ and β.

Similarly, the concept guide can be utilized to take into account the second condition of Theorem 3 for additions. As discussed earlier, one can trivially observe the clashes that are dependent on β when updating each $\mathsf{G} \in \mathit{Comp}(\mathsf{K})$; further, the actual disjunctions that the clashes are dependent on (if any) can also be observed. If the disjunction dependencies of labels are tracked during the tableau algorithm, then one can easily determine the root nodes N which have a label that is dependent on one of the disjunctions. Thus, the concept guide can then be used to determine the root nodes $n \in N$ reachable due to $\neg C$. Given the discussion presented in section 6.2.2, it is clear that deletion updates can be handled in a similar manner.

Next, a variety of notation is introduced, which will be utilized to show the correctness of the approach. Given addition β, $\mathsf{G} \in \mathit{Comp}(\mathsf{K})$ and G' the result of $\mathsf{G} \uplus \beta$ (containing a clash or clash-free), denote by $\mathit{Dep}(\beta, \mathsf{G}, \mathsf{G}')$ the set of named individuals whose corresponding root nodes have a node label, or incoming/outgoing edge or edge label changed when constructing G'. Also, given some node x with $D_1 \sqcup D_2 \in \mathcal{L}(x)$ and completion graph G, denote by $\mathit{Disj}_{\mathsf{G}}((D_1 \sqcup D_2, x))$ the set of named individuals whose corresponding root nodes have a label that is dependent on disjunction $D_1 \sqcup D_2 \in \mathcal{L}(x)$ in G. Further, denote by $\mathit{Disj}_{\mathsf{K}}((D_1 \sqcup D_2, x))$ the set of named individuals whose corresponding root nodes have a label that is dependent on disjunction $D_1 \sqcup D_2 \in \mathcal{L}(x)$ in some completion graph $\mathsf{G} \in \mathit{Comp}(\mathsf{K})$. As in section 6.1, it is important to reiterate that the node x corresponds to the same node in the different completion graphs. Again, the case in which x corresponds to an existential is not problematic to maintain, as nodes are not merged (i.e., number restrictions are not supported in $\mathcal{SHI}$) and all completion graphs are maintained.

Given this, I define the set of *concept candidates*, which intuitively is an overestimate of the individuals which instantiate (or no longer instantiate) a concept after an addition (respectively deletion).

Definition 18 *(Concept Candidates): Let* K *be a* $\mathcal{SHI}$ *KB,* β *an ABox addition (or deletion),* C *be a* $\mathcal{SHI}$ *concept that is safe with respect to* K *and* β*, and* $\mathcal{G} = guide(\neg C)$*. Then define the concept candidates, denoted* $CC(\mathsf{K}, C, \beta)$*, to be the set of all named individuals* $a \in \mathbf{I}_{\mathsf{K}} \cup \mathbf{I}_{\beta}$ *such that adding* β *to some* $\mathsf{G} \in Comp(\mathsf{K})$ *(respectively* $\mathsf{G} \in Comp(\mathsf{K} - \beta)$*) results in* G' *and one of the following conditions is satisfied:*

1. $a \in \mathbf{I}_{\beta}$
2. G' *clash-free,* $b \in Dep(\beta, \mathsf{G}, \mathsf{G}')$ *and there is a concept guide path* $path(n, m, x_a, x_b, \mathcal{G}, \mathsf{G}')$
3. *a clash is observed that is dependent on* $D_1 \sqcup D_2 \in \mathcal{L}(y)$ *and for some* $b \in Disj_{\mathsf{K}}((D_1 \sqcup D_2, y))$ *(respectively* $b \in Disj_{\mathsf{K}-\beta}((D_1 \sqcup D_2, y))$*) or* $b \in Dep(\beta, \mathsf{G}, \mathsf{G}')$ *there is a concept guide path* $path(n, m, x_a, x_b, \mathcal{G}, \mathsf{G}'')$ *in some* $\mathsf{G}'' \in Comp(\mathsf{K}) \setminus \mathsf{G}$

for some $n, m \in N_{\mathcal{G}}$*, where* $\neg C \in \mathcal{L}_{\mathcal{G}}(n)$*.*

Theorem 5 implies the correctness of the approach using the concept guide for determining the candidate new (respectively invalidated) bindings for a retrieval query consisting of a $\mathcal{SHI}$ concept.

Theorem 5 *Let* K *be a* $\mathcal{SHI}$ *KB,* β *an ABox addition (or deletion),* C *be a* $\mathcal{SHI}$ *concept that is safe with respect to* K *and* β*, and* $\mathcal{G} = guide(\neg C)$*. If for some* $a \in \mathbf{I}_{\mathsf{K}} \cup \mathbf{I}_{\beta}$*,* $\mathsf{K} \not\models C(a)$ *and* $\mathsf{K} + \beta \models C(a)$ *(respectively* $\mathsf{K} \models C(a)$ *and* $\mathsf{K} - \beta \not\models C(a)$*), then* $a \in CC(\mathsf{K}, C, \beta)$

Proof See Appendix A.2.3 for the proof of this theorem. □

6.4.3 Discussion

The construction of the concept guide assumes the standard $\mathcal{SHI}$ tableau expansion rules [74]. In many DL reasoning systems, a variety of optimizations are utilized, some of which introduce additional tableau expansion rules (see Chapter 2.4.3 for additional details). The approach can easily be extended to take into account the unfolding and domain/range expansion rules via a simple extension to the definition of the concept guide construction. In order to support the unfolding rule, whenever an atomic concept is encountered during the construction of the concept guide, the concept is unfolded and its label is be added to the current concept guide node; in order ensure that the same concept is not repeatedly unfolded (i.e., termination), nodes with the same concept names can be merged, capturing the cycle. For the domain rule, whenever a label of the form $\exists R.D$ is encountered during the construction of the concept guide, then the domain of the role R is added to the current node. Observe that handling the range-rule is un-necessary, due to the restriction on the query form.

6.5 Summary Completion Graph

Incrementally maintaining all completion graphs for a given KB is not practical; further, in the presence of a reasonable degree of non-determinism in a KB, constructing all completion graphs is a very expensive process. To overcome this issue, an approach is developed in which a completion graph structure is constructed that represents a summary of the structures present in all completion graphs for the KB; this structure is referred to as a *summary completion graph*. The general idea is to maintain the summary completion graph through updates in a similar manner as maintaining a regular completion graph. Importantly, this structure can be utilized to locate the candidate individuals, and therefore all completion graphs for the KB do not have to be maintained. Given this brief overview, a summary completion graph is defined as follows:

Definition 19 *(Initial Summary Completion Graph Construction): Define the summary completion graph for $\mathcal{SHI}$ KB K, denoted S_{G}, to be constructed by applying the $\mathcal{SHI}$ tableau algorithm to K, however with the following modifications:*

1. *the $\sqcup$-rule is replaced as follows:*
   if $C_1 \sqcup C_2 \in \mathcal{L}(x)$, x is not indirectly blocked and $\{C_1, C_2\} \not\subseteq \mathcal{L}(x)$ then $\mathcal{L}(x) \leftarrow \mathcal{L}(x) \cup \{C_1, C_2\}$

2. *if a clash is encountered, it is ignored and the algorithm continues*

The construction of the summary completion graph proceeds in an similar manner as the regular tableau algorithm, however when the $\sqcup$-rule is applied, all concept names of the disjunction are added to the node label (in the same manner as the $\sqcap$-rule). Note that condition 2 of the definition is required, as adding all concept names from a disjunction can introduce clashes which would not occur in the different completion graphs.

Termination of the construction of the summary completion graph follows easily from the fact that the termination for the $\mathcal{SHI}$ tableau algorithm is independent of clash detection; therefore it can be shown in an identical manner as termination for the $\mathcal{SHI}$ tableau algorithm [79, 74].

6.5.1 Summary Completion Graph Properties

In this section, various properties of the summary completion graph for a KB are shown. Specifically, it is shown that if a root node has a label in some completion graph corresponding to a model for the KB, then that concept name will be in the label for that individual in the summary completion graph. Further, properties regarding the tree and graph-like structures rooted at root nodes are shown. These properties will be utilized later when showing the correctness of the approach.

First, the following notation is introduced: given a completion graph G with root node x, denote by $Tree(x)$ the tree rooted at x that is composed of x, all non-root descendants of x, and the labels for nodes and edges for the tree; this tree is referred to as the *root tree* for x in G. Additionally, given root tree T, denote by $Root(T)$, V_T, E_T, and L_T the unique root, set of nodes, edges, and label function for the tree respectively. The notion of sub-graphs of root nodes in a completion graph is also introduced. This structure can be viewed as a generalization of a root tree in which node neighbors, edges from the predecessor of a blocked node to the blocking node, and edges between root nodes are also considered.

Definition 20 *(Root Graph) A root graph G is composed of a set of nodes V_G, edges E_G, and labeling function L_G for the nodes and edges; additionally, there is a uniquely defined root node of the graph, $Root(G)$. Given a completion graph G and root node x, the root graph G for x, denoted $Graph(x)$, is defined as follows:*

1. *$Root(G) = x$, $x \in V_G$, and $L_G(x) = \mathcal{L}(x)$*
2. *if $y \in V_G$ and y has R-neighbor z in G s.t. z not blocked, then $z \in V_G$, $\langle y, z\rangle \in E_G$, $L_G(z) = \mathcal{L}(z)$ and $L_G(\langle y, z\rangle) = L_G(\langle y, z\rangle) \cup \{R\}$*
3. *if $w, y \in V_G$, $z \in \mathcal{V}$ a non-root node, y the predecessor of z, y has R-neighbor z, and w blocks z in G, then $\langle y, w\rangle \in E_G$ and $L_G(\langle y, w\rangle) = L_G(\langle y, w\rangle) \cup \{R\}$*

Next, the notion of *tree containment* in a root graph is introduced; intuitively, this implies that the tree structure of the root node is included in the graph structure of the node.

Definition 21 *(Tree Containment) Let T, G be a root tree and root graph respectively. Inductively define node $x \in V_T$ to be contained in $y \in V_G$, denoted $con(x, y)$, if the following holds:*

1. *$L_T(x) \subseteq L_G(y)$*
2. *for each $\langle x, z\rangle \in E_T$ there exists an edge $\langle y, w\rangle \in E_G$ s.t.*
   (a) *for all $R \in L_T(\langle x, z\rangle)$ there is some $S \in L_G(\langle y, w\rangle)$ s.t. $S \sqsubseteq R$ and*
   (b) *$L_T(z) \subseteq L_G(w)$ and*
   (c) *$con(z, w)$*

Then, T is said to be contained in G, denoted $contain(T, G)$, if $con(Root(T), Root(G))$. Lastly, given a root tree T and root graph G s.t. $contain(T, G)$, denote by $y \rightarrow_{T,G} z$ a mapping of node $y \in V_T$ into $z \in V_G$ s.t. the containment relationship is satisfied.

Importantly, each root tree in a complete and clash-free completion graph for K must be contained in the root graph for the corresponding node in the summary completion graph for K.

Lemma 2 *Let K be a $\mathcal{SHI}$ KB and S_{G} be the summary completion graph for K. Then for all $\mathsf{G} \in Comp(\mathsf{K})$ and each $a \in \mathbf{I}_{\mathsf{K}}$, $x_a \in \mathcal{V}$ and $x'_a \in \mathcal{V}_{S_{\mathsf{G}}}$, it is the case that $contain(Tree(x_a), Graph(x'_a))$.*

Proof See Appendix A.2.4 for the proof of this lemma. □

Because root nodes are never blocked and the tableau expansion rules do not add edges or edge labels between root nodes, it must be the case that all edges and their labels between root nodes that exist in any complete and clash free completion graph for the KB also exist in the summary completion graph. Intuitively, this and Lemma 2 imply that the summary completion subsumes the information in all complete and clash-free completion graphs for K.

6.5.2 Using the Summary Completion Graph

In this section, I show how the summary completion graph can be used to avoid maintaining all completion graphs for the purpose of exploiting Theorem 3. This is accomplished by showing that an overestimate of the concept candidates introduced in Definition 18 can be determined by simply using the summary completion graph. In the following two sections, I address each of the three conditions in Definition 18 separately. Clearly, the first condition of the definition is trivial; therefore, only conditions 2 and 3 are addressed. It is noted that in the following two subsections, I only address addition updates; this is because the approach for deletions follows in a similar manner and will be addressed later.

Condition 2

For the second condition of Definition 18 to be taken into account using the summary completion graph, intuitively it must shown that we can determine the propagation of labels due to the addition in all complete and clash-free completion graphs for a KB by simply using the summary completion graph. The main idea of the approach is that given a set of assertions β, the structures for β can be added to the summary completion graph and the expansion rules can be applied to the added labels in a similar manner as when incrementally updating a complete graph (as in Chapter 5). Then, it can be shown that the propagation of labels to root nodes in the summary completion graph subsumes the propagation in all completion graphs. Further, I show that the concept guide paths must also exist in the summary completion graph; collectively, this implies the completeness of the approach.

Due to the specialized treatment of the $\sqcup$-rule, there may be labels present in the summary completion graph which prohibit the application of an expansion rule. For example, β may include a type assertion of the form $\forall R.C(a)$ and in the summary completion graph all R-neighbors of a already have C in their label; however, in the different complete and clash-free completion graphs for the KB prior to the addition of β, there could exists some R-neighbor that does not contain C in its label (this is due to the non-determinism in the tableau algorithm). In this case, this neighbor would in fact have a label change, causing it to be considered when detecting concept guide paths.

This problem is overcome by a modification when checking if the expansion rules can be applied to a node. Specifically, if when updating the summary completion graph with β, it is the case that a node label exists which prevents the application of an expansion rule, then it is applied anyway. In order to ensure that the algorithm still terminates, rule applications are tracked using a marking function θ when they have been applied to a specific node during the update of the summary completion graph (shown in Table 6.3); therefore, the re-application will only happen once.

$\sqcap$-rule:	if 1) $C_1 \sqcap C_2 \in \mathcal{L}(x)$, x is not indirectly blocked and 2) either a) $\theta((C_1 \sqcap C_2, x)) == false$ or b) $\{C_1, C_2\} \not\subseteq \mathcal{L}(x)$ then set $\theta((C_1 \sqcap C_2, x)) = true$ and $\mathcal{L}(x) = \mathcal{L}(x) \cup \{C_1, C_2\}$
$\sqcup$-rule:	if 1) $C_1 \sqcup C_2 \in \mathcal{L}(x)$, x is not indirectly blocked and 2) either a) $\theta((C_1 \sqcup C_2, x)) == false$ or b) $\{C_1, C_2\} \not\subseteq \mathcal{L}(x)$ then set $\theta((C_1 \sqcup C_2, x)) = true$ and $\mathcal{L}(x) = \mathcal{L}(x) \cup \{C_1, C_2\}$
$\exists$-rule:	if 1) $\exists S.C \in \mathcal{L}(x)$, x is not blocked and 2) either a) $\theta((\exists S.C, x)) == false$ or b) x has no S-neighbor y with $C \in \mathcal{L}(y)$ then set $\theta((\exists S.C, x)) = true$ and create a new node y with $\mathcal{L}(\langle x, y\rangle) = S$ and $\mathcal{L}(y) = C$
$\forall$-rule:	if 1) $\forall S.C \in \mathcal{L}(x)$, x is not indirectly blocked and 2) either a) $\theta((\forall S.C, x, y)) == false$ and there is an S-neighbor y of x with $C \in \mathcal{L}(y)$ or b) there is an S-neighbor y of x with $C \notin \mathcal{L}(y)$ then set $\theta((\forall S.C, x, y)) = true$ and $\mathcal{L}(y) = \mathcal{L}(y) \cup C$
$\forall_+$-rule:	if 1) $\forall S.C \in \mathcal{L}(x)$, x is not indirectly blocked and 2) there is some R with $\mathsf{Trans}(R)$ and $R \sqsubseteq^{*} S$, 3) either a) $\theta((\forall S.C, R, x, y)) == false$ and there is an R-neighbor y of x with $\forall R.C \in \mathcal{L}(y)$ b) there is an R-neighbor y of x with $\forall R.C \notin \mathcal{L}(y)$ then set $\theta((\forall S.C, R, x, y)) = true$ and $\mathcal{L}(y) = \mathcal{L}(y) \cup \{\forall R.C\}$

Table 6.3: Modified Tableau Expansion Rules for the Summary Completion Graph.

Given this, the approach for updating of the summary completion graph is defined as follows.

Definition 22 *(Summary Completion Graph Update): Let* K *be a $\mathcal{SHI}$ KB, S_G be the summary completion graph for* K*, and β a set of ABox assertions. Then S_G is incrementally updated with β using the approach presented in Chapter 5, however:*

- *clashes are ignored*
- *the modified expansion rules defined in Table 6.3 are assumed and are applied to the following nodes:*
  1. *each node x_a corresponding to some individual $a \in \mathbf{I}_\beta$*
  2. *any node subsequently reached by the application of an expansion rule due to condition 1–3*
  3. *any node that was previously blocked, yet the block is invalidated because of the addition of a node label due to condition 1–3*

Denote by $Update(\beta, S_{\mathsf{G}})$ the update of summary completion graph S_{G} with β according to Definition 22. Additionally, denote by $Dep(\beta, S_{\mathsf{G}})$ the set of named individuals whose corresponding root nodes have a node label, or incoming/outgoing edge or edge label that is (re)added during $Update(\beta, S_{\mathsf{G}})$. It can be shown that the approach for updating the summary completion graph terminates and that after the update Lemma 2 still holds.

Lemma 3 *Let* K *be a* $\mathcal{SHI}$ *KB,* $\mathsf{G} \in Comp(\mathsf{K})$*,* S_{G} *the summary completion graph for* K*,* β *an ABox addition,* G' *the result of adding* β *to* G *(either containing a clash, or complete and clash-free), and* $S'_{\mathsf{G}} = Update(\beta, S_{\mathsf{G}})$*. Then* $Update(\beta, S_{\mathsf{G}})$ *terminates and Lemma 2 holds for* S'_{G} *and* G'*.*

Proof See Appendix A.2.5 for the proof of this lemma. □

In order to show completeness of the approach, we must demonstrate that S_{G} can be used to find all $a \in Dep(\beta, \mathsf{G}, \mathsf{G}')$ for some $\mathsf{G} \in Comp(\mathsf{K})$ and G' the result of adding β to G. First, note that if an edge $\langle x, y\rangle \in \mathcal{E}$, where x, y are root nodes, is added to G, it must have been added due to a role assertion in β (the same holds for edge labels for edges between root nodes); this is due to the fact that the tableau expansion rules do not add edges or edge labels between named individuals. Therefore, it suffices to show that if a label is added to some root node x_a during the update of some $\mathsf{G} \in Comp(\mathsf{K})$, then a label will be (re)added to the corresponding root node in the summary completion graph.

Lemma 4 *Let* K *be a* $\mathcal{SHI}$ *KB,* $\mathsf{G} \in Comp(\mathsf{K})$*,* S_{G} *be the summary completion graph for* K*, and* β *a set of ABox assertions. If when adding* β *to* G*, a root node* x *has a concept name added to* $\mathcal{L}(x)$*, then* x *will have a concept name (re)added to* $\mathcal{L}_{S_{\mathsf{G}}}(x)$ *when updating* S_{G} *with* β*.*

Proof See Appendix A.2.6 for the proof of this lemma. □

Given this, the summary completion graph can be used to determine $Dep(\beta, \mathsf{G}, \mathsf{G}')$ by tracking the nodes that are reached during the update of the summary completion graph. However, one must then find the concept guide paths involving these individuals in each G' (i.e., the result of adding β to $\mathsf{G} \in \mathit{Comp}(\mathsf{K})$). Once again, it can be shown that it suffices to simply use the updated summary completion graph, rather than all G'; this is a consequence of Lemma 3 and the following theorem, which intuitively states that if there is a concept guide path in a completion graph, then there will also exist a concept guide path in the summary completion graph.

Lemma 5 *Let* K *be a* $\mathcal{SHI}$ *KB,* $\mathsf{G} \in \mathit{Comp}(\mathsf{K})$*,* S_{G} *be the summary completion graph for* K*,* C *be a* $\mathcal{SHI}$ *concept that is safe with respect to* K*, and* $\mathcal{G} = \mathit{guide}(\neg C)$*. If for some* $a \in \mathbf{I}_{\mathsf{K}}$*,* $n, m \in N_{\mathcal{G}}$ *s.t.* $\neg C \in \mathcal{L}_{\mathcal{G}}(n)$ *there is a concept guide path* $\mathit{path}(n, m, x_a, x_b, \mathcal{G}, \mathsf{G})$*, then there is a concept guide path* $\mathit{path}(n, m, x_a, x_b, \mathcal{G}, S_{\mathsf{G}})$

Proof See Appendix A.2.7 for the proof of this lemma. □

Condition 3

Now, let us consider the third condition of Definition 18 for addition updates. Intuitively, an approach must be developed that uses the summary completion graph to determine the clashes observed when adding β to $\mathsf{G} \in \mathit{Comp}(\mathsf{K})$ that that are dependent on some $D_1 \sqcup D_2 \in \mathcal{L}(x)$; additionally, the root nodes with a label that is also dependent on $D_1 \sqcup D_2 \in \mathcal{L}(x)$ in a completion graph $\mathsf{G}' \in \mathit{Comp}(\mathsf{K}) \setminus \mathsf{G}$ must also be determined. The general idea of the approach, and the focus of the remainder of this section, is that a specialized disjunction dependency function can be introduced for the summary completion graph, such that the dependencies subsume those in all completion graphs. Given this, it is also shown that all clashes observed when updating a completion graph for the KB will also be observed when updating the summary completion graph. This implies that we can use the summary completion graph to detect all clashes dependent on a disjunction and all root nodes with a label dependent on those disjunctions. This in conjunction with Lemma 5 implies that the summary completion graph can be used to take into account the third condition.

First, the topic of maintaining the dependencies on disjunctions in the summary completion graph is addressed; due to the fact that labels from different completion graphs are essentially merged in the summary completion graph, care must be taken when maintaining the dependence of labels on disjunctions. In particular, we must ensure that if a label *could* be added due to a disjunction, then this is reflected in the disjunction dependencies. To account for this, a disjunction dependency function for the summary completion graph is defined in Definition 23.

Definition 23 *(Summary Disjunction Dependency) Given summary completion graph* S_{G} *and node* x *with* $D_1 \sqcup D_2 \in \mathcal{L}(x)$*, inductively define the set* S *of concept/node pairs* (C, x) *that are dependent on* $D_1 \sqcup D_2 \in \mathcal{L}(x)$ *as follows:*

- $(D_1 \sqcup D_2, x) \in S$
- *if* $(C, y) \in S$*, then if* C *of the form:*
  1. $C_1 \sqcap C_2$*, then* $\{(C_1, y), (C_2, y)\} \subseteq S$
  2. $C_1 \sqcup C_2$*, then* $\{(C_1, y), (C_2, y)\} \subseteq S$
  3. $\exists R.D$*, then for each* z *s.t.*
     (a) z *a* R*-neighbor of* y *and* $D \in \mathcal{L}(z)$*, then* $(B, z) \in S$ *for all* $B \in \mathcal{L}(z)$
     (b) z *a* R*-neighbor of* y*,* z *blocked by* m *and* $D \in \mathcal{L}(m)$*, then* $(B, m) \in S$ *for all* $B \in \mathcal{L}(m)$
     (c) z *a* R*-neighbor of* y*,* z *blocked by* m*,* $\forall S.D \in \mathcal{L}(m)$ *s.t.* $\mathsf{Inv}(R) \sqsubseteq^{*} S$*, and* $D \in \mathcal{L}(m)$*, then* $(B, y) \in S$ *for all* $B \in \mathcal{L}(y)$
  4. $\forall R.D$*, then for each* z *s.t.*
     (a) z *a* R*-neighbor of* y *and* $D \in \mathcal{L}(z)$*, then* $(D, z) \in S$
     (b) z *a* R*-neighbor of* y*,* z *blocked by* m *and* $D \in \mathcal{L}(m)$*, then* $(D, m) \in S$
     (c) y *blocks* z*,* m *the predecessor of* z*,* m *a* R*-neighbor of* z*, and* $D \in \mathcal{L}(m)$*, then* $(D, m) \in S$
  5. $\forall R.D$ *and there is some* P *s.t.* $\mathsf{Trans}(P)$ *and* $P \sqsubseteq^{*} R$*, then for each* z *s.t.*
     (a) z *a* P*-neighbor of* y *and* $\forall P.D \in \mathcal{L}(z)$*, then* $(\forall P.D, z) \in S$
     (b) z *a* P*-neighbor of* y*,* z *blocked by* m *and* $\forall P.D \in \mathcal{L}(m)$*, then* $(\forall P.D, m) \in S$
     (c) y *blocks* z*,* m *the predecessor of* z*,* m *a* P*-neighbor of* z*, and* $\forall P.D \in \mathcal{L}(m)$*, then* $(\forall P.D, m) \in S$

In the remainder of this chapter, denote by $Disj_{S_{\mathsf{G}}}((D_1 \sqcup D_2, x))$ the named individuals whose corresponding root nodes in S_{G} have a node label dependent on $D_1 \sqcup D_2 \in \mathcal{L}(x)$.

Note that the neighbor relation is used in condition 3 of Definition 23 because it could be the case that during the construction of S_{G}, the $\exists$-rule is prohibited from firing due to an existing R-neighbor. Additionally, the $(b), (c)$ conditions account for traversal through blocked nodes, which is necessary to show completeness.

The disjunction dependency function can easily be constructed after building or updating a summary completion graph. Further, if a new disjunction is introduced after an update, then its dependencies can be easily identified after the update. Given this, it is simply assumed that the disjunction dependency function is maintained. Importantly, it can be shown that if some root node x in a completion graph for the KB has a node label that is dependent on a disjunction in some node label $\mathcal{L}(y)$, then the dependency for the disjunction (using Definition 23) of the nodes in the summary completion graph corresponding to y (via the containment relationship) will contain x. This intuitively implies that the disjunction dependencies of the completion graphs are subsumed by those in the summary completion graph.

Lemma 6 *Let* K *be a* $\mathcal{SHI}$ *KB,* β *a set of ABox assertions,* $\mathsf{G} \in \mathit{Comp}(\mathsf{K})$*, and* S_{G} *the summary completion graph for* K*. Also, let* y *be some node in* G *s.t.* $D_1 \sqcup D_2 \in \mathcal{L}(y)$*,* x *be the unique root of* y *(possibly* y *itself),* $T = \mathit{Tree}(x)$ *in* G*, and* $G = \mathit{Graph}(x)$ *in* S_{G}*. Then for all* $y \rightarrow_{T,G} z$*,* $\mathit{Disj}_{\mathsf{G}}((D_1 \sqcup D_2, y)) \subseteq \mathit{Disj}_{S_{\mathsf{G}}}((D_1 \sqcup D_2, z))$*.*

Proof See Appendix A.2.8 for the proof of this lemma. □

Observe that Lemma 3 implies that after adding β to some $\mathsf{G} \in \mathit{Comp}(\mathsf{K})$ resulting in G' (possibly containing a clash), the tree containment relationship still holds after updating S_{G} with β. Therefore, it is a direct consequence of Lemmas 3 & 6 that the disjunctions dependencies in the updated summary completion graph subsume those for each completion graph after it is updated.

Corollary 1 *Let* K *be a* $\mathcal{SHI}$ *KB,* β *a set of ABox assertions,* $\mathsf{G} \in \mathit{Comp}(\mathsf{K})$*,* G' *the result of* $\mathsf{G} \uplus \beta$ *(possibly containing a clash),* S_{G} *the summary completion graph for* K*, and*

$S'_{\mathsf{G}} = Update(\beta, S_{\mathsf{G}})$. Also, let y be some node in G' s.t. $D_1 \sqcup D_2 \in \mathcal{L}(y)$, x be the unique root of y (possibly y itself), $T = Tree(x)$ in G', and $G = Graph(x)$ in S'_{G}. Then for all $y \rightarrow_{T,G} z$, $Disj_{\mathsf{G}}((D_1 \sqcup D_2, y)) \subseteq Disj_{S_{\mathsf{G}}}((D_1 \sqcup D_2, z))$.

Lastly, all clashes observed when updating each $\mathsf{G} \in Comp(\mathsf{K})$ will be observed when updating the summary completion graph. While clashes are ignored during the construction and update of the summary completion graph, they still will be present and therefore can be tracked.

Lemma 7 *Let K be a $\mathcal{SHI}$ KB, $\mathsf{G} \in Comp(\mathsf{K})$, S_{G} be the summary completion graph for K, and β a set of ABox assertions. Also assume that there is a clash $c = (y, C, \neg C)$ observed when β is added to G and let x be the unique root of y (possibly y itself). Additionally, let $T = Tree(x)$ in G at the time of the clash and $G = Graph(x)$ in S_{G} after S_{G} is updated with β. Then, for some $y \rightarrow_{T,G} z$, the expansion rules will be applied to z when updating updating S_{G} and $\{\neg C, C\} \subseteq \mathcal{L}_{S_{\mathsf{G}}}(z)$.*

Proof See Appendix A.2.9 for the proof of this lemma. □

Given this, we have effectively shown the summary completion graph can be used to determine all of the clashes that would be observed when updating some $\mathsf{G} \in Comp(\mathsf{K})$ with β, as well as the disjunction dependencies for these clashes. This is because all clashes that would be observed in some $\mathsf{G} \in Comp(\mathsf{K})$ will be observed in S_{G} and the disjunction dependencies of the clashes subsume those that would be found in all completion graphs. Then to take into account the third condition of Definition 18, one must then find concept guide paths involving the individuals in the disjunction dependencies (or in $Dep(\beta, \mathsf{G}, \mathsf{G}')$, which has been addressed in the previous section) in some $\mathsf{G}' \in Comp(\mathsf{K}) \setminus \mathsf{G}$. It is a direct consequence of Lemma 5 that this can be performed simply using the summary completion graph.

ABox Deletions

Until this point, I have only discussed how the summary completion graph can be used to support addition updates. Thus, deletion updates are now addressed; following this, correctness of the approach is shown in Theorem 6.

Given the axiom tracing function presented in Chapter 5, incremental deletions can be supported using the summary completion graph in a similar manner as the approach for addition updates. This is because the completeness of the axiom tracing function when applying it to the summary completion graph directly follows from Theorem 1. Given this, a straightforward approach to support deletion updates can be accomplished by first reverting the change events in the summary completion graph that are dependent on the deletion (as in Chapter 5). Then, the necessary expansion rules can be applied to the summary completion graph; importantly, the modified $\sqcup$-rule must be used (as presented in Definition 19) and clashes must be ignored during the re-application of the expansion rules. Lastly, the disjunction dependencies for the summary completion graph can be updated to reflect the deletion of β. Given this, the summary completion graph for $\mathsf{K} - \beta$ is easily obtained; for ease of presentation denote this process by $Del(\beta, S_{\mathsf{G}})$. Therefore, deletions updates can be supported by then adding β back to the summary completion corresponding to $\mathsf{K} - \beta$ and performing the same approach as in the case for additions. It is noted that after a deletion has been processed using this technique, the summary completion graph must again be updated to reflect the deletion. As in the original retraction of β, the axiom tracing function can be used for this purpose.

Given this, the completeness of the over-estimate for the set of candidate individuals is shown; first, the over-estimate is formally defined.

Definition 24 *(Concept Candidates Overestimate): Let* K *be a* $\mathcal{SHI}$ *KB,* β *an ABox addition (or deletion),* S_{G} *the summary completion graph for* K*,* $S'_{\mathsf{G}} = Del(\beta, S_{\mathsf{G}})$*,* $S''_{\mathsf{G}} = Update(\beta, S_{\mathsf{G}})$ *(respectively* $S''_{\mathsf{G}} = Update(\beta, S'_{\mathsf{G}})$*),* C *a* $\mathcal{SHI}$ *concept that is safe with respect to* K *and* β*, and* $\mathcal{G} = guide(\neg C)$*. Define the overestimate of candidate individuals,* *denoted $CC_{S_{\mathsf{G}}}(\mathsf{K}, C, \beta)$, to be the set of named individuals $a \in \mathbf{I}_{\mathsf{K}} \cup \mathbf{I}_{\beta}$ such that for some $n, m \in N_{\mathcal{G}}$ s.t. $\neg C \in \mathcal{L}_{\mathcal{G}}(n)$, one of the following conditions is satisfied:*

1. *$a \in \mathbf{I}_{\beta}$*
2. *$b \in Dep(\beta, S_{\mathsf{G}})$ (respectively $b \in Dep(\beta, S'_{\mathsf{G}})$) and there is a concept guide path $path(n, m, x_a, x_b, \mathcal{G}, S''_{\mathsf{G}})$*
3. *the expansion rules are applied to a node x during $Update(\beta, S_{\mathsf{G}})$ (respectively $Update(\beta, S'_{\mathsf{G}})$) such that $\{A, \neg A\} \subseteq \mathcal{L}(x)$, $A \in \mathcal{L}(x)$ or $\neg A \in \mathcal{L}(x)$ is dependent on $D_1 \sqcup D_2 \in \mathcal{L}(y)$ (determined using Definition 23), and for some $b \in Disjs_{S''_{\mathsf{G}}}((D_1 \sqcup D_2, y))$ there is a concept guide path $path(n, m, x_a, x_b, \mathcal{G}, S''_{\mathsf{G}})$*

Given this, it is shown that this over-estimate is in fact complete.

Theorem 6 *Given a $\mathcal{SHI}$ KB K, ABox update β, C be a $\mathcal{SHI}$ concept that is safe with respect to K and β, and summary completion graph S_{G} for K, then $CC(\mathsf{K}, C, \beta) \subseteq CC_{S_G}(\mathsf{K}, C, \beta)$.*

Proof See Appendix A.2.10 for the proof of this theorem. □

6.6 Supporting Complex Query Patterns

Thus far, the techniques presented have only addressed queries that are simply composed of a single DL concept. Given this, the approach is now extended to support complex query patterns. As discussed previously, the general approach for supporting complex query patterns is to transform each role atom in the query into a concept atom, which is referred to as *rolling-up* the query; for example, the query $(x) \leftarrow hasProduct(x, y)$ can be transformed into the equivalent concept term $\exists hasProduct.\top$ [81, 140] (see Chapter 2.4.2 for further details).

This rolling-up process if often enabled via the use of nominals, however the DL $\mathcal{SHI}$ does not provide such expressivity. Fortunately, as discussed in Chapter 2.4.2, there is a well known workaround, in which the use of nominals can be simulated. The approach is to substitute each constant in the rolled-up query concept with a new concept name that does not occur in the knowledge base; additionally, an assertion is added to ensure that the individual instantiates its representative concept. The general query answering technique for retrieval queries is then to iteratively introduce a new concept names for the each individual substitution for distinguished variables and further extend the KB with concept assertions for these representative concepts/individual pairs; then, for each of these possible substitutions, a consistency check is performed [81, 140].

It can be shown that the techniques developed in the previous sections can be used to find the candidate bindings for the distinguished variable that the query is rolled-up into if the resulting query concept is safe w.r.t. the KB and updates. Further, it can be shown that the query can simply be rolled-up once using a single set of new concept names and the additional type assertions for the representative concepts can be ignored when determining the candidates. Thus, if the query can be rolled-up into a distinguished variable x, the same approach can be used to find the candidates using the single rolled-up concept.

