Semantic Web Foundations for Representing, Reasoning, and Traversing Contextualized Knowledge Graphs

A dissertation submitted in partial fulfilment of the requirements for the degree of Doctor of Philosophy

By

VINH THI KIM NGUYEN
B.E., Vietnam National University, 2007

2017
Wright State University

Amit Sheth, Ph.D.
Dissertation Director

Michael Raymer, Ph.D.
Director, Computer Science and Engineering Ph.D. Program

Barry Milligan, Ph.D.
Interim Dean of the Graduate School

Committee on
Final Examination

Amit Sheth, Ph.D.

Krishnaprasad Thirunarayan, Ph.D.

Olivier Bodenreider, Ph.D.

Kemafor Anyanwu, Ph.D.

Ramanathan V. Guha, Ph.D.
ABSTRACT


Semantic Web technologies such as RDF and OWL have become World Wide Web Consortium (W3C) standards for knowledge representation and reasoning. RDF triples about triples, or meta triples, form the basis for a contextualized knowledge graph. They represent the contextual information about individual triples such as the source, the occurring time or place, or the certainty.

However, an efficient RDF representation for such meta-knowledge of triples remains a major limitation of the RDF data model. The existing reification approach allows such meta-knowledge of RDF triples to be expressed in RDF by using four triples per reified triple. While reification is simple and intuitive, this approach does not have a formal foundation and is not commonly used in practice as described in the RDF Primer.

This dissertation presents the foundations for representing, querying, reasoning and traversing the contextualized knowledge graphs (CKG) using Semantic Web technologies.

A triple-based compact representation for CKGs. We propose a principled approach and construct RDF triples about triples by extending the current RDF data model with a new concept, called singleton property (SP), as a triple identifier. The SP representation needs two triples to the RDF datasets and can be queried with SPARQL.

A formal model-theoretic semantics for CKGs. We formalize the semantics of the singleton property and its relationships with the triple it represents. We extend the current RDF model-theoretic semantics to capture the semantics of the singleton properties and provide the interpretation at three levels: simple, RDF, and RDFS. It provides a single interpretation of the singleton property semantics across applications and systems.

A sound and complete inference mechanism for CKGs. Based on the semantics we propose, we develop a set of inference rules for validating and inferring new triples based on the SP syntax. We also derive different sets of context-based inference rules for provenance, time, and uncertainty.

A graph-based formalism for CKGs. We propose a formal contextualized graph model for the SP representation. We formalize the RDF triples as a mathematical graph by combining the model theory and the graph theory into the same RDF formal semantics. The unified semantics allows the RDF formal semantics to be leveraged in the graph-based algorithms.
# Contents

1 Introduction .......................................................... 1  
   1.1 Contextualized Knowledge Graphs .............................. 3  
   1.2 Motivating Examples ........................................... 5  
   1.3 Existing approaches ........................................... 7  
   1.4 Thesis Statement ................................................ 8  
   1.5 Our Approach .................................................... 9  
      1.5.1 Knowledge Representation ................................. 9  
      1.5.2 Reasoning .................................................. 13  
      1.5.3 Graph Traversal .......................................... 14  
   1.6 Our Contributions ............................................... 16  
      1.6.1 Foundational contributions .............................. 16  
      1.6.2 Implementation .......................................... 16  
   1.7 Outline .......................................................... 17  

I Foundations .................................................................. 19  

2 Knowledge Representation for Contextualized Knowledge Graphs ............................. 20  
   2.1 Singleton Property Concept ................................... 20  
      2.1.1 Singleton property as unique key for statement within a context .... 20  
      2.1.2 Asserting meta knowledge for triples .......................... 23  
      2.1.3 Enforcing the singleton-ness of property instances .............. 24  
   2.2 Model-Theoretic Semantics ....................................... 25  
   2.3 RDF 1.1 Singleton Property Model-Theoretic Semantics ............ 28  
   2.4 Related Work ...................................................... 29  

3 Reasoning for Contextualized Knowledge Graphs ........................................... 31  
   3.0.1 Introduction ..................................................... 31  
   3.1 Conceptual Modeling ............................................... 34
List of Figures

1.1 The n-ary approach for representing Example 3 (from Noy et al. 2006). ............ 8
1.2 Node-LabeledArc-Node diagram (NLAN) for Example 4. ................................. 15
1.3 A contextualized knowledge graph (labeled directed multigraph) for Table 1.9 ....... 15

4.1 Subgraphs G1 and G2 are disconnected in the Node-LabeledArc-Node diagram (NLAN). 50
4.2 Node-LabeledArc-Node diagram (NLAN) for the example 4 ............................. 52
4.3 Subgraphs G1 and G2 from Figure 4.1 are now connected in a CKG graph with labeled directed multigraph with triple nodes. ............................ 53
4.4 CKG graph with labeled directed multigraph with triple nodes for Example 4. ...... 54
4.5 A step by step construction of a CKG graph for the two triples T_3 and T_4. .......... 56

5.1 Number of triples in million of 3 categories contributing to the total number of triples. 71
5.2 Query performance in msec.: fixed values. ....................................................... 72
5.3 Query performance in msec.: 100 values. ......................................................... 73
5.4 Number of triple patterns within three queries Q1, Q2, and Q3 of query set B. ......... 73
5.5 Query performance in msec. of the set B. ......................................................... 74

6.1 Total number of triples (top), total number of singleton triples (middle), and run time (bottom) for each dataset: with vs. without reasoning. ......................... 79
6.2 Run time vs. number of triples (a) and run time vs. number of inferred triples (b) across datasets in two cases: with vs. without reasoning. ......................... 80

7.1 Average time taken per shortest distance within CS. ........................................ 88
7.2 Average time taken per shortest distance within SS. ........................................ 88
7.3 Average time taken per shortest distance within HR. ...................................... 89
7.4 The shortest resource path and its respective triple path from Andrew Card to Jacob Lew in Yago2S-SP. The resource path includes all the edges along the dashed line in the periphery of the figure, and the triple path includes all the triples where three nodes from a triple form a straight line. Traversing the resource path from start to end will find a sequence of five politicians (Andrew Card, Joshua Bolten, Rahm Emanuel, Pete Rouse, William M. Daley, Jacob Lew) in consecutive terms.
List of Tables