Given a retrieval query Q with $DVar(Q) = \{x_1, ..., x_n\}$, denote the rolling-up of Q into $x_i \in DVar(Q)$ by $Rollup(i, Q)$, such that it produces a $\mathcal{SHI}$ concept C where each x_j, $i \neq j$, has been replaced by a new atomic concept D_j not appearing in the KB, any update, or the query. It is assumed that the concept obtained by $Rollup(i, Q)$ is safe with respect to K and all updates. Given this, for a new (respectively invalidated) binding $\{a_1, ..., a_n\}$ to occur, it must be the case that the individual bound to the distinguished variable that the query is rolled-up into is in the set of concept candidates for the rolled-up query concept.

Theorem 7 *Let K be a $\mathcal{SHI}$ KB, Q a conjunctive retrieval query that can be rolled-up into a distinguished variable $x_i \in DVar(Q)$, $C = Rollup(i, Q)$, and β an ABox addition (or deletion). If $\mathsf{K} \not\models Q[x_1/a_1, ..., x_n/a_n]$ and $\mathsf{K} + \beta \models Q[x_1/a_1, ..., x_n/a_n]$ (respectively $\mathsf{K} \models Q[x_1/a_1, ..., x_n/a_n]$ and $\mathsf{K} - \beta \not\models Q[x_1/a_1, ..., x_n/a_n]$), then $a_i \in CC(\mathsf{K}, C, \beta)$.*

Proof See Appendix A.2.11 for the proof of this theorem. □

It is a direct consequence of Lemma 5, Theorems 6 & 7 and the fact that each D_j is a new atomic concept that the summary completion graph can be used to determine an overestimate of concept candidates for the rolled-up query concept.

6.6.1 Query Impact

It is easy to show that one cannot simply consider the candidates for the variable that the query is rolled-up into as the only candidates for the other distinguished variables in the query. Consider the query $(x, y) \leftarrow (\exists hasProduct.(\exists causes.Infection))(x) \land hasCEO(x, y)$; if it is assumed that the query is rolled-up into the variable x, then the resulting query concept would be $\exists hasProduct.(\exists causes.Infection)) \sqcap \exists hasCEO.Nom_y$, where Nom_y is the representative concept for distinguished variable y. In this case, simply considering the candidates for x as the candidates for y is insufficient, as the previously described techniques do not allow us to make any statements regarding the candidates for y. Therefore, in this section two techniques are developed to determine the remaining candidates; this is referred to as the *query impact* on the candidates.

It is pointed out that given Theorem 7 and the monotonicity of $\mathcal{SHI}$, in the event of deletions, all that must be considered after the update are the previous answer sets which have some individuals that is in the set of query concept candidates. Therefore, one simply needs to re-check these answer sets to ensure that the entailment still holds. Given this, the remainder of this section only addresses ABox additions.

Basic Approach

The first approach to take into account the query impact exploits properties about $\mathcal{SHI}$ completion graphs and the assumptions made regarding the query. It has previously been shown that if if two elements from the domain of interpretation are in the interpretation of a role, then there must be an edge between the completion graph nodes corresponding to these domain elements [136]. This, in conjunction with the assumption that the query is connected and only contains simple roles implies that the roles between named individuals that satisfy the query role atoms must exist in each complete and clash-free completion graph for the updated KB[9].

Given this, a straightforward approach can be used to find the remaining named individuals in a new binding. In particular, the initial set of query concept candidates can be expanded to also included any individual b such that x_b is reachable by at most n edge traversals (ignoring direction) from some $a \in CC(\mathsf{K}, C, \beta)$ in *some* completion graph $\mathsf{G} \in Comp(\mathsf{K} + \beta)$, where n is the length of the longest path in the query graph between the node that the query was rolled-up into and some other node. Note that the individual must be reachable in all completion graphs, because the entailment must hold in all models; therefore, it suffices to only consider one completion graph for the KB when expanding the candidate set. Denote by $Impact(\mathsf{K}, Q, \beta)$ the extended set of individuals.

Mapping Approach

The previous technique does not take into account the actual role names when determining the additional candidates; given this, a second approach is presented that accomplishes this, which will in turn result in a smaller set of candidates. It has previously been shown that a conjunctive query can be answered be syntactically *mapping* the query into all completion graphs for the KB [114, 92]. More specifically, [114] defines a syntactic mapping from a query Q (restricted to only simple roles) into a completion graph G, denoted $Q \hookrightarrow \mathsf{G}$, using a mapping μ from the variables (both distinguished and non-distinguished) and individuals in Q into the nodes of G such that:

- $\mu(a) = x_a$ for each individual $a \in Q$,
- for each atom $\mathrm{C}(x)$ in Q, $C \in \mathcal{L}(\mu(x))$, and

[9]Note that inverse and transitive roles, as well as role hierarchies have to be accounted for.

- for each atom R(x,y) in Q, $\mu(y)$ is an R-neighbor of $\mu(x)$

If the query can be mapped into all completion graphs, then the KB satisfies the query [114]. In order for such an approach to be complete, a special blocking condition, tree-blocking, must be used during the tableau algorithm, in which blocking is delayed to take into account the longest path in the query (see Chapter 2.4.3 for a discussion regarding blocking in general); this is accomplished by using two isomorphic trees such that the depth of the tree corresponds to the longest path of the query [114, 92], and it ensures that such a mapping will be possible in the presence of blocking. It is important to note that if the query contains only distinguished variables, then tree-blocking is not necessary and simply dynamic blocking can be used. This is because root nodes corresponding to named individuals are never blocked during the tableau algorithm. Additionally, a TBox axiom $\top \sqsubseteq C \sqcup \neg C$ must be added to the KB for each concept atom C in the query; this is necessary as the query concepts are syntactically mapped into the completion graph.

A straightforward application of this technique can be leveraged for our purpose. First, it is noted that extending the KB with $\top \sqsubseteq C \sqcup \neg C$ for each query concept may be impractical in the syndication framework when dealing with a substantial number of registered subscriptions. Therefore, a slight modification of the approach is used, in which the mapping of concept names in the query is ignored. Specifically, given a conjunctive retrieval query Q with $DVar = \{z_1, ..., z_n\}$ that has been rolled-up into a distinguished variable $z_i \in DVar(Q)$ resulting in concept C and the set of query concept candidates $CC(\mathsf{K}, C, \beta)$, the following mapping is checked in some $\mathsf{G} \in Comp(\mathsf{K} + \beta)$ for each x_b s.t. $b \in CC(\mathsf{K}, C, \beta)$:

- $\mu(a) = x_a$ for each individual $a \in Q$,
- $\mu(z_i) = x_b$,
- $\mu(z_j) = x_c$ for $1 \leq j < i$, $i < j \leq n$ and some root node x_c
- for each atom R(x,y) in Q, $\mu(y)$ is an R-neighbor of $\mu(x)$

If the original candidate individual b cannot be mapped into x_b such that there is a valid mapping for the remaining query nodes, then this individual does not need to be considered as a candidate; this follows as a completion graph (i.e., model) has just been found in which the query cannot be mapped [114]. However, if the query can be mapped into the completion graph such that the candidate individual, b, is mapped into x_b, then this individual must be considered as a candidate binding; importantly, any named individual in the completion graph that can be mapped into the remaining distinguished variables of the query graph must also be added to the candidate set. As in the previous query impact approach, the mapping only needs to be performed for one completion graph, as for the entailed to occur, it must be satisfied by all completion graphs. Denote by $Map\_Impact(\mathsf{K}, Q, \beta)$ the set of all named individuals corresponding to the root nodes that are mapped into a distinguished variable in a valid mapping.

To illustrate the approach further, consider the query $(x, y) \leftarrow$ Company(x) $\wedge$ onSellList(x,y) $\wedge$ hasCEO(y,z), whose query graph is presented in Figure 6.5.

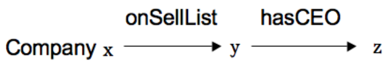


Figure 6.5: Sample Query Graph

Let us assume that the query is rolled up into the variable x; then, when determining the additional candidates, any individual $a \in CC(\mathsf{K}, C, \beta)$ that does not have a *onSellList*-neighbor, which in turn has a *hasCEO* neighbor cannot be a candidate binding for the variable x. Again, this is because a model has just be found in which the entailment does not hold. However, if a has *onSellList*-neighbors b and c (both of which are root-nodes) that also have *hasCEO*-neighbors d, e respectively, then a, b, c are considered in the candidate set. Lastly, it is shown that the query impact approaches are complete.

Theorem 8 *Let* K *be a* $\mathcal{SHI}$ *KB,* Q *a conjunctive retrieval query that can be rolled-up into a distinguished variable* $x_i \in DVar(Q)$*,* $C = Rollup(i, Q)$*, and* β *an ABox addition. If*

$\mathsf{K} \not\models Q[x_1/a_1, ..., x_n/a_n]$ *and* $\mathsf{K}+\beta \models Q[x_1/a_1, ..., x_n/a_n]$, *then* $\{a_i, ..., a_n\} \subseteq$ *Impact*(K, Q, β) *and* $\{a_i, ..., a_n\} \subseteq$ *Map_Impact*(K, Q, β).

Proof See Appendix A.2.12 for the proof of this theorem. □

6.7 Finding Concept Guide Paths

Until now, the task of finding concept guide paths in a completion graph has not been addressed. This is clearly necessary in order to exploit the concept guide to avoid adding the negated query concept to each completion graph (or summary completion graph) for the KB. Therefore, this topic is addressed in this section.

It is a fairly straightforward observation that the task of determining if there is a concept guide path in the summary completion graph can be reduced to evaluating regular path expressions over a labeled directed graph. This follows as clearly the summary completion graph is a labeled directed graph, and it is easily seen that a set of regular path expressions can be constructed from the concept guide; specifically, for each node x in the concept guide, a regular path expression can be created that starts at the node that corresponds to the negated query concept and terminates at x. Then, in the most naïve approach, for each pair of nodes in the completion graph, one needs to test if there is a path in the completion graph that satisfies one of the regular path expressions.

There exist known results regarding the complexity of evaluating regular path expression over graph databases; specifically, it has been shown that deciding if a graph G contains a directed path from nodes x to y satisfying regular expression R can be performed in polynomial time [103]. Given this, assume there are m nodes in the concept guide and s nodes in the summary completion graph. Then, there are $\binom{s}{2}$ pairs of nodes in the summary completion graph, which is $O(n^2)$. Thus, the concept guide paths can be found in $O(m * n^2)$ time.

From a practical point of view, it is important to note that there has been recent work in XML database literature that addresses the evaluation of regular path expressions over graph-based XML data (XML documents with IDREFs) [83, 84]. Therefore, there exist known algorithms which can be utilized to locate concept guide paths.

6.8 Incremental Query Answering Algorithm

Prior to presenting the incremental query answering algorithm, the pseudo code for updating the summary completion graph and determining the set of individuals that must be considered in concept guide paths when determining the concept candidates is provided (shown in Algorithm 3). It is assumed that the summary completion graph S_{G} is created at startup. For ease of exposition, given $S'_{\mathsf{G}} = Update(\beta, S_{\mathsf{G}})$ and a clash c observed during $Update(\beta, S_{\mathsf{G}})$ that is dependent on a set of disjunctions in node labels of S'_{G}, denote by $Disj(c)$ the set of named individuals a s.t. $a \in Disj_{S'_{\mathsf{G}}}((D_1 \sqcup D_2, x))$ for some disjunction $D_1 \sqcup D_2 \in \mathcal{L}_{S'_{\mathsf{G}}}(x)$ that c is dependent on.

Algorithm 3 $Update\_Summary(S_{\mathsf{G}},\beta)$

```
Input:
    $S_{\mathsf{G}}$: Summary completion graph
    $\beta$: Set of ABox assertions
Output:
    $AI$: Set of named individuals
    $S_{\mathsf{G}}$: Updated summary completion graph
 1: if $\beta$ is a deletion then
 2:     $S_{\mathsf{G}} \leftarrow Del(\beta, S_{\mathsf{G}})$
 3: end if
 4: $AI \leftarrow \mathbf{I}_\beta$
 5: $S_{\mathsf{G}} \leftarrow Update(\beta, S_{\mathsf{G}})$
 6: for all root nodes $x_a$ such that the tableau expansion rules are applied to $x_a$ during $Update(\beta, S_{\mathsf{G}})$ do
 7:     $AI \leftarrow AI \cup \{a\}$
 8: end for
 9: for all clashes $c$ observed during $Update(\beta, S_{\mathsf{G}})$ do
10:     $AI \leftarrow AI \cup Disj(c)$
11: end for
12: return $AI, S_{\mathsf{G}}$
```

The general algorithm for the incremental query answering is presented in Algorithm 4. Similar to the discussion above, the algorithm is presented in terms of a single query; additionally, it is assumed the query can be rolled up into the distinguished variable $x_i \in DVar(Q)$. The presentation of the algorithm does not dictate which approach will be used for handling complex query patterns; simply, the function $Query\_Impact(A, Q)$ is used to denote the extension of a set of named individuals A to include all candidates for distinguished variables in the query Q in some complete and clash free completion graph for the updated KB. It is also assumed that the initial set of answers for Q_c is previously determined.

The algorithm first integrates the update into a consistent KB; if the update is an addition, it is also assumed that KB is consistent after the update. Following this, the set of candidate individuals is found using the summary completion graph and concept guide searches (lines 2–3); for simplicity, given a set of individuals AI and concept guide $\mathcal{G}$, the location of concept guide paths is denoted as $Guide\_Search(AI, \mathcal{G})$.

If the update is an addition, the remaining candidates are found by taking into account the query impact; after this, the set of candidate distinguished variable bindings is iterated over and checked for entailment. Standard techniques for query answering are used (see Chapter 2.4.2 for a discussion).

In contrast, if the update is a deletion, each tuple in the previous answer set is iterated over and tuples that do not contain some individual in the set of affected individuals are still entailed, as the conditions for the invalidation of the entailment were not satisfied; otherwise, the tuples are re-checked for entailment.

Theorem 9 *Algorithm 4 is sound, complete, and terminating.*

Proof See Appendix A.2.13 for the proof of this theorem. □

6.9 Empirical Results

A prototype of the algorithm developed in this chapter has been implemented as an extension to the OWL DL reasoner Pellet. The advanced mapping technique has been

Algorithm 4 $Update\_Query\_Results(\mathsf{K}, S_{\mathsf{G}}, Q, \mathcal{G}, R, \beta)$

```
Input:
    $\mathsf{K}$: Consistent $\mathcal{SHI}$ KB
    $S_{\mathsf{G}}$: Summary completion graph for $\mathsf{K}$
    $Q$: Conjunctive query
    $\mathcal{G}$: Concept guide for $Rollup(i, Q)$
    $R$: Set of all current bindings (answer set)
    $\beta$: ABox update
Output:
    $\mathsf{K}$: Consistent $\mathcal{SHI}$ KB
    $S_{\mathsf{G}}$: Updated summary completion graph
    $R$: Updated bindings (answer set)
 1: $\mathsf{K} \leftarrow \mathsf{K} \oplus \beta$
 2: $AI, S_{\mathsf{G}} \leftarrow Update(S_{\mathsf{G}}, \beta)$
 3: $CC \leftarrow Guide\_Search(AI, \mathcal{G})$
 4: if $\beta$ is an addition then
 5:     $QC \leftarrow Query\_Impact(CC, Q)$
 6:     for all $\{a_1, ..., a_n\} \in QC^n$ s.t. $\sharp DVar(Q) = n$ do
 7:         if $\mathsf{K} \models Q_c[x_1/a_1, ..., x_n/a_n]$ then
 8:             $R \leftarrow R \cup \{(a_1, ..., a_n)\}$
 9:         end if
10:     end for
11: else if $\beta$ is an deletion then
12:     for all $(a_1, ..., a_n) \in R$ do
13:         if $CC \cap \{a_1, ..., a_n\} = \emptyset$ then
14:             continue
15:         else if $\mathsf{K} \not\models Q_c[x_1/a_1, ..., x_n/a_n]$ then
16:             $R \leftarrow R \setminus \{(a_1, ..., a_n)\}$
17:         end if
18:     end for
19:     $S_{\mathsf{G}} \leftarrow Del(\beta, S_{\mathsf{G}})$
20: end if
21: return $\mathsf{K}, S_G, R$
```

implemented to take into account query impact, as it is more effective.

In order to evaluate the technique the same ontology test-suite from Chapter 5.5 has been used. For each of the ontologies, a set of queries has been used to simulate subscriptions and to evaluate the effectiveness of the approach. For the LUBM and UOB benchmarks, a set of sample queries accompanies the benchmarks; therefore, a representative subset of these queries are used. In contrast, the VICODI and SEMINTEC ontologies do not have such a query suite. However, there has been recent work in literature [104] in which a set of queries has been obtained from the authors of these ontologies; therefore, these queries have been used, as they will provide insights into response times for expected queries over these datasets. Because the approach is applicable to $\mathcal{SHI}$, functional role assertions have been removed from the SEMINTEC and UOB ontologies.

In the experiments, varying sized ABox additions and deletions were randomly selected from each dataset. Update sizes include 1, 5, 15, 25, and 50 assertions, and as noted in the previous evaluation, these sizes were selected as they align with publication sizes expected in realistic syndication systems. In the experiments, the initial set of answers for the query was first determined, and then the randomly selected assertions were added (or removed) to the KB and the query results were updated. The aim behind the evaluation was to simulate new publications arriving at the syndication broker.

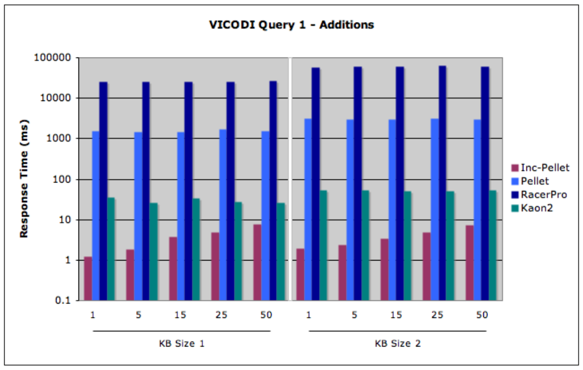
In the experiments, two versions of the DL reasoner Pellet were used; a regular version of the reasoner and a version that has been extended with the algorithm to reduce the candidate individuals. Additionally, RacerPro and KAON2 were used in the evaluation. The experiments were run on a Linux machine with 2Gb of RAM and a 3.06GHz Intel Xeon CPU. Pellet v1.5, RacerPro v1.9.0, and KAON2 release 2007-09-07 were used in the experiments and all results were averaged over 75 iterations. Note that in all of the figures showing the query response time results, the X-axis corresponds to the update size, and the Y-axis is the response time in milliseconds for query answering (the scale is logarithmic). Additionally, the response times shown only include the time to determine the query results. To clarify further, the response times do not include consistency checking times or query preparation times performed by Pellet and RacerPro. The response times for the incremental algorithm developed in this chapter include both the time to find the candidates and execute the query over the candidate set (i.e., Algorithm 4). In the experiments a maximum response time of 100 seconds was imposed in the tests.

As discussed earlier, the summary completion graph must be built so that the technique can be used. The total time to construct the initial summary completion graph is of interest, as well as the potential memory overhead imposed by the structure. Table 6.4 addresses the both of these topics and specifically shows the total time (in seconds) to construct the initial summary completion for each of the different datasets, as well as the memory overhead; note that the results were averaged over 50 iterations.

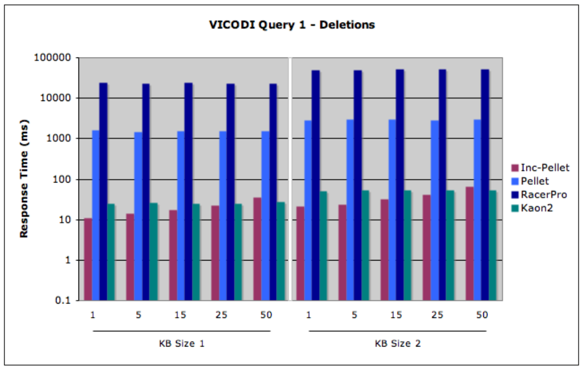
Ontology	**Construction Time (sec)**	**Memory (mb)**
VICODI1	1.7	57
VICODI2	3.7	115
SEMINTEC1	2.4	82
SEMINTEC2	5.3	165
LUBM1	2.3	59
LUBM2	9.2	228
UOB1	15.2	183
UOB2	30.4	331

Table 6.4: Summary Completion Graph Initial Construction Results

As expected, the initial construction of the summary completion graphs introduces overhead. However, the process must only be performed once at startup. It is also clear that there is a memory impact of the summary completion graph. However, in the implementation of the algorithm, the overlap of between the summary completion graph and the cached completion graph corresponding to a model of the KB is not exploited. Today's tableau-based reasoners typically cache the completion graph constructed during the initial consistency check, as it is used in optimizations for other reasoning tasks (e.g., classification). A substantial portion of the structure in the cached completion graph will overlap with the summary completion graph. Therefore, with further engineering, this



(a)



(b)

Figure 6.6: (a) VICODI query 1 - additions (b) VICODI query 1 - deletions

memory overhead can be dramatically reduced.

For the VOCODI ontology, the following queries have been used:

1. $(x) \leftarrow Individual(x)$
2. $(x, y, z) \leftarrow Military - Person(x) \land hasRole(y, x) \land related(x, z)$
3. $(x, y) \leftarrow Military - Person(x) \land hasRole(y, x)$

Queries 1 and 2 have been used in previous literature [104] to evaluate DL query answering and were suggested by the ontology authors. Query 3 has been included as it provides additional insights into the approach (discussed later).

Figure 6.6 presents the response times for query 1 for addition and deletion updates (note that *Inc-Pellet* corresponds to Algorithm 4). The query answering times for the regular version of Pellet is comparable through update sizes and is between 1.4 to 3.08 seconds, depending on the dataset size. The response time remains comparable through update sizes, as the query is re-performed from scratch and the update sizes are small relative to the overall KB size. Response time for RacerPro exhibits similar properties as Pellet.

The incremental query answering approach demonstrates substantial performance improvements over both the regular version of Pellet and RacerPro. For both datasets, approximately 1.5 to 3 orders of magnitude performance improvements over Pellet are exhibited, and the response time is in the 10s of milliseconds (in many cases less than 10ms). This is clearly due to the reduction in the portion of the KB that must be considered for the query after the update. Table 6.5 presents the actual number of candidates for addition updates using the approach for the different update sizes and queries (the results for deletions are very similar and are therefore omitted). This table also shows the percentage of the KB that this candidate set represents. For the first query, the technique provides a dramatic reduction in the portion of the KB that must be considered (always below 1% of the original KB).

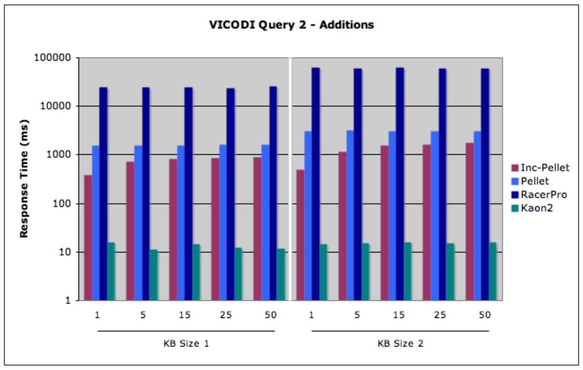
KB / Query	**1 (%)**	**5 (%)**	**15 (%)**	**25 (%)**	**50 (%)**
VIC-1 / 1	1.6 (.009)	7.02 (.04)	20.7 (.12)	34.4 (.2)	65.6 (.38)
VIC-2 / 1	1.7 (.01)	7.4 (.02)	21 (.06)	35.4 (.1)	67.6 (.19)
VIC-1 / 2	7,128 (42)	11,440 (67)	11,553 (68)	11,560 (68)	11,536 (68)
VIC-2 / 2	8,375 (24)	19,358 (57)	22,958 (67)	23,062 (68)	23,126 (68)
VIC-1 / 3	1.9 (.01)	7.16 (.04)	21.6 (.12)	34.9 (.2)	72.8 (.43)
VIC-2 / 3	1.9 (.005)	7.2 (.02)	21.5 (.06)	35.3 (.1)	71.6 (.2)

Table 6.5: VICODI Candidate Sizes for Different Queries and Update Sizes

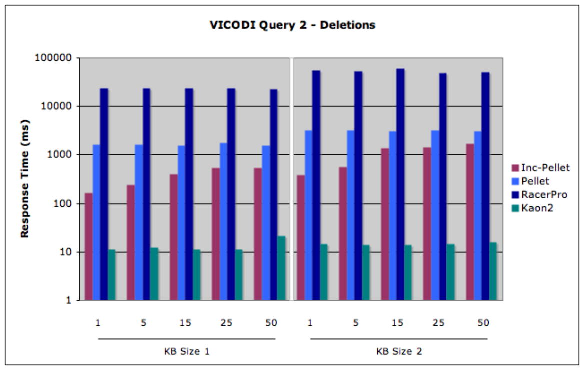
It can also be seen in Figure 6.6 that as the update size is increased, the performance of the approach scales well. Deletions take slightly longer than additions because the deleted assertions must be retracted from the summary completion graph, whereas this process is not necessary for additions.

KAON2 outperforms the regular tableau-based reasoners (i.e., Pellet and RacerPro) and exhibits comparable results through the update sizes. However, the incremental version of Pellet performs better than KAON2 in this experiment.

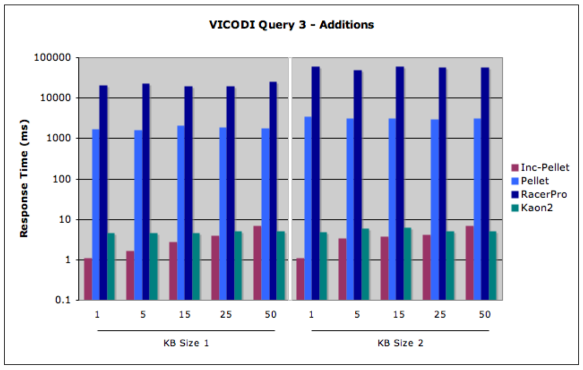
Figure 6.7 presents the results for queries 2 and 3. In the second query, the query answering times for the regular version of Pellet and RacerPro exhibit similar characteristic as for the first query. Further, KAON2 again exhibits comparable response times through updates sizes and in general performs better than in query 1. However, the incremental approach does not provide as dramatic performance improvements as it did in the first query. This can be explained by inspecting Table 6.5; in particular, it can be seen that



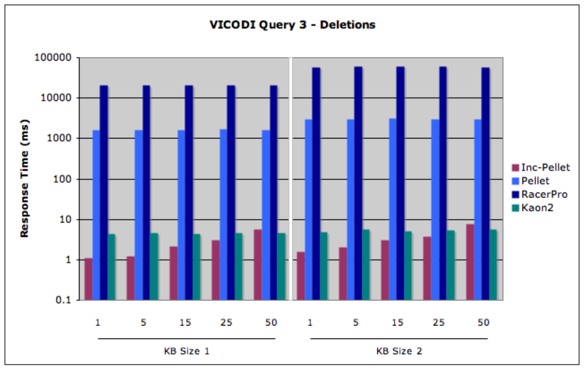
(a)



(b)



(c)



(d)

Figure 6.7: (a) VICODI query 2 - additions (b) VICODI query 2 - deletions (c) VICODI query 3 - additions (d) VICODI query 3 - deletions

a large portion of the knowledge base is considered after each update, and therefore, it is like re-running the query from scratch.

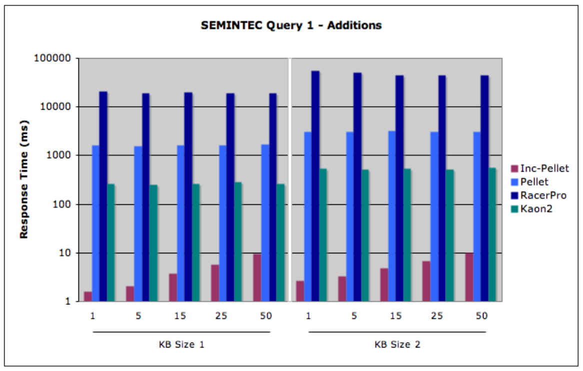
If we inspect the results of query 3, additional insights into the results for the second query are provided. Query 3 is actually a subset of the second query, in which the last role atom from query 2 is excluded. Once again, the incremental algorithm exhibits dramatic performance improvements, as the candidates considered are a small subset of the original KB. This sheds light on the previous query, as there are individuals in the KB that are related to an extremely large portion of the KB by the *related* role. Therefore, when taking into account the query impact, almost all of the knowledge base is included as a candidate. As we will see in the remainder of the evaluation, this was the only query for any of the ontologies that demonstrated this behavior, indicating the approach should be effective in general.

Table 6.6 presents a summary of the distribution of response times for publications of size 50 observed in the experiments using the VICODI ontology. In the table, the notation "VIC-1" and "VIC-2" are used to denote the VICODI KB of size 1 and 2 respectively; additionally, the table presents the minimum, median, maximum, and average response times observed, as well as the standard deviation. In general, it can be observed that the response times are very close to the average.

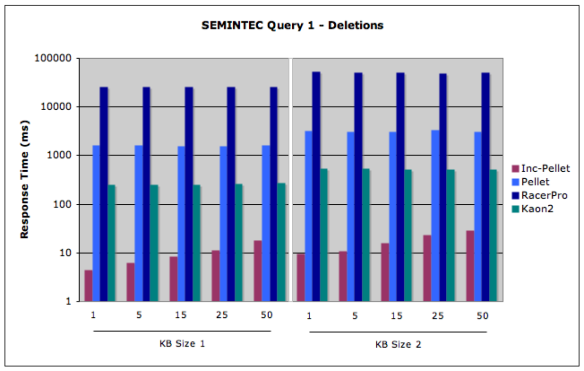
	Additions					**Deletions**				
KB / Query	**Min**	**Med**	**Max**	**Avg**	**Stdv**	**Min**	**Med**	**Max**	**Avg**	**Stdv**
VIC-1 / 1	5	7	52	7	5.2	31	36	70	36	5
VIC-2 / 1	6	7	10	7.4	.7	57	65	76	65	3
VIC-1 / 2	795	854	1236	881	83	303	480	916	534	132
VIC-2 / 2	1541	1736	2461	1800	180	911	1629	2556	1672	246
VIC-1 / 3	5	7	10	6.8	1	3	5	9	5.5	1.1
VIC-2 / 3	6	7	33	7	3	4	6	97	7.7	10

Table 6.6: Distribution of VICODI Response Times for Updates of Size 50 (time in milliseconds)

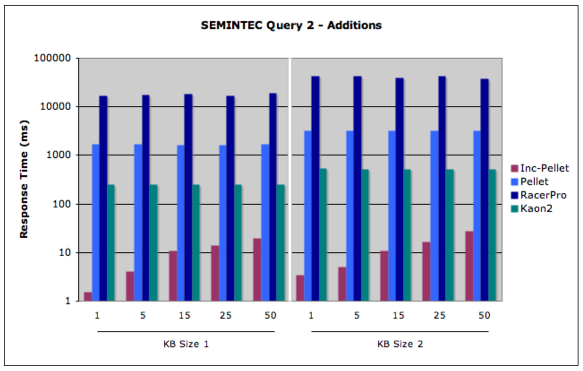
For the SEMINTEC datasets, the following queries have been used:



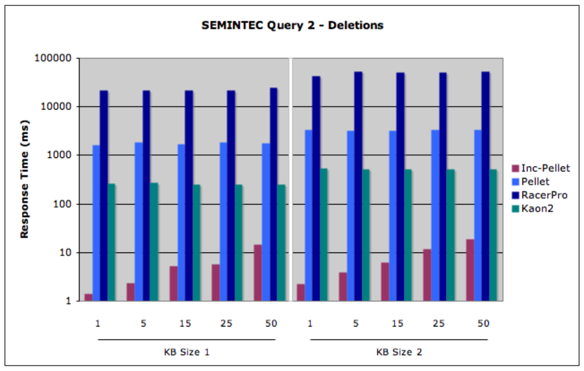
(a)



(b)



(c)



(d)

Figure 6.8: (a) SEMINTEC query 2 - additions (b) SEMINTEC query 2 - deletions (c) SEMINTEC query 3 - additions (d) SEMINTEC query 3 - deletions

1. $(x) \leftarrow Man(x)$
2. $(x, y, z) \leftarrow Man(x) \wedge Gold(y, x) \wedge Region(z) \wedge isCreditCardOf(y, x) \wedge livesIn(x, z)$

Similar to the VICODI queries, the SEMINTEC queries have been used previously in literature to evaluate DL query answering and were suggested by the ontology authors.

Figure 6.8 presents the response times for both queries. Pellet, RacerPro, and KAON2 exhibit similar characteristics as for the VICODI ontologies. Further, the algorithm developed in this chapter again demonstrates dramatic performance improvements for all update sizes. In many cases, the results are below 10ms. Table 6.7 presents the actual number of candidates considered on average for addition updates; again, there is a dramatic reduction in the portion of the KB that must be considered for the query after the updates. Table 6.8 presents a summary of the distribution of response times for publications of size 50 observed in the experiments using the SEMINTEC ontology. As in the case for VICODI, it can be observed that the response times are typically very close to the average.

KB / Query	**1 (%)**	**5 (%)**	**15 (%)**	**25 (%)**	**50 (%)**
SEM-1 / 1	1.5 (.008)	7.2 (.04)	21.8 (.12)	36.8 (.2)	71.8 (.4)
SEM-2 / 1	1.55 (.004)	7.5 (.02)	22.1 (.06)	37 (.1)	73.4 (.2)
SEM-1 / 2	6.2 (.03)	27.2 (.15)	95.4 (.53)	147.6 (.82)	206.6 (1.1)
SEM-2 / 2	22 (.06)	27.8 (.07)	86.6 (.24)	157 (.43)	243 (.67)

Table 6.7: SEMINTEC Candidate Sizes for Different Queries and Update Sizes

	Additions					**Deletions**				
KB / Query	**Min**	**Med**	**Max**	**Avg**	**Stdv**	**Min**	**Med**	**Max**	**Avg**	**Stdv**
SEM-1 / 1	7	9	40	9.5	4.5	15	17	37	17.7	2.9
SEM-2 / 1	8	10	12	10.1	.9	26	29	45	29.5	2.8
SEM-1 / 2	3	20	56	19.5	16.3	6	11	89	14.5	14.6
SEM-2 / 2	4	23	93	27.7	23.3	6	15	87	18.5	12.4

Table 6.8: Distribution of SEMINTEC Response Times for Updates of Size 50 (time in milliseconds)

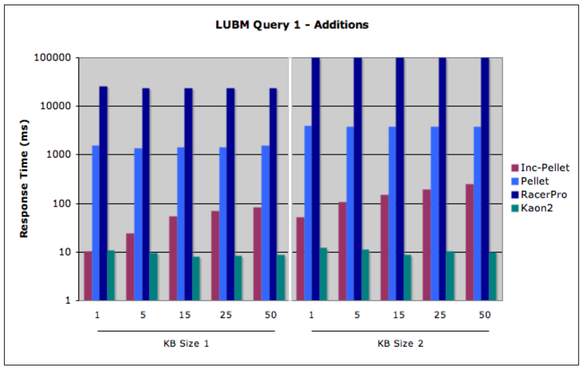
As discussed, the LUBM benchmark includes a set of 14 queries for performance analysis of DL systems. From this set of queries, the results of the following 6 queries[10] are presented.