1.1 Sample queries for different types of contextual knowledge, each query example is assigned an identifier \((P_i, T_i, S_i, \text{and } C_i)\) for references ........................................ 4

1.2 The same fact has different provenance and temporal information associated with it .................................................. 6

1.3 Contextualized knowledge base for employee information \((sp: \text{subPropertyOf, and } sc: \text{subClassOf})\) .................................................. 6

1.4 Reified statements and their meta knowledge assertions for the same fact \text{BobDylan isMarriedTo SaraLownds} occuring in two documents from the example 1 (Table 1.2) .................................................. 7

1.5 The RDF representation of singleton properties and their meta knowledge assertions for the same fact \text{BobDylan isMarriedTo SaraLownds} occuring in two documents .................................................. 10

1.6 The SP representation for the n-ary relationship from Example 3 .................................................. 11

1.7 The SP representation for the facts about the American politician Bill Clinton and his successors from Example 4 .................................................. 11

1.8 Singleton property model representation for the contextualized knowledge base from Example 2 \((sp: \text{subPropertyOf, and } sc: \text{subClassOf})\) .................................................. 12

1.9 Example triples using singleton properties to represent the facts about the American politician Bill Clinton and his successors .................................................. 14

2.1 Singleton properties and their meta knowledge assertions for the same fact \text{BobDylan isMarriedTo SaraLownds} occuring in two documents .................................................. 21

2.2 Singleton graph pattern asserting meta knowledge for data triple \((s,p,o)\) .................................................. 24

2.3 Singleton property approach representing facts and their temporal assertions .................................................. 26

2.4 RDF interpretation for the vocabulary \(V_{EX}\) from Table 2.3 .................................................. 27

3.1 Singleton property model representation for the contextualized knowledge base from Table 1.3 \((sp: \text{subPropertyOf, and } sc: \text{subClassOf})\) .................................................. 32

4.1 Example triples using singleton properties to represent the facts about the American politician Bill Clinton and his successors .................................................. 52
5.1 PaCE approach for \((s, p, o)\) with meta knowledge \((\text{PMID?id}=1, 0.3)\) and \((\text{PMID?id}=2, 0.8)\) ........................................ 67

5.2 Singleton Property approach for \((s, p, o)\) with meta knowledge \((\text{PMID?id}=1, 0.3)\) and \((\text{PMID?id}=2, 0.8)\) ........................................ 67

5.3 Overall statistics of the SP-YAGO2 dataset ........................................ 68

5.4 Sample meta properties in SP-YAGO2 including temporal, spatial and provenance ........................................ 68

5.5 YAGO2 uses fact ID for representing fact and asserting meta knowledge ........................................ 69

5.6 Singleton property replaces fact ID in asserting meta knowledge ........................................ 69

6.1 Number of RDF quads and unique quads per dataset with NCBI Genes (NCB), DBpedia (DBP), PharmGKB (PHA), CTD, and GO Annotations (GOA) ........................................ 78

7.1 Number of politicians and pairs for each input group ........................................ 86

7.2 Number of reachable pairs (R) and time taken (T) per group in the CKG model and the NLAN model, ran in 5 threads in GraphKE ........................................ 87

7.3 Total time taken and average time taken in seconds in the CKG model running in 1 thread vs. 5 threads in GraphKE, and 1 thread in RDF-3X ........................................ 90

8.1 The compact OKN representation of singleton properties and their meta knowledge assertions for the fact \(\text{BobDylan isMarriedTo SaraLownds}\) occurring in two documents ........................................ 94

8.2 The compact OKN representation for the contextualized knowledge base from Example 2 \((sp: \text{subPropertyOf}, \text{and sc: subClassOf})\) ........................................ 95

8.3 The compact OKN representation the n-ary relationship from Example 3 ........................................ 96
ACKNOWLEDGMENTS

During my doctoral program, I have had the opportunities to work with and learn from many great mentors, colleagues, and friends.

I would first like to thank my advisor, Dr. Amit Sheth, for several years of supporting me and my research. He has created a unique environment for our professional and personal development. At the beginning of my program, I joined the T.cruzi project, and thanks to this project, I learned how to collaborate with biologists and learned how to write funding proposals. I learned how to define and refine the research problems, and I learned what it takes for a project to be funded. From the first days, he has been inspiring me to find ways to become a world-class scientist, and this has become my goal. I have gradually raised the bar for myself. During that journey, he has supported me with all he could. Many would be jealous of me if they know how supportive he has been. I had the opportunities to intern at the best places for my research (National Library of Medicine - NLM, Oracle, IBM Watson) thanks to his networking and the eco-system he has created. He provided the best hardware for my big data experiments. I have had the most flexibility for my unusual working style, which has tested his patience to the extreme, and he has proven himself to be the most patient person on earth. More importantly, he has always been available whenever I needed his help. Despite years without major publications, he has still invested in me and my research. Luckily, our patience has paid off, and the fruits come at the end of my program. In so many ways he has influenced me and my research. The aspiration from him to work on the big problems with high impact for humankind will continue with me in my post-graduation career. His dedication to students will be appreciated for life.

Aside from my advisor, the most influential person to me and my research has been my mentor at the NLM, Dr. Olivier Bodenreider. In my first year, I came across his research proposal on the biomedical knowledge repository (BKR). This has been the driving force of my research. With the ideas I had from discussions with Dr. Jagannathan Srinivasan during my Oracle internship, Dr. Bodenreider and I developed them into the singleton property approach, which was published at the prestigious WWW conference. During both internships I have done with him at the NLM, we were very productive with two publications. They were the results of many endless discussions during my internships at the NLM. I still remember the days we started our discussions at 3 PM, and we did not finish until his wife called him on the phone around 8 PM. He has been the sounding board for all my ideas over the years, and discussing with him was always a pleasure. He not only used our
SP approach in his projects but also introduced our work to other teams. This made me so proud of our work since I believe that a product is no good if the creators do not use it themselves.