1. $(x, y, z) \leftarrow \mathit{GraduateStudent}(x) \wedge \mathit{University}(y) \wedge \mathit{Department}(z) \wedge \mathit{memberOf}(x, z) \wedge \mathit{subOrganization}(z, y) \wedge \mathit{undergraduateDegreeFrom}(x, y)$
2. $(x) \leftarrow \mathit{Student}(x)$
3. $(x, y, z) \leftarrow \mathit{Student}(x) \wedge \mathit{Faculty}(y) \wedge \mathit{Course}(z) \wedge \mathit{advisor}(x, z) \wedge \mathit{takesCourse}(x, z) \wedge \mathit{teacherOf}(y, z)$
4. $(x) \leftarrow \mathit{Student}(x) \wedge \mathit{takesCourse}(x, \mathit{GraduateCourse0})$
5. $(x) \leftarrow \mathit{ResearchGroup}(x) \wedge \mathit{subOrganizationOf}(x, \mathit{University0})$
6. $(x) \leftarrow \mathit{UnderGraduateStudent}(x)$

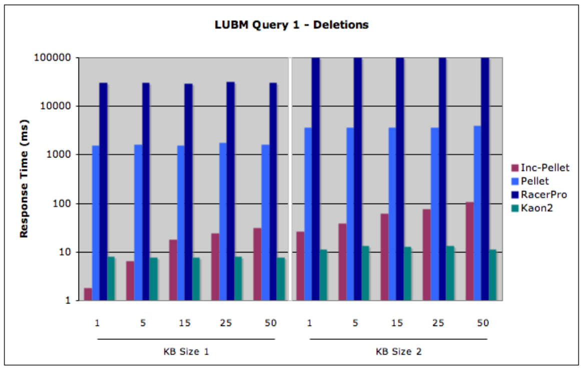
[10]Note that these correspond to LUBM queries 2, 6, 9, 10, 11, & 14 respectively

This subset was selected because the results for the remaining queries are similar. Figure 6.9 presents the results for queries 1–3. The incremental algorithm again demonstrates dramatic performance improvements, as the portion of the KB that must be considered after the update is very small (shown in Table 6.9). In the queries, the approach typically exhibits response times in the 10s of milliseconds and in all cases shows orders of magnitude improvements over the regular version of Pellet and RacerPro. Further, the approach outperforms KAON2 in many cases. An interesting observation can be made from the number of candidates under additions presented in Table 6.9; in particular, for query 1 for the smaller of the LUBM datasets, there are 0 candidates. In this case, there are no answers for the query in the entire KB, and when performing the syntactic mapping to take into account the query impact, the initial candidates can never be mapped into the cached completion graph. In the case of the larger dataset, there are only a few mappings in the entire KB, which are located during the approach. Another interesting observation can be made regarding query 3; specifically, the number of candidates is actually smaller on average in the experiments involving the larger of the two datasets. This can be explained by the fact that in the larger dataset there is a larger number of individuals which do not participate in a concept guide path (for the negated, rolled-up query concept) with some other individual, and there is a larger number of individuals that are not mappable into the query (under query impact). Thus, when the random updates are selected, there is a greater chance that these individuals will be selected.

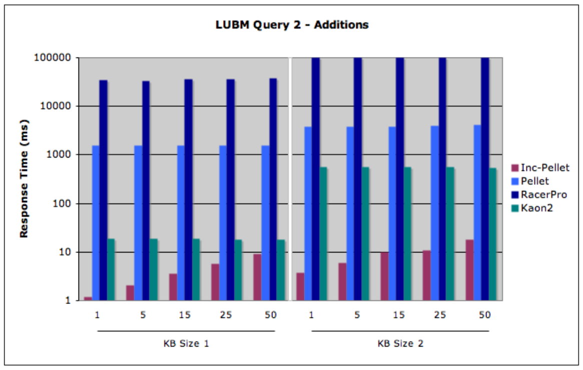
Figure 6.10 presents the results for queries 4–6. Queries 5 & 6 exhibit similar properties as the previous three queries. Interestingly, in query 4, the regular version of Pellet outperforms the version which exploits the optimizations presented in this chapter, even though there is over a 98% reduction in the portion of the KB that must be considered for the query (shown in Table 6.9). This is because query answering for this query in Pellet is extremely fast; therefore, locating the candidates introduces this extra overhead. However, even in this case, the overall response time of the incremental algorithm is still



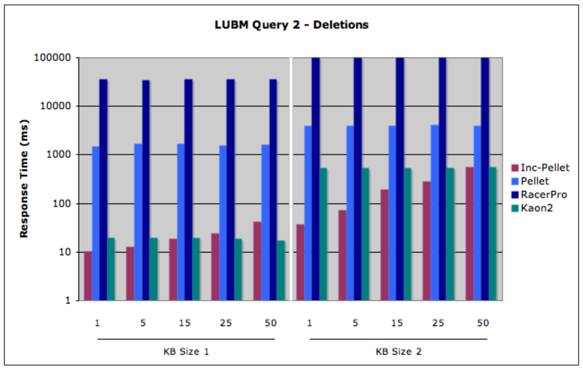
(a)



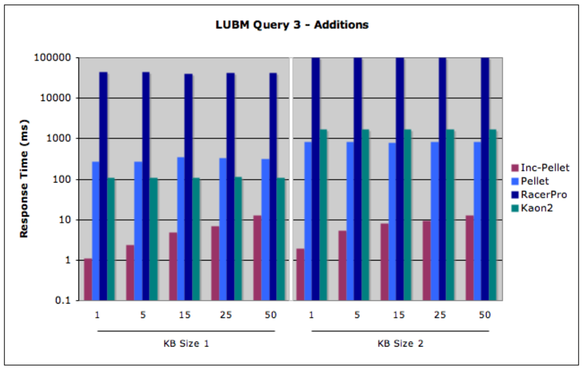
(b)



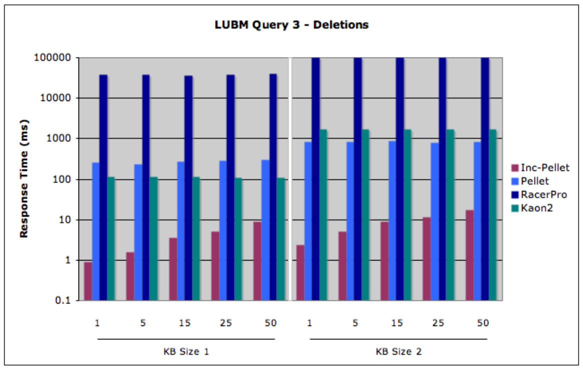
(c)



(d)



(e)



(f)

Figure 6.9: (a) LUBM query 1 - additions (b) LUBM query 1 - deletions (c) LUBM query 2 - additions (d) LUBM query 2 - deletions (e) LUBM query 3 - additions (f) LUBM query 3 - deletions

generally under 10ms and is fast enough for matching in syndication systems.

KB / Query	1 (%)	5 (%)	15 (%)	25 (%)	50 (%)
LUBM-1 / 1	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
LUBM-2 / 1	9.97 (.02)	17 (.04)	17 (.04)	17 (.04)	17 (.04)
LUBM-1 / 2	1.4 (.008)	7.3 (.04)	22.3 (.13)	36.7 (.2)	74.1 (.45)
LUBM-2 / 2	2.8 (.007)	28.5 (.07)	80 (.2)	133.8 (.3)	263.2 (.7)
LUBM-1 / 3	14.6 (.09)	50.2 (.3)	138.8 (.8)	216 (1.3)	404 (2.4)
LUBM-2 / 3	2 (.005)	7.2 (.01)	19.5 (.05)	32.7 (.08)	67.6 (.18)
LUBM-1 / 4	1.6 (.009)	25.3 (.15)	72.3 (.4)	120.8 (.7)	245.5 (1.5)
LUBM-2 / 4	12.9 (.03)	27.6 (.07)	88.2 (.2)	131.8 (.35)	271.1 (.72)
LUBM-1 / 5	3.2 (.01)	14.3 (.08)	32 (.19)	61. (.3)	118.2 (.7)
LUBM-2 / 5	3.5 (.009)	10.9 (.02)	36.6 (.09)	62.4 (.16)	129.3 (.3)
LUBM-1 / 6	1.5 (.009)	8 (.04)	22.7 (.1)	37.1 (.2)	74.3 (.4)
LUBM-2 / 6	1.4 (.003)	7.1 (.01)	22.8 (.06)	37.4 (.1)	75.1 (.2)

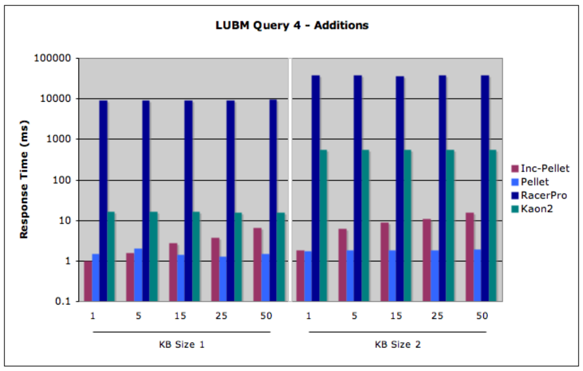
Table 6.9: LUBM Candidate Sizes for Different Queries and Update Sizes

Next, Table 6.10 presents a summary of the distribution of response times for publications of size 50 observed in the experiments for LUBM. As in the previous cases, it can be observed that the response times are close to the average.

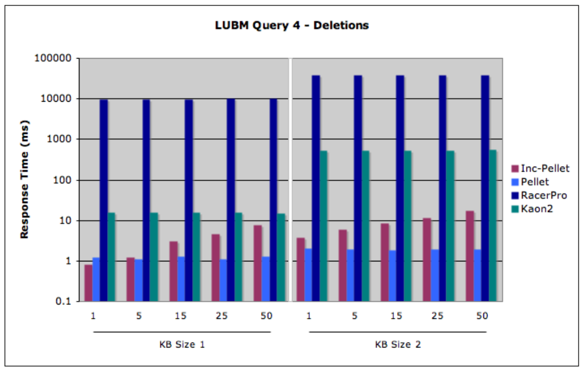
	Additions					Deletions				
KB / Query	Min	Med	Max	Avg	Stdv	Min	Med	Max	Avg	Stdv
LUBM-1 / 1	69	83	119	83.6	9.7	26	30	64	31.1	4.7
LUBM-2 / 1	180	239	346	246.1	33.6	89	105	257	108.1	22.2
LUBM-1 / 2	6	8	38	9	4.7	36	42	52	41.9	2.5
LUBM-2 / 2	13	15	89	18.9	10.3	346	552	720	552.2	75.5
LUBM-1 / 3	7	13	30	12.6	2.8	6	8	16	8.8	1.5
LUBM-2 / 3	9	12	25	13.6	3.4	13	17	31	17.7	3.8
LUBM-1 / 4	4	6	9	6.4	1.02	5	7	28	7.5	2.7
LUBM-2 / 4	12	14	26	15.5	3.1	11	17	26	17.3	3.07
LUBM-1 / 5	2	4	10	4.5	1.04	6	18	60	19.6	8.4
LUBM-2 / 5	10	14	58	15.8	6.3	13	22	45	22.9	5.9
LUBM-1 / 6	7	8	38	8.8	3.4	33	38	70	38.2	4.2
LUBM-2 / 6	13	16	23	16.4	2.2	85	94	167	96	10.1

Table 6.10: Distribution of LUBM Response Times for Updates of Size 50 (time in milliseconds)

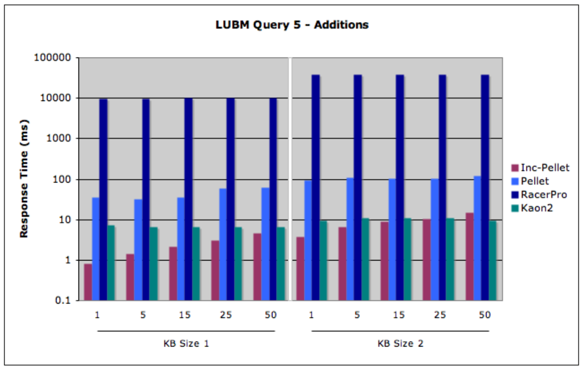
As with the case for LUBM, the UOB benchmark provides a suite of queries, and



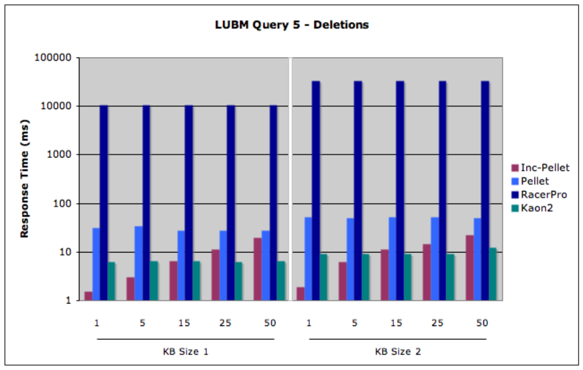
(a)



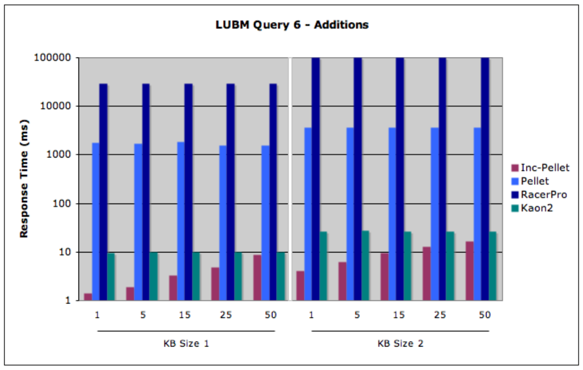
(b)



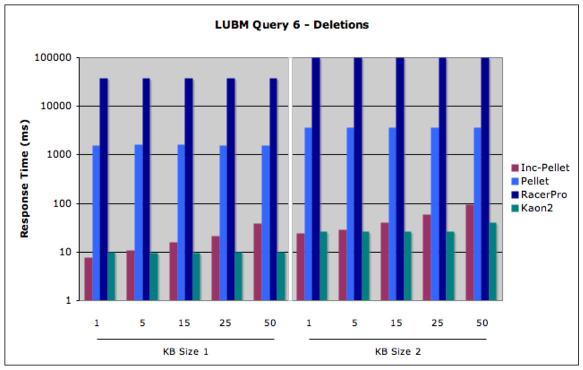
(c)



(d)



(e)



(f)

Figure 6.10: (a) LUBM query 4 - additions (b) LUBM query 4 - deletions (c) LUBM query 5 - additions (d) LUBM query 5 - deletions (e) LUBM query 6 - additions (f) LUBM query 6 - deletions

the following queries[11] have been used:

1. $(x) \leftarrow \mathit{UndergraduateStudent}(x) \wedge \mathit{takesCourse}(x, \mathit{Course0})$
2. $(x) \leftarrow \mathit{Employee}(x)$
3. $(x, y) \leftarrow \mathit{Publication}(x) \wedge \mathit{Faculty}(y) \wedge \mathit{isMemberOf}(y, \mathit{University0}) \wedge$ $\mathit{publicationAuthor}(x, y)$
4. $(x) \leftarrow \mathit{ResearchGroup}(x) \wedge \mathit{subOrganizationOf}(x, \mathit{University0})$
5. $(x, y, z) \leftarrow \mathit{GraduateCourse}(x) \wedge \mathit{isTaughtBy}(x, y) \wedge \mathit{isMemberOf}(y, z) \wedge$ $\mathit{subOrganizationOf}(z, \mathit{University0})$
6. $(x) \leftarrow \mathit{PeopleWithHobby}(x) \wedge \mathit{isMemberOf}(x, \mathit{University0})$

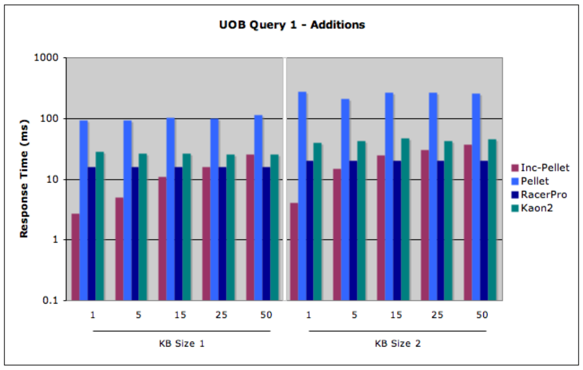
These queries were selected as the results for the other queries are similar.

Figure 6.11 presents the results for queries 1–3. The response times of all reasoners were comparable in query 1. The technique presented in this chapter resulted in a dramatic reduction in the candidates considered (shown for addition updates in Table 6.11), and the incremental version of Pellet outperformed re-running the query from scratch. In queries 2 & 3, the developed approach substantially outperforms the other reasoners, demonstrating orders of magnitude performance improvement and generally response times in the 10s of milliseconds. Interestingly, KAON2 was able to process all of the queries, even though consistency checking performance times in Chapter 5.5 indicated significant overhead. This is potentially explained due to the removal of functional roles in the TBox.

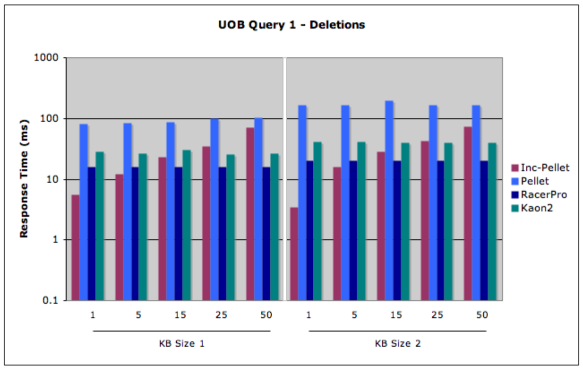
Figure 6.12 presents the results for queries 4–6. Again, there incremental version of Pellet demonstrated substantial performance improvements in many of the queries and generally exhibited response times in the 10s of milliseconds. It is also noted that similar observations as those found in the case for LUBM can be made regarding the number of candidates under additions in the UOB datasets (shown in Table 6.11).

Lastly, Table 6.12 presents a summary of the distribution of response times for publications of size 50 observed in the experiments for UOB. Again, the response times are close to the average.

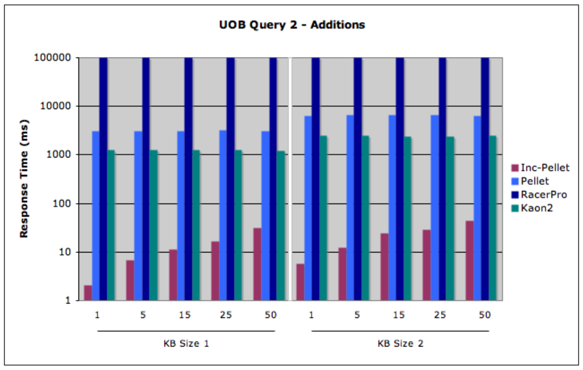
[11]Note that these correspond to UOB queries 1, 2, 4, 5, 9, & 13 respectively



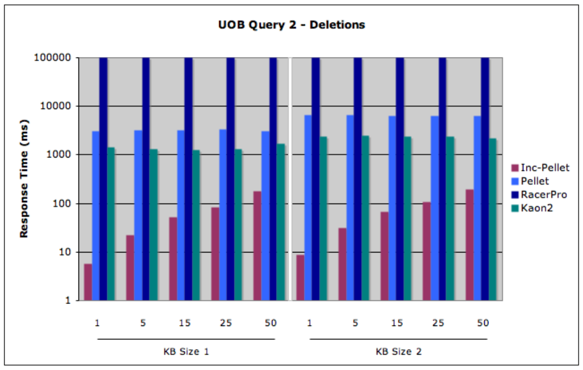
(a)



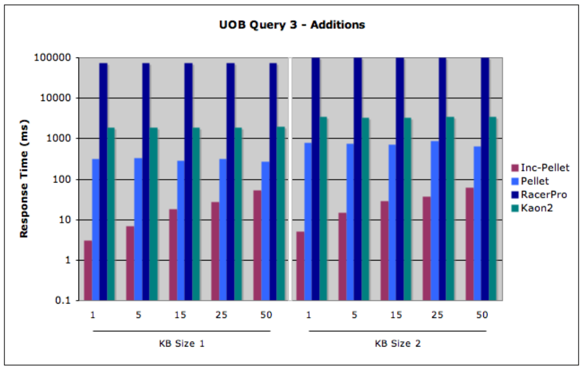
(b)



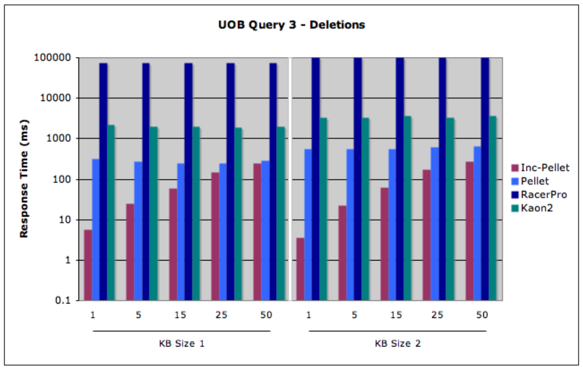
(c)



(d)

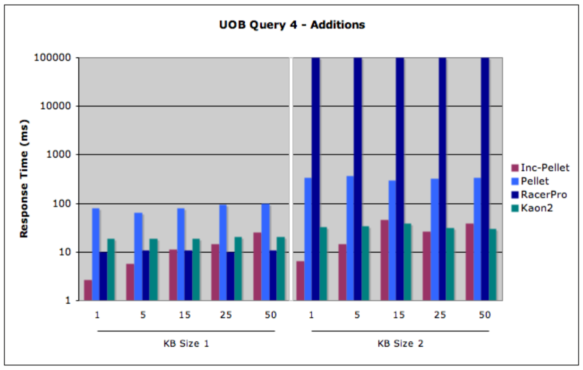


(e)

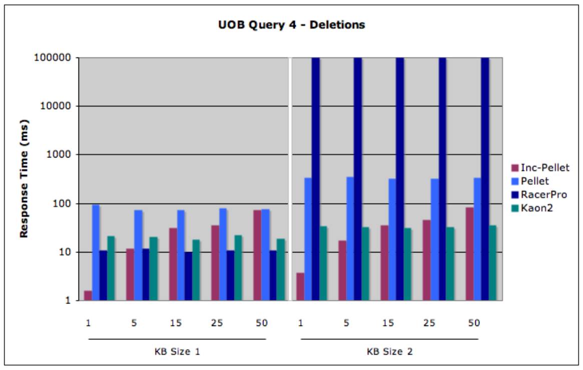


(f)

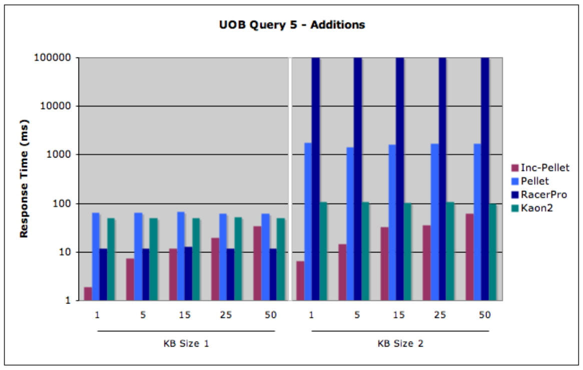
Figure 6.11: (a) UOB query 1 - additions (b) UOB query 1 - deletions (c) UOB query 2 - additions (d) UOB query 2 - deletions (e) UOB query 3 - additions (f) UOB query 3 - deletions



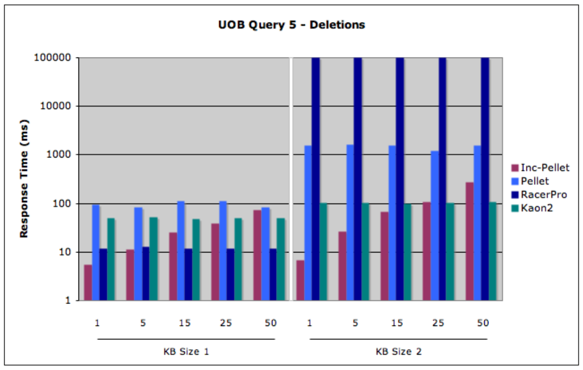
(a)



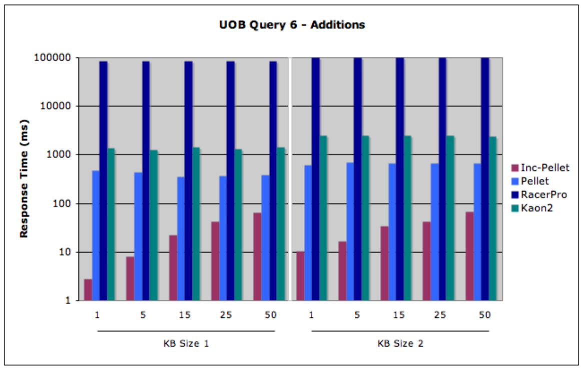
(b)



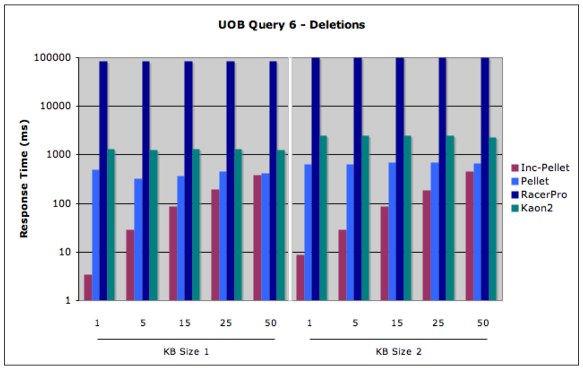
(c)



(d)



(e)



(f)

Figure 6.12: (a) UOB query 4 - additions (b) UOB query 4 - deletions (c) UOB query 5 - additions (d) UOB query 5 - deletions (e) LUBM UOB 6 - additions (f) UOB query 6 - deletions

KB / Query	1 (%)	5 (%)	15 (%)	25 (%)	50 (%)
UOB-1 / 1	1.9 (.007)	29 (.11)	66.8 (.26)	111.4 (.4)	238.3 (.9)
UOB-2 / 1	1.5 (.002)	21.7 (.04)	58.8 (.1)	98.3 (.1)	201.6 (.3)
UOB-1 / 2	3.2 (.01)	17.6 (.06)	52.3 (.2)	85.7 (.33)	174.3 (.6)
UOB-2 / 2	3.2 (.006)	12.3 (.02)	55.1 (.1)	87.1 (.16)	157.1 (.3)
UOB-1 / 3	.9 (.003)	1.1 (.004)	35.1 (.13)	25 (.09)	80 (.3)
UOB-2 / 3	.04 (.0005)	.3 (.0006)	10.5 (.02)	20.8 (.04)	46.4 (.08)
UOB-1 / 4	2 (.007)	10.8 (.04)	29.8 (.1)	49.9 (.2)	96.8 (.38)
UOB-2 / 4	2.1 (.004)	9.3 (.01)	31.7 (.06)	47.5 (.09)	106.9 (.2)
UOB-1 / 5	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
UOB-2 / 5	6.1 (.001)	14.02 (.02)	69.5 (.13)	69.9 (.13)	187.8 (.36)
UOB-1 / 6	10.9 (.04)	162.2 (.6)	528.1 (2.1)	1,052 (4.1)	2,080 (8.2)
UOB-2 / 6	118.5 (.2)	164.4 (.3)	586.8 (1.1)	936.8 (1.8)	1,776 (3.4)

Table 6.11: UOB Candidate Sizes for Different Queries and Update Sizes

6.10 Discussion

It is clear that the empirical results demonstrate that the approach developed is very effective for optimizing continuous query answering. One issue regarding the approach is related to the syntactic restriction imposed on the query; specifically, it is reasonable to question what the real-world impact of this restriction will be. In order to investigate this, recent results of a survey of the publically available OWL ontologies available on the Web were utilized [146]. In particular, a subset of the OWL ontologies gathered during this survey was used to gauge the actual impact of the restriction; note that only a subset was used as a number of the ontologies were no longer available on the Web. However, this subset still totaled 460 ontologies, which provides reasonable insight into the expected impact of the restriction.

In the experiment, each concept in the ontology was selected and tested to see if it violated the restrictions imposed by the approach. Given that today's state of the art DL reasoners typically use highly optimized tableau algorithms, involving the unfolding and domain/range expansion rules, it was assumed that the unfolding and domain/range rules were utilized; therefore, the syntactic restriction was extended to cover the concept encountered when unfolding the query concept. The statistics were gathered after ab-

	Additions					**Deletions**				
KB / Query	**Min**	**Med**	**Max**	**Avg**	**Stdv**	**Min**	**Med**	**Max**	**Avg**	**Stdv**
UOB-1 / 1	24	25	30	25.9	.8	27	70	131	69.7	17
UOB-2 / 1	29	35	60	37.3	7.3	32	69	196	72.1	24.2
UOB-1 / 2	20	30	61	31.3	8.1	72	170	392	181.6	66.3
UOB-2 / 2	31	39	145	43.2	18	77	188	383	196.1	55.1
UOB-1 / 3	17	49	121	54.09	22.8	86	235	504	244	92.5
UOB-2 / 3	32	56	144	62.3	24.3	27	256	616	276.3	117.3
UOB-1 / 4	15	24	48	25.3	7.1	30	67	161	73.5	23.4
UOB-2 / 4	27	37	62	38.9	8.9	36	83	172	84.8	27.6
UOB-1 / 5	16	30	104	33.7	14	35	71	126	73	19.1
UOB-2 / 5	29	54	347	61.2	40.2	86	225	1078	270.8	164.7
UOB-1 / 6	19	63	130	64.4	25.2	136	385	703	386.6	129.5
UOB-2 / 6	30	61	146	68	24.2	85	436	1101	455.9	174.2

Table 6.12: Distribution of UOB Response Times for Updates of Size 50 (time in milliseconds)

sorption and internalization of the general TBox. That said, of the 460 ontologies, only 4 had at least one concept that violated the restriction. This indicates that the restriction should not have a large impact when attempting to use the technique for many ontologies. Further, in the 4 violating ontologies, on average 6.8 concepts actually violated the restriction. This is promising, as the number of concepts concepts in the ontology that cannot be used in queries is very small.

I have investigated lifting the syntactic restriction on the queries by further restricting the expressivity of the KB. In particular, if the KB is restricted to definitorial $\mathcal{SHI}$, then the restriction can be removed [64] and a similar approach as the one presented here can be used. However, discussing this approach has been omitted as assuming the KB to definitorial $\mathcal{SHI}$ greatly restricts the expressivity of the KB.

Chapter 7

Maintaining Consistency at the Syndication Broker

While Chapter 5 provides an efficient approach to determine if an inconsistency has occurred as a result of the integration of a new publication into the broker's KB, thus far a mechanism to recover from such an inconsistency has not been addressed. This is clearly an issue, as if the broker's KB is inconsistent, then all publications will trivially satisfy all registered subscriptions.

There have been numerous approaches investigated in literature to address this problem. Unfortunately, when applying many of the common theories for updating and revising logical KBs to the OWL, negative results are encountered; for example, it has been shown that the traditional minimal change, model-based update semantics cannot be represented in the DLs which OWL is aligned with [96]. Further, it has been shown that the traditional AGM theory of belief revision cannot be directly applied to OWL, as the postulates for contraction cannot be satisfied [45, 46]. As noted in Chapter 4, this has been one of the motivations for adopting syntactic updates for integrating new publication into the broker's KB.

In this chapter, two techniques to recover from an inconsistency are introduced. The first is a straightforward approach, which will be discussed next in section 7.1. Secondly, a belief-base semi-revision algorithm for OWL DL is developed, which provides a flexible mechanism to select the assertions to retract from the KB to regain consistency. To illustrate this, an approach using the notion of trust in publications to perform the selection process is presented at the end of the chapter.

7.1 Rejection-Based Approach

In this section, a straightforward technique to regain consistency is introduced. First, observe that if a publication p composed of a set of ABox assertions β is received by the broker such that it causes the broker's KB K to become inconsistent (i.e., $\mathsf{K} \not\models \perp$ and $\mathsf{K}+\beta \models \perp$), it must be the case that the inconsistency is dependent on β. This implies that if β were simply retracted from $\mathsf{K}+\beta$, then the KB would trivially be consistent. Therefore, in the first proposed approach, if a publication is received by the broker such that it causes the broker's KB to become inconsistent, then the publication is simply rejected.

From a practical point of view, supporting such an approach will introduce very little overhead, as the previous technique for incremental consistency checking can be used. Specifically, given an addition publication which leads to an inconsistency, the effects of the publication can simply be rolled-back using the previously described approach in Chapter 5; given the performance results observed in Chapter 5, such an approach will impose very little overhead.

7.2 Belief Base Revision in OWL-DL

Simply rejecting a publication because it causes the broker's KB to become inconsistent may not be ideal. This is because the entire publication may not contribute to the inconsistency, therefore rejecting all assertions in the publication is un-necessary. Further, there could be other assertions contained in other publications which additionally contribute to the inconsistency, which may be better candidates to revise/remove; for example, if one views newer publications as more important, then it may be better to retract assertions from older publications that contribute to the inconsistency. Additionally, certain publishers may be more trust-worthy than others; in this case, one may want to retain inferences deduced from the trust-worthy sources, implying that assertions from highly trusted publications should not be retracted during the revision process.

Given this motivation, a belief-base revision algorithm for OWL-DL is developed in this chapter; specifically, an algorithm for kernel semi-revision is presented. As noted in Chapter 2.5, the difference between belief-base revision and semi-revision is that semi-revision does not ensure that the newly added belief will be accepted. While there have been many different belief-base revision operators investigated in literature, a semi-revison approach has been developed as it is fitting for the syndication framework. In particular, the construction of the kernel semi-revision operator [68, 69] allows for an arbitrary function to be defined to actually select the assertions to be retracted from the minimal set of assertions causing the inconsistency. Therefore, the previous notions of publication time and trust-worthiness could be leveraged during the revision process. Additionally, the fact that the recently added publication (or assertions contained within it) can be rejected is actually ideal for our need; for example, if a trust-based approach is used and the most recent publication is trusted the least, then one would want to reject it.

7.2.1 Overview

As discussed in Chapter 2.5, the construction of the kernel semi-revision operator requires two main tasks. First, one must be able to compute all of the minimal sets of beliefs (i.e., assertions) in the KB that entail the inconsistency; as discussed in Chapter 2.5, such an operator is referred to as a kernel operator. Secondly, a function must be defined that selects for removal at least one belief from each kernel, which then results in recovering consistency of the KB; this selection function is referred to as an incision function (again introduced in Chapter 2.5).

In the next section, a kernel operator for OWL DL is first provided. This is accomplished by leveraging recent work in literature on finding all justification for entailments for OWL DL KBs [85]. Following this, a kernel semi-revision operator is defined.