I would never forget the initial steps leading me to the direction of my dissertation research. Dr. Satya Sahoo was my mentor during my first year and introduced me to the BKR he initiated with Dr. Bodenreider, and our singleton property approach has been developed on top of his work. Dr. Pascal Hitzler taught the course providing the first introduction to the Semantic Web as well as to the Knowledge Representation and Reasoning field with his book “Foundations of Semantic Web technologies”, which my dissertation is built upon. Dr. Jagannathan Srinivasan (my mentor at Oracle) has enlightened me with his wisdom when I was too confused in my beliefs, and our discussions have motivated me to develop the initial ideas of the SP model. Mr. Clare Paul (AFRL at Wright-Patterson) and I had many long, enthusiastic discussions on the adoption of our SP model into the material science domain. Dr. Ramakanth Kavuluru has worked with me patiently on the development of the access control model, even though it was yet to be published. It helped me to develop perspectives on different kinds of metadata for the SP model later. I started following Dr. Ramanathan V. Guha’s work after I generalized the specific provenance metadata into the more general concept, context. Dr. Guha’s work on context and his many admirable accomplishments in creating impactful technologies have influenced my research, and it was a great honor to have him on my dissertation committee. I am also thankful for Dr. Kemafor Anyanwu to serve on my committee.

I really appreciate the supportive eco-system we have had at Kno.e.sis over the years. Dr. Krishnaprasad Thirunarayan’s door is always open for us with his dedication. I have a feeling that we can comfortably discuss anything with him and seek his advice from research projects, research problems, philosophy, and career goals to daily life chit chat with his open mind and compassionate heart. My colleagues and friends from Kno.e.sis, I have always been able to talk to them and share with them my happiness and concerns. We have spent many memorable years together and our friendships do not stop after graduation!

I am thankful to my dear friend Franklin for always standing beside me and caring for me with his sympathetic ears. He has always been listening to each and every problem I encountered and giving me the emotional support whenever I needed. He has always been patient listening to my ideas and helping me to polish my manuscripts although they are not in his major. His positive attitude has influenced me a lot, and my viewpoints have become more positive and balanced because of him.

Last but not least, my big family has always been there for me. To my parents, my brothers, my sisters, my nieces, and my nephew, who always love me and support me unconditionally, I love them all!
1

Introduction

Since 2012, Knowledge Graph has become a common name for many enterprise knowledge bases with their question answering and search services. The Knowledge Graph term has gained popularity since its first use by Google in 2012 [Singhal 2012]. Although other companies such as IBM, Microsoft, Amazon, and Facebook use different names for their knowledge bases, the Knowledge Graph term can be alternatively used for these knowledge bases as well because they all refer to the knowledge bases with entities and relationships that can be utilized for search engines, personal assistant, and other services. Particularly, Google uses its Knowledge Graph to enhance its understanding of search queries by moving from strings to things (e.g. people, places, and movies) and generating summaries (or information box) based on the topic of the search keyword. Similar to Google search engine, the other major search engine Microsoft Bing [1] leverages its knowledge base Satori for understanding the query input and generating information summaries.

Beside the use in search engines, the Knowledge Graphs have also become the core components of major commercial technologies such as intelligent personal assistant services with Amazon Alexa [2], Google Assistant [3], Microsoft Cortana [4] and Apple Siri [5]. These personal assistant services come with a natural language interface (speech) and use the knowledge graphs as the underlying knowledge repositories for question answering services, from managing calls, making appointments, playing music, and asking for weather conditions to managing smart devices around the houses. Google’s Knowledge Graph (and Knowledge Vault later), Facebook’s Entity Graph, Microsoft’s Satori, and Amazon’s True Knowledge (Evi) [6] provide the entities and relationships extracted from Wikipedia and Web pages, these Knowledge Graphs have become an integral part of the technologies that are

shifted from keyword-based to entity-based services, such as question answering and search engines. We believe that these Knowledge Graphs will aspire and enable more and more intelligent services and applications that influence our lives in the broader and deeper ways. And these commercial products are just the beginning.

The Knowledge Graphs mentioned above are the well-known examples that are commercialized by the giant companies. Many other knowledge graphs are created and shared on the Web. The Linked Data [Bizer et al. 2009; Speicher et al. 2015] is a major hub of such knowledge graphs (or alternatively, knowledge bases). By February 2017, the Linked Data contains 1,163 knowledge graphs from various domains such as Life Sciences, Geography, Government, Media, Social Networking, Linguistics, Publications, and User Generated. The links between any two knowledge graphs in the Linked Data are created if they share common entities. DBpedia and NCI Thesaurus are currently among the most influential knowledge graphs in the Linked Data (2017).

Indeed, Knowledge Graph is not a new concept. It has deep roots from the Knowledge Representation and Reasoning field, a sub-field of Artificial Intelligence with a long history. Back in 1980’s, Cyc was started as a massive symbolic artificial intelligence project that created a knowledge base of human common-sense knowledge in a machine-understandable form [Lenat et al. 1985; Lenat 1995; Matuszek et al. 2006]. The rules and assertions of the Cyc knowledge base are mostly hand-written using the CycL knowledge representation language. Evolving from the CycL language, Guha developed the Meta Content Framework (MCF) [Guha 1996], and the MCF using the XML [Guha and Bray 1997], which has become the technical precursor for the Resource Description Framework (RDF), the World Wide Web Consortium (W3C) standard for knowledge representation in the Semantic Web.

Semantic Web technologies with a suite of W3C recommendations have become standard languages for machine-understandable knowledge representation and reasoning. They provide a common framework for data to be shared and reused across applications. The RDF data model consists of RDF triples, each of which is in the form subject-predicate-object. The current RDF 1.1 standards come with a set of specifications such as RDF 1.1 Concept and Abstract Syntax [Cyganiak et al. 2014], RDF 1.1 Semantics [Hayes and Patel-Schneider 2014]. The RDF Schema [Brickley et al. 2014] is a set of classes and properties for constructing RDF resources. Web Ontology Language (OWL) [Consortium et al. 2012] is an ontology language for Semantic Web with formally defined meaning. OWL 2 ontologies provide classes, properties, individuals, and data values for RDF documents. The RDF dataset, a set of RDF triples, can be loaded into the triple stores and queried using SPARQL [Harris et al. 2013].

http://linkeddata.org/
1.1. CONTEXTUALIZED KNOWLEDGE GRAPHS

With the maturity of Semantic Web technologies, more and more knowledge graphs describing people, places, genes, or diseases are being generated for various domains. For supporting the interoperability among knowledge graphs and applications, several ontologies have been created by the Semantic Web community. Two major outcomes are schema.org and BioPortal. The schema.org \cite{Guha2016} provides a single schema across a wide range of topics with 638 classes and 965 relations being used by 12 million sites. BioPortal currently has 686 ontologies with 8,387,594 classes and 95,468,433,792 direct annotations (retrieved by Jan 10, 2018). The deployment of Knowledge Graphs in popular commercial systems like Google Search \cite{Singhal2012}, IBM Watson \cite{Kalyanpur2012}, and Talee \cite{Sheth2001,Sheth2002} has shown the initial successes of the Semantic Web research, moving from academic research to enterprise scale.

### 1.1 Contextualized Knowledge Graphs

The facts in a knowledge graph are usually in the form of triples, or subject-predicate-object such as:

\[
\text{bobDylan} \quad \text{isMarriedTo} \quad \text{saraLownds}
\]

\[
\text{chadHurley} \quad \text{worksAt} \quad \text{google}.
\]

Here we simplify the syntax of resource URIs for readability by eliminating their prefixes. While these facts are useful for finding working companies of a person, they do not provide sufficient information for answering many types of challenging questions involving contextual knowledge. Such lists of query types and their examples are listed in Table 1.1.

Additional information about the triples must be provided in order to address those queries. A Semantic Web knowledge base, also called a knowledge graph, is created using different methods. Existing data from a structured form (e.g., relational databases, text files, XML documents, or HTML pages) can be transformed into RDF with ontologies describing the database schema (e.g., Bio2RDF \cite{Belleau2008} and PubChem \cite{Fu2015}). A knowledge graph can also be created by extracting the triples in the form of (subject, predicate, object) from unstructured data using natural language processing algorithms (e.g., Google Knowledge Vault \cite{Dong2014}, Yago2S \cite{Hoffart2013}, and DBpedia \cite{Lehmann2015}). In either method, each RDF triple in the resulting knowledge graphs can be associated and enriched with different types of contextual information, such as the time duration in which the triple holds true, and the provenance specifying the source of the triple.

Furthermore, if a knowledge graph is created and maintained by an organization, the knowledge integrity can be validated within that organization. Nowadays, it is commonplace for a knowledge
Table 1.1: Sample queries for different types of contextual knowledge, each query example is assigned an identifier ($P_i$, $T_i$, $S_i$, and $C_i$) for references

<table>
<thead>
<tr>
<th>Query type</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Provenance</td>
<td>P1. Where is this fact from?</td>
</tr>
<tr>
<td></td>
<td>P2. When was it created?</td>
</tr>
<tr>
<td></td>
<td>P3. Who created this fact?</td>
</tr>
<tr>
<td>Temporal</td>
<td>T1. When did this event occur?</td>
</tr>
<tr>
<td></td>
<td>T2. What is the time span of this event?</td>
</tr>
<tr>
<td></td>
<td>T3. Which events were in the same year?</td>
</tr>
<tr>
<td>Spatial</td>
<td>S1. What is the location of this event?</td>
</tr>
<tr>
<td></td>
<td>S2. Which events occurred at the same place?</td>
</tr>
<tr>
<td>Certainty</td>
<td>C1. What is the confidence in this fact?</td>
</tr>
</tbody>
</table>

Graph to be created and shared by anyone on the Web, or in the Linked Open Data. Therefore, we believe that the context of every fact or assertion should be provided to enable both end users and machines to validate and assess the reliability of the knowledge before using it. The contextual information associated with a triple may provide the time interval or the time instant when the triple holds true so that the consumers can validate it. It may also provide the Web page or the article from which the triple was extracted, or any provenance information that allows for tracking the origin of the triples so that the reliability of the triples can be assessed. Since a fact would not hold true in every context, the context in which a fact holds true needs to be presented in the knowledge bases so that the fact can be validated and reused.

Such knowledge graphs are called contextualized knowledge graphs. A contextualized knowledge graph (CKG) consists of RDF contextualized statements. An RDF contextualized statement is an RDF triple associated and enriched with different types of contexts, such as the time duration in which the triple holds true, and the provenance specifying the source of the triple. As we discussed above, Semantic Web technologies are matured for supporting the representation and reasoning of knowledge graphs. However, the technologies for supporting contextualized knowledge graphs are still under development and no standard has been created yet.

Particularly, at the heart of Semantic Web technologies is the RDF data model which contains two basic concepts: triples and meta-triples about triples. An RDF triple $t$ consists of a subject $s$, a predicate $p$, and an object $o$. An RDF triple has been formally defined within the latest W3C Recommendations such as the RDF 1.1 concept and abstract syntax [Cyganiak et al. 2014] and the
1.2. MOTIVATING EXAMPLES

RDF 1.1 formal semantics [Hayes and Patel-Schneider 2014]. The latest RDF standard data model is incapable of representing contextual metadata. The RDF reification was previously presented as a part of the RDF standard data model in the W3C 2004 Recommendations [Hayes and McBride 2004]. However, this RDF reification has been withdrawn from the normative sections in the latest RDF Recommendation [Hayes and Patel-Schneider 2014] and has become non-standard because of its limitations, which will be discussed later in this chapter. Representing triples about triples is a basic requirement for the RDF data model but remains one of the most important open problems.

Despite the lack of support in the RDF standard data model for representing contexts of RDF triples, a growing number of Semantic Web contextual knowledge graphs created in the recent years show the emerging demand for such support. Several knowledge bases such as Bio2RDF from release 2 [Callahan et al. 2013], PubChem [Fu et al. 2015], Wikidata [Hernández et al. 2015], Yago2S [Hoffart et al. 2013], and DBpedia [Lehmann et al. 2015], DisGenNet [Queralt-Rosinach et al. 2016] require the contextual metadata for triples including provenance, trust, certainty, time, and location to be represented and queried. These knowledge graphs being created with context incorporated into individual triples show the demand for a standard RDF data model to accommodate it. Open Knowledge Network [9], a community effort led by the Big Data Interagency Working Group (IWG) [10] at NITRD, is also looking for such an RDF data model [11]. We will present our proposal to the OKN data model in Chapter 8.