7.2.2 Kernel Semi-Revision Operator

As previously mentioned, an approach is needed for computing all minimal justifications for an entailment (in our case $\mathsf{K} \models \bot$) in an OWL DL knowledge base. To accomplish this, related work in literature is leveraged; specifically, as discussed in Chapter 2.5, [85] has developed an algorithm for finding all minimum sets of assertions in an OWL DL KB responsible for an entailment, referred to as *justifications*. The notation *Just*(K, α) is used to denote the set of all justifications (see Chapter 2.5 for a more formal definition). As we are only concerned with ABox assertions (as the TBox is assumed to be fixed in the framework), it is assumed *Just*(K, α) simply contains the ABox assertions in the justification. Given this, it can easily be shown that this function can be used as a kernel operator, implying that this previous work can be leveraged for our purpose.

Lemma 8 *Let* K *be a* $\mathcal{SHOIN}$ *KB and* α *some belief, such that* $\mathsf{K} \models \alpha$*. Then,* *Just*$(\mathsf{K}, \alpha) = \bigcup(\mathsf{K} \perp\!\!\!\perp \alpha)$.

Proof See Appendix A.3.1 for the proof of this lemma. □

Next, an incision function must be defined in order to select the assertions from each kernel for removal. Given an incision function, the semi-revision operator can easily be defined. The general requirements of an incision function are introduced by Definition 2 in Chapter 2.5. This function does not ensure minimality, implying that an arbitrary number of assertions from each justification can be selected for removal.

Given this, a general purpose minimal incision function is defined, which is a adaptation of the minimal incision function presented in [147].

Definition 25 *($\mathcal{SHOIN}$–Minimal Incision Function) Let* K *be a* $\mathcal{SHOIN}$ *KB. Define a* $\mathcal{SHOIN}$*–minimal incision function to be any function* $\sigma_{\mathcal{SHOIN}}$ *that maps a set of sets of formulae into a set of formulae, such that for any set of formulae* S*:*

1. $\sigma_{\mathcal{SHOIN}}(S) \subseteq \bigcup S$

2. *If $X \neq \emptyset$ and $X \in S$ then $X \cap \sigma_{\mathcal{SHOIN}}(S) \neq \emptyset$*

3. *If for all $X \in S$, $X \cap \mathsf{K} \neq \emptyset$, then $\sigma_{\mathcal{SHOIN}}(S) \subseteq \mathsf{K}$, and*

4. *$\sigma_{\mathcal{SHOIN}}(S)$ is the smallest set satisfying conditions 1–3*

Now, we are in a position to define a semi-revision operator for OWL DL.

Definition 26 *Let K be a $\mathcal{SHOIN}$ KB and α some formula. Define the operator $?_{\sigma_{\mathcal{SHOIN}}}$ such that:*

$$\mathsf{K}?_{\sigma_{\mathcal{SHOIN}}}\alpha = (\mathsf{K} + \{\alpha\}) \setminus \sigma_{\mathcal{SHOIN}}(\mathit{Just}(\mathsf{K} + \{\alpha\}, \perp))$$

It directly follows from this definition and Lemma 8 that $?_{\sigma_{\mathcal{SHOIN}}}$ satisfies the kernel semi-revision postulates and is a kernel semi-revision operator.

Theorem 10 *$?_{\sigma_{\mathcal{SHOIN}}}$ satisfies the kernel semi-revision postulates.*

7.2.3 Semi-Revision Algorithm

The belief base semi-revision algorithm is shown in Algorithm 5. First, the underlying KB is expanded with the update; if the KB is inconsistent after the expansion, then all justifications for the inconsistency are found using the previously discussed approach for computing *Just*$(\mathsf{K}, \perp)$. Following this, the incision function selects the assertions to retract; given that an arbitrary incision function can be used, the incision function is simply denoted by *Incision*.

7.2.4 Trust-Based Incision Function

It has been pointed out that any incision function that satisfies the conditions of Definition 2 can be used to select the assertions for removal. In this section, an incision

Algorithm 5 *Semi_Revision*(K,β)

```
Input:
    K: Initial $\mathcal{SHOIN}$ KB
    $\beta$: Set of $\mathcal{SHOIN}$ assertions
Output:
    K: Consistent KB
1: $\mathsf{K} \leftarrow \mathsf{K} + \beta$
2: if $\mathsf{K} \models \perp$ then
3:     $J \leftarrow Just(\mathsf{K}, \perp)$
4:     $\mathsf{K} \leftarrow \mathsf{K} \setminus Incision(J)$
5: end if
6: return K
```

function that is based on the trust and recency (i.e. time since publication) of assertions contained in publications is presented [1].

The motivation and insight into using such an approach is that certain publishers will be more trustworthy than others; for example, if content is being disseminated from within the financial domain, publications from a reputable publisher such as the Dow Jones Newswires will be more trustworthy than Bob's financial news blog. Therefore, if an inconsistency occurs, it is intuitive to select an assertion for retraction that was published by Bob's blog, as opposed to content from the Dow Jones Newswires. Further, the incision function presented takes into account the age of the statement; this is additionally intuitive as even statements from highly trusted source will become obsolete over time.

In the approach presented, it is assumed that the syndication broker maintains the trust value for each source. When major publishers are involved, it is likely that the number of sources will be small enough that the broker can manually decide the trust value for sources. In other situations where potentially thousands of publishers are involved (e.g., syndication of blog posts), the broker can likely rely on algorithms that compute trust automatically. Developing such algorithms is out of scope of this dissertation, however there exists substantial work on inferring trust in social networks (e.g., [54, 90]) and it is likely that these approaches could be applied to this problem as well. It is pointed out

[1] See [55] for the original presentation of this algorithm.

that a philosophical discussion of how to determine trust ratings for published content is not presented, as it is out of scope of this dissertation; however, the interested reader is referred to [54], which provide detailed discussion of this topic.

First, the notion of the trust for a statement is introduced. Specifically, given an assertion α, let τ_{source_α} be the broker defined trust in the source of α and ρ_α be a measure of the recency of the statement (i.e., time since publication). Trust in α, denoted τ_α, is determined by an arbitrary function of the trust in its source and it recency. However, higher trust in the source or a more recent statement must be more trusted. This is formally defined in Definition 27; note that the notation $source_\alpha$ and $source\prime_\alpha$ denote different sources of α, whereas ρ_α and $\rho\prime_\alpha$ denotes publications of α with different measures of recency (i.e., the greater ρ_α, the longer the information has existed).

Definition 27 *(Trust Function) Trust in α, denoted τ_α, is given by a function ζ such that:*

1. $\tau_\alpha = \zeta(\tau_{source_\alpha}, \rho_\alpha)$
2. $\zeta(\tau_{source_\alpha}, \rho_\alpha) < \zeta(\tau_{source\prime_\alpha}, \rho_\alpha)$ *if* $\tau_{source_\alpha} < \tau_{source\prime_\alpha}$
3. $\zeta(\tau_{source_\alpha}, \rho_\alpha) < \zeta(\tau_{source_\alpha}, \rho\prime_\alpha)$ *if* $\rho\prime_\alpha < \rho_\alpha$
4. *if* $\zeta(\tau_{source_\alpha}, \rho_\alpha) < \zeta(\tau_{source_\beta}, \rho_\beta)$ *then* $\zeta(\tau_{source_\alpha}, \rho_\alpha + i) < \zeta(\tau_{source_\beta}, \rho_\beta + i)$

It is noted that condition 4 requires that ρ degrades τ linearly. This ensures that if $\tau_\alpha < \tau_\beta$ at time t, then at time $t + i$, $\tau_\alpha < \tau_\beta$; that is, condition 4 ensures that our choice at time t remains the same at all times in the future. Lastly, it is re-iterated that the precise trust function for these inputs is not specified, as it will likely differ per deployment of the syndication framework.

Given this, a trust-based selection function is introduced:

Definition 28 *(Trust-Based Selection Function) Define a trust-based selection function λ as any function mapping a set of $\mathcal{SHOIN}$ assertions into an assertion α, such that for any set X of $\mathcal{SHOIN}$ assertions:*

1. *If* $X \neq \emptyset$ *then* $\alpha \in X$

2. *For each* $\gamma \in X$*,* $\tau_\alpha \leq \tau_\gamma$

Intuitively, the selection function returns (one of) the least trusted assertion. Given this, a trust-based incision function is defined in Definition 29.

Definition 29 *(Trust-Based Incision Function) Given KB* K*, a trust-based incision function* σ_τ *is any function mapping a set of sets of assertions into a set of assertions, such that for any set* S *of sets of assertions:*

1. $\sigma_\tau(S) \subseteq \bigcup S$

2. *If* $X \neq \emptyset$ *and* $X \in S$ *then* $X \cap \sigma_\tau(S) \neq \emptyset$

3. *If for all* $X \in S$*,* $X \cap \mathsf{K} \neq \emptyset$*, then* $\sigma_\tau(S) \subseteq \mathsf{K}$*, and*

4. *For each* $X \in S$*,* $\lambda(X) \in \sigma_\tau(S)$

5. $\sigma_\tau(S)$ *is the smallest set satisfying conditions 1–4*

The incision function will select the least trusted statement from each kernel. Clearly there can be situations in which the intersection of two kernels A, B is non-empty. In this case, it could be that assertion $\alpha \in A \cap B$ and α is the least trustworthy assertion in A, however not in B. Clearly α will be removed from A, as it is necessary to regain consistency; similarly the least-trusted assertion in B will be retracted as well. This demonstrates that the trust-based incision function is not a minimal-incision function, as it does not satisfy the conditions of Definition 25. That said, it is a direct consequence of the definition that the trust-based incision function satisfies the condition of a general incision function defined in Definition 2. This implies that it can be used in the previously described belief-base semi-revsion algorithm.

Lastly, the pseudo-code of an algorithm for σ_τ is provided in Algorithm 6.

Algorithm 6 *Trust_Incision*(S)

Input:
S: Set of sets of $\mathcal{SHOIN}$ assertions

Output:
R: Set of assertions to discard

```
1:  $R \leftarrow \emptyset$
2:  for all $X \in S$ do
3:      $discard \leftarrow null$
4:      for all $x \in X$ do
5:          if $\tau_x < \tau_{discard}$ then
6:              $discard = x$
7:          end if
8:      end for
9:      $R \leftarrow R \cup \{discard\}$
10: end for
11: return $R$
```

Example

In this section, a brief example of the trust-based incision function is provided. First, assume that trust values range over the positive integers from 1 to 10. Additionally, for simplicity assume that given an assertion α, the trust function is defined as follows:

$$\zeta(\tau_{source_\alpha}, \rho_\alpha) = \frac{\tau_{source_\alpha}}{\rho_\alpha + 1}$$

That is, the trust in α is simply the result of dividing the trust value for a given assertion by one plus the number of time units since it was published; note that the time since publication is incremented by one to ensure division by a non-zero number.

For ease of exposition, simply assume that the broker's initial KB is empty. Additionally, let there be two publishers, P_1 and P_2, registered with a syndication broker, such that P_1 is a highly trusted source with trust value of 9. Further, P_2 is a far less trust-worthy publisher and therefore has a trust value of 3.

Now, say that at time 1, P_1 publishes the assertion *RiskyCompany(BauschAndLomb)*. Given that this is the only assertion in the KB at time 1, the KB is clearly consistent. Next assume that at time 2, publisher P_2 publishes the assertion $\neg$*RiskyCompany(BauschAndLomb)*, clearly resulting in an inconsistency. Using the previously described revision approach, the justifications (constituting the kernels) for $\mathsf{K} \models \bot$ will be determined. In this simple example, there is only one justification composed of the two assertions. Next, the following trust values would be found for the assertions at time 2:

$$\tau_{RiskyCompany(BauschAndLomb)} = \frac{9}{2}$$
$$\tau_{\neg RiskyCompany(BauschAndLomb)} = \frac{3}{1}$$

Given this, the assertion *¬RiskyCompany(BauschAndLomb)* would be removed and consistency would be regained.

7.3 Discussion

It is clear that the previously described approaches provide mechanisms to recover from an inconsistency encountered when updating the broker's KB. However, a comment is in order regarding the practicality of adopting the revision algorithm for the syndication framework; specifically, an open question is whether the approach will impose unacceptable performance overhead.

The main performance issue with the approach is related to efficiently computing all kernels (i.e., computing all justifications for an entailment). In [85] an implementation of the algorithm for finding all justifications has been provided; further, the results presented indicate that the algorithm performs well on average, and in some cases response times are even less than 1 second. However, in a syndication environment in which there are frequent publications, it is quite clear that response times in the seconds will not scale.

In the algorithm provided in [85], the main performance overhead is in computing the minimal hitting sets (see Chapter 2.5 for a more detailed discussion regarding the algorithm); specifically, this requires a substantial number of consistency checks. This is evident as the algorithm incrementally removes assertions and performs consistency checks at each node in the hitting set tree. In order to make this approach more practical, the technique from Chapter 5 on incremental consistency checking can be directly applied to this problem; that is, rather than checking for consistency from scratch at each iteration of the algorithm, the incremental approach can be used. Given the empirical results demonstrated in Chapter 5, this will clearly reduce the overhead required in re-checking consistency from scratch for each new node of the tree. Because of the direct application of this technique, further discussion of this issue is not presented.

Chapter 8

Implementation and Evaluation

In this chapter, a prototype of the syndication framework is presented. The discussion includes an overview of the system architecture, as well as a variety of details related to the implementation itself. Additionally, results from performance evaluations using the implemented system are presented; the aim is to demonstrate the practicality of the syndication framework. This is accomplished by showing that the system can match subscriptions in 10s to 100s of milliseconds for realistic subscriptions and publications. Given the actual publication frequencies observed in real domains with high publication rates (shown in section 8.3), this demonstrates practicality because the system can support substantial numbers of subscriptions efficiently.

The evaluation is conducted in two parts; first, experiments using synthetic datasets are provided; specifically, the previously discussed ontology test-suite is used to simulate newly arriving publications in the system and matching times are assessed. The second part of the evaluation investigates the practicality of the framework for syndicating real-world content from the financial domain. The financial domain has been selected as the publication frequency in this domain is very high (up to 12,000 publications per day[1]). To this end, we have worked with the Dow Jones Newswires[2], who has provided sample datasets from their historical news feed archive[3]. This allows the simulation of real publications which align with the actual publication times. Additionally, realistic subscriptions have been constructed in collaboration with the Dow Jones Newswires and used in the experiments, demonstrating the practicality of the framework using subscriptions one might

[1]Source: http://www.dj.com/Products_Services/ElectronicPublishing/DJNewswires.htm

[2]http://www.djnewswires.com/

[3]Dow Jones News & Archive For Algorithmic Applications: http://www.djnewswires.com/us/djnaap.htm

expect in the financial domain.

8.1 System Architecture

A prototype of the syndication framework presented in Chapter 4 has been implemented (in Java). The main components of the system are shown in Figure 8.1. When a new publisher joins with the system, it first registers with a publication manager. Following this, new publications can be sent to the publication manager, whose main task is to add and retract new publications from the broker's knowledge base. In a similar manner, when a new subscriber joins the system, it first registers with a subscription manager. After this, the subscriber can register new subscriptions with the subscription manger; it is additionally the subscription manager's task to maintain the list of subscribers and their subscriptions.

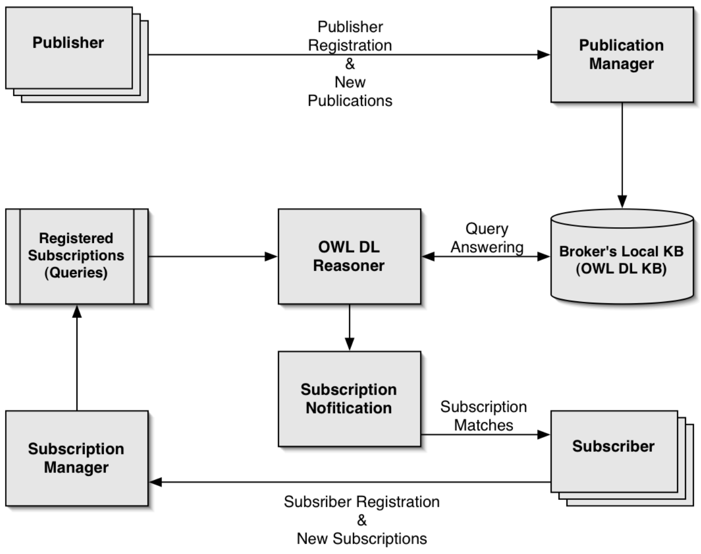

Publisher
Publisher Registration & New Publications
Publication Manager
Registered Subscriptions (Queries)
OWL DL Reasoner
Query Answering
Broker's Local KB (OWL DL KB)
Subscription Nofitication
Subscription Matches
Subscriber
Subscription Manager
Subsriber Registration & New Subscriptions

Figure 8.1: Syndication System Architecture

As discussed throughout this dissertation, the main matching engine is an OWL DL reasoner. Given a new publication, the reasoner is notified to update the results for registered subscriptions (i.e., queries). In the current implementation, the version of the

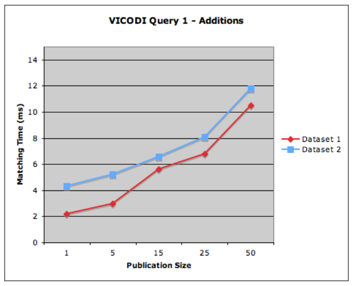
OWL DL reasoner Pellet that has been extended with the algorithms developed in this dissertation serves as the matching engine. Lastly, when there is a new match for a subscription, the subscription notification component sends the actual alerts to the necessary subscribers; note that only new matches are forwarded to the subscribers.

8.2 Synthetic Datasets

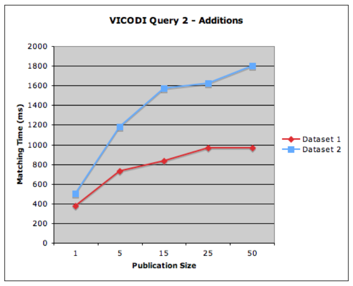
In this section, an empirical evaluation of the syndication system using the synthetic datasets discussed in Chapter 6.9 is presented. In the experiments, the same set of queries from Chapter 6.9 has been used to simulate registered subscriptions in the system; in the remainder, the queries will simply be referred to as subscriptions. As in the evaluation in Chapter 5.5, varying sized ABox additions and deletions were randomly selected from each dataset. The remainder of the dataset was used to simulate persistent background information and publications in the broker's KB. Publication sizes include 1, 5, 15, 25, and 50 assertions. In the experiments, the subscription is first registered with the subscription manager; it is noted that in all of the experiments, it was assumed that the subscriptions where for information matches, as publication matches can easily be found from new information matches. Following this, the randomly selected publications where submitted to the publication manager. It was assumed that all subscriptions and publications persisted throughout the evaluation. The aim behind the experiment was to simulate new publications arriving at the syndication broker and to asses the practicality of the syndication system.

The experiments where run on a Linux machine with 2Gb of RAM and a 3.06GHz Intel Xeon CPU. In all of the figures, the X-axis corresponds to the publication size, while the Y-axis is the average time in milliseconds for processing a new publication (i.e., consistency checking, followed by query answering) after it has been integrated into the OWL KB; the scale is logarithmic.

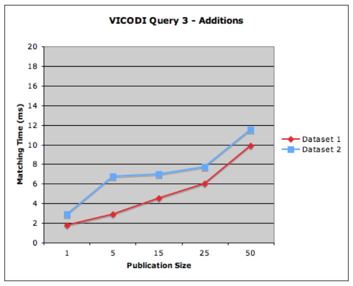
Figure 8.2 presents the average matching time for all of the VICODI subscriptions



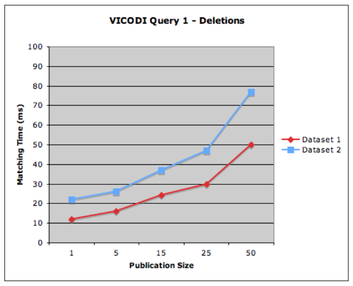
(a)



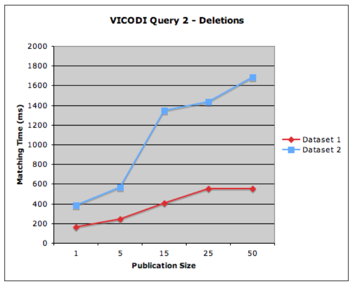
(b)



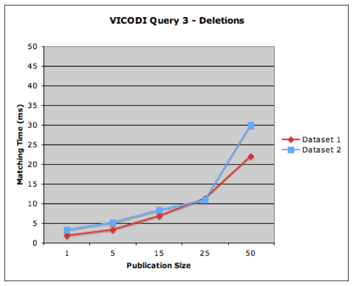
(c)



(d)



(e)

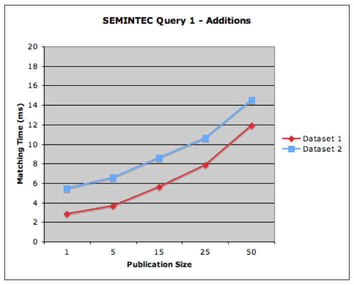


(f)

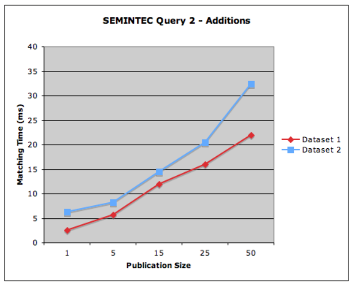
Figure 8.2: (a) VICODI subscription 1 - additions (b) VICODI subscription 2 - additions (c) VICODI subscription 3 - additions (d) VICODI subscription 1 - deletions (e) VICODI subscription 2 - deletions (f) VICODI subscription 3 - deletions

(see Chapter 6.9 for a discussion of these queries). It can be observed that for addition publications, subscription 1 and 3 are always matched in 10s of milliseconds; this demonstrates the practicality of the syndication system. Deletion publications introduce slightly more overhead, as the incremental query answering technique is more costly for deletions (see Chapter 6.9 for a discussion). However, it can be observed that deletion publications are still matched in 10s of milliseconds; further, in realistic syndication systems publications are much more likely to be additions (i.e., new publications).

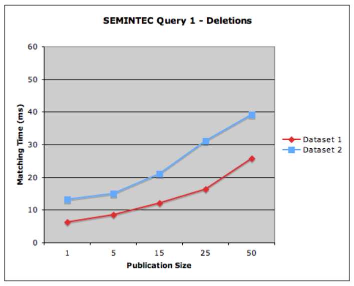
It can be observed that subscription 2 introduces additional overhead; as discussed in Chapter 6.9, this is because the incremental query answering technique is not as effective for this subscription, essentially resulting in the subscription being re-evaluated from scratch over the entire broker's KB. If such a problematic subscriptions were encountered in a real deployment of the syndication system, one could potentially batch process the subscription (i.e., re-evaluate the subscription more infrequently). While this would intro-



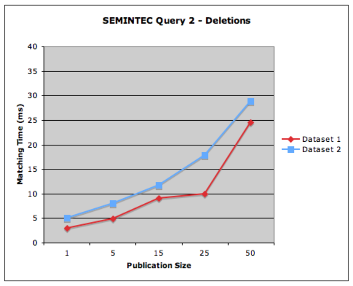
(a)



(b)



(c)



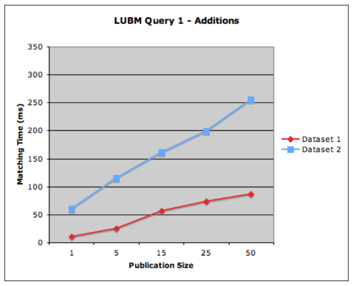
(d)

Figure 8.3: (a) SEMINTEC subscription 1 - additions (b) SEMINTEC subscription 2 - additions (c) SEMINTEC subscription 1 - deletions (d) SEMINTEC subscription 2 - deletions

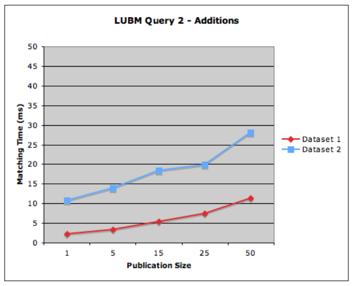
duce a time delay when matching the publication with the necessary subscribers, it would reduce the overhead. However, in the remainder of this chapter, the subscriptions do not demonstrate this behavior.

Figure 8.3 presents the average matching times for the SEMINTEC subscriptions. It can be observed that for addition publications matching is performed in 10s of milliseconds. Further, the same is observed for deletions. This clearly demonstrates that the practicality of the system.

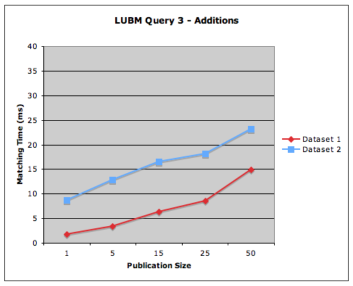
Figures 8.4 & 8.5 present the average matching times for LUBM subscriptions 1–3 and 4–6 respectively. It can be observed that for addition publications using the smaller dataset, all matching times are in the 10s of milliseconds; the same is also observed for the larger dataset for subscriptions 2–6. It can be seen that for the first subscription over the larger dataset, slightly more overhead is introduced; however, the response time is well below one second.



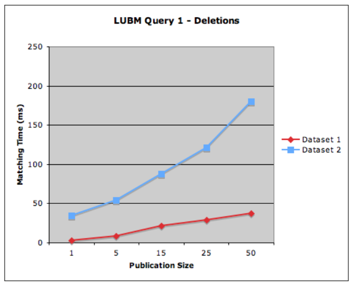
(a)



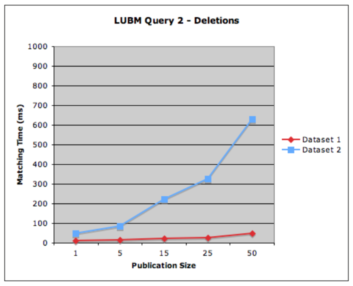
(b)



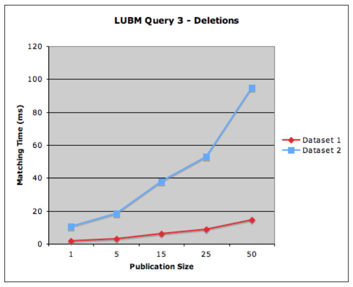
(c)



(d)

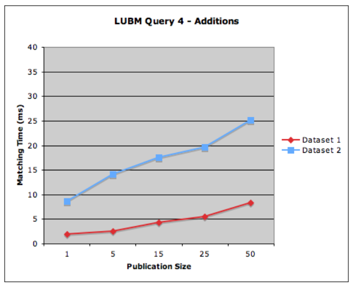


(e)

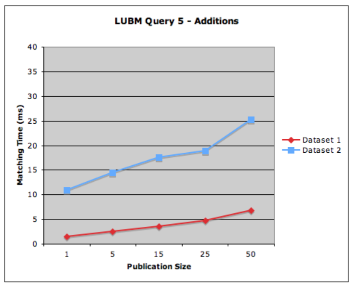


(f)

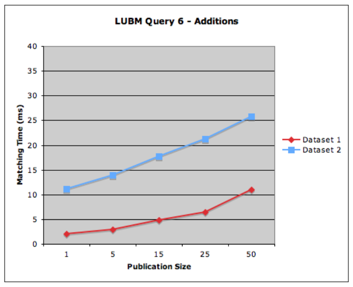
Figure 8.4: (a) LUBM subscription 1 - additions (b) LUBM subscription 2 - additions (c) LUBM subscription 3 - additions (d) LUBM subscription 1 - deletions (e) LUBM subscription 2 - deletions (f) LUBM subscription 3 - deletions



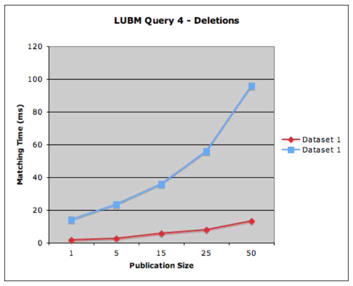
(a)



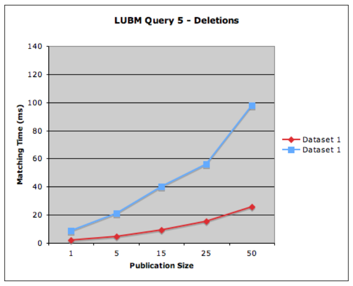
(b)



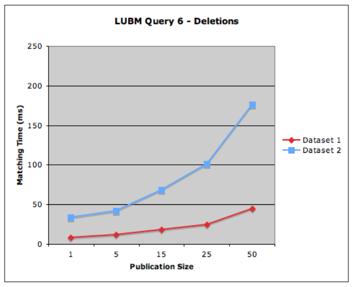
(c)



(d)



(e)



(f)

Figure 8.5: (a) LUBM subscription 4 - additions (b) LUBM subscription 5 - additions (c) LUBM subscription 6 - additions (d) LUBM subscription 4 - deletions (e) LUBM subscription 5 - deletions (f) LUBM subscription 6 - deletions

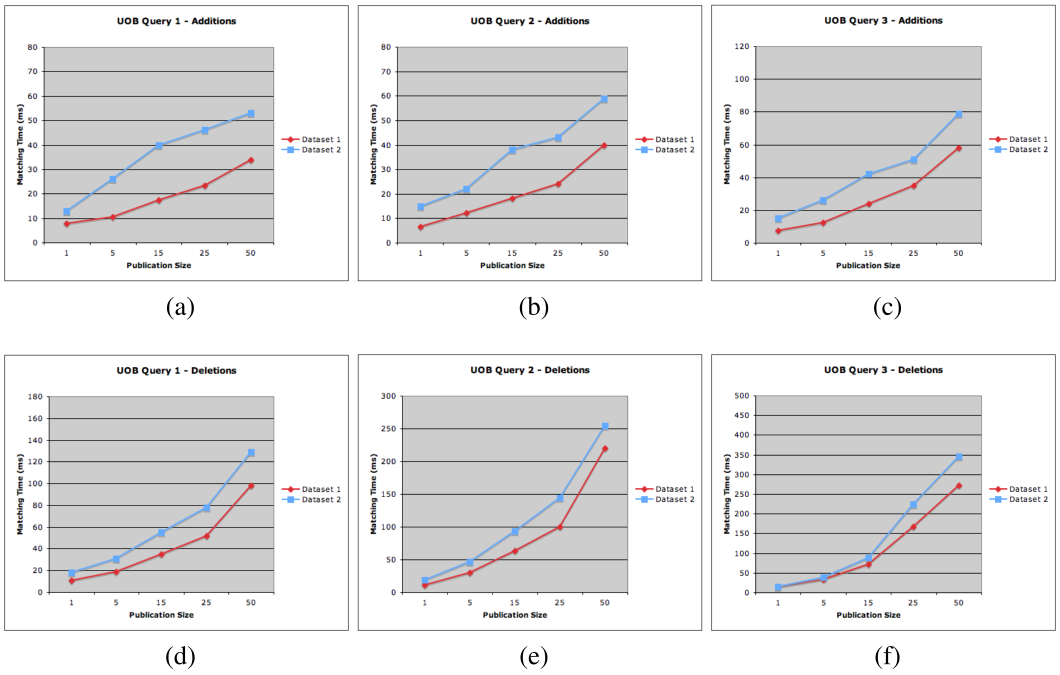
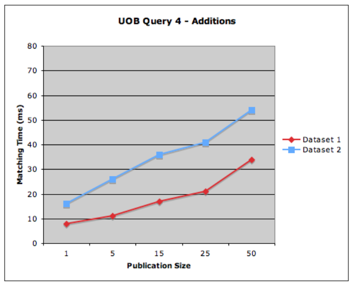


Figure 8.6: (a) UOB subscription 1 - additions (b) UOB subscription 2 - additions (c) UOB subscription 3 - additions (d) UOB subscription 1 - deletions (e) UOB subscription 2 - deletions (f) UOB subscription 3 - deletions

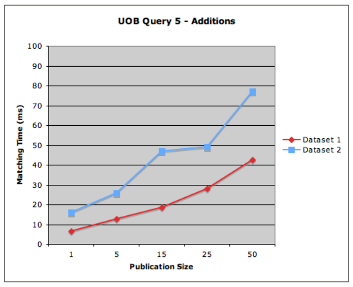
Lets us now turn our attention to deletion publications; as in the case for additions, all subscriptions over the smaller dataset are matched in 10s of milliseconds. For the larger dataset, it can be observed that slightly more overhead is introduced; however, again, it is expected that deletions are far more infrequent and the current matching times are still very promising, as they are well below one second.

Lastly, Figures 8.6 & 8.7 present the average matching times for UOB subscriptions 1–3 and 4–6 respectively. Similarly to LUBM, all matching times for addition publications are in the 10s of milliseconds. Similarly, many of the deletion publications over the smaller dataset are matched in 10s of milliseconds.

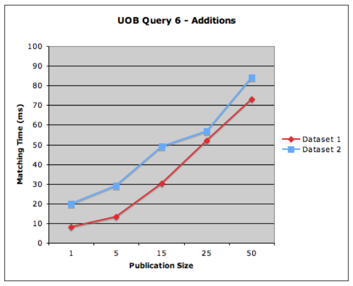
In summary, the empirical results demonstrate that the syndication framework will be practical for realistic domain models (i.e., OWL ontologies). Further, the results indicate the approach will scale even when substantial information is persisted (e.g., background information or publications) in the broker's knowledge base.



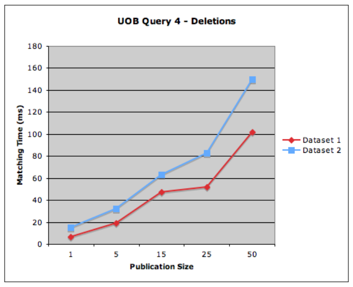
(a)



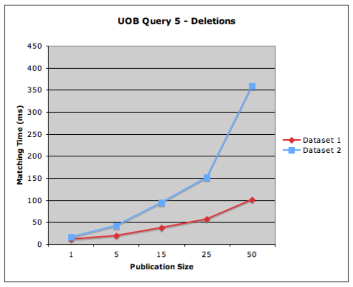
(b)



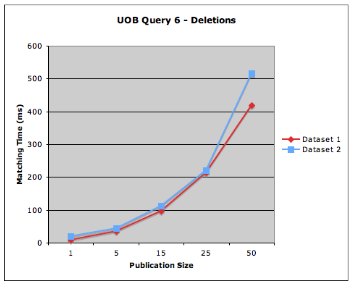
(c)



(d)



(e)



(f)

Figure 8.7: (a) UOB subscription 4 - additions (b) UOB subscription 5 - additions (c) UOB subscription 6 - additions (d) UOB subscription 4 - deletions (e) LUBM UOB 5 - deletions (f) UOB subscription 6 - deletions

8.3 Real-World Financial Dataset

While the usage of the synthetic datasets in the previous section indicates that the syndication framework will be practical, it is still an open question as to whether such a system will meet the performance demands of real-world, high demand syndication environments. Therefore, in this section we explore the application of the system to disseminating real-world news feed information from the financial domain. This domain was selected as there exists a very high publication frequency (e.g., > 12,000 publications per day) and there is a critical need to perform matching very efficiently (e.g., for stock trading purposes).