1.2 Motivating Examples

An RDF triple may be associated with different types of context, including time, location, provenance, trust, and so on. Lenat and Guha [Lenat 1998; Guha et al. 2004] discussed various dimensions in the context space that are learned from the Cyc and other projects. The dimensions are time, type of time, geolocation, type of place, culture, sophistication/security, topic, granularity, modality, argument-preference, justification, and let’s.

Here we introduce four examples for demonstrating the requirements of contextualized knowledge graphs that are common in the knowledge integration and acquisition tasks. We will use this KB as our running examples throughout the dissertation.

Example 1. (Multiple occurrences). Given the fact BobDylan isMarriedTo SaraLownds, we can extract from the wiki pages of Bob_Dylan with timestamp 2009-06-07 and Sara_Dylan with timestamp 2009-08-08 as presented in Table 1.2. The information about the sources and the dates

of this fact would help find the answer for the question P1 (where is the fact from?) and P2 (when was it created?) given in Table 1.1.

Table 1.2: The same fact has different provenance and temporal information associated with it

<table>
<thead>
<tr>
<th>Triple</th>
<th>Provenance</th>
<th>Extraction Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>(S_1):((Bob,Dylan, \text{isMarriedTo}, Sara,Lownds))</td>
<td>wiki:Bob_Dylan</td>
<td>2009-06-07</td>
</tr>
<tr>
<td>(S_2):((Bob,Dylan, \text{isMarriedTo}, Sara,Lownds))</td>
<td>wiki:Sara_Dylan</td>
<td>2009-08-08</td>
</tr>
</tbody>
</table>

The same fact occurring in different contexts is a common case. It may happen when the fact is extracted from different unstructured documents such as Wiki pages or scientific articles.

**Example 2. (Multiple annotations).** It is also very common for a triple to be associated with multiple types of contextual information such as the combination of provenance, time, and score. Table 1.3 shows an example of such a contextualized KB adapted from Zimmermann et al. 2012.

Table 1.3: Contextualized knowledge base for employee information (sp: subPropertyOf, and sc: subClassOf)

<table>
<thead>
<tr>
<th>Triple</th>
<th>Time</th>
<th>Prov</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>(C_1)::((chad,Hurley, \text{type}, youtube,Emp))</td>
<td>[2005, 2010]</td>
<td>0.8</td>
<td></td>
</tr>
<tr>
<td>(C_2)::((youtube,Emp, \text{sc}, google,Emp))</td>
<td>[2006, 2011]</td>
<td>0.9</td>
<td></td>
</tr>
<tr>
<td>(C_3)::((chad,Hurley, \text{ceo}, youtube))</td>
<td>[2005, 2010]</td>
<td>0.7</td>
<td></td>
</tr>
<tr>
<td>(C_4)::((\text{ceo, sp, worksFor}))</td>
<td>work</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>(C_5)::((\text{worksFor, sp, member}))</td>
<td>work</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>(C_6)::((\text{ceo, domain, Person}))</td>
<td>work</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>(C_7)::((\text{ceo, range, Company}))</td>
<td>work</td>
<td>1.0</td>
<td></td>
</tr>
</tbody>
</table>

**Example 3. (N-ary relationship).** John buys “Lenny the Lion” book from books.example.com for $15 as a birthday gift.

This example is taken from W3C Working Note Noy et al. 2006. Although the RDF triple represents a binary relationship, human knowledge is usually more complex and is in the form of n-ary relationship.

**Example 4. (Time duration and n-ary relationship)** Bill Clinton held the political position as US President from Jan 20, 1993 to Jan 20, 2001 and George W. Bush is the successor for this position. Bill Clinton held the political position as Arkansas Governor from January 11, 1983 to December 12, 1992 and Frank White is the successor for this position.
1.3. EXISTING APPROACHES

Throughout the dissertation, we will discuss how we represent and infer with these examples in RDF.

1.3 Existing approaches

We consider two existing approaches for representing the motivating examples: reification and n-ary.

Reification. Using the reification approach to represent the example in Table 1.2, the list of reified triples are provided in Table 1.4. The fact is represented as an instance of class Statement with three different properties for its subject, predicate and object.

Table 1.4: Reified statements and their meta knowledge assertions for the same fact BobDylan isMarriedTo SaraLownds occuring in two documents from the example 1 (Table 1.2)

<table>
<thead>
<tr>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>BobDylan</td>
<td>isMarriedTo</td>
<td>SaraLownds</td>
</tr>
<tr>
<td>stmt#1</td>
<td>rdf:type</td>
<td>Statement</td>
</tr>
<tr>
<td>stmt#1</td>
<td>rdf:subject</td>
<td>BobDylan</td>
</tr>
<tr>
<td>stmt#1</td>
<td>rdf:predicate</td>
<td>isMarriedTo</td>
</tr>
<tr>
<td>stmt#1</td>
<td>rdf:object</td>
<td>SaraLownds</td>
</tr>
<tr>
<td>stmt#1</td>
<td>hasSource</td>
<td>wk:Bob_Dylan</td>
</tr>
<tr>
<td>stmt#1</td>
<td>extractedOn</td>
<td>2009-06-07</td>
</tr>
<tr>
<td>stmt#2</td>
<td>rdf:type</td>
<td>Statement</td>
</tr>
<tr>
<td>stmt#2</td>
<td>rdf:subject</td>
<td>BobDylan</td>
</tr>
<tr>
<td>stmt#2</td>
<td>rdf:predicate</td>
<td>isMarriedTo</td>
</tr>
<tr>
<td>stmt#2</td>
<td>rdf:object</td>
<td>SaraLownds</td>
</tr>
<tr>
<td>stmt#2</td>
<td>hasSource</td>
<td>wk:Sara_Dylan</td>
</tr>
<tr>
<td>stmt#2</td>
<td>extractedOn</td>
<td>2009-08-08</td>
</tr>
</tbody>
</table>

Since we need to represent two occurrences of the same statement in two different documents, we create two resources: stmt#1 and stmt#2 because if we create only one stmt#1 for both occurrences, the association of each occurrence with its source and date of extraction together is not distinguishable. The meta information about the fact is represented by hasSource and extractedOn properties.

The RDF reification does not have a formal semantics for capturing the relationship between a statement instance and the triple it represents. The lack of formal semantics connecting a statement and the resource describing it is one of the main drawbacks of using reification for describing triples.
1.4. THESIS STATEMENT

Since the resource stmt#1 describing a statement is not associated with that statement, assertions created for this resource are not the same as assertions created for the original statement as explained in the RDF Primer [Manola et al. 2004]. Moreover, the reification approach requires four additional triples for representing one statement per document as a resource. This increases the size of the data sets by at least four times, which is not a scalable approach. It would also make query patterns lengthy for finding when the couple was married or divorced.

N-ary relationship. The sentence from Example 2 is represented by the graphs in Figure 1.1 taken from W3C Working Note [Noy et al. 2006]. This approach creates a new entity Purchase_1 and links it to other entities within the sentence. This modeling is not a principled approach. It does not come with a formal semantics explaining each element of the n-ary relationship.

![Figure 1.1: The n-ary approach for representing Example 3 (from [Noy et al. 2006]).](image)

1.4 Thesis Statement

Although the RDF data model is widely adopted as W3C standards for representing knowledge, it does not have the foundational support for representing, reasoning, and traversing contextualized knowledge graphs. Here we analyze some of the foundational challenges as follows.

The current formal semantics cannot represent the contextualized KGs. Among the existing approaches (reification and n-ary), none of them comes with a formal semantics formalizing the relationship between the triple identifier and the triple it represents.

A sound and complete reasoning mechanism for contextualized KG does not exist. In order for machines to infer contextual triples, we believe that two requirements need to be fulfilled. First, these contextual statements must be represented in machine-understandable form, with
explicitly defined semantics for the relationship between the triple and its contextual information. Second, we need an entailment mechanism that takes into account the semantics of the contextual statements so that it can validate the existing statements and entail the new ones. There is no sound and complete proof system that involves RDF contextual triples about triples.

We need to investigate whether or not a more efficient mechanism for describing a statement using RDF exists. A good design should provide a formal semantics, use existing syntax and be compatible with existing Semantic Web languages, tools, and methods. The proposed formal semantics should be compatible with the existing model-theoretic semantics in order to avoid conflicts in the RDF/RDFS interpretation. Using the existing RDF syntax would ensure the compatibility of meta triples and existing triple datasets. Such design would overcome the need to develop new or revise available tools and methods for making them work with new contextual triples.

The current RDF semantics is not formally defined with a graph-based formalism for contextualized KGs. The reified statements are not formalized and its bipartite model represents a hyper-graph instead of a simple graph.

This dissertation addresses these challenges and we develop our thesis statement as follows.

It is possible to develop (1) a compact and formal representation, (2) a sound and complete inference mechanism, and (3) a model-theoretic graph formalism for contextualized knowledge graphs that can be efficiently implemented.

1.5 Our Approach

Since the foundations of the current RDF data model lacks the support for the contextualized KGs, my dissertation develops the foundations for the RDF data model to support the representation, reasoning, and traversal of the contextualized KGs.

1.5.1 Knowledge Representation

This dissertation proposes a novel approach called Singleton Property for representing and reasoning with RDF statements about statements. Our approach is based on the intuition that the nature of every relationship is universally unique. The uniqueness of the relationship can be a key for any statement using the new type of property called singleton property. A singleton property is a property instance representing one specific relationship between two particular entities under one specific context. Singleton properties can be viewed as instances of generic properties whose extensions contain a set of entity pairs. In order to represent the uniqueness for a singleton property, we assign a unique URI for each singleton property.
Next, we present the representation of the three examples using the SP modeling approach in Table 1.5, Table 1.8, and Table 1.6.

In Example 1, for the statement BobDylan isMarriedTo SaraLownds, we can create two singleton property instances describing the occurrences of this statement in two documents as provided in Table 1.5.

Table 1.5: The RDF representation of singleton properties and their meta knowledge assertions for the same fact BobDylan isMarriedTo SaraLownds occurring in two documents

<table>
<thead>
<tr>
<th>No</th>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>BobDylan</td>
<td>isMarriedTo?id=1</td>
<td>SaraLownds</td>
</tr>
<tr>
<td>2</td>
<td>isMarriedTo?</td>
<td>hasSource</td>
<td>wk:Bob_Dylan</td>
</tr>
<tr>
<td>3</td>
<td>isMarriedTo?</td>
<td>extractedOn</td>
<td>2009-06-07</td>
</tr>
<tr>
<td>4</td>
<td>isMarriedTo?</td>
<td>rdf:singletonPropertyOf</td>
<td>isMarriedTo</td>
</tr>
<tr>
<td>5</td>
<td>BobDylan</td>
<td>isMarriedTo?id=2</td>
<td>SaraLownds</td>
</tr>
<tr>
<td>6</td>
<td>isMarriedTo?</td>
<td>hasSource</td>
<td>wk:Sara_Dylan</td>
</tr>
<tr>
<td>7</td>
<td>isMarriedTo?</td>
<td>extractedOn</td>
<td>2009-08-08</td>
</tr>
<tr>
<td>8</td>
<td>isMarriedTo?</td>
<td>rdf:singletonPropertyOf</td>
<td>isMarriedTo</td>
</tr>
</tbody>
</table>

For each document (or context of the fact), we create one separate singleton property representing that fact (in $T_1^1$, $T_5^1$). Particularly we create two singleton properties isMarriedTo?id=1 and isMarriedTo?id=2 for the relationships extracted from the Wiki pages of Bob Dylan (wk:Bob_Dylan) and Sara Dylan (wk:Sara_Dylan), respectively. Meta knowledge about the fact from one document can be added as assertions for the singleton property from that document (in $T_2^1$, $T_3^1$, $T_6^1$, and $T_4^1$).