To this end, we have worked with the Dow Jones Newswires to obtain historical news feed content from the financial domain; fortunately, the Dow Jones Newswires maintains a 20+ year historical news feed archive, which is utilized by their customers for a variety of uses (e.g., building automated trading systems). Further, the archive is represented in a numerous representation formats, including XML, and has metadata associated with each news item; as we will see later in this chapter, this allows a straightforward translation to an OWL representation and therefore the utilization of the developed syndication framework. Additionally, there exists time stamps associated with the actual publications, allowing realistic simulations to be performed in which we can assess the practicality of the framework for this domain.

Given this, an OWL domain model has been constructed, which is an extension of the current metadata category codes (taxonomy-like) that the Dow Jones Newswires has created and associates with published news items; this serves as the syndication broker's fixed schema and allows for subscriptions to be issued over the classes and properties defined in this ontology. Further, we have worked with the Dow Jones Newswires to develop realistic subscriptions for the financial domain. Collectively, this allows a real-world assessment of the practicality of the syndication framework for a high-frequency publication domain.

The remainder of this chapter is organized as follows; first, background information related to the Dow Jones Newswires historical news feed archive is presented. Following this, a discussion of the construction of the OWL domain model is provided. A brief discussion of an extension of the previously described system architecture is discussed. Then an overview of the subscriptions used in the simulations is presented. Lastly, we presented the empirical results from the evaluation.

8.3.1 News Feed Background and Overview

The Dow Jones Newswires News & Archive[4] (simply referred to as archive) is 20+ year historical news archive dating back to January 1986 that contains news stories

[4]Dow Jones News and Archives for Algorithmic Applications: http://www.djnewswires.com/us/djnaap.htm

from the Dow Jones Newswires, The Wall Street Journal[5], and Barron's[6]. The archive is represented in various formats, including the Dow Jones News Mark-up Language, Dow Jones Composite Feed (text-based), and NewsML/NITF; note that the later is a standardized XML representation for news content.

Importantly, each news item contains a variety of metadata associated with it; this includes the date, time, story headline, company ticker symbols, and Dow Jones category codes. The later are a collection of subject-specific metadata codes (with a taxonomic-like structure), which details the topic of the article (i.e., specific industry, region, government statistics, etc.). Due to proprietary issues, further discussion regarding the metadata codes is omitted.

8.3.2 OWL Domain Model

To utilize the archive for empirical evaluations of the developed syndication framework, the category codes associated with published articles have been translated to an OWL ontology. This has been performed by inspecting the codes and their accompanying XML DTD (provided by the Dow Jones Newswires) and creating an automatic translation to an OWL ontology. During this conversion, implicit relations from the category codes have been created as well (e.g., the containment of sub-regions in regions).

After converting the initial category codes, additional modeling was performed to enrich the ontology. This included constructing classes and properties describing relations between organizations, employees, government organizations, and regions. Modeling was also performed to describe the general investment domain, including securities, portfolios, etc. More complex modeling related to government regulatory decisions (e.g., FDA clearances) was performed to allow subscriptions of interest for the Dow Jones Newswires.

[5]http://www.wsj.com/
[6]http://www.barrons.com/

The resulting OWL ontology was additionally populated with instances from freely available data-sources. This included the population of all of the listed securities and their corresponding companies for the NASDAQ, NYSE, and AMEX stock exchanges. This allowed stock ticker symbols associated with published articles to be integrated with additional background information about the companies and the securities; this in turn, allows for more complex OWL inferences when determining subscription matches. Additionally, geographic regions were populated using commonly available OWL ontologies (which also include instances) describing the domain; specifically, this includes the usage of a countries ontology[7] developed by Jenz & Partner GmbH[8]. Given that there was a dramatic overlap with the regions in the countries ontology and the region specific Dow Jones Newswires category codes, OWL constructs were additionally utilized to state that regions were equivalent (using the OWL *sameAs* constructor).

Due to proprietary Dow Jones Newswires content, the ontology is not made available. However, it is noted that in the ontology, the following constructs are used: concept and role subclass axioms, negation, universal and existential property restrictions, inverse roles, transitive roles and datatypes. Table 8.1 provides additional details regarding the ontology, including the DL expressivity, number of classes, properties, individuals and triples; additionally, the average time to perform the initial consistency check using Pellet (in seconds) of the ontology is shown. The later is provided as it demonstrates the overhead of checking the consistency of the broker's KB from scratch.

Expressivity	**♯ Classes**	**♯ Properties**	**♯ Individuals**	**♯ Triples**	**Init. Cons. (sec.)**
$\mathcal{SHI}(\mathcal{D})$	58	53	21,683	68,238	4.3

Table 8.1: Financial Ontology Overview

[7]Available at http://www.w3.org/Consortium/Offices/Presentations/RDFTutorial/rdfs/Countries.owl
[8]Jenz & Partner GmbH Homepage: http://www.jenzundpartner.de/

8.3.3 Extended System Architecture

It is clear that a conversion step must be performed when a new publication is received. In the evaluation, we have used the NewsML/NITF format of the archive, as there exists freely available APIs for manipulating NewsML documents[9]. Given this, it is a trivial process to convert the existing metadata associated with published articles to the OWL model.

In order to augment the previously existing metadata and provide richer filtering based on semantic content, an information extraction module has been implemented as well. While simplistic, the module developed exploits the observation that many of the headlines for news articles have a similar structure; this is particularly true for specific government statistics, company earnings, and regulatory decision releases. For example, the phrase "*XYZCompany Gets FDA OK For XYZDrug*" is often used in the headline of news articles discussing regulatory decision by the Food and Drug Administration. Clearly, more sophisticated information extraction and natural language processing techniques would be effective as well, however further exploring their use is out of scope of this dissertation.

The previous system architecture presented in section 8.1 is slightly extended to include this functionality. Figure 8.8 shows the main components of this extension; specifically, a new publication is first processed by information conversion and extraction modules prior to their receipt by the publication manager.

8.3.4 Real-World Subscriptions

We have additionally consulted our collaborators at the Dow Jones Newswires when selecting the subscriptions to use in the simulations. The following four queries have been selected.

[9]For example, see the NewsML Toolkit: http://newsml-toolkit.sourceforge.net/

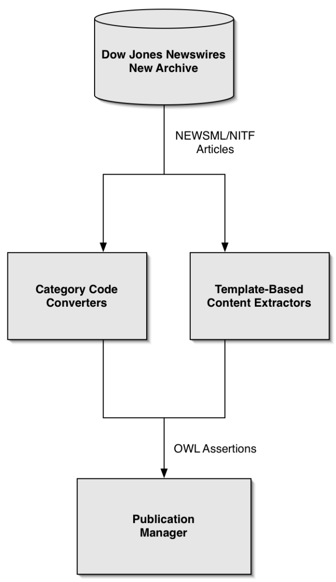


Figure 8.8: Metadata Converter Architecture

1. $(x) \leftarrow \mathit{CompanyWithPositiveFDARegulatoryDecision}(x)$
2. $(x, y) \leftarrow \mathit{CompanyWithPositiveFDARegulatoryDecision}(x) \wedge \mathit{issuesSecurity}(x, y) \wedge \mathit{listedOnExchange}(y, \mathit{NASDAQ})$
3. $(x) \leftarrow \mathit{Publication}(x) \wedge \mathit{hasTopic}(x, \mathit{AAPL})$
4. $(x, y) \leftarrow \mathit{Publication}(x) \wedge \mathit{hasTopic}(x, y) \wedge \mathit{containedIn}(y, \mathit{HalaschekPortfolio})$

The first subscription is a retrieval query for all instances of that are companies that have some positive FDA regulatory decision. This subscription is of interest as it will likely indicate a stock which will become volatile due to new regulatory decisions. It is noted that a variety of OWL inferences are necessary to conclude that an instance is actually a member of this class; specifically the concept is defined in terms of a conjunction of the concept *Company* and an existential property restriction on the role *receivesRegulatoryNotification* and role filler *GrantedFDARegulatoryDecision*. Additionally, *receivesRegulatoryNotification* is an inverse role, and *GrantedFDARegulatoryDecision* is also defined in terms of a complex concept.

Subscription 2 is an extension of the first subscription, and specifically only matches instances that are of type *CompanyWithPositiveFDARegulatoryDecision*, which additionally issue a security on the NASDAQ stock exchange; it is noted that *issuesSecurity* has an inverse role.

The third subscription matches publications with a topic that includes the common stock for *Apple Inc*. Such a subscription is intended to filter information such that only news related to a specific company is returned to the subscriber. The last subscription is for any publication that has some topic that is contained in a specific type of investment portfolio, namely *HalaschekPortfolio*. This subscription is intended to filter information that is only relevant to a collection of securities.

8.3.5 Empirical Results

In this section, the results of the simulations using the Dow Jones Newswires news archive, previously described domain model, and subscriptions are presented. In the evaluation, three different one week subsets of the archive have been used, namely from Oct. 20–26, 2001, Oct. 21–27, 2002, and Oct. 17–23, 2004. In each experiment, the previous four subscriptions are initially registered with the syndication broker. Then, three different simulations have been run using the different one-week datasets; in particular, the historical news items from the datasets are streamed into the syndication in real-time using the time-stamps associated with publications. Additionally, in the experiments each publications is persisted in the syndication broker's KB for 2 days and then expired. In the event of an inconsistency after receiving a new publication, the rejection-based approach discussed in the previous chapter was used. As in the evaluation with the synthetic datasets, response times for information matches have been assessed for the different subscriptions.

Prior to presenting the average matching times for the various subscriptions, a discussion regarding the actual publication frequencies observed in the datasets is presented. This will later provide insights into the practicality of the syndication framework. First, Table 8.2 presents an overview of the average number of publications observed over various timeframes in the different datasets; specifically, the average number of publications per second, minute, hour, and day is presented. In the table, different subsets of the various one-week time periods is also shown. The first row corresponds to the publication frequencies for the weekend days (i.e., Saturday and Sunday) for each dataset. It can be observed that during these days, the publication frequency is the lowest, given that the financial markets are not open. The next row presents the average publication frequencies for the weekdays in the datasets. Importantly, it can been observed that on average .08–.13 news items are published per second. Next, the publications frequencies for the weekdays between 7:00am–7:00pm EST is provided. These have been shown, as it is intuitive to expect there to be higher publication frequencies during actual trading hours; it can be seen that this is in fact observed. The maximum publication frequencies during the 7:00am–7:00pm EST weekday timeframe have also been provided. Interestingly, there appears to be periodic bursts of a high number of publications within one second.

Time	Dataset	Pub. / Sec.	Pub. / Min.	Pub. / Hour	Pub. / Day
Weekend	10/21/01-10/27/01	.005	.29	16	276
	10/20/02-10/26/02	.01	.31	17.59	325
	10/17/04-10/23/04	.01	.38	21.52	358
Weekdays	10/21/01-10/27/01	.11	7	403.9	10,098
	10/20/02-10/26/02	.08	5.02	289.3	7,234
	10/17/04-10/23/04	.13	8.28	477.8	11,925
Peak Hours	10/21/01-10/27/01	.15	9.2	542.8	6,691
	10/20/02-10/26/02	.11	6.69	371.1	4,825
	10/17/04-10/23/04	.18	11.1	645	8,014
Max. Peak	10/21/01-10/27/01	34	69	1,110	7,682
	10/20/02-10/26/02	63	63	1,254	5,849
	10/17/04-10/23/04	13	85	1,530	9,361

Table 8.2: Publication Frequency Distribution

Table 8.3 provides additional insights into the cases where a high number of publications is received within one seconds. First, the table shows the average number of occurrences during each weekday in which more than 20 publications are produced within one second (the third one-week period is omitted as this never occurs). In general, it can be seen that this does not occur very often. The observation that there are times when a high number of publications are disseminated within one second might suggest that there could potentially be scalability issues if a subsequent high number of publications is immediately received. In light of this concern, Table 8.3 additionally provides the average number of publications produced in the following minute after a one second period with more than 20 publications. It can be seen that the frequency essentially returns to the average publication frequency per second shown in Table 8.2. This indicates that typically these bursts do not appear to be prolonged.

Dataset	**Freq. of Pub./Sec. > 20**	**Avg. ♯ Pub. in Next Min.**
10/21/01-10/27/01	47.5	15
10/20/02-10/26/02	63	16.08

Table 8.3: Frequency of High Publication Rates in One Second

Next, Table 8.4 presents the average number of OWL assertions obtained when converting and extracting information from each article from the archive. This indicates that the conversion/extraction process is successfully able to generate OWL assertions for matching purposes.

10/20/2002–10/26/2002	10/21/2001–10/27/2002	10/17/2004–10/23/2004
27.46	29.45	34.62

Table 8.4: Average Number of OWL Assertions per Publication

Let us now turn our attention to matching time for the various subcrptions. It is first noted that in the figures, the matching time consists of both the time to check the consistency of the broker's KB and re-evaluate the registered subscriptions.

Consider the first subscription; it is pointed out that initially executing the subscription over the broker's KB takes on average 540 milliseconds. Figure 8.9 presents the average matching time for the first subscription given an new publication. The response times range from under 10ms to just over 12ms, depending on the dataset; this clearly indicates the effectiveness of the incremental reasoning algorithms developed in previous chapters. This also shows that the framework will be practical for realistic domains.

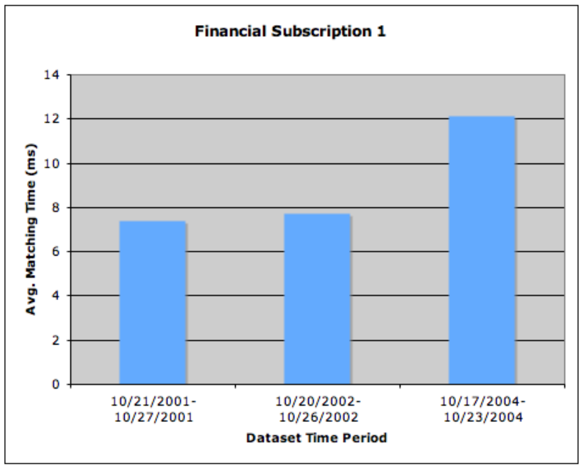


Figure 8.9: Finance Subscription 1 Average Matching Time

Next, let us consider the second subscription. Running this subscription (i.e., query) from scratch over the broker's initial KB takes approximately 2.08 seconds; clearly, re-running this subscription from scratch in the event of a new publications will not scale for even a small number of subscriptions. Figure 8.10 presents the average matching times for the second subscription. It can be observed that this subscription introduces slightly more overhead, however matching time is still well below 100ms, demonstrating the utility of the developed algorithms.

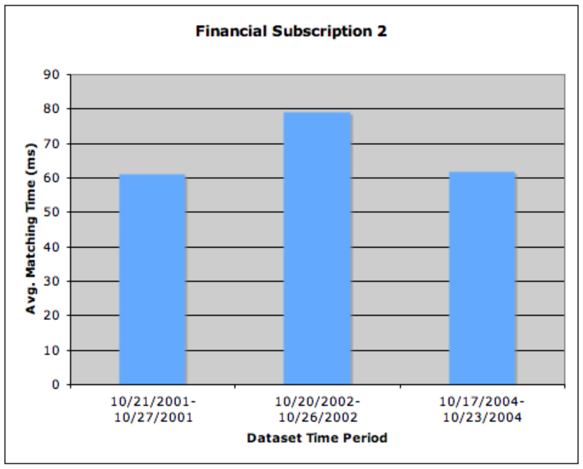


Figure 8.10: Finance Subscription 2 Average Matching Time

The third and fourth subscriptions are initially evaluated more efficiently then the previous two subscriptions; namely, they are answered in 7 and 8 milliseconds respectively on average (however, note that the initial consistency check takes more than 2 seconds). Figures 8.11 & 8.11 presents the average matching time for these subscriptions. It can be seen that the matching times range from approximately 30 milliseconds to just over 80 milliseconds.

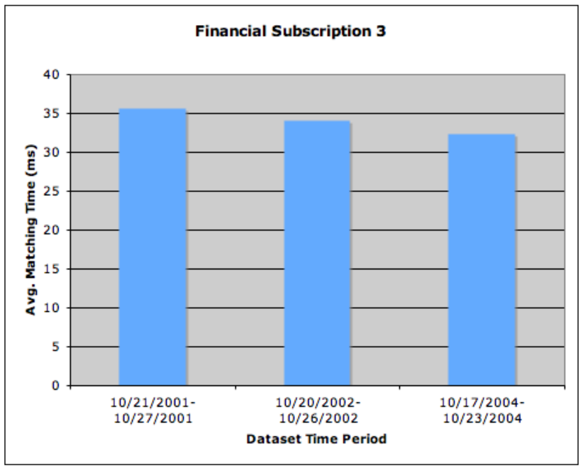


Figure 8.11: Finance Subscription 3 Average Matching Time

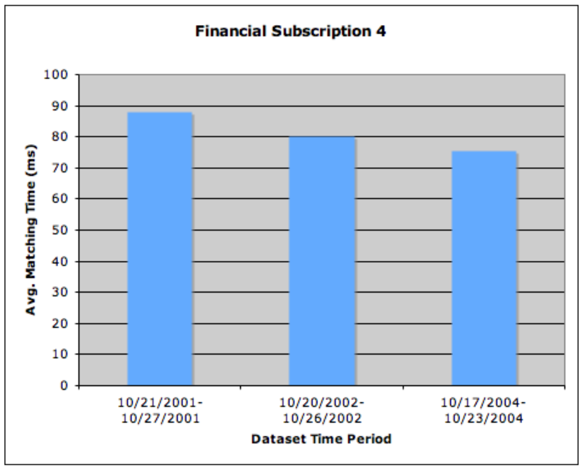


Figure 8.12: Finance Subscription 4 Average Matching Time

It is important to note that during all of the simulations, the system did not encounter a situation in which it effectively got stuck evaluating a subscription for a period long enough to cause a bottleneck at the matching engine (i.e., OWL reasoner). Even during the (infrequent) times in which a very high number of publications was received within one second, the publication buffer still cleared; this is explained from the observation from Table 8.3, which shows that subsequent publication frequencies return to the average case.

8.3.6 Discussion

The empirical results demonstrated in the previous section indicate that a substantial number of subscriptions can be supported in the framework without creating a bottleneck in the matching engine. For example, assuming the average publication frequency during the peak weekday hours, then potentially 100s of subscriptions can be supported. On the other hand, assuming the average number of publications per second observed during the weekend, then it is clear that 1,000s of subscriptions can be supported. This in turn demonstrates the practicality of the framework, as the results show that my techniques work for realistic OWL domain models, publication frequencies, and subscriptions.

Such a number of subscriptions may be acceptable for small to medium deployments of the syndication framework, depending on the domain models and publication frequencies. In order to achieve increased numbers of subscriptions, a production level deployment of the framework involving multiple servers dedicated to performing matching for a subset of the registered subscriptions can be developed. This would dramatically increase the number of subscriptions supported. However, further addressing this issue is out of scope of this dissertation. Additional future work to further increase the scalability of the framework is presented in Chapter 9.2.

Chapter 9

Conclusions and Future Work

9.1 Conclusions

The main goal of this dissertation was to develop a syndication framework that utilizes a rich semantics-based mechanism for matching that is practical for real world use. This provides finer control for filtering published information by using automated reasoning, resulting in subscription matches not found using traditional syntactic syndication approaches.

Given this motivation, a syndication framework has been developed in Chapter 4, in which published information is represented using OWL. Given OWL's alignment with description logics, matching newly published information with subscription requests in the framework is reduced to DL query answering. Therefore, the previous goal of using automated reasoning for matching within the framework is accomplished via DL reasoning algorithms.

While the developed framework is more expressive more traditional approaches, using standard DL reasoning algorithms introduces overhead which ultimately prohibits the framework from being practical for many real-world domains and publication frequencies. This is primarily due to reasoning through the frequent changes to the KB caused by new publications.

Within the syndication framework, two DL reasoning services are required, namely consistency checking and query answering. Therefore, in Chapter 5, incremental consistency checking techniques have been developed for the DLs $\mathcal{SHIQ}$ and $\mathcal{SHOQ}$. The main idea behind the approach is to leverage information related to the model constructed in previous consistency checks. Specifically, the developed algorithms incrementally maintain tableau completion graphs, which correspond to a model for the KB. Then, in the event of an addition or deletion to the KB, the previous completion graph can be updated to reflect the changes.

Following this, the topic of query answering in the presence of updates to DL KBs has been addressed. Specifically, Chapter 6 presents an algorithm for reducing the portion of the KB that must be considered after an update for the DL $\mathcal{SHI}$. The developed approach safely prunes candidate answers by exploiting the interactions between the effects of the update and the query concepts on the KB. This provides an effective technique for incrementally maintaining query results as the underlying KB is updated.

Given the likelihood that logical contradictions will be encountered as the syndication broker's KB is updated with new publications, two approaches for recovering from inconsistencies have been developed in Chapter 7. The first is a straightforward approach in which problematic publications are simply discarded. The second technique is a belief-base revision algorithm which is flexible in that arbitrary functions can be defined to determine the assertions to retract from the KB to regain consistency. A specific function using the notion of trust in publication sources has also been presented.

An important goal of this dissertation was to demonstrate the practicality of the syndication framework. To this end, a comprehensive empirical evaluation has been conducted, the results of which have been presented in Chapter 8. In the evaluation, experiments have been performed using synthetic datasets to simulate publications in the system. Additionally, real-world simulations have been performed using historical news items from the financial domain obtained by collaborating with the Dow Jones Newswires. In these evaluations, historical time-stamps have been used to simulate the actual publication frequencies observed. This allows the assessment of whether the syndication system can accommodate high frequency publication rates observed in the financial domain. In the end, the utility of the algorithms and practicality of the syndication framework developed in this dissertation are shown.

A summary of the contributions in this dissertation are as follows:

- A formalization of a syndication framework for the Web, which is based upon the Web Ontology Language and description logic reasoning.
- A set of techniques for efficient consistency checking through incremental updates to the KB. This provides a practical approach for integrating new publications into the broker's KB.
- A set of techniques for reducing the portion of the KB that must be considered for query answering after an update to the KB. This provides efficient query answering through incremental changes to the KB.
- An algorithm for recovering from logical inconsistencies for DL knowledge bases. This allows the syndication broker to regain consistency in the event a publications which cause a logical contradiction in its KB.
- Demonstrated the practicality of the OWL-based syndication framework by performing a comprehensive evaluation of syndication framework using synthetic datasets, as well as real world data from the financial domain.

9.2 Open Issues and Future Work

In this section, the limitations and open issues of this dissertation are discussed. Additionally, areas for future work are outlined. First, discussions are provided regarding the extension of the syndication framework developed in this dissertation.

9.2.1 Extending the Syndication Framework

Even More Expressivity

Supporting Boolean Queries

In the syndication framework developed in Chapter 4, subscriptions have been represented as conjunctive retrieval queries (i.e., they contain at least one distinguished variable), and therefore boolean queries (i.e., no distinguished variables) are not supported. As stated in Chapter 4, boolean queries are not supported as there is no notion of an information match when subscribers' interests are represented as boolean queries. Additionally, in many real world applications, queries typically have some number of distinguished variables. In general, it is straightforward to extend the framework to support boolean queries. The notion of continuous conjunctive boolean queries can be defined in a similar manner as continuous retrieval queries, and publication matches can be extended to support boolean queries as well.

Publication Representation

In this dissertation, I have proposed the use of the subsets of the Web Ontology Language that align with description logics as the means for encoding published information in the developed syndication framework. Currently, there are efforts to extend OWL with additional DL constructs, resulting in OWL 1.1[1]. Clearly, there can be extensions of the developed syndication framework to allow the use of these additional constructs provided by this and future extensions to OWL. There are also ongoing efforts to standardize rules languages for the Web (e.g., the Rules Interchange Format[2]), and therefore, another interesting avenue for future work includes supporting the results of such standardization efforts within the syndication framework. Interesting directions for future work also include investigating syndication frameworks that support full first order logic or higher

[1]W3C OWL Working Group: http://www.w3.org/2007/OWL/
[2]W3C Rule Interchange Format homepage: http://www.w3.org/2005/rules/

order logics, as well as non-monotonic logics (e.g., there have been non-monotonic extensions of descriptions logics; for example see [24]). In general, potential interesting research problems include formalizing an integration of these different rules languages within the framework and addressing the scalability issues introduced by the inclusion of additional formalisms.

Distributed Architectures

In the syndication framework, it has been assumed that there is a single syndication broker. However, it is clear that the scalability of the approach can be further increased by extending the framework to a distributed architecture. This is a common technique investigated in literature when further extending previous syndication approaches (e.g., [21, 150, 38, 151]). Simply utilizing additional servers that are dedicated to a subset of the registered subscriptions can be supported in a fairly straightforward manner and is largely an engineering issue. However, supporting distributed matching is much more difficult problem, as this reduces to distributed DL reasoning. This topic has recently gained attention in literature (e.g., see [134, 135]), and these techniques may eventually lead to possible extensions of the framework.

Exploiting Subscription Overlap

Similar to the previous section, exploiting the overlap of registered subscriptions has been investigated in syndication literature for the purpose improving scalability. The intuition is to exploit the containment relationship between queries to determine more optimal subscription evaluation orderings when a new publication is received. The notion of query containment in DLs has been investigated in literature (e.g., see [92]) and can certainly be leveraged in the developed syndication framework. Given the straightforward application of such an approach, it has been left as future work. It is also noted that recent work on determining more optimal subscription evaluation orders for general syndication systems [98] can be leveraged as well.

9.2.2 Enhancing Incremental Reasoning Techniques

In this dissertation, incremental reasoning algorithms have been developed for a variety of description logics. In the following sections, some limitations and tradeoffs of these techniques are addressed and future work related to these issues is identified.

Consistency Checking

The incremental consistency checking algorithm developed in Chapter 5 is applicable to the DLs $\mathcal{SHIQ}$ and $\mathcal{SHOQ}$. However, given that the framework is applicable to all of OWL DL, extending the algorithm to $\mathcal{SHOIQ}$ would allow efficient support for ontologies that utilize all of the constructs available in OWL DL. The restriction to $\mathcal{SHIQ}$ and $\mathcal{SHOQ}$ is primarily due to fact that there is no expansion rule ordering imposed by the tableau algorithm when these logics are considered; therefore, the correctness of the algorithm can be shown. It should be possible to extend the technique to all of OWL DL, however this is left as future work, as a very large subset of OWL has been covered using the algorithm just described, and by using this subset a far more expressive syndication framework is provided when compared to XML and RDF/S-based approaches.

Related to this is additionally investigating the extension of the technique to more expressive formalisms discussed in section 9.2.1, such as OWL 1.1, non-monotonic extensions to DLs, or even full first order logic.

As noted in Chapter 5, in the worst case, the performance of the incremental consistency checking algorithm is the same as reasoning from scratch. A near worst-case scenario could potentially occur if the updated completion graph (i.e., model) contains a clash, which requires a substantial portion of the completion graph to be reverted to a previous non-determinstic choice. Additionally, this could possibly occur if the update itself causes structures to propagate to a large portion of the existing completion graph. For example, if there is an individual *a* in the KB that is highly connected to the other individuals via a *knows* role and the update states that *a* instantiates the concept $\forall knows.Person$, then this will cause the propagation of the $Person$ concept name to all other individuals; this in turn, may then cause events to occur due to the fact that all individuals now satisfy the $Person$ concept. While this worst case was not encountered during experiments, it is possible that it can occur and therefore additional empirical evaluations to determine when and how frequently this occurs is left as future work. Additional ontologies can be utilized to investigate this problem as well.

Query Answering

Similar to the consistency checking algorithm, the approach for incremental query answering is applicable to a subset of OWL DL, namely $\mathcal{SHI}$. The main issue with respect to supporting all of OWL DL is that the construction and maintenance of the summary completion graph becomes difficult to accomplish. For example, extending the approach to deal with function roles or number restrictions is currently an issue, due to the merging of arbitrary nodes in the completion graphs during the tableau algorithm. Again, by using this portion of OWL DL, a far more expressive syndication framework is provided when compared to XML and RDF/S-based approaches. Given this, extending the technique to support a larger part of OWL is left as future work. As in the case of the consistency checking algorithm, related to this is additionally investigating the extension of the technique to more expressive formalisms discussed in section 9.2.1.

While the approach demonstrates very impressive results, there are some potential limitations with the technique. One issue is related to the size of the overestimate of individuals affected by updates. In particular, if the approach produces an overestimate that is too large, the value of the approach will degrade (note that in the worse case, the number of individuals one would have to check is the same as in the non-incremental case). This in fact was observed in the case with the second query of the VICODI test ontology, and similar to the previously pointed out limitation with the incremental consistency checking approach, this is caused by individuals in the KB that are connected to a large number of other individuals via a role in the query. While this is problematic, in general the empirical results indicate that the candidate set obtained provides a dramatic reduction in the search space. However, further empirical evaluations using additional ontologies to investigate this issue can certainly be performed.

As mentioned in section 9.2.1, it is possible to extend the framework to support subscriptions represented as boolean queries. However, in the current incremental query answering algorithm, only retrieval queries are supported. This is mainly due to the fact that the general technique to answering boolean queries is to extend the KB with a TBox axiom and then to check for consistency. In this case, extending the approach will require further research.

Other interesting extensions of the approach include lifting the additional restrictions on the form of the query. The restriction which dis-allows transitive roles in the query is imposed as it allows the query impact techniques to be used. With additional research, it should be possible to lift this restriction, however the topic of query answering in general for OWL DL in the presence of transitive roles is still a relatively open issue.

Lifting the syntactic restriction on the query is additionally a very interesting direction for future work. This restriction is imposed as it allows the isolation of the propagation of labels during the tableau algorithm; this essentially allows the correctness of the concept guide technique to be shown. Lifting this restriction will require additional investigation.

9.2.3 Evaluating Belief-Base Semi-Revision

While comprehensive performance evaluations have been preformed for the incremental reasoning algorithms developed in Chapters 5 & 6, such an evaluation was not performed for the revision approach presented in Chapter 7.2. This was primarily because the main mechanism for implementing the semi-revision operator follows from previously developed algorithms on finding justifications for entailments in OWL KBs, which has been extensively evaluated in literature. However, as noted in Chapter 7.2, the incremental consistency checking technique can be directly applied to the approach for finding all justifications, and it is expected that similar performance improvements as demonstrated in Chapter 5 will be observed. Given this, additional future work includes implementing such extensions and performing additional empirical evaluations.

9.2.4 Information Extraction

In this dissertation, the content creation problem has not been addressed; that is, it has been assumed that published information is encoded in OWL. As demonstrated in Chapter 8.3, in some domains this may not be the case, as information is encoded in other formats. To this end, in Chapter 8.3 simple converters and extractors have been utilized for converting non-OWL publications to the necessary representation when performing simulations using historical news items. While natural language processing and information extraction is out of scope of this dissertation, it certainly is an interesting avenue for future work. This can involve both the research of novel extraction techniques for the syndication framework and the application of existing extraction techniques which can be applied to textual information that is then encoded in domain ontologies (e.g., see [6, 39]).

9.3 Summary

In summary, in this dissertation I have shown that it is possible to provide an expressive syndication framework that uses an OWL/DL-based approach for matching newly published information with registered subscriptions. The key results of this dissertation are the formalization of the syndication framework and the development of algorithms for incremental description logic reasoning and resolving inconsistencies encountered due to new publications. Given this, it has been shown that the syndication framework is practical for use in real-world domains and publications frequencies. Future work involves extending the syndication framework and developed incremental reasoning algorithms, and performing additional empirical evaluations.

Appendix A

Proofs

A.1 Proofs for Chapter 5

A.1.1 Lemma 1: Correctness of Modified Tableau Algorithm

Proof It is first noted that soundness and completeness of the tableau algorithm are independent of the expansion rule ordering [78]; therefore, they trivially hold. Next consider termination. We first present the following properties of the assumptions of the lemma and the $\mathcal{SHOIQ}$ tableau algorithm:

1. By assumption, assertions in the ABox are not transformed into TBox assertions involving nominals.
2. K is assumed to be a $\mathcal{SHIQ}$ or $\mathcal{SHOQ}$ KB. This in conjunction with property 1, implies that if the TBox includes nominals, then there will not exist inverses, and if there exists inverse roles, then there will not exist nominals in the TBox. This in turn implies that the NN-rule will never be applied and is not necessary [78].

It is a direct consequence of these properties that the modified tableau algorithm terminates; this follows as the first three conditions in the termination proof shown in Lemma 6 of [78] are sufficient to show termination of the algorithm, as the fourth property is un-necessary because the NN-rule is never applied. The three conditions are omitted here as they are identical to those in [78]. □

A.1.2 Theorem 1: Completeness of $\mathcal{SHOIQ}$ Axiom Tracing

Proof We must show the following claim: if e occurred due to α, then $e \in \psi(\alpha)$. This will be shown by inductively considering the application of expansion rules. For the base, consider the initial completion graph which corresponds to the ABox; observe that this completion graph and the ψ function constructed from the initialized τ function (from Algorithm 1) satisfy this claim. The only non-trivial cases regarding this initial condition are 1) adding root nodes to the completion graph and 2) the dependence of root node edges on multiple role assertions. The first case is trivially handled as the root nodes in $\mathcal{V}$ correspond to the named individuals $\mathbf{I}_\mathsf{A}$ and not an ABox assertion. The second case is clearly handled by the special condition in the definition of τ for edges between root nodes.

For the inductive step, we show that if a given completion graph G and its corresponding functions τ, ψ satisfy the claim, then after applying some expansion rule to G the resulting τ, ψ functions satisfy the claim as well. This can be shown by considering the application of a tableau expansion rules based on the form of a concept C:

- $\sqcap$-rule: The only events that occur are the addition of C_1, C_2 to $\mathcal{L}(x)$, both of which are clearly due to the existence of x and the addition of $C_1 \sqcap C_2$ to $\mathcal{L}(x)$. These are precisely the events added to τ. By induction $\tau((C_1 \sqcap C_2), x)$ and $\tau(x, \mathcal{V})$ are complete, which implies the claim still holds after the application of the expansion rule.
- $\sqcup$-rule: This condition follows in a similar manner as the $\sqcap$-rule.
- $\exists$-rule: The only event that occurs is the addition of a new node y, edge $\langle x, y\rangle$, role name S to $\mathcal{L}(\langle x, y\rangle)$, and concept name C to $\mathcal{L}(y)$; these are precisely the events added to τ. All of these events are clearly dependent on the existence of x and the label $\exists R.C$ in $\mathcal{L}(x)$, both of which are taken into account when constructing τ. Lastly, by induction $\tau((\exists R.C), x)$ and $\tau(x, \mathcal{V})$ are complete, implying the claim still holds.
- $\forall$-rule: The only events that occur are the addition of C to $\mathcal{L}(y)$, and clearly this event is only dependent on the existence of the nodes x and y, concept name $\forall S.C$ in $\mathcal{L}(x)$, as well as the edge (to or from y) and edge label satisfying the S-neighbor

relation; these events and dependencies are precisely those taken into account when constructing τ. By induction, τ is complete for these events, implying the claim still holds after the application of the expansion rule.