Note that one singleton property is created for each meta triple as well, following the two modeling principles.

In Example 2, consider the contextualized statement $C_1$ “(chadHurley, type, youtubeEmp): [2005, 2010] and score 0.8” given in Table 1.3. The singleton property model represents it with 5 triples $T_1^2$, $T_2^2$, $T_3^2$, $T_4^2$, and $T_5^2$ in Table 1.8. The singleton property type?id=1 uniquely represents the type relationship between chadHurley and youtubeEmp in the triple $T_1^2$. The singleton property type?id=1 can be asserted with contextual information (time [2005, 2010] and confidence score 0.8) about the relationships as shown in the triples $T_2^2$, $T_3^2$, and $T_4^2$.

In Example 3, we represent the n-ary relationship by the singleton property buys?id=1 with the set of triples shown in Table 1.6. We represent the example 4 in Table 1.7.

We distinguish the regular predicates from the singleton properties by naming them generic properties. Similar to class-instance relationships, the singleton properties are considered as property
instances of a generic property. We define the predicate \texttt{singletonPropertyOf} between a singleton property and a generic property.

\[
isMarriedTo?id=1 \text{ singletonPropertyOf } \text{isMarriedTo}.
\]

We capture the semantics of this relationship between a triple and its identifier and express it as a first-class citizen in the formal semantics. We will describe the model-theoretic semantics for interpreting the singleton properties, in Chapter 2. It allows the semantics of the contextual statements to be expressed in the formal model.
Table 1.8: Singleton property model representation for the contextualized knowledge base from Example 2 (sp: subPropertyOf, and sc: subClassOf)

<table>
<thead>
<tr>
<th>Triple</th>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_2^2$</td>
<td>chadHurley</td>
<td>type? id=1</td>
<td>youtubeEmp</td>
</tr>
<tr>
<td>$T_2^3$</td>
<td>type? id=1</td>
<td>from</td>
<td>2005</td>
</tr>
<tr>
<td>$T_2^4$</td>
<td>type? id=1</td>
<td>to</td>
<td>2010</td>
</tr>
<tr>
<td>$T_2^5$</td>
<td>type? id=1</td>
<td>score</td>
<td>0.8</td>
</tr>
<tr>
<td>$T_2^6$</td>
<td>type? id=1</td>
<td>rdf:singletonPropertyOf type</td>
<td></td>
</tr>
<tr>
<td>$T_2^7$</td>
<td>youtubeEmp</td>
<td>sc? id=6</td>
<td>googleEmp</td>
</tr>
<tr>
<td>$T_2^8$</td>
<td>sc? id=6</td>
<td>from</td>
<td>2006</td>
</tr>
<tr>
<td>$T_2^9$</td>
<td>sc? id=6</td>
<td>to</td>
<td>2017</td>
</tr>
<tr>
<td>$T_2^{10}$</td>
<td>sc? id=6</td>
<td>score</td>
<td>0.9</td>
</tr>
<tr>
<td>$T_2^{11}$</td>
<td>sc? id=6</td>
<td>rdf:singletonPropertyOf sc</td>
<td></td>
</tr>
<tr>
<td>$T_2^{12}$</td>
<td>chadHurley</td>
<td>ceo? id=11</td>
<td>youtube</td>
</tr>
<tr>
<td>$T_2^{13}$</td>
<td>ceo? id=11</td>
<td>from</td>
<td>2005</td>
</tr>
<tr>
<td>$T_2^{14}$</td>
<td>ceo? id=11</td>
<td>to</td>
<td>2010</td>
</tr>
<tr>
<td>$T_2^{15}$</td>
<td>ceo? id=11</td>
<td>score</td>
<td>0.7</td>
</tr>
<tr>
<td>$T_2^{16}$</td>
<td>ceo? id=11</td>
<td>rdf:singletonPropertyOf ceo</td>
<td></td>
</tr>
<tr>
<td>$T_2^{17}$</td>
<td>ceo</td>
<td>sp? id=16</td>
<td>worksFor</td>
</tr>
<tr>
<td>$T_2^{18}$</td>
<td>sp? id=16</td>
<td>derivedFrom</td>
<td>work</td>
</tr>
<tr>
<td>$T_2^{19}$</td>
<td>sp? id=16</td>
<td>score</td>
<td>1.0</td>
</tr>
<tr>
<td>$T_2^{20}$</td>
<td>sp? id=16</td>
<td>rdf:singletonPropertyOf sp</td>
<td></td>
</tr>
<tr>
<td>$T_2^{21}$</td>
<td>worksFor</td>
<td>sp? id=20</td>
<td>member</td>
</tr>
<tr>
<td>$T_2^{22}$</td>
<td>sp? id=20</td>
<td>derivedFrom</td>
<td>work</td>
</tr>
<tr>
<td>$T_2^{23}$</td>
<td>sp? id=20</td>
<td>score</td>
<td>1.0</td>
</tr>
<tr>
<td>$T_2^{24}$</td>
<td>sp? id=20</td>
<td>rdf:singletonPropertyOf sp</td>
<td></td>
</tr>
<tr>
<td>$T_2^{25}$</td>
<td>ceo</td>
<td>domain? id=24</td>
<td>Person</td>
</tr>
<tr>
<td>$T_2^{26}$</td>
<td>domain? id=24</td>
<td>derivedFrom</td>
<td>work</td>
</tr>
<tr>
<td>$T_2^{27}$</td>
<td>domain? id=24</td>
<td>score</td>
<td>1.0</td>
</tr>
<tr>
<td>$T_2^{28}$</td>
<td>domain? id=24</td>
<td>rdf:singletonPropertyOf domain</td>
<td></td>
</tr>
<tr>
<td>$T_2^{29}$</td>
<td>ceo</td>
<td>range? id=28</td>
<td>Company</td>
</tr>
<tr>
<td>$T_2^{30}$</td>
<td>range? id=28</td>
<td>derivedFrom</td>
<td>work</td>
</tr>
<tr>
<td>$T_2^{31}$</td>
<td>range? id=28</td>
<td>score</td>
<td>1.0</td>
</tr>
</tbody>
</table>
1.5. OUR APPROACH

1.5.2 Reasoning

Here we propose our approach for addressing the reasoning support from the proposed SP model-theoretic semantics.