- $\forall_+$-rule: This condition follows in a similar manner as the $\forall$-rule.
- *choose*-rule: This condition follows in a similar manner as the $\forall$-rule.
- $\geqslant$-rule: This condition follows in a similar manner to the $\exists$-rule, however an additional event occurs which adds the inequality relation between y_i, y_j. Clearly this event is handled as well.
- $\leqslant$-rule: It suffices to show that the claim holds after the application of the merge function. First, note that clearly the application of this rule is dependent on the existence of more than n S-neighbors, implying that it is dependent on the existence of these node, edges, and edge labels; additionally, the rule application is dependent on the existence of the node x and concept name $\leqslant nS.C$ in $\mathcal{L}(x)$. Note that all of these dependencies are included in the definition of M. We now consider the subcases of the merge function:
  - 1a) The only events that occur are the creation of an edge $\langle z, x\rangle$ and the addition of all role names from $\mathcal{L}(\langle z, y\rangle)$. This is dependent on satisfaction of the conditions for applying the rule itself, as well as the existence of $\langle z, y\rangle$ and its labels. The first of these dependencies is captured by M, while the latter of these conditions is subsumed by T. This in conjunction with the completeness of τ before the application of this rule, implies that the claim still holds.
  - 1b) This follows in a similar manner as 1a, however only edge labels not in $\mathcal{L}(\langle z, x\rangle)$ will be added. This is taken into account during the construction of τ.
  - 1c) This follows in a similar manner as 1b.
  - 1d) The only events that occur in this condition are the removal of the edge

$\langle z, y\rangle$ and its labels; these are precisely the events added to τ. Additionally, all of these events have the same dependencies as 1a. Therefore, the claim still holds after this condition.

  - 2a–d) These cases follow in a similar manner as cases 1a–d.
  - 3) The only events which occur are the addition of concept names from $\mathcal{L}(y)$ to $\mathcal{L}(x)$ if they do not already occur in $\mathcal{L}(x)$. These are precisely the events added to τ. This is dependent on satisfaction of the conditions for applying the rule itself, as well as the existence of the node y and its labels. As in case 1a, these dependencies are captured by M and T. This in conjunction with the completeness of τ before the application of this rule, implies that the claim still holds.
  - 4) This can be shown in a similar manner as case 3.
  - 5) We must show that the claim still holds after the prune function. Consider the first condition of the prune function; the only events which occur are the removal of edges $\langle y, z\rangle$ and their labels. These are precisely the events added to τ. This is clearly dependent on satisfaction of the conditions for applying the rule itself, as well as the existence of the edges and their labels. These dependencies are captured by M and T. This in conjunction with the completeness of the τ function before the application of this rule, implies that the claim still holds. The second condition of the prune function can be shown in a similar manner.

- $\mathcal{O}$-rule: The only difference with this rule and the $\leqslant$-rule is the cause of the rule application. In this case, the rule is dependent on the existence of the nodes x, y and the label o in both of their labels. The definition of M captures this. Therefore, this case follows in a similar manner as the $\leqslant$-rule.
- NN-rule: The only events that occur are the addition of the label $\leqslant mS.C$ to $\mathcal{L}(x)$

label, the creation of new nodes $y_1, ..., y_m$, the creation of edges $\langle x, y_i \rangle$ which are labeled with S, the addition of $\{C, o_i\}$ to each $\mathcal{L}(y_i)$, and the addition of $y_i \dot{\neq} y_j$; observe that these are precisely the events added to τ. All of these events are clearly dependent on the existence of x and y, the label $\leqslant nS.C$ in $\mathcal{L}(x)$, the label C in $\mathcal{L}(y)$, the edge $\langle x, y \rangle$, and the label R of $\mathcal{L}(\langle y, x \rangle)$ that satisfies the S-neighbor relation. These dependencies are used when constructing τ. By induction, τ is complete before the application of the rule, impling the claim still holds.

Given this, it has been shown that the claim holds. □

A.1.3 Theorem 2: Correctness of Algorithm 2

Proof First termination is shown. β is a finite set of ABox assertions; additionally, in the case of deletions, the set of events dependent on these removed assertions is finite. This implies that the for loop in Algorithm 2 (lines 4–40) will terminate. Therefore, it suffices to show that firing the necessary tableau expansion rules on the updated completion graph will terminate. Consider addition updates. Clearly the completion graph will be extended with additional structures corresponding to the new ABox assertions. Note that these actions results in a valid completion graph structure. This in conjunction with Lemma 1 implies that the application of the tableau expansion rules will terminate. Next, consider deletion updates. Prior to re-firing the tableau expansion rules, nodes, edges, and labels can be removed to the existing completion graph. Additionally, due to the roll-back of merges, structures can be added to the completion graph. However, these actions still result in a valid completion graph. This in conjunction with Lemma 1 implies that the application of the tableau expansion rules will terminate.

Next, correctness of the algorithm is shown. Consider additions first. First note the following properties of Algorithm 2 and the $\mathcal{SHOIQ}$ tableau algorithm:

1. By assumption K is consistent and G is a compete and clash-free completion graph for K.

2. Algorithm 2 adds the structure of β to G in an identical manner as when the completion graph corresponding to the initial ABox is constructed. Also when performing the tableau algorithm for $\mathsf{K} + \beta$ these structures will be added and the expansion rules will be applied to them.
3. It is a consequence of Lemma 1 that the rule applications for structures in a completion graph G' for $\mathsf{K}+\beta$ due to β can be delayed until after all expansion rule have been applied to all other labels.
4. Due to generating tableau expansion rules (i.e., $\exists$-rule and $\geqslant$-rule), nodes and edges could have been introduced to G that would not have been added when constructing a completion graph for $\mathsf{K} + \beta$. However, due to the completeness of the tableau algorithm [78], when re-applying the expansion rules to G after the structures from β have been added to it, the necessary nodes and edges will be merged, thereby eliminating these structures.

Now we address the two possible cases. Suppose that Algorithm 2 returns a completion graph G' that contains a clash, yet running the tableau algorithm from scratch for $\mathsf{K} + \beta$ returns a complete and clash-free completion graph G''. This implies that there was some sequence of expansion rule applications and choices at non-determinstic points that does not cause a clash when constructing G''. The assumption that G' contains a clash, implies that when the expansion rules are re-applied in Algorithm 2, the regular tableau algorithm must have backtracked and tried the sequence of rule applications that allowed the construction of G''. This in conjunction with the previous properties results in a contradiction.

Next, suppose that Algorithm 2 returns a complete and clash-free completion graph G', yet running the tableau algorithm from scratch for $\mathsf{K} + \beta$ returns a completion graph G'' with a clash. When applying the algorithm from scratch, all non-determinstric choices must have been explored. Properties 2–4 and the fact that G'' contains a clash, imply that after updating G with the structure from β and reapplying the expansion rules, a clash was observed and the algorithm back-jumped to a previous non-determinstic choice; this follows as after G is updated with β, it is a intermediate state (with possibly extra nodes and edges) of some completion graph built for $\mathsf{K} + \beta$ (all of which contain a clash). Therefore, the clash must also be observed in when updating G. This, the fact that G'' contains a clash, and the completeness of the tableau algorithm implies that there is a contradiction, as the exploration of all other non-determinstic choices must result in a clash as well since they did when constructing G''.

Next consider deletions. We consider two cases.

- G contains a clash. First note the following properties:

    1. There does not exist a complete clash-free completion graph for K; otherwise, G would not contain a clash
    2. By Theorem 1, if a previously observed clash is dependent on some $\alpha \in \beta$ then the structures causing the clash will be rolled-back.
    3. The previous property and the definition of the tracing function imply that if a structure (node, edge, or label) that was referenced during a merge operation is dependent on a removed assertion, then the entire merge operation will be rolled-back.
    4. During the rollback, structures corresponding to explicit ABox assertions are only removed if those ABox assertions are removed.
    5. The standard tableau algorithm is applied to G after the events dependent on β are rolled-back, and all previously invalidated non-determinstic choices are re-considered.

  Suppose that Algorithm 2 returns a completion graph that contains a clash, yet running the tableau algorithm from scratch for $\mathsf{K} - \beta$ returns a complete and clash-free completion graph G''. This implies that when back-jumping in Algorithm 2, there does not exist a sequence of expansion rule applications and choices at non-determinstic points that can construct a complete and clash-free completion graph. This in conjunction with properties 2–5 and the soundness and completeness of the tableau algorithm results in a contradiction, as there must exist some sequence of expansion rule applications and choices at non-determinstic points that constructs a complete and clash-free completion graph (precisely those that constructed G'').

Next, suppose that Algorithm 2 returns a complete and clash-free completion graph, yet running the tableau algorithm from scratch for $\mathsf{K} - \beta$ returns a completion graph with a clash. The previous properties imply that the reapplication of the expansion rules found some sequence of expansion rule applications that constructed a complete and clash-free completion graph; however, by properties 3–5, if when running the tableau algorithm from scratch all sequences of expansion rule applications and choices at non-determinstic points result in a clash, it must be the case when re-applying the expansion rules in Algorithm 2 these clashes must be observed. Therefore, we have arrived at a contradiction.

- G complete and clash-free. Suppose that Algorithm 2 returns a completion graph that contains a clash, yet running the tableau algorithm from scratch for $\mathsf{K} - \beta$ returns a complete clash-free completion graph. By the assumption that G complete and clash free, it must be the case that the previous KB was consistent. Further, by monotonicity of $\mathcal{SHIQ}$ & $\mathcal{SHOQ}$ and soundness and completeness of the tableau algorithm it must be that $\mathsf{K} - \beta$ is consistent. Given the previous properties and the soundness and completeness of the tableau algorithm, there is an immediate contradiction as all dependent structures must be retracted and all non-determistic choices are reconsidered when back-jumping. Next, suppose that Algorithm 2 returns a complete clash-free completion graph, yet running the tableau algorithm from scratch for $\mathsf{K} - \beta$ returns a completion graph with a clash. There is an immediate contradiction, as by soundness and completeness of the tableau algorithm and the mononticity of $\mathcal{SHIQ}$ & $\mathcal{SHOQ}$, running the tableau algorithm from scratch must results in a complete clash-free completion graph.

Thus, the theorem holds. □

A.2 Proofs for Chapter 6

A.2.1 Theorem 3: Conditions for $\mathcal{SHI}$ Concept Instantiation

Proof First note the following properties:

1. $Comp(\mathsf{K} + \{\neg C(a)\}) \neq \emptyset$, as $K \not\models C(a)$
2. $Comp(\mathsf{K} + \beta) \neq \emptyset$, as it is assumed $\mathsf{K} + \beta$ is consistent.
3. $Comp(\mathsf{K} + \{\neg C(a)\} + \beta) = \emptyset$, as $\mathsf{K} + \beta \models C(a)$
4. The additional expansion rule firings caused by adding β and $\neg C(a)$ to each $G \in Comp(\mathsf{K})$ must cause a clash in all possible completion graphs; this follows from Theorem 2 & property 3

Consider addition updates. This case will be shown by contradiction. Assume that 1) $\mathsf{K} \not\models C(a)$, $\mathsf{K} + \beta \not\models \bot$, $\mathsf{K} + \beta \models C(a)$ and 2) there does not exist $G \in Comp(\mathsf{K})$ s.t. the first condition of the theorem holds and 3) there does note exist the same node x with $D_1 \sqcup D_2 \in \mathcal{L}(x)$ in $\{\mathsf{G}_1, \mathsf{G}_2\} \subseteq Comp(\mathsf{K})$ s.t. the second condition of the theorem holds. Assumption 2 implies that every clash observed when updating each $\mathsf{G} \in Comp(\mathsf{K})$ with $\neg C(a)$ and β is not dependent on both $\neg C(a)$ and β (note that property 4 implies a clash must occur). This implies that if the clash is dependent on β, then a clash would be observed if only β were added to G; the same can be said for $\neg C(a)$. This in conjunction with properties 1–2 implies that each observed clash c must be dependent on $\neg C(a)$ or β (but not both), as well as some non-determinstic choice (i.e., some $D_1 \sqcup D_2 \in \mathcal{L}(x)$ for some $x \in \mathcal{V}$); otherwise, $\mathsf{K} \models C(a)$ or $\mathsf{K} + \beta \models \bot$ must hold, resulting in a contradiction. Observe that it clearly cannot be the case that every clash observed is dependent on β (or $\neg C(a)$), as this would imply $\mathsf{K} + \beta \models \bot$ (respectively $\mathsf{K} \models C(a)$). Next, it must be the case that for some set of clashes $\{c_1, ..., c_n\}$ observed, any c_i, $1 \leq i \leq n$, is dependent on the same non-deterministic choice as any c_j, $i \neq j$; this is a consequence of the previous properties, the assumption that $\mathsf{K} + \beta \models C(a)$ and the fact that each clash observed is dependent on a non-determinstic choice. It suffices to show that for some set of observed clashes $\{c_1, ..., c_n\}$ that are dependent on the same non-determinstic choice, there exists c_i, $1 \leq i \leq n$, that is dependent on β and there exists c_j, $i \neq j$, that is dependent on $\neg C(a)$. Assume this is not the case; that is, for each set of observed clashes $\{c_1, ..., c_n\}$ that are dependent on the same non-determinstic choice, it is the case that there does not exist some c_i, $1 \leq i \leq n$, that is dependent on β and c_j, $i \neq j$, that is dependent on $\neg C(a)$. If this is the case there is a contradiction as this would imply that either $\mathsf{K} + \beta \models \perp$ or $\mathsf{K} \models C(a)$. Next, note that the only cause of non-determinsm in the $\mathcal{SHI}$ tableau algorithm are disjunctions; thus the observed clashes c_i and c_j must be dependent on the same disjunction. This implies that there exists $\{\mathsf{G}_1, \mathsf{G}_2\} \subseteq Comp(\mathsf{K})$ with the same node x with $D_1 \sqcup D_2 \in \mathcal{L}(x)$ that c_i and c_j are dependent on, and c_i and c_j are dependent on β and $\neg C(a)$ respectively. Thus, we have arrived at a contradiction.

Next consider the case for deletions; due to the assumption that $\mathsf{K} \models C(a)$ and $\mathsf{K} - \alpha \not\models C(a)$, it is the case that $(\mathsf{K} - \alpha) + \alpha \models C(a)$. Therefore, the case for deletions is a consequence of the case for additions. $\square$

A.2.2 Theorem 4: Concept Guide Label Transfer

Proof First, note the following properties:

1. G is complete, implying no more expansion rules are applicable prior to the addition of $\neg C \in \mathcal{L}(x_a)$.
2. It is assumed that if $\forall R.D \in clos(\mathsf{K}) \cup clos(\neg C)$, then $\exists P.B \notin clos(\neg C)$ s.t. $\mathsf{Inv}(P) \sqsubseteq\!\!\!\!\sqsupseteq R$. This implies that a concept name will never be propagated across the $\mathsf{Inv}(P)$ edge added to the completion graph due to $\neg C \in \mathcal{L}(x_a)$.

This will be shown by induction on the application of expansion rules based on the structure of the concept $\neg C$. By assumption, $\neg C$ is in NNF; therefore, for ease of exposition in the remainder of this proof, we simply denote the concept by C. First, consider the base case, in which it is assumed the expansion rules are applied to $C \in \mathcal{L}(x_a)$. We address the expansion rules case-by-case:

- C is an atomic concept: this case is trivial, as none the tableau expansion rules are applicable to $C \in \mathcal{L}(x_a)$.
- C is of the form $\exists S.A$: the $\exists$-rule will add a S edge from x_a to a new individual y; observe that x_a will never be blocked as it is a root node. Property 2 implies a concept name cannot be propagated back up this edge and the tree-like model property of $\mathcal{SHI}$ implies that y has one unique root node (in this case x_a). Therefore, this case is irrelevant.
- C is of the form $\forall S.A$ and there does not exist some R s.t. $\mathsf{Trans}(R)$ and $R \mathrel{\underline{\circledast}} S$: the $\forall$-rule will add the concept name A to the label of all of x_a's S-neighbors that do not include A in their label; assume the existence of some S-neighbor z. By definition of the construction for the concept guide, if C is of the form $\forall S.A$, then there will exist an edge labeled with S between the nodes n and m s.t. $\forall S.A \in \mathcal{L}_{\mathcal{G}}(n), A \in \mathcal{L}_{\mathcal{G}}(m)$. Thus, a concept guide path will exist starting from x_a and concept guide node n and ending at z and m s.t. $A \in \mathcal{L}_{\mathcal{G}}(m)$.
- C is of the form $\forall S.A$ and there exists some R s.t. $\mathsf{Trans}(R)$ and $R \mathrel{\underline{\circledast}} S$: as in the previous case, the $\forall$-rule will add A to all of x_a's S-neighbors that do not include A in their label. This propagation follows in a similar manner as the previous case. Given the transitive role, the $\forall_+$-rule will add the concept name $\forall R.A$ to each R-neighbor of y for all R s.t. $\mathsf{Trans}(R)$ and $R \mathrel{\underline{\circledast}} S$. Observe that by definition, these edges will also exist in the concept guide; thus, this case holds in the same manner as the previous case. Note that a self looping edge labeled with each R is added to concept guide node m s.t. $A \in \mathcal{L}_{\mathcal{G}}(m)$.

- C is of the form $A_1 \sqcap A_2$: the $\sqcap$-rule will add all conjuncts to the label of x_a. By definition of the concept guide, if a conjunction is encountered, all conjuncts are added to the label of the node. Observe, the tableau expansion rules will not reach a new node due to application of the $\sqcap$-rule; thus this case holds.
- C is of the form $A_1 \sqcup A_2$: the $\sqcup$-rule will add one of the disjuncts to the label of x_a. By definition of the concept guide, if a disjunction is encountered, all disjuncts are added to the label of the node. As in the previous case, the tableau expansion rules will not reach a new node due to application of the $\sqcup$-rule, implying this case holds.

Observe that we can ignore the case in which $C \in \mathcal{L}(x_a)$ invalidates a blocking condition, allowing the propagation of labels due to the original structures in G; this follows from a) property 1, b) the completeness of dynamic blocking which implies that all labels have propagated back up the trees rooted at root nodes, and c) the fact that breaking the block will simply repeatedly replicate the isomorphic sub-tree spanning from the blocking node to the blocked node until the cycle is blocked again. In the inductive step below we address the case in which the breaking of the blocking condition generates the structure necessary to propagate some concept name added due to the query concept.

Next, consider the inductive step in which it is assumed that a concept D has just been added to $\mathcal{L}(y)$ and there is a concept guide path $x_a/n, ...z/m, y/o$, where x_a/n denotes the mapping of $x_a \in \mathcal{V}$ to $n \in \mathcal{V}_{\mathcal{G}}$. Again, consider the expansion rule applications; we only address the application of the $\forall$-rule or $\forall_+$-rule to a concept of the form $\forall S.A$, as the cases for the remaining expansion rules follow in a similar manner as these cases and their bases cases. We address both expansion rules together and consider the case in which the node y in G is blocked and not blocked separately:

- y not blocked: if there exists a S neighbor b s.t. $A \notin \mathcal{L}(b)$, the expansion rules would add a A label to the label of b; additionally if there exists some role P s.t. $\mathsf{Trans}(P)$ and $P \sqsubseteq\!\!\!\!\ast\ S$, then $\forall P.A$ will be added to $\mathcal{L}(b)$. Due to property 2 and the fact that y is not blocked, it suffices to only consider the case that b exists in G prior to adding $C \in \mathcal{L}(x_a)$. Next, consider two sub-cases:

- $\forall S.A$ was not propagated to $\mathcal{L}(y)$ due to the existence of an $\mathsf{Inv}(S)$-neighbor z with $\forall P.D \in \mathcal{L}(z)$, $\mathsf{Trans}(S)$ and $S \sqsubseteq\!\!\!\!\!* \; P$: by induction, it must be the case that $\forall S.A \in L_{\mathcal{G}}(o)$ (the concept guide node y is mapped to), and therefore this case follows similar to the base case.
- $\forall S.A$ was propagated to $\mathcal{L}(y)$ due to the existence of an $\mathsf{Inv}(S)$-neighbor z with $\forall P.A \in \mathcal{L}(z)$, $\mathsf{Trans}(S)$ and $S \sqsubseteq\!\!\!\!\!* \; P$: by induction, there must be a self-looping edge (o, o) s.t. $S \in L_{\mathcal{G}}((o, o))$ for all S s.t. $\mathsf{Trans}(S)$, $S \sqsubseteq\!\!\!\!\!* \; P$, and $A \in L_{\mathcal{G}}(o)$. Therefore, this edge can be selected for the next transition in the concept guide path and the hypothesis holds in a similar manner as the base case.

- y directly or indirectly blocked: first, consider the case where y directly blocked. The only non-trivial case is if adding $C \in \mathcal{L}(x_a)$ leads to the block being broken. Therefore, it could be that the necessary S-neighbor does not explicitly exist in G, due to the blocking condition. This is because the blocked node prevents the repeated generation of the isomorphic subtree spanning from the blocking node to the blocked node; if C were actually added to $\mathcal{L}(x_a)$, then the block would be invalidated leading to the addition of the necessary S-neighbor. Observe that the tree-like model property of $\mathcal{SHI}$ and the fact that root nodes are not blocked, imply that y has one unique predecessor z; assume $\mathcal{L}(\langle z, y\rangle) = \{R\}$ (by definition there can only be one such label). Next, let w be the blocking node s.t. $\mathcal{L}(y) = \mathcal{L}(w)$. Condition 2a of Definition 17 considers the edge $\langle z, w\rangle$ as a valid traversal in G due to the blocking condition; this implies that there also exists a concept guide path $x_a/n, ...z/m, w/o$ which satisfies the hypothesis. Further, because w is not blocked, the S neighbor must exist and the traversal can be made. Therefore, this case follows in a similar manner as the case where y is not blocked. This is sufficient, as the completeness of dynamic blocking implies concept names have been fully propagated prior to the addition of $C \in \mathcal{L}(x_a)$, and property 2 implies that concept names will not be propagated back up an edge added to G as a result of adding C to $\mathcal{L}(x_a)$; this in turn implies that only concept names $\forall S.D \in clos(C)$ added due to $C(a)$ can cause labels to be propagated through pre-existing cyclic structures in the completion graph and potentially back up to the root node. The case where y is indirectly blocked follows from the directly blocked case, as 1) y is indirectly blocked, implying that it has an ancestor that is directly blocked and 2) the cyclic structure is captured by considering the traversal from the predecessor of the directly blocked node to the blocking node.

□

A.2.3 Theorem 5: $\mathcal{SHI}$ Concept Instantiation Overestimate

Proof First consider addition updates; this case will be shown by contradiction. Assume that $\mathsf{K} \not\models C(a)$, $\mathsf{K} + \beta \models C(a)$ and that a does not satisfy either of the three conditions in Definition 18. It suffices to show that this implies that the conditions of Theorem 3 cannot be satisfied for a. Consider the first condition from Theorem 3, which states a clash will be dependent on both β and $\neg C(a)$ when applying any valid sequence of expansion rules. Consider the case where there is an immediate clash after adding the structure of β and $\neg C(a)$ to $\mathsf{G} \in Comp(\mathsf{K})$ prior to applying any expansion rules. Clearly, case 1 of Definition 18 trivially captures this case, as it must be the case that $a \in \mathbf{I}_\beta$ for this to occur. Next consider the case where expansion rules are applied prior to observing the clash. Note that a valid sequence of expansion rule applications can be obtained by first adding only β to $\mathsf{G} \in Comp(\mathsf{K})$ and then adding $\neg C(a)$ to the resulting complete and clash free completion graphs; this is a consequence of the fact that the $\mathcal{SHI}$ tableau algorithm does not impose an ordering when applying the expansion rules. This is sufficient to take into account condition one of Theorem 3 as the clash must be observed for any valid sequence of expansion rules. In general, for there to be a clash on node y that is dependent on β and $\neg C(a)$, there must be some concept name $l \in \mathcal{L}(y)$ that is dependent on β and some $l' \in \mathcal{L}(y)$ that is dependent on $\neg C(a)$. Further, the tree model property of $\mathcal{SHI}$ and definition of the tableau algorithm imply that for there to be a label of a non-root node y that is dependent on some ABox assertion α, then there must be some $l \in \mathcal{L}(x)$ s.t. l is also dependent on α and x is the unique root node of y. Case 2 of Definition 18 considers all individuals whose corresponding root nodes have a node label, or incoming/outgoing edge or edge label dependent on β when updating some G resulting in G'; this follows from the definition of $Dep(\beta, \mathsf{G}, \mathsf{G}')$. Thus, for there to be a clash observed that is dependent on both β and $\neg C(a)$, it must be the case that G' is clash-free and adding $\neg C(a)$ to G' causes a label to propagate to some $b \in Dep(\beta, \mathsf{G}, \mathsf{G}')$. Theorem 4 implies that if adding $\neg C$ to $\mathcal{L}(x_a)$ causes a label to propagate to some $b \in Dep(\beta, \mathsf{G}, \mathsf{G}')$, then a concept guide path will exist. Therefore, condition 1 of Theorem 3 cannot be satisfied, as it has been assumed the concept guide path does not exist between a and any $b \in Dep(\beta, \mathsf{G}, \mathsf{G}')$.

Next consider the second condition from Theorem 3. Clearly, for this condition to be satisfied, it must be the case that when adding β to some $\mathsf{G} \in Comp(\mathsf{K})$ a clash is caused that is dependent on some disjunction $D_1 \sqcup D_2 \in \mathcal{L}(y)$, and in some other $\mathsf{G}' \in Comp(\mathsf{K}) \setminus \mathsf{G}$, when adding $\neg C(a)$ to G' a clash is observed that is dependent on the same disjunction; similar to the previous condition, we can ignore $\neg C(a)$ when adding β to G (and β when adding $\neg C(a)$ to G'), as for the condition to be satisfied, the clash must be independent of $\neg C(a)$ (respectively β). We consider the various cases in which this can occur and show that they cannot be satisfied under our assumptions. First assume y is a non-root node and the unique root node x_b for y does not have a label dependent on $D_1 \sqcup D_2 \in \mathcal{L}(y)$ in any $\mathsf{G} \in Comp(\mathsf{K})$. This in conjunction with the tree model property of $\mathcal{SHI}$ implies that only the subtree rooted at x_b has structures dependent on the disjunction. In general, this implies that for there to be a clash dependent on this disjunction and any assertion α, then α must cause a label to be added to $\mathcal{L}(x_b)$. Since β causes a clash dependent on the disjunction, β must have caused a concept name to be added to $\mathcal{L}(x_b)$. Thus, $b \in Dep(\beta, \mathsf{G}, \mathsf{G}')$ where G' is the result of adding β to some G that results in a clash. In this case, for condition 2 of Theorem 3 to be satisfied, it must be that adding $\neg C(a)$ causes a concept name to be added to $\mathcal{L}(x_b)$ for some $b \in Dep(\beta, \mathsf{G}, \mathsf{G}')$. However, given our assumptions this cannot occur, as Theorem 4 implies there must exist a concept guide path involving x_a and x_b.

Next consider the case where y is a root node or y's unique root x_b does contain a node label dependent on the disjunction in all $\mathsf{G} \in Comp(\mathsf{K})$. Clearly, other root nodes and their non-root node descendants can also have node labels dependent on the disjunction. By definition, the third case of $CC(\mathsf{K}, C, \beta)$ considers all individuals whose corresponding root nodes have a label dependent on $D_1 \sqcup D_2 \in \mathcal{L}(y)$ in some $\mathsf{G}' \in Comp(\mathsf{K}) \setminus \mathsf{G}$. Again, for condition 2 of Theorem 3 to be satisfied, adding $\neg C(a)$ must cause a concept name to be added to one of these root nodes; however, Theorem 4 and the assumption that the concept guide path does not exist, implies there would be a contradiction if this case holds.

Lastly, consider the case in which y is a root node or y's unique root x_b does contain a node label dependent on the disjunction in some $\mathsf{G} \in Comp(\mathsf{K})$. The only non-trivial case is if in G x_b has a node label dependent on $D_1 \sqcup D_2 \in \mathcal{L}(y)$ and $\mathsf{G} \uplus \beta$ contains a clash dependent on the disjunction. Given $\mathsf{G}' \in Comp(\mathsf{K}) \setminus \mathsf{G}$, it could be the case that in G' x_b does not contain a node label dependent on $D_1 \sqcup D_2 \in \mathcal{L}(y)$, yet $\neg C(a)$ still causes a clash that is dependent on the disjunction when updating G' (i.e., a non-root descendant of x_b has a node label with the dependency). However, the third case of Definition 18 clearly considers x_b as it has a node label dependent on the disjunction in G; therefore, it can be shown that this case cannot not hold in the same manner as the previous cases. Note that the case in which x_b does have a node label dependent on the disjunction in G' easily follows from the previous cases. Thus, a contradiction has been shown as neither condition of Theorem 3 can be satisfied.

Next consider deletions. Again, it suffices to show that the previous assumptions imply that the conditions of Theorem 3 cannot be met. Due to the assumption that $\mathsf{K} \models C(a)$ and $\mathsf{K} - \alpha \not\models C(a)$, it is the case that $(\mathsf{K} - \alpha) + \alpha \models C(a)$. Therefore, the case for deletions is a consequence of the case for additions. □

A.2.4 Lemma 2: Tree Containment in Summary Completion Graph

Proof This will be shown by inductively considering the construction of a completion graph and summary completion graph for the same KB. As the base case, consider the initial completion graph and summary completion graph corresponding to the initial ABox, prior to the application of any expansion rules. Clearly, they are structurally the same, therefore the hypothesis is satisfied. Next, we address the inductive step. Assume that concept name C has just been added to $\mathcal{L}(y)$ (possibly x itself) s.t. y has unique root node x in completion graph G, and that G and S_{G} satisfy the hypothesis for each root node z. Let $T = \mathit{Tree}(x)$ in G, $G = \mathit{Graph}(x)$ in S_{G}, and $y' \in V_{S_{\mathsf{G}}}$ be a node such that $y \rightarrow_{T,G} y'$ where $\mathit{contain}(T, G)$ holds. By induction, it must be that $C \in \mathcal{L}_{S_{\mathsf{G}}}(y')$. We show that the hypothesis holds through the application of an expansion rule to $C \in \mathcal{L}(y)$ based on the form of C.

- $\sqcap$-rule: C_1, C_2 will be added $\mathcal{L}(y)$ if the labels do not exist. Consider two cases depending on whether y' is blocked at the time the $\sqcap$-rule is applied to it:
  - y' not blocked: because y' not blocked, the expansion rule will be applied when constructing S_{G}, maintaining the label relationship. No new edges are added, thus the containment relationship is maintained.
  - y' indirectly or directly blocked: consider the case in which y' is directly blocked. The expansion rule will still be applied; thus, the label relationship will be maintained. However, we must show that a valid mapping exists as blocked nodes are not in V_G. This follows easily as there must exist some predecessor w of y' and blocking node z s.t. $\mathcal{L}_{S_{\mathsf{G}}}(y') = \mathcal{L}_{S_{\mathsf{G}}}(z)$; by definition

of $\mathit{Graph}(x)$, there will be an edge from w to z with the same labels as $\langle w, y' \rangle$. This implies that y can be mapped to z rather than y' (i.e., $y \rightarrow_{T,G} z$). Because z is not blocked, the relationship will be maintained. Next, consider y' indirectly blocked; first, observe that if the application of the expansion rule to y' would cause the propagation of a label back up $\mathcal{L}_{S_{\mathsf{G}}}(x)$, then the correctness of dynamic blocking would be contradicted. Next, the indirect blocking of y' implies there exists an ancestor z s.t. for some $m \in N_T$, $m \rightarrow_{T,G} z$ and z is directly blocked by an ancestor n in S_{G}. This and the definition of $\mathit{Graph}(x)$ implies that the mapping of $m \rightarrow_{T,G} z$ must be replaced with $m \rightarrow_{T,G} n$, as z and y' are blocked. Given that the block represents a cycle, there must then be a mapping $y \rightarrow_{T,G} w$ s.t. w is not blocked and w is in a path between n and z. Since w not blocked, the relationship is maintained.

- $\sqcup$-rule: C_1 or C_2 will be added to $\mathcal{L}(y)$; clearly multiple new completion graphs will be created. It suffices to consider one such completion graph, G'', as the remainder follow in a similar manner. Observe that when constructing S_{G}, both labels of a disjunction are added to the node label. Therefore, this case follows in a similar manner as the previous case for the $\sqcap$-rule.
- $\exists$-rule: consider two cases depending on whether y' is blocked at the time the $\exists$-rule is applied to it:
  - y' not blocked: due to the label relationship, this rule may not be applicable to y', as a S-neighbor w labeled with C already exists in S_{G}; this case is trivial, as simply $z \rightarrow_{T,G} w$ if w is not blocked, and $z \rightarrow_{T,G} m$ if w blocked by node m. Additionally, consider the case where such a neighbor does not exist; clearly the rule will be applied to y' and again the relationship will hold.
  - y' indirectly or directly blocked: this can be shown in a similar manner as the $\sqcap$-rule; thus, there exists a valid mapping to z through the cycle introduced by the blocking condition and z will not be blocked. Thus, the necessary S

neighbor will exist and the containment relationship will hold.

- $\forall$-rule: all S-neighbors z of y in G s.t. $C \notin \mathcal{L}(z)$ will have C added to $\mathcal{L}(z)$. Consider two cases:
  - z a root node: by definition of $Tree(x)$ and the tree model property of $\mathcal{SHI}$, it must be the case that $z = x$, as y was added to the application of the $\exists$-rule to a label of x. By induction, there is mapping s.t. x in S_{G} is a S-neighbor of y'. Observe that y' cannot be indirectly blocked, as its neighbor is a root node, implying the expansion rule will be applied. Therefore, the label relationship is maintained. Note that if y' blocked, the mapping of y into a non-blocked node follows in a similar manner as the previous cases.
  - z a non-root node: consider two cases depending on whether y' is blocked at the time the $\forall$-rule is applied to it: 1) y' not blocked: by induction the S-neighbor exists in S_{G}, and since the node is not blocked the rule with be applied. Therefore, the relationship is maintained. 2) y' indirectly or directly blocked blocked: this follows in a similar manner as the previous case and the $\sqcap$-rule.
- $\forall_{+}$-rule: This condition follows in a similar manner as the $\forall$-rule.

Lastly, the general case is addressed in which $y \in \mathcal{V}$ is previously blocked, yet due to the addition of a concept name C to an ancestor z of y or simply y itself, the block is invalidated resulting in the application of expansion rules to y. By induction, prior to the blocking of y, there is some y' s.t. $y \rightarrow_{T,G} y'$. We show that a valid mapping will still exist. Consider two cases: 1) y' is not blocked: by induction $\mathcal{L}(y) \subseteq \mathcal{L}_{S_{\mathsf{G}}}(y')$ and the mapping still holds. Since y' not blocked, this case follows from the cases shown above. 2) y' directly or indirectly blocked: again by induction $\mathcal{L}(y) \subseteq \mathcal{L}_{S_{\mathsf{G}}}(y')$ prior to applying an expansion rule to y. We must show that there exists a mapping $y \rightarrow_{T,G} z$ s.t. z not blocked; again, this can be shown similar manner to the cases shown above. □

A.2.5 Lemma 3: Summary Completion Graph Update

Proof First consider termination; this can be shown as a simple extension to the termination proof of the $\mathcal{SHI}$ tableau algorithm [79]. Note that because the KB is expressed in $\mathcal{SHI}$ dynamic blocking is assumed. Let $m = \sharp(clos(\mathsf{K}) \cup clos(\beta))$; then termination is a consequence of the following properties:

1. The marking function θ ensures the expansion rules will be applied to a node label at most once.
2. The expansion rules never remove nodes from the summary completion graph or concept from node labels.
3. Successors are only generated for existential value restrictions. For any node, during each update each of these restrictions can only trigger the generation of at most one additional successor; this is a direct consequence of the marking function. Assume there have been n previous additions. Since $clos(\mathsf{K}) \cup clos(\beta)$ cannot contain more than m existential value restrictions, the out-degree of the tree is bound by $(n + 1)m$.
4. Node are labeled with a nonempty subsets of $clos(\mathsf{K}) \cup clos(\beta)$. If a path p is of least length 2^m, then there are 2 nodes x, y of p with $\mathcal{L}(x) = \mathcal{L}(y)$, and therefore blocking occurs. Since a path on which nodes are blocked cannot become longer, paths are of length at most 2^m.

Next consider the containment relationship after the update. It is a consequence of Theorem 2, that in order to construct all complete and clash-free completion graphs for $\mathsf{K} + \beta$, one can add the structure from β to each $\mathsf{G} \in Comp(\mathsf{K})$ and apply the necessary expansion rules. Therefore, it suffices to show that when updating each $\mathsf{G} \in Comp(\mathsf{K})$ with β the lemma holds after updating S_{G}; this can be shown as a straightforward extension of the proof for Lemma 2. Consider each update type for some $\alpha \in \beta$:

1. $\alpha = C(a)$, where a is an existing individual. By Lemma 2, $\mathcal{L}_{\mathsf{G}}(x_a) \subseteq \mathcal{L}_{S_{\mathsf{G}}}(x_a)$;

therefore, when adding C to $\mathcal{L}(x_a)$ and $\mathcal{L}_{S_{\mathsf{G}}}(x_a)$ the label relationship is clearly maintained. Consider the different expansion rules which could be applicable depending on the form of C:

- $\sqcap$-rule: the expansion rule will add C_1, C_2 to $\mathcal{L}(x_a)$ if that label does not exist. By definition, C_1, C_2 will be re-added to $\mathcal{L}_{S_{\mathsf{G}}}(x_a)$, and the expansion rules will be applied to both labels. Clearly, the root label relationship is maintained. Therefore, this case can inductively be shown in the same manner as in Lemma 2.
- $\sqcup$-rule: the expansion rule will select C_1 or C_2 to add to $\mathcal{L}(x_a)$; clearly, multiple new completion graphs will be created, as it is assumed that all complete and clash-free completion graphs are constructed. It suffices to consider one such completion graph, G'', as the remainder follow in a similar manner. Note that $C \in \{C_1, C_2\}$ will be added to $\mathcal{L}_{\mathsf{G}''}(x_a)$. When updating S_{G}, both labels are be re-added even if they exists. Therefore, this case follows in a similar manner as the $\sqcap$-rule.
- $\exists$-rule: the expansion rule will add a new edge and node and set the appropriate labels, if x does not have an S-neighbor labeled with C. Note that root nodes cannot be blocked. Observe that due to the marking condition, a new S-successor labeled with C will be added to S_{G} and the expansion rule will be applied to this node. Therefore, this case can be shown as in Lemma 2.
- $\forall$-rule: the concept name C will be added to the label of all of y's S-neighbors z. Consider the case that z is a root node. By definition, the tableau expansion rules do not added edges between root nodes; therefore, there must be the corresponding edge in S_{G}. This, in conjunction with the marking scheme implies that the label will be re-added to the corresponding S-neighbor, and the modified expansion rules will be applied to all labels of this node. Thus, this case can be shown in a similar manner as Lemma 2. Consider the case

that C is added to a non-root node $y \in \mathcal{V}$. Lemma 2 implies there is some y' in S_{G} where $y \rightarrow_{T,G} y'$ such that the containment is satisfied. Due to the marking scheme, the C concept name will be propagated to $\mathcal{L}(y')$. Therefore, this can be shown in a similar manner as Lemma 2.

- $\forall_{+}$-rule: This condition follows in a similar manner as the $\forall$-rule.

Lastly, consider the general case in which y is previously blocked, yet due to the addition of a concept name C to an ancestor z of y, the blocking condition is invalidated resulting in the application of expansion rules to y. Given the previous cases for the various expansion rules, this case can be shown in a similar to the same case in the proof for Lemma 2.

2. $\alpha = C(a)$, where a is a new individual. This follows from Lemma 2, as the only rule applications will be for C and the universal concept C_T.

3. $\alpha = R(a, b)$. There are 4 cases to consider:

   – a and b are new individuals. This follows from Lemma 2, as the only rule applications will be those applied to a, b due to the universal concept C_T and generated existential individuals with unique root node a or b.

   – a and b are existing individuals. The only rules applicable are either the $\forall$-rule or $\forall_{+}$-rule for some label in either $\mathcal{L}(a)$ or $\mathcal{L}(b)$; this is a direct consequence as G and S_{G} are complete prior to the update. By Lemma 2, $\mathcal{L}_{\mathsf{G}}(a) \subseteq \mathcal{L}_{S_{\mathsf{G}}}(a)$ and $\mathcal{L}_{\mathsf{G}}(b) \subseteq \mathcal{L}_{S_{\mathsf{G}}}(b)$. Therefore, this case can be shown in the same manner as shown for these expansion rules in condition 1.

   – a is a existing individual and b is an new individual. This can be shown in a similar manner as previous case.

   – a is a new individual and b is an existing individual. This can be shown in a similar manner as previous case.

4. $\alpha = (a = b)$. By definition of the approach, under ABox equality updates, the completion graph nodes will be merged; then, expansion rules are applied. Note that this occurs when updating S_{G}. By Lemma 2, $\mathcal{L}_{\mathsf{G}}(a) \subseteq \mathcal{L}_{S_{\mathsf{G}}}(a)$ and $\mathcal{L}_{\mathsf{G}}(b) \subseteq \mathcal{L}_{S_{\mathsf{G}}}(b)$. Thus, the relationship holds for the merged node; trivially the containment relationship holds as well. Then, expansion rules are applied to all node labels of a and b. Thus, this can be shown in a similar manner as the previous cases.

5. $\alpha = (a \neq b)$. Observe that by definition of the approach, the inequality relation will be updated. The same will occur for the summary completion graph. Therefore, this case trivially holds.

□

A.2.6 Lemma 4: Label Propagation of Summary Completion Graph

Proof Assume that x has concept name added to $\mathcal{L}(x)$ when updating some $\mathsf{G} \in \mathit{Comp}(\mathsf{K})$ with β, yet x is not reached when applying the modified expansion rules while updating S_{G}. Let G' be the completion graph resulting from the update of G with β such that a concept name is added to $\mathcal{L}(x)$ and S'_{G} be the result of updating S_{G} with β. We will show that there is a contradiction by considering the different update types for some $\alpha \in \beta$ and showing that the propagation must be observed in S_{G}:

1. $\alpha = C(a)$, where a is an existing individual: Lemma 3 implies that after the update $\mathcal{L}_{\mathsf{G}'}(x_a) \subseteq \mathcal{L}_{S'_{\mathsf{G}}}(x_a)$. By the tree model property of $\mathcal{SHI}$, it is the case that labels can only be propagated to some other root node by the application of the $\forall$-rule for some other root node. Further, by definition the tableau expansion rules do not add edges or edge labels between root nodes; so the corresponding edge must also exist in S_{G}. This, in conjunction with the marking scheme, implies that if the tableau expansion rules are applied to a root node neighbor of x_a when updating G, the same expansion rule application must occur when updating S_{G}. The same can

inductively be said for subsequently reached root nodes.

2. $\alpha = C(a)$, where a is a new individual. This follows trivially as the only rule applications will be for C and the universal concept C_T; this in conjunction with the tree model property for $\mathcal{SHI}$ implies the propagation holds.

3. $\alpha = R(a, b)$. There are 4 cases to consider:

   – a and b are new individuals. This follows trivially as the only rule applications will be those applied to x_a, x_b due to the universal concept C_T and generated existential individuals rooted at x_a or x_b.

   – a and b are existing individuals. Observe that the only rules applicable are either the $\forall$-rule or $\forall_+$-rule for some label in either $\mathcal{L}(y_a)$ or $\mathcal{L}(y_b)$; this is a direct consequence as G and S_G are complete prior to the update. By Lemma 3, $\mathcal{L}_{\mathsf{G}'}(y_a) \subseteq \mathcal{L}_{S'_\mathsf{G}}(y_a)$ and $\mathcal{L}_{\mathsf{G}'}(y_b) \subseteq \mathcal{L}_{S'_\mathsf{G}}(b)$. Therefore, this case can be shown in the same manner as case 1.

   – a is a existing individual and b is an new individual. This can be shown in a similar manner as previous case.

   – a is a new individual and b is an existing individual. This can be shown in a similar manner as previous case.

4. $\alpha = (a = b)$. Observe that by definition of the approach, under ABox equality updates, the completion graph nodes will be merged; then expansion rules are applied. Note that this occurs when updating S_G. By Lemma 3, $\mathcal{L}_{\mathsf{G}'}(y_a) \subseteq \mathcal{L}_{S'_\mathsf{G}}(y_a)$ and $\mathcal{L}_{\mathsf{G}'}(y_b) \subseteq \mathcal{L}_{S'_\mathsf{G}}(y_b)$. Therefore, this can be shown in a similar manner as the previous cases.

5. $\alpha = (a \neq b)$. Observe that by definition of the approach, the inequality relation will be updated. The same will occur for the summary completion graph. Therefore, this can be shown in a similar manner as the previous cases.

□

A.2.7 Lemma 5: Concept Guide Paths in Summary Completion Graph

Proof First, observe that the edges and edge labels between root nodes in the summary completion graph will be the same as those in any $\mathsf{G} \in \mathit{Comp}(\mathsf{K})$; this is a direct consequence of the construction of the summary completion graph and the fact that the expansion rules never add edges or edge labels between root nodes. Therefore, it suffices to show that the graph structure rooted at root nodes in any completion graph is contained in the graph structure rooted at the root node in the summary completion graph. To show this, a simplification of the graph structure introduced in Definition 20 is used. Define a *basic root graph* to composed of a set of nodes V, edges E, and a labeling function L for the nodes and edges; given root node x and completion graph G, the basic root graph, denoted $\mathit{Graph}_B(x)$, is inductively defined as follows:

1. $x \in V$, and $L(x) = \mathcal{L}(x)$
2. if $y \in V$ and y has R-neighbor z s.t. z a non-root node in G that is not blocked, then $z \in V$, $\langle y, z\rangle \in E$, $L(z) = \mathcal{L}(z)$ and $L(\langle y, z\rangle) = L(\langle y, z\rangle) \cup \{R\}$
3. if $w, y \in V$ s.t. z a non-root node, y is the predecessor or z, y has R-neighbor z, and w blocks z in G, then $\langle y, w\rangle \in E$ and $L(\langle y, w\rangle) = L(\langle y, w\rangle) \cup \{R\}$

Define a basic root graph G_B for root node x_a to be contained in root graph G for x_a if $\mathit{contain}(G_B, G)$. Given basic root graph G_B and root graph G, denote by $y \rightarrow_{G_B,G} z$ the mapping of node $y \in V_{G_B}$ into $z \in V_G$ s.t. the containment relationship is satisfied. It suffices to show that given a completion graph $\mathsf{G} \in \mathit{Comp}(\mathsf{K})$ and summary completion graph S_{G} for K, for each root node $x \in \mathcal{V}$ and corresponding root node x' in S_{G}, $\mathit{contain}(G_B, G)$ holds for $G_B = \mathit{Graph}_B(x)$, $G = \mathit{Graph}(x')$. This will be shown by inductively considering the application of expansion rules to a completion graph and summary completion graph for the same KB. As the base case, consider the initial completion graph and summary completion graph corresponding to the initial ABox, prior to the application of any expansion rules. Clearly, they are structurally the same, therefore the hypothesis is satisfied.

Now we address the inductive step. Assume that a completion graph G and summary completion graph S_{G} satisfy the hypothesis for root node x, and concept name C has just been added to $\mathcal{L}(y)$, s.t. y is either a non-root descendent of x or x itself. We show that the hypothesis holds through the application of an expansion rule to $C \in \mathcal{L}(y)$ based on the form of C. Let $y' \in V_{S_{\mathsf{G}}}$ be a node such that $y \rightarrow_{G_B,G} y'$.

- $\sqcap$-rule: first, consider the case in which y' is directly blocked; this implies there exists some blocking node z such that $\mathcal{L}(z) = \mathcal{L}(y')$ and z not blocked. Therefore, $C_1 \sqcap C_2 \in \mathcal{L}(z)$, as $C_1 \sqcap C_2 \in \mathcal{L}(y')$ and $C_1 \sqcap C_2 \in \mathcal{L}(y)$. Let w be the predecessor of y'. By definition of $Graph(x)$, there will be an edge from w to z with the same labels as $\langle w, y\rangle$. This implies that y is mapped to z rather than y' (i.e., $y \rightarrow_{G_B,G} z$). Because z is not blocked, then the $\sqcap$-rule can be applied to $C_1 \sqcap C_2 \in \mathcal{L}(z)$, resulting in $\{C_1, C_2\} \subseteq \mathcal{L}(z)$. The case in which y' indirectly blocked follows similar to the case just addressed and the $\sqcap$-rule case from the proof for Lemma 2. Additionally, the case in which y' is not blocked follows trivially, as the expansion rules will apply to the node.

  Lastly, consider the case that a node $n \in \mathcal{V}$ is blocked after the application of this rule. Therefore, a new edge $\langle m, w\rangle$ is added to $Graph_B(x)$ s.t. m is the predecessor of n and w is the blocking node. By definition of blocking, $\mathcal{L}(w) = \mathcal{L}(n)$. By induction, there exists some p s.t. $n \rightarrow_{G_B,G} p$; assume p not blocked after the rule application. Clearly, $\mathcal{L}(w) \subseteq \mathcal{L}(p)$; therefore, the cycle in G must eventually be observed in S_{G} implying this case holds. This follows as the expansion rules will apply to p and thus the necessary the sequence of neighbors $z_1, ..., z_n$ will exist that satisfy the hypothesis until the block occurs in S_{G}. Then, the edge from the predecessor of the blocked node to the blocking node will exist in S_{G}, which will satisfy the graph containment relationship. The case where p is blocked is trivial, as the same edge is added $Graph(x')$. The case where p is indirectly blocked follows from the indirectly blocked case in the regular inductive step for the $\sqcap$-rule discussed above and the two cases just considered where p not blocked and blocked.

- $\sqcup$-rule: by definition of the construction of the summary completion graph, when a disjunction is encountered, all disjuncts are added to the node label; therefore, this case follows in a similar manner as the previous case for the $\sqcap$-rule.
- $\exists$-rule: assume the rule applies to y and creates a new node z. Consider two cases; 1) y' not blocked: this case trivially holds as the same edge would be added to $Graph(x)$. 2) y' directly blocked: this implies there exists some blocking node m such that $\mathcal{L}(m) = \mathcal{L}(y')$ and m not blocked; further, because m is not blocked, it must either have a S-neighbor labeled with C or the $\exists$-rule can be applied to the $\exists S.C$ creating the neighbor. Therefore, this case follows in the same manner as the $\sqcap$-rule. The case in which y' is indirectly blocked follows similar to this case and the $\exists$-rule case from the proof for Lemma 2. Lastly, the condition where some node in G is blocked after the application of this rule follows in a similar manner as in the $\sqcap$-rule.
- $\forall$-rule: consider the following cases which suffice to show the hypothesis holds:
  - z a root node: by definition of $Graph_B(x)$, it must be the case that $z = x$, as y was added due to the application of the $\exists$-rule to a label for x. By induction, it must be the case that there exists a mapping s.t. x in S_{G} is a S-neighbor of y'. Observe that y' cannot be indirectly blocked, as its neighbor is a root node, imply the expansion rule will be applied. Therefore, the label relationship is maintained. Note that if y' blocked, the mapping of y into a non-blocked node follows in a similar manner as this case and the $\forall$-rule case from the proof for Lemma 2. Again, the condition where some node is blocked after the application of this rule follows in a similar manner as in the $\sqcap$-rule.
  - z a non-root node: we now consider two cases:
    - y' not blocked: by induction the S-neighbor exists in S_{G}, and since the

node is not blocked, the rule with be applied. Therefore, the relationship is maintained.

  * y' directly blocked: as in the $\sqcap$-rule, this implies there exists some blocking node w such that $\mathcal{L}(w) = \mathcal{L}(y')$ and w not blocked; further, by induction, w must have the necessary S-neighbor. Select one such S-neighbor m. Clearly, the root graph containment relationship is satisfied, due to the definition of *neighbor* and the fact that m is labeled with C as w is not blocked. The case in which y' indirectly blocked follows similar to this case and the $\forall$-rule case from the proof for Lemma 2.

  Again, the condition where some node in G is blocked after the application of this rule follows in a similar manner as in the $\sqcap$-rule.

- $\forall_+$-rule: This condition follows in a similar manner as the $\forall$-rule.

Lastly, consider the general case in which $y \in \mathcal{V}$ is previously blocked, yet due to the addition of a concept name C to an ancestor z of y or simply y itself, the block is invalidated resulting in the application of expansion rules to y. It is a consequence of this case in the proof for Lemma 2 and the previously addressed expansion rules that the relationship still holds. □

A.2.8 Lemma 6: Dependencies in Summary Completion Graph

Proof Let $y \in \mathcal{V}$ s.t. $D_1 \sqcup D_2 \in \mathcal{L}(y)$ and x be the unique root node for y (possible y itself). First consider $Disj_{\mathsf{G}}((D_1 \sqcup D_2, y)) \subseteq Disj_{S_{\mathsf{G}}}((D_1 \sqcup D_2, z))$ for $T = Tree(x)$ in G and $G = Graph(x)$ in S_{G}. Assume there exists some $y \rightarrow_{T,G} z$, s.t. $Disj_{\mathsf{G}}((D_1 \sqcup D_2, y)) \nsubseteq Disj_{S_{\mathsf{G}}}((D_1 \sqcup D_2, z))$. This will be shown by contradiction. Observe that Lemma 2, the tree-model property of $\mathcal{SHI}$, and the fact that the tableau expansion rules do not add edges or edge labels between root nodes imply that it suffices to inductively show the rule applications that can occur in G as a result of $D_1 \sqcup D_2 \in \mathcal{L}(y)$ are captured by the modified disjunction dependencies of the nodes in S_{G} for each $y \rightarrow_{T,G} z$. This shows there is a contradiction as same propagations up or down the tree in G must be possible in S_{G}, implying the dependency will by captured.

The base case is simply the initial application of the expansion rule to the disjunct selected in G. Clearly, Lemma 2 implies that the disjunct selected will be in $\mathcal{L}_{S_{\mathsf{G}}}(z)$ and thus the disjunction dependency easily holds. We now inductively consider the application of expansion rules to some concept name in the node label $\mathcal{L}(y)$ for some node $y \in \mathcal{V}$ and show that the dependencies hold through the expansion rule applications for each mapping of $y \rightarrow_{T,G} z$, implying there is a contradiction.

- $\sqcap$-rule: The expansion rule could have added C_1, C_2 to $\mathcal{L}(y)$. By definition, the dependency in S_{G} simply includes both concept names. Due to $y \rightarrow_{T,G} z$, it must be the case that $\mathcal{L}(y) \subseteq \mathcal{L}_{S_{\mathsf{G}}}(z)$, implying that both concept names are in $\mathcal{L}_{S_{\mathsf{G}}}(z)$ as well. Thus, the dependency relation holds.
- $\sqcup$-rule: one of C_1, C_2 are added to $\mathcal{L}(y)$. As in the previous case, by definition of S_{G}, $\{C_1, C_2\} \subseteq \mathcal{L}_{S_{\mathsf{G}}}(z)$. Thus the dependency relation holds.
- $\exists$-rule: The expansion rule will add a node w labeled with C and edge $\langle y, w\rangle$ labeled with S. Clearly, the C label is dependent on the disjunction. By Lemma 2, $\exists S.C \in \mathcal{L}_{S_{\mathsf{G}}}(z)$; further, the S-neighbor m in $Graph(x)$ must exist, s.t. x is the unique root of z. By definition all S-neighbors with C in their label are in the dependency, and the definition of the disjunction dependency considers the mapping $w \rightarrow_{T,G} m$ through cycles introduced due to blocking in S_G. Therefore, this case holds.
- $\forall$-rule and $\forall_+$-rule: the concept name C will be added to all S-neighbors $w \in \mathcal{V}$ s.t. $C \notin \mathcal{L}(w)$; each of these label additions are obviously in the dependency in G. Further, if there exists some R s.t. $\mathsf{Trans}(R)$ and $R \sqsubseteq\!\!\!\!\ast\; S$, then $\forall R.C$ will be propagated to all R-neighbors, implying these concept names could be in the dependency as well. Lastly, if C (or $\forall R.C$) is propagated across an edge added due to $\exists S.D \in \mathcal{L}(y)$ that is dependent on the disjunction, then C (respectively $\forall R.C$)

is also dependent on the disjunction. Lemma 2 implies $\mathcal{L}(y) \subseteq \mathcal{L}_{S_{\mathsf{G}}}(z)$ and the S-successors m of z must exist in $Graph(x)$, s.t. x is the unique root of z. By Definition 23, all S-neighbors with C in their label are in the dependency, and the dependency considers the mapping $w \rightarrow_{T,G} m$ through cycles introduced due to blocking. Further, condition 3a of Definition 23 clearly accounts for the case in which a concept is propagated across an edge added as a result of $\exists S.C \in \mathcal{L}(y)$ (as all labels of the S-neighbors of y are in the dependency). Observe that this additionally covers the case where some concept of the form $\forall R.D$ is propagated to a R neighbor due to the existence of a transitive role R s.t. $R \sqsubseteq S$; further, conditions 4b, 4c, 5b and 5c clearly account for the case in which the propagation is through a cycle introduced by a blocked node. Collectively, this implies that the dependencies for the newly added labels will be captured.

□

A.2.9 Lemma 7: Clashes in Summary Completion Graph

Proof Assume that there is a clash $c = (y, \neg C, C)$ caused when updating some $\mathsf{G} \in Comp(\mathsf{K})$ with β due to the addition of $\neg C$ to $\mathcal{L}(y)$, yet either the expansion rules do not reach z or $\{\neg C, C\} \not\subseteq \mathcal{L}(z)$ from all z s.t. y can be mapped into z satisfying the containment relationship. A contradiction will be shown by considering the different update types for some $\alpha \in \beta$ and showing that the expansion rules will reach some z and the clash will be observed.

1. $\alpha = C(a)$, where a is an existing individual. It suffices to inductively consider the different expansion rules which will be applicable depending on the form of C and show that the expansion rules will reach some z and the clash will be observed. As the base case, consider the initial addition of the concept name C to $\mathcal{L}(x_a)$. Lemma 3 implies that $\mathcal{L}(x_a) \subseteq \mathcal{L}_{S_{\mathsf{G}}}(x_a)$. Therefore, if a clash is observed when

adding C to $\mathcal{L}(x_a)$, it will be observed when added C to $\mathcal{L}_{S_{\mathsf{G}}}(x_a)$. Because nodes are not merged, there is a unique mapping for root nodes; thus this case holds. Next consider the inductive step; assume that the hypothesis holds for all label additions thus far. Further, assume that for some node y in G with unique root node x (possibly y itself), an expansion rule is now applicable to concept name C in $\mathcal{L}(y)$. Let $T' = Tree(x)$ and $G' = Graph(x)$ at this point, and z be a node in S_{G} where $y \rightarrow_{T',G'} z$ s.t. the containment is satisfied. We now consider the different expansion rules applicable based on the form of C:

- $\sqcap$-rule: the expansion rule will add C_1, C_2 to $\mathcal{L}(y)$ if that label does not exist. Consider two cases:
  - z not blocked: By definition, C_1, C_2 will be (re)added to $\mathcal{L}_{S_{\mathsf{G}}}(z)$; therefore, if the clash is observed in $\mathcal{L}(y)$, clearly it will be observed in $\mathcal{L}(z)$.
  - z directly or indirectly blocked: consider the case where z directly blocked. By definition of S_G, z is no longer a valid mapping, as it is not in $V_{S_{\mathsf{G}}}$; however, as shown in the proof for Lemma 3, there must exist a blocking node n s.t. $\mathcal{L}(n) = \mathcal{L}(z)$. Further, there must be some predecessor p of z. By definition, y must be mapped to n rather than z (as z blocked), resulting in the containment relationship; therefore, assume that y is mapped to n rather than z. Next observe that due to the tree model property of $\mathcal{SHI}$, for the rules to be applied to z, they must have been applied to n, as n is an ancestor of z. This implies the clash will be observed when the expansion rules are applied to n, as $\mathcal{L}(n) = \mathcal{L}(z)$. A similar case can be made when z indirectly blocked, as there must be some ancestor of z that is directly blocked.
- $\sqcup$-rule: the expansion rule will select C_1 or C_2 to add to $\mathcal{L}(y)$; clearly, multiple new completion graphs will be created. It suffices to consider one such

completion graph, G'', as the remainder follow in a similar manner. Note that $C \in \{C_1, C_2\}$ will be added to $\mathcal{L}_{\mathsf{G}''}(y)$. Observe that by definition of the summary completion graph, both disjuncts are added to a node label if a disjunction is encountered. Therefore, this case can be shown in a similar manner as the $\sqcap$-rule.

- $\exists$-rule: if y does not have a S-neighbor labeled with C, then the expansion rule will add a new node w and edge $\langle y, w\rangle$ labeled with C and S respectively; additionally, the universal concept C_T will be added to $\mathcal{L}(w)$. Observe that no atomic clashes can occur directly as a result of this action. However, we must show that the expansion rules will subsequently be applied to a valid mapping of w into S_{G}. Consider the case that z not blocked. By Lemma 3, $\exists S.C \in \mathcal{L}_{S_{\mathsf{G}}}(z)$; further, by definition of the marking scheme, a new S-neighbor m will be added to the summary completion graph. Thus, w can trivially be mapped to m and this case case holds. The cases in which z directly or indirectly blocked can be shown similar to this case and the blocked cases for the $\sqcap$-rule.
- $\forall$-rule: in G the label C will be added to all S-neighbors w of y. Consider the following two cases:
  - z not blocked: Lemma 3 implies that the corresponding S-neighbors m of z also exist in S_{G} s.t. $\mathcal{L}(w) \subseteq \mathcal{L}(m)$ for each $w \rightarrow_{T,G} m$. Further, the marking function implies that the label will be re-added to m. Therefore, if the clash is observed in $\mathcal{L}(w)$, clearly it will be observed in $\mathcal{L}(m)$, as $\mathcal{L}(w) \subseteq \mathcal{L}(m)$. Lastly, note that due to blocking in S_G, z itself could be mapped to w; however, this case trivially follows as $\mathcal{L}(w) \subseteq \mathcal{L}_{S_{\mathsf{G}}}(z)$ and the expansion rules are applied to $z \in \mathcal{V}_{S_{\mathsf{G}}}$.
  - z directly or indirectly: this case follows in a similar manner as the blocked case for the $\sqcap$-rule and the previously considered case in which

z is not blocked.

- $\forall_+$-rule: This condition follows in a similar manner as the $\forall$-rule.

Lastly, consider the case where the blocking condition of a non-root node y in G is invalidated due to the addition of some concept name C to $\mathcal{L}(w)$ for some $w \in \mathcal{V}$. Observe that for there to be a clash due to the application of expansion rules to the labels in $\mathcal{L}(y)$, the blocking node z must be reached via an expansion rule application in G; this follows from the tree-model property of $\mathcal{SHI}$ and completeness of dynamic blocking, which implies further applying the expansion rules to the concept names in $\mathcal{L}(y)$ will not propagate any node label back up the tree. Further, the previously considered inductive cases for the expansion rules imply that some m in S_{G}, s.t. $z \rightarrow_{T,G} m$, will be reached when updating S_{G}. Therefore, the previous cases, Lemma 3, and the usage of the marking scheme, imply that any clashes in G observed because of newly added structures due to the breaking of the blocking condition will also be observed in S_{G}.

2. $\alpha = C(a)$, where a is a new individual. This follows easily from the previous cases as the only rule applications will be for C and the universal concept C_T.

3. $\alpha = R(a, b)$. There are 4 cases to consider:

   – a and b are new individuals. Again, this follows easily from the previous cases as the only rule applications will be those applied to a, b due to the universal concept C_T and generated existential individuals rooted at a or b.

   – a and b are existing individuals. Observe that the only rules applicable are either the $\forall$-rule or $\forall_+$-rule for some label in either $\mathcal{L}(x_a)$ or $\mathcal{L}(x_b)$; this is a direct consequence as G and S_{G} are complete prior to the update. By Lemma 3, $\mathcal{L}_{\mathsf{G}}(x_a) \subseteq \mathcal{L}_{S_{\mathsf{G}}}(x_a)$ and $\mathcal{L}_{\mathsf{G}}(b) \subseteq \mathcal{L}_{S_{\mathsf{G}}}(x_b)$. Therefore, this case can be shown in the same manner as discussed above for these expansion rules.

– a is a existing individual and b is an new individual. This can be shown in a similar manner as previous case.

– a is a new individual and b is an existing individual. This can be shown in a similar manner as previous case.

4. $\alpha = (a = b)$. Under ABox equality updates, the completion graph nodes will be merged; then expansion rules are applied. Note that this occurs when updating S_{G}. By Lemma 3, $\mathcal{L}_{\mathsf{G}}(x_a) \subseteq \mathcal{L}_{S_{\mathsf{G}}}(x_a)$ and $\mathcal{L}_{\mathsf{G}}(x_b) \subseteq \mathcal{L}_{S_{\mathsf{G}}}(x_b)$. Thus, the relationship holds for the merged node. Therefore, if a clash is observed as a result of the merge in G, it will be observed in S_{G}. Then, expansion rules are applied to all node labels of x_a and x_b. Therefore, this case is a consequence of the previous cases.

5. $\alpha = (a \neq b)$. By definition of the approach, the inequality relation will be updated. The same will occur for the summary completion graph. Therefore, this case holds as well.

□

A.2.10 Theorem 6: Completeness of Summary Completion Graph

Proof Consider addition updates first. This will be shown by contradiction. Assume there exists some $a \in CC(\mathsf{K}, C, \beta)$ and $a \notin CC_{S_{\mathsf{G}}}(\mathsf{K}, C, \beta)$. It suffices to show that if a satisfies a condition in $CC(\mathsf{K}, C, \beta)$, then it must also satisfy the same condition in $CC_{S_{\mathsf{G}}}(\mathsf{K}, C, \beta)$. Clearly the first condition is trivial. Consider the second condition of Definition 18; it is a direct consequence of Lemma 4 that if $b \in Dep(\beta, \mathsf{G}, \mathsf{G}')$, then $b \in Dep(\beta, S_{\mathsf{G}})$. This in conjunction with Lemmas 3 & 5 implies that there is a contradiction if this case causes $a \in CC(\mathsf{K}, C, \beta)$, as the concept guide path must also exist in S''_{G}. Consider the third condition. It is a direct consequence of Lemma 6, Corollary 1, and the fact that the expansion rules do not remove nodes, edges or labels that the disjunction dependencies in the updated summary completion graph will subsume those for all possible mappings in some

$\mathsf{G} \in Comp(\mathsf{K})$ and $\mathsf{G} \uplus \beta$; this implies the completeness of using the summary completion graph for detecting the disjunction dependencies of clashes, as well as root nodes. Additionally, Lemma 7 implies that each clash $c = (y, \neg A, A)$ observed when updating some $\mathsf{G} \in Comp(\mathsf{K})$ with β will also be observed for some mapping of y into z in S_{G} as it is updated. This in conjunction with the previous case and Lemmas 3 & 5 imply there is a contradiction if the third condition of Definition 18 causes $a \in CC(\mathsf{K}, C, \beta)$, as the individuals considered in concept guide paths in the second condition will be located using the summary completion graph and the concept guide path must exist in the summary completion graph if it exists in a completion graph. Thus, a contradiction has been shown, as for each $a \in CC(\mathsf{K}, C, \beta)$ it must be the case that $a \in CC_{S_{\mathsf{G}}}(\mathsf{K}, C, \beta)$.

Consider deletions. By definition, the change events in S_{G} dependent on β are reverted and the necessary expansion rules are then applied to the summary completion graph. Therefore, it is a consequence of Theorem 1 that this results in the summary completion graph for $\mathsf{K} - \beta$. Thus, this case is a consequence of the case for additions. □

A.2.11 Theorem 7: $\mathcal{SHI}$ Binding Requirement

Proof This will be shown by contradiction. Consider additions first; assume that Q can be rolled-up into variable $x_i \in DVar(Q)$ resulting in concept C, $\mathsf{K} \not\models Q[x_1/a_1, ..., x_n/a_n]$, and $\mathsf{K} \oplus \beta \models Q[x_1/a_1, ..., x_n/a_n]$, but $a_i \notin CC(\mathsf{K}, C, \beta)$. Let C_P be the new concept obtained by substituting new concept names D_j into C for each distinguished variable x_j, $j \neq i$. Additionally let $\mathsf{K}' = \mathsf{K} + \{D_1(a_1)\} + ... + \{D_{i-1}(a_{i-1})\} + \{D_{i+1}(a_{i+1})\} + ... + \{D_n(a_n)\}$. It is shown in [140] that $\mathsf{K} \models Q[x_1/a_1, ..., x_n/a_n]$ if $\mathsf{K}' \models C_P(a_i)$; further, $\mathsf{K}' \models C_P(a_i)$ if $\mathsf{K}' \cup \{\neg C_P(a_i)\}$ is inconsistent [140]. Therefore, it is a consequence of Theorem 3 that one of two necessary conditions must be satisfied for a_i. A contradiction can be demonstrated by showing that if $a \in CC(\mathsf{K}', C, \beta)$ then $a \in CC(\mathsf{K}, C, \beta)$.

It suffices to show that if one of the three conditions of Definition 18 are satisfied for K', then they will be satisfied for K. Clearly, the first condition is trivial. Consider the second condition. Observe that each D_j is a new concept, which implies that no concepts will propagate in a completion graph due to the existence of D_j; hence D_j will only exist in $\mathcal{L}(x_{a_j})$. This implies that when updating $\mathsf{G} \in Comp(\mathsf{K})$ with $\{D_1(a_1)\} + ... + \{D_{i-1}(a_{i-1})\} + \{D_{i+1}(a_{i+1})\} + ... + \{D_n(a_n)\}$, no expansion rules will be applicable, nor will any clashes occur. Similarly when adding β to some $\mathsf{G} \in Comp(\mathsf{K}')$, no expansion rules will be applicable, nor will any clashes will occur as a result of some $D_j \in \mathcal{L}(x_{a_j})$. Thus, it suffices to show that $\{D_1(a_1)\} + ... + \{D_{i-1}(a_{i-1})\} + \{D_{i+1}(a_{i+1})\} + ... + \{D_n(a_n)\}$ does not affect the construction of the concept guide, nor the creation of a concept guide path. This easily follows from the previously discussed implications of D_j being a new concept and the fact that all D_j are atomic concepts, implying they will not cause any new nodes, edges, or labels when constructing the concept guide for $\neg C_P$.

Next consider the third condition. Note that the previous observations regarding each D_j being a new concept implies that they will not introduce any disjunctions during the tableau algorithm; thus, the disjunction dependencies for K will be the same as those for K'. Additionally, this implies that when updating $\mathsf{G} \in Comp(\mathsf{K} + \{D_1(a_1)\} + ... + \{D_{i-1}(a_{i-1})\} + \{D_{i+1}(a_{i+1})\} + ... + \{D_n(a_n)\})$ with β, if there is a clash, it cannot be dependent on any $D_j \in \mathcal{L}(x_{a_j})$. Collectively, this implies that if the third condition in Definition 18 is satisfied for some individual a for K', it will be also be satisfied for K. Thus we have arrived at a contradiction, as we have shown that if a condition from Definition 18 is satisfied for a in K', then it will be satisfied for a in K.

The case for deletions is a direct consequence of the case for additions. □

A.2.12 Theorem 8: Completeness of Query Impact

Proof First consider *Impact*; we must show that all variable bindings not included in $CC(\mathsf{K}, C, \beta)$ must be found. We will show this by contradiction. Assume that there is a new binding $\{a_1, ..., a_n\}$ and $\{a_1, ..., a_n\} \not\subseteq Impact(\mathsf{K}, Q, \beta)$. Theorem 7 implies that $a_i \in CC(\mathsf{K}, C, \beta)$. It is a direct consequence of Lemma A.2 in [136], that if two named individuals are in the interpretation of some simple role, then an edge will exist between those individuals. Also note that for the entailment to occur, each interpretation (and therefore each completion graph) must satisfy the entailment. This in conjunction with the connectedness of the query, restriction to simple roles in the query, and the fact that the maximal query depth is used to expand each $a_i \in CC(\mathsf{K}, C, \beta)$, implies that there is a contradiction.

Next consider *Map_Impact*; again, we must show that all candidate distinguished variables binding not included in $CC(\mathsf{K}, C, \beta)$ must be found. We will show this by contradiction. Assume there is a new binding $\{a_1, ..., a_n\}$ and $\{a_1, ..., a_n\} \nsubseteq Map\_Impact(\mathsf{K}, Q, \beta)$. Again, it is a direct consequence of Theorem 7 that $a_i \in CC(\mathsf{K}, C, \beta)$. Note that tree blocking is assumed, s.t. blocking is delayed to take into account the longest path in the query. Next, observe that not adding $\top \sqsubseteq \neg D \sqcup D$ for each concept atom D in the query does impact the completeness of the mapping technique as query concepts are not taken into account during the mapping. Thus, there is a contradiction of the completeness of Theorem 3 in [114], as it is shown that if the query is entailed, then it must be syntactically mappable in all complete and clash free completion graphs for the KB. □

A.2.13 Theorem 9: Correctness of Algorithm 4

Proof Consider additions. First, observe that by montonicity of $\mathcal{SHI}$, under additions only new bindings can be found; therefore, we do not have to recheck previous bindings. Thus, soundness is trivial as it follows from the soundness of the query answering algorithm provided in [82]. Termination follows from the following properties:

- Lemma 3 implies that the update of the summary completion graph terminates
- the summary root graph and concept guide are of finite size; further, evaluating regular path expressions over graph databases can be decided in polynomial time [103]. This implies the concept guide search will terminate.
- standard query answering are assumed, which reduce to query answering to consis-

tency checking; this is known to terminate [74, 80, 78]

In order to show completeness, we must show that all new answer tuples will be found. Theorem 6 implies the completeness of using the summary root graph to determine $CC(\mathsf{K}, C, \beta)$; note that this set of individuals is located in lines 2–3. Therefore, it is a consequence of Theorems 3, 7, and 8 that all named individuals in new answer sets will be found. Thus, completeness is a consequence of the fact that the algorithm iterates over all combinations of candidate variable bindings and uses standard query answering techniques to check for entailment [82].

Next consider deletions. Termination can be shown in a similar manner as the case for additions. By montonicity of $\mathcal{SHI}$, under deletions only previous answer tuples can be invalidated (no new bindings can be found); therefore, only previous bindings need to be rechecked, implying that completeness trivially follows. In order to show soundness we must show that all tuples in the answer set after the deletion are in fact entailed. Algorithm 4 admits previous binding under two conditions; first is when none of the bound individuals of a previous binding are in $CC(\mathsf{K}, C, \beta)$. It is a consequence of Theorems 3, 6, 7, & 8 that if a previous binding is invalidated, then one of the bound individuals will be in the set of query concept candidates. Therefore, this case clearly holds. The second case only admits previous tuples if they are entailed; therefore soundness follows from the soundness of standard query answering algorithms [82]. □

A.3 Proofs for Chapter 7

A.3.1 Lemma 8: $\mathcal{SHOIN}$ Kernel Operator

Proof By definition, $Just(\mathsf{K}, \alpha)$ contains all justifications for $K \models \alpha$. It suffices to show that criteria 1, 2, and 3 from Definition 1 are satisfied. Let $j \in Just(\mathsf{K}, \alpha)$. By definition, $j \subseteq \mathsf{K}$, thereby satisfying criteria 1. Further, by definition $j \models \alpha$; thus criteria 2 is satisfied. Finally, since justifications are minimal, we obtain condition 3, namely $j' \subset j$, $j' \not\models \alpha$.

Thus, *Just*(K, α) is a kernel operator for $\mathcal{SHOIN}$ KBs. $\square$

Bibliography

[1] System s project homepage:
http://domino.research.ibm.com/comm/research_projects.nsf/pages/esps.index.html.

[2] W3c semantic web activity. In *http://www.w3.org/2001/sw/*.

[3] D. Abadi, D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik. Aurora: A new model and architecture for data stream management. *VLDB Journal*, 12, 2003.

[4] Serge Abiteboul and Gösta Grahne. Update semantics for incomplete databases. In Alain Pirotte and Yannis Vassiliou, editors, *VLDB'85, Proceedings of 11th International Conference on Very Large Data Bases, August 21-23, 1985, Stockholm, Sweden*, pages 1–12. Morgan Kaufmann, 1985.

[5] Marcos Kawazoe Aguilera, Robert E. Strom, Daniel C. Sturman, Mark Astley, and Tushar Deepak Chandra. Matching events in a content-based subscription system. In *Symposium on Principles of Distributed Computing*, pages 53–61, 1999.

[6] Massimiliano Albanese and V.S. Subrahmanian. T-rex: A domain-independent system for automated cultural information extraction. In *Proceedings of the First International Conference on Computational Cultural Dynamics (ICCCD 2007)*, 2007.

[7] Carlos E. Alchourrón, Peter Gärdenfors, and David Makinson. On the logic of theory change: Partial meet contraction and revision functions. *Journal of Symbolic Logic*, 50(2):510–530, 1985.

[8] Mehmet Altinel and Michael J. Franklin. Efficient filtering of XML documents for selective dissemination of information. In *The VLDB Journal*, pages 53–64, 2000.

[9] K. Apt and J. M. Pugin. Maintenance of stratified databases viewed as a belief revision system. In *PODS '87: Proceedings of the sixth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 136–145, 1987.

[10] C. Areces and M. de Rijke. From description to hybrid logic, and back. In *Advances in Modal Logic, Volume 3*. CSLI Publications, 2001.

[11] Ron Avnur and Joseph M. Hellerstein. Eddies: Continuously adaptive query processing. In *Proc. of the 2000 ACM SIGMOD Intl. Conf. on Management of Data*, pages 261–272, 2000.

[12] F. Baader, M. Buchheit, and B. Hollunder. Cardinality restrictions on concepts. *Artificial Intelligence*, 88(1–2):195–213, 1996.

[13] F. Baader, E. Franconi, B. Hollunder, B. Nebel, and H.J. Profitlich. An empirical analysis of optimization techniques for terminological representation systems or: Making KRIS get a move on. *Applied Artificial Intelligence. Special Issue on Knowledge Base Management*, 4:109–132, 1994.

[14] F. Baader and B. Hollunder. Embedding defaults into terminological representation systems. *J. Automated Reasoning*, 14:149–180, 1995.

[15] F. Baader, M. Milicic, C. Lutz, U. Sattler, and F. Wolter. Integrating description logics and action formalisms for reasoning about web services. LTCS-Report LTCS-05-02, Chair for Automata Theory, Institute for Theoretical Computer Science, Dresden University of Technology, Germany, 2005. See http://lat.inf.tu-dresden.de/research/reports.html.

[16] F. Baader and W. Nutt. Basic description logics. In Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors, *The Description Logic Handbook: Theory, Implementation, and Applications*, pages 43–95. Cambridge University Press, 2003.

[17] F. Baader and U. Sattler. An overview of tableau algorithms for description logics. *Studia Logica*, 69:5–40, 2001.

[18] Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[19] Franz Baader and Philipp Hanschke. A scheme for integrating concrete domains into concept languages. In *Twelfth International Conference on Artificial Intelligence*, pages 452–257, 1991.

[20] S. Babu and J. Widom. Continuous queries over data streams. In *SIGMOD Record*, 2001.

[21] Guruduth Banavar, Tushar Deepak Chandra, Bodhi Mukherjee, Jay Nagarajarao, Robert E. Strom, and Daniel C. Stur. An efficient multicast protocol for content-based publish-subscribe systems. In *ICDCS '99: Proceedings of the 19th IEEE International Conference on Distributed Computing Systems*, page 262, Washington, DC, USA, 1999. IEEE Computer Society.

[22] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web, scientific american, 284(5):34-43. 2001.

[23] Jose A. Blakeley, Per-Ake Larson, and Frank Wm Tompa. Efficiently updating materialized views. In *Proc. of SIGMOD '86: ACM SIGMOD International Conference on Management of Data*, pages 61–71, 1986.

[24] P. Bonatti, C. Lutz, and F. Wolter. Expressive non-monotonic description logics based on circumscription. In Patrick Doherty, John Mylopoulos, and Christopher

Welty, editors, *Proceedings of the Tenth International Conference on Principles of Knowledge Representation and Reasoning (KR'06)*, pages 400–410. AAAI Press, 2006.

[25] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. DL-Lite: Tractable description logics for ontologies. In *AAAI-05*, pages 602–607, 2005.

[26] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Daniele Nardi. Reasoning in expressive description logics. In Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, pages 1581–1634. Elsevier Science Publishers, 2001.

[27] D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, G. Seidman, M. Stonebraker, N. Tatbul, and S. Ldonik. Monitoring streams - a new class of data management applications. In *Proceedings of the International conference on Very large data bases (VLDB2002)*, 2002.

[28] A. Carzaniga, M. Rutherford, and A. Wolf. A routing scheme for content-based networking. In *Technical Report CU-CS-953-03, Department of Computer Science, University of Colorado, June 2003.*, 2003.

[29] A. Carzaniga and A. Wolf. Forwarding in a content-based network. In *In SIGCOMM '03, Karlsruhe, Germany, Aug.*, 2003.

[30] Chee Yong Chan, Pascal Felber, Minos N. Garofalakis, and Rajeev Rastogi. Efficient filtering of XML documents with XPath expressions. *The VLDB Journal*, 11:354–379, 2002.

[31] J. Chen, D. DeWitt, F. Tian, and Y. Wang. Niagaracq: A scalable continuous query system for the internet databases. In *Proc. of the 2000 ACM SIGMOD Intl. Conf. on Management of Data*, 2000.

[32] Paul Alexandru Chirita, Stratos Idreos, Manolis Koubarakis, and Wolfgang Nejdl. Publish/subscribe for rdf-based p2p networks. In *Proceedings of the 1st European Semantic Web Symposium. (2004)*, 2004.

[33] M. Cilia, C. Bornhvd, and A. P. Buchmann. Cream: An infrastructure for distributed, heterogeneous event-based applications. In *Proceedings of the International Conference on Cooperative Information Systems*, 2003.

[34] Arturo Crespo and Hector Garcia-Molina. Routing indices for peer-to-peer systems. In *ICDCS '02: Proceedings of the 22 nd International Conference on Distributed Computing Systems (ICDCS'02)*, page 23, Washington, DC, USA, 2002. IEEE Computer Society.

[35] Brickley D. and R.V. Guha. RDF Vocabulary Description Language 1.0: RDF Schema. http://www.w3.org/tr/rdf-schema/. February 2004.

[36] Johan de Kleer. An assumption-based TMS. *Artif. Intell.*, 28(2):127–162, 1986.

[37] M. Dean and G. Schreiber. OWL Web Ontology Language Reference W3C Recommendation. http://www.w3.org/tr/owl-ref/. February 2004.

[38] Y. Diao, S. Rizvi, and M. Franklin. Towards an internet-scale xml dissemination service. In *Proceedings of VLDB2004, August 2004.*, 2004.

[39] Yihong Ding and David W. Embley. Using data-extraction ontologies to foster automating semantic annotation. In *Proceedings of the 22nd International Conference on Data Engineering Workshops (ICDEW06)*, 2006.

[40] Guozhu Dong, Jianwen Su, and Rodney W. Topor. Nonrecursive incremental evaluation of datalog queries. *Annals of Mathematics and Artificial Intelligence*, 14(2-4):187–223, 1995.

[41] Guozhu Dong and Rodney W. Topor. Incremental evaluation of datalog queries. In *Proc. of the 4th Int. Conference on Database Theory*, pages 282–296, 1992.

[42] J. Doyle. A truth maintenance system. *Readings in nonmonotonic reasoning*, pages 259–279, 1987.

[43] Françoise Fabret, H. Arno Jacobsen, François Llirbat, João Pereira, Kenneth A. Ross, and Dennis Shasha. Filtering algorithms and implementation for very fast publish/subscribe systems. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 30(2):115–126, 2001.

[44] G. Flouris. On belief change and ontology evolution. In *Ph.D. Dissertation, University of Crete*, 2006.

[45] G. Flouris, D. Plexousakis, and G. Antoniou. On applying the agm theory to dls and owl. In *4th International Semantic Web Conference (ISWC 2005)*, 2005.

[46] G. Flouris, D. Plexousakis, and G. Antoniou. Updating description logic using the agm theory. In *7th International Symposium on Logical Formalizations of Commonsense Reasoning*, 2005.

[47] Giorgos Flouris, Dimitris Plexousakis, and Grigoris Antoniou. Generalizing the agm postulates: Preliminary results and applications. In *Proceedings of the 10th International Workshop on Non-Monotonic Reasoning (NMR 2004)*, Whistler, Canada, June 2004.

[48] Andre Furmann. Theory contraction through base contraction. *Journal of Philosophical Logic*, 20:175–203, 1991.

[49] Giuseppe De Giacomo, Maurizio Lenzerini, Antonella Poggi, and Riccardo Rosati. On the update of description logic ontologies at the instance level. In *Proceedings, The Twenty-First National Conference on Artificial Intelligence (AAAI 2006)*, 2006.

[50] Birte Glimm, Ian Horrocks, Carsten Lutz, and Uli Sattler. Conjunctive query answering for the description logic $\mathcal{SHIQ}$. In *Proc. of the 20th Int. Joint Conf. on Artificial Intelligence (IJCAI 2007)*, pages 399–404, 2007.

[51] Birte Glimm, Ian Horrocks, and Uli Sattler. Conjunctive query entailment for $\mathcal{SHOQ}$. In *Proc. of the 2007 Description Logic Workshop (DL 2007)*, volume 250 of *CEUR (`http://ceur-ws.org/`)*, 2007.

[52] Birte Glimm, Ian Horrocks, and Ulrike Sattler. Conjunctive query answering for description logics with transitive roles. In *Proc. of the Int. Description Logic Workshop (DL 2006)*, 2006.

[53] Birte Glimm, Ian Horrocks, and Ulrike Sattler. Conjunctive query answering for the description logic $\mathcal{SHOIQ}$. Technical report, University of Manchester, School of Computer Science, 2006.

[54] Jennifer Golbeck. *Computing and Applying Trust in Web-based Social Networks*. PhD thesis, University of Maryland, College Park, MD, USA, April 2005.

[55] Jennifer Golbeck and Christian Halaschek-Wiener. Trust-based revision for expressive web syndication. *MINDSWAP Technical Report TR-MS1293, University of Maryland, College Park.*

[56] Bernardo Cuenca Grau, Christian Halaschek-Wiener, and Yevgeny Kazakov. Historymatters: Incremental ontology reasoning using modules. In *Proceedings of the 6th International Semantic Web Conference*, 2007.

[57] J. Gregorio and B. de hra. The atom publishing protocol. In *IETF Internet Draft*, 2005.

[58] Y. Guo, Z. Pan, and J. Heflin. Lubm: A benchmark for owl knowledge base systems. *Journal of Web Semantics*, 3(2):158–182, 2005.

[59] Ashish Gupta, Inderpal Singh Mumick, and V. S. Subrahmanian. Maintaining views incrementally. In *SIGMOD '93: Proceedings of the 1993 ACM SIGMOD international conference on Management of data*, pages 157–166, 1993.

[60] Ashish Kumar Gupta and Dan Suciu. Stream processing of xpath queries with predicates. In *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 419–430, New York, NY, USA, 2003. ACM Press.

[61] V. Haarslev and R. Moller. Description logic systems with concrete domains: Applications for the semantic web. In *Int. Workshop on KR meets Databases, 2003.*, 2003.

[62] V. Haarslev and R. Möller. Incremental query answering for implementing document retrieval services. In *Proceedings of the International Workshop on Description Logics (DL-2003), Rome, Italy, September 5-7*, pages 85–94, 2003.

[63] Volker Haarslev, Ralf Moller, and Anni-Yasmin Turhan. Exploiting pseudo models for tbox and abox reasoning in expressive description logics. In *IJCAR '01: Proceedings of the First International Joint Conference on Automated Reasoning*, pages 61–75, London, UK, 2001. Springer-Verlag.

[64] Christian Halaschek-Weiner and James Hendler. Toward expressive syndication on the web. In *Proceedings of the 16th World Wide Web Conference*, 2007.

[65] Christian Halaschek-Wiener, Aditya Kalyanpur, and Bijan Parsia. Extending tableau tracing for abox updates. In *UMIACS Tech Report*, 2006. http://www.mindswap.org/papers/2006/aboxTracingTR2006.pdf.

[66] Sven Ove Hansson. New operators for theory change. *Theoria*, 55:114–133, 1989.

[67] Sven Ove Hansson. Belief base dynamics. In *PhD Theis, Uppsala University*. 1991.

[68] Sven Ove Hansson. Kernel contraction. *Journal of Symbolic Logic*, 59(3):845–859, 1994.

[69] Sven Ove Hansson. Semi-reivion. *Journal of Applied Non-Classical Logics*, 7(2), 1997.

[70] Sven Ove Hansson. A textbook on belief dynamics. Kluwer Academic Press, 1999.

[71] J. Harrison and S. Dietrich. Maintenance of materialized views in a deductive database: An update propagation approach. In *Workshop on Deductive Databases held in conjunction with the Joint International Conference and Symposium on Logic Programming (JICSLP)*, pages 56–65, 1992.

[72] I. Horrocks. *Optimising Tableaux Decision Procedures for Description Logics*. PhD thesis, University of Manchester, 1997.

[73] I. Horrocks. Implementation and optimisation techniques. In Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors, *The Description Logic Handbook: Theory, Implementation, and Applications*, pages 313–355. Cambridge University Press, 2003.

[74] I. Horrocks, U. Sattler, and S. Tobies. Practical reasoning for expressive description logics. In *Proc. of the 6th Int. Conference on Logic for Programming and Automated Reasoning (LPAR'99)*, number 1705 in Lecture Notes in Artificial Intelligence, pages 161–180. Springer-Verlag, 1999.

[75] I. Horrocks and S. Tobies. Reasoning with axioms: Theory and practice. In *Proc. of the 7th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 2000)*, pages 285–296, 2000.

[76] Ian Horrocks and Ulrike Sattler. A description logic with transitive and inverse roles and role hierarchies. *J. of Logic and Computation*, 9(3):385–410, 1999.

[77] Ian Horrocks and Ulrike Sattler. Ontology reasoning in the $\mathcal{SHOQ}$(D) description logic. In *Proc. of the 17th Int. Joint Conf. on Artificial Intelligence (IJCAI 2001)*, pages 199–204. Morgan Kaufmann, Los Altos, 2001.

[78] Ian Horrocks and Ulrike Sattler. A tableaux decision procedure for SHOIQ. In *Proc. of the 19th Int. Joint Conf. on Artificial Intelligence (IJCAI 2005)*. Morgan Kaufman, 2005.

[79] Ian Horrocks, Ulrike Sattler, and Stephan Tobies. A description logic with transitive and converse roles, role hierarchies and qualifying number restrictions. LTCS-Report 99-08, LuFg Theoretical Computer Science, RWTH Aachen, Germany, 1999.

[80] Ian Horrocks, Ulrike Sattler, and Stephan Tobies. Reasoning with individuals for the description logic $\mathcal{SHIQ}$. In David McAllester, editor, *Proc. of the 17th Int. Conf. on Automated Deduction (CADE 2000)*, volume 1831 of *Lecture Notes in Computer Science*, pages 482–496. Springer, 2000.

[81] Ian Horrocks and Sergio Tessaris. A conjunctive query language for description logic aboxes. In *National conference on artificial intelligence (AAAI 2000)*, pages 399–404, 2000.

[82] Ian Horrocks and Sergio Tessaris. Querying the semantic web: a formal approach. In *Proc. of the 13th Int. Semantic Web Conf. (ISWC 2002)*, pages 177–191, 2002.

[83] Z. Ives, A. Levy, and D. Weld. Efficient evaluation of regular path expressions on streaming xml data. In *Univ. of Washington Tech. Rep. CSE000502.*, 2000.

[84] Zachary Ives, Alon Levy, Daniel Weld, Daniela Florescu, and Marc Friedman. Adaptive query processing for internet applications. *IEEE Data Engineering Bulletin*, 23(2):19–26, 2000.

[85] Aditya Kalyanpur. Debugging and repair of owl ontologies. In *Ph.D. Dissertation, University of Maryland, College Park*, 2006. http://www.mindswap.org/papers/2006/AdityaThesis-DebuggingOWL.pdf.

[86] Aditya Kalyanpur, Bijan Parsia, Evren Sirin, and Bernardo Cuenca Grau. Repairing unsatisfiable concepts in owl ontologies. In *Proceedings of the European Semantic Web Conference (ESWC2006)*, pages 170–184, 2006.

[87] Aditya Kalyanpur, Bijan Parsia, Evren Sirin, and James Hendler. Debugging unsatisfiable classes in owl ontologies. In *Journal of Web Semantics - Special Issue of the Semantic Web Track of WWW2005*, 2005.

[88] Hirofumi Katsuno and Alberto Mendelzon. On the difference between updating a knowledge base and revising it. In *International Conference of Principles of Knowledge Representation and Reasoning(KR)*, 1991.

[89] V. Kuchenhoff. On the efficient computation of the difference betwen consecutive database states. In *Proc. of 2nd Int. Conf. on Deductive and Object-Oriented Databases*, pages 478–502, 1991.

[90] Ugur Kuter and Jennifer Golbeck. Sunny: A new algorithm for trust inference in social networks using probabilistic confidence models. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 2007.

[91] Laks V. S. Lakshmanan and Sailaja Parthasarathy. On efficient matching of streaming XML documents and queries. In *Extending Database Technology*, pages 142–160, 2002.

[92] Alon Y. Levy and Marie-Christine Rousset. CARIN: A representation language combining horn rules and description logics. In *European Conference on Artificial Intelligence*, pages 323–327, 1996.

[93] Isaac Levy. Subjunctives, dispositions, and chances. *Synthese*, 34:423–455, 1977.

[94] L. Li and I. Horrocks. A software framework for matchmaking based on semantic web technology. In *Proceedings of the Twelfth International World Wide Web Conference (WWW 2003), 2003.*, 2003.

[95] D. Libby. Rss 0.91 spec, revision 3. In *Netscape Comm.*, 1999.

[96] H. Liu, C. Lutz, M. Milicic, and F. Wolter. Updating description logic aboxes. In *International Conference of Principles of Knowledge Representation and Reasoning(KR)*, 2006.

[97] L. Liu, C. Pu, and W. Tang. Continual queries for internet scale event-driven information delivery. *IEEE Trans. on Knowledge and Data Engineering*, 11, 1999.

[98] Zhen Liu, Srinivasan Parthasarthy, Anand Ranganathan, and Hao Yang. Scalable event matching for overlapping subscriptions in pub/sub systems. In *Proceedings of International Conference on Distributed Event-Based Systems (DEBS07)*, pages 250–261, 2007.

[99] Sam Madden, Mehul A. Shah, Joseph M. Hellerstein, and Vijayshankar Raman. Continuously adaptive continuous queries over streams. In *SIGMOD Conference*, 2002.

[100] F. Manola and E. Miller. RDF Primer W3C Recommendation. http://www.w3.org/tr/rdf-primer/. February 2004.

[101] David McAllester. Truth maintenance. In Reid Smith and Tom Mitchell, editors, *Proceedings of the Eighth National Conference on Artificial Intelligence*, volume 2, pages 1109–1116. AAAI Press, 1990.

[102] John McCarthy and Patrick J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence 4*, pages 463–502. Edinburgh University Press, 1969. reprinted in McC90.

[103] A. O. Mendelzon and P. T. Wood. Finding regular simple paths in graph databases. In *VLDB '89: Proceedings of the 15th international conference on Very large data bases*, pages 185–193, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.

[104] Boris Motik and Ulrike Sattler. A Comparison of Reasoning Techniques for Querying Large Description Logic ABoxes. In Miki Hermann and Andrei Voronkov, editors, *Proc. of the 13th Int. Conference on Logic for Programming Artificial Intelligence and Reasoning (LPAR 2006)*, volume 4246 of *LNCS*, pages 227–241, Phnom Penh, Cambodia, November 13–17 2006. Springer.

[105] Boris Motik, Ulrike Sattler, and Rudi Studer. Query Answering for OWL-DL with rules. *Journal of Web Semantics: Science, Services and Agents on the World Wide Web*, 3(1):41–60, 2005.

[106] Bernhard Nebel. A knowledge level analysis of belief revision. In R. Brachman, H. J. Levesque, and R. Reiter, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the 1st International Conference*, pages 301–311, San Mateo, 1989. Morgan Kaufmann.

[107] Bernhard Nebel. Syntax-based approaches to belief revision. In P. Gärdenfors, editor, *Belief Revision*, volume 29, pages 52–88. Cambridge University Press, Cambridge, UK, 1992.

[108] Bernhard Nebel. Base revision operations and schemes: Semantics, representation, and complexity. In *Cohn A.G. (eds.), Proc. 11th European Conference on Artificial Intelligence.*, 1994.

[109] Bernhard Nebel. How hard is it to revise a belief base? In Didier Dubois and Henri Prade, editors, *Handbook of Defeasible Reasoning and Uncertainty Management Systems, Volume 3: Belief Change*, pages 77–145. Kluwer Academic Publishers, Dordrecht, 1998.

[110] Wolfgang Nejdl, Martin Wolpers, Wolf Siberski, Christoph Schmitz, Mario Schlosser, Ingo Brunkhorst, and Alexander Loser. Super-peer-based routing and clustering strategies for rdf-based peer-to-peer networks. In *Proceedings of the 12th International World Wide Web Conference*, 2003.

[111] B. Nguyen, S. Abiteboul, G. Cobena, and M. Preda. Monitoring xml data on the web. In *Proc. of the 2001 ACM SIGMOD Intl. Conf. on Management of Data*, 2001.

[112] M. Nottingham and R. Sayre. The atom syndication format. In *IETF Internet Draft*, 2005.

[113] B. Oki, M. Pfluegl, and Dale Skeen. The information bus: An architecture for extensible distributed systems. In *In Proc. 14th SOSP*, 1993.

[114] Maria Magdalena Ortiz de la Fuente, Diego Calvanese, and Thomas Eiter. Data complexity of answering unions of conjunctive queries in shiq. In *Proc. of the 2006 Description Logic Workshop (DL 2006)*. CEUR Electronic Workshop Proceedings, http://ceur-ws.org/, 2006.

[115] Massimo Paolucci, Takahiro Kawamura, Terry R. Payne, and Katia Sycara. Semantic matching of web services capabilities. In *The First International Semantic Web Conference*, 2002.

[116] Bijan Parsia, Christian Halaschek-Wiener, and Evren Sirin. Towards incremental reasoning through updates in owl-dl. In *Reasoning on the Web - Workshop at 15th International World Wide Web Confence*, 2006.

[117] Bijan Parsia and Evren Sirin. Pellet: An owl dl reasoner. In *Third International Semantic Web Conference - Poster*, 2004.

[118] P.F. Patel-Schneider, P. Hayes, and I.Horrocks. Web ontology language OWL Abstract Syntax and Semantics. *W3C Recommendation*, 2004.

[119] Pavlos Peppas, Abhaya C. Nayak, Maurice Pagnucco, Norman Y. Foo, Rex Bing Hung Kwok, and Mikhail Prokopenko. Revision vs. update: Taking a closer look. In *Proceedings of the 12th European Conference on Artificial Intelligence*, pages 95–99, 1996.

[120] Milenko Petrovic, Ioana Burcea, and Hans-Arno Jacobsen. S-topss: Semantic toronto publish/subscribe system. In *VLDB '03: Proceedings of the 29th international conference on Very large data bases*, 2003.

[121] Milenko Petrovic, Haifeng Liu, and Hans-Arno Jacobsen. Cms-topss: Efficient dissemination of rss documents. In *VLDB '05: Proceedings of the 31st international conference on Very large data bases*, pages 1279–1282. VLDB Endowment, 2005.

[122] Eric Prud'hommeaux and Andy Seaborne (editors). Sparql query language for rdf. W3C Working Draft (21 July 2005) http://www.w3.org/TR/rdf-sparql-query/, 2005.

[123] R Reiter. A theory of diagnosis from first principles. *Artif. Intell.*, 32(1):57–95, 1987.

[124] R. Reiter. Knowledge in action. *MIT Press*, 2001.

[125] Marcio Moretto Ribeiro and Renata Wasserman. Base revision in description logics - preliminary results. In *Proc. of the Int. Workshop on Ontology Dynamics (IWOD 2007)*, 2007.

[126] Marcio Moretto Ribeiro and Renato Wasserman. First steps towards revising ontologies. In *Proceedings of 2nd Workshop on Ontologies and their Applications (WONTO'2006)*, 2006.

[127] Mathieu Roger, Ana Simonet, and Michel Simonet. Toward updates in description logics. In *International Workshop on Knowledge Representation meets Databases*, 2002.

[128] Richard B. Scherl and Hector J. Levesque. Knowledge, action, and the frame problem. *Artif. Intell.*, 144(1-2):1–39, 2003.

[129] S. Schlobach. Debugging and semantic clarification by pinpointing. In *Proceedings of the European Semantic Web Conference (ESWC2005)*, 2005.

[130] S. Schlobach. Diagnosing terminologies. In *Proceedings of National Conference on Artificial Intelligence (AAAI05)*, 2005.

[131] S. Schlobach and R. Cornet. Non-standard reasoning services for the debugging of description logic terminologies. In *Proceedings of IJCAI, 2003*, 2003.

[132] U. Schreier, H. Pirahesh, R. Agrawal, and C. Mohan. Alert: An architecture for transforming a passive dbms into an active dbms. In *Proceedings of the International conference on Very large data bases (VLDB1991)*, 1991.

[133] Andy Seaborne. RDQL - A Query Language for RDF. http://www.w3.org/submission/2004/subm-rdql-20040109/. January 2004.

[134] Luciano Serafini and Andrei Tamilin. Local tableaux for reasoning in distributed description logics. In *Proceedings of the International Workshop on Description Logics*, 2004.

[135] Luciano Serafini and Andrei Tamilin. Drago: Distributed reasoning architecture for the semantic web. In *Proceedings European Semantic Web Conference*, 2005.

[136] Evren Sirin. Combining description login reasoning with ai planning for composition of web services. In *Ph.D. Dissertation, University of Maryland, College Park*, 2006.

[137] Martin Staudt and Matthias Jarke. Incremental maintenance of externally materialized views. In *VLDB '96: Proceedings of the 22th International Conference on Very Large Data Bases*, pages 75–86, 1996.

[138] Michael Stonebraker. Implementation of integrity constraints and views by query modification. In *SIGMOD '75: Proc. of the 1975 ACM SIGMOD international conference on Management of data*, pages 65–78, New York, NY, USA, 1975.

[139] D. B. Terry, D. Goldberg, D. Nichols, and B. M Oki. Continuous queries over append-only databases. In *Proc. of the Intl. Conf. on Management of Data*, pages 321–330, 1992.

[140] Sergio Tessaris. *Questions and answers: reasoning and querying in Description Logic*. PhD thesis, University of Manchester, 2001.

[141] Stephan Tobies. The complexity of reasoning with cardinality restrictions and nominals in expressive description logics. *Journal of Artificial Intelligence Research*, 12:199–217, 2000.

[142] Dmitry Tsarkov and Ian Horrocks. Efficient reasoning with range and domain constraints. In *Proc. of the Int. Description Logic Workshop (DL 2004)*, pages 41–50, 2004.

[143] Michael Uschold, Peter Clark, Fred Dickey, Casey Fung, Sonia Smith, Stephen Uczekaj Michael Wilke, Sean Bechhofer, and Ian Horrocks. A semantic infosphere. In Dieter Fensel, Katia Sycara, and John Mylopoulos, editors, *Proc. of the 2003 International Semantic Web Conference (ISWC 2003)*, number 2870 in Lecture Notes in Computer Science, pages 882–896. Springer, 2003.

[144] Raphael Volz, Steffen Staab, and Boris Motik. Incrementally Maintaining Materializations of Ontologies Stored in Logic Databases. *Journal of Data Semantics II*, 3360:1–34, 2005.

[145] Jinling Wang, Beihong Jin, and Jing Li. An ontology-based publish/subscribe system. In *Middleware '04: Proceedings of the 5th ACM/IFIP/USENIX international conference on Middleware*, pages 232–253, New York, NY, USA, 2004.

[146] Taowei David Wang, Bijan Parsia, and James Hendler. A survey of the web ontology landscape. In *Proceedings of the International Semantic Web Conference (ISWC2006*, 2006.

[147] R. Wassermann. An algorithm for belief revision. In *Proceedings of the Seventh International Conference on Principles of Knowledge Representation and Reasoning (KR2000)*, 2000.

[148] J. Widom and R. Motwani. Query processing, resource management, and approximation in a data stream management system. In *Proceedings of Conference on Innovative Data Systems Research (CIDR2003)*, pages 245–256, 2003.

[149] M. Winslett. Updating logical databases. In *Updating Logical Databases*. Cambridge University Press, 1990.

[150] Tak W. Yan and Hector Garcia-Molina. The SIFT information dissemination system. *ACM Transactions on Database Systems*, 24(4):529–565, 1999.

[151] Eiko Yoneki and Jean Bacon. Distributed multicast grouping for publish/subscribe over mobile ad hoc networks. In *Wireless Communications and Networking Conference*, pages 2293–2299, 2005.