Validating singletonPropertyOf. Since the property singletonPropertyOf connects a singleton property and a generic property, we need to formally define the semantics of this property. What should be the domain and range of this property? What is the relationship between the triple \( T^2_1 \) (chadHurley, type?=1, youtubeEmp) and the original triple (chadHurley, type, youtubeEmp)? What do they share in common and what are the differences between them?

Hierarchy inferences. Given a class hierarchy, what kinds of inferences can be performed on the singleton triples? For example, the original triples (chadHurley, type, youtubeEmp) and (youtubeEmp, sc, googleEmp) infer the new triple (chadHurley, type, googleEmp). What do their corresponding singleton triples \( T^2_1 \) (chadHurley, type?=1, youtubeEmp) and \( T^2_2 \) (youtubeEmp, sc?=6, googleEmp) entail? Similarly, for the property hierarchy inference, the original triples (ceo, sp, worksFor) and (worksFor, sp, member) entail the new triple (ceo, sp, member). What do the singleton triples (ceo, sp?=16, worksFor) and (worksFor, sp?=20, member) entail?

Contextual inferences. Given the two contextualized statements \( C_1 \) “(chadHurley, type, youtubeEmp) : [2005,2010]” and \( C_2 \) “(youtubeEmp, sc, googleEmp) : [2006,2011]”, we can infer the new contextualized triple (chadHurley, type, googleEmp):[2006,2011] using the inference rules for the annotated RDF from [Zimmermann et al. 2012; Udrea et al. 2010]. Since these two contextualized statements are represented using the set of SP triples \( T^2_1 \) to \( T^2_{10} \), what can be concluded from this set of SP triples?

In the dissertation, we investigate the questions above and propose a principled approach to develop the inference rules for the singleton property model in Chapter 3. We identify the new concepts and formalize their semantic associations with singleton properties using a model theory (Section 3.1). Based on the semantic associations, we develop a set of inference rules with their proofs derived from the model theory (Section 3.2). We adapt the inference rules from the annotated RDF work [Zimmermann et al. 2012] into the SP representation of RDFS rules to derive the set of contextual rules (for time-interval and fuzzy). The formalism allows the proposed reasoning mechanism to be implemented in Semantic Web reasoners and allow the contextualized KBs to be reasoned with via these reasoners.

Our reasoning mechanism does not limit the support to only contextualized KBs with SP representation. Any contextualized KB available in other representations such as the named graph or RDF reification can also be reasoned with our mechanism. These KBs need to be transformed from non-SP to SP representation. To support such a transformation, we need to develop a tool that
1.5. OUR APPROACH

Table 1.9: Example triples using singleton properties to represent the facts about the American politician Bill Clinton and his successors

<table>
<thead>
<tr>
<th>No</th>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>BillClinton</td>
<td>holdsPos?id=1</td>
<td>U.S.President</td>
</tr>
<tr>
<td>T2</td>
<td>BillClinton</td>
<td>holdsPos?id=2</td>
<td>ArkansasGovernor</td>
</tr>
<tr>
<td>T3</td>
<td>holdsPos?id=1</td>
<td>hasSuccessor</td>
<td>GeorgeBush</td>
</tr>
<tr>
<td>T4</td>
<td>holdsPos?id=2</td>
<td>hasSuccessor</td>
<td>FrankWhite</td>
</tr>
</tbody>
</table>

transforms the syntax and adds explicit semantic relationships between the statements and their context. This tool, which is described in Chapter 6, allows existing contextualized KBs such as Bio2RDF with the named graph provenance to be transformed and inferred with our mechanism.

1.5.3 Graph Traversal

With contexts for RDF statements represented in the SP representation, each singleton property uniquely representing a statement can be annotated with different kinds of metadata describing that statement such as provenance, time, and location. Therefore, the singleton properties also become subjects and objects of other meta triples. Using the SP representation for contextualizing a knowledge graph with the set of triples from Table 1.9 extracted from Example 4, the resulting SP graphs are disconnected as shown in the Figure 1.2.

A reachability query verifies if a path exists between any two nodes in a graph. A shortest path query returns the shortest distance in terms of the number of edges between any two nodes in a graph if the path exists. In the NLAN graph model, both query types traverse the graph from subject node to object node of a triple. As traversing the NLAN graph starts from a node and explores its adjacent nodes, predicates as arcs never get explored. Being unable to traverse among triples in the scenarios described above due to the limited connectivity of the NLAN graph limits the capability of answering reachability and shortest path queries in the contextualized KGs.

We propose a new approach to represent the CKGs in a labeled directed multigraph with triple nodes (CKG). Similar to the bipartite model, all subjects, predicates, and objects are mapped to nodes of a CKG graph. This model, however, differs from the bipartite model in that it adds one pair of directed edges (subject-predicate, predicate-object) to directly connect three nodes of the same triple. Furthermore, this approach differs from the NLAN diagram in that the predicates are mapped to nodes instead of labeled edges. As they are adjacent to the subjects, they are explored after the subject. The objects now become adjacent to the predicates and get explored after the
1.5. **OUR APPROACH**

![Diagram](image)

Figure 1.2: Node-Labeled Arc-Node diagram (NLAN) for Example 4.

We formalize the CKG graph model by combining the formal semantics from a model theory and a graph theory into a single model that enables the graph algorithms to be developed on the semantics of the contextualized KGs.

The CKG graph representation of Example 4 shown in Table 1.9 is illustrated in Fig. 1.3. For the set of four triples, we map all subjects, predicates, and objects to the set of nine nodes. We link every three nodes representing the same triple by one pair of directed edges. For instance, the three nodes \{BillClinton, holdsPos?id=1, U.S.President\} are connected by the pair of edges \(e_1^I, e_I^T\). Traversing the graph this way starting from the node BillClinton will reach the node GeorgeW.Bush after 3 hops, as shown in Fig. 1.3.

Compared to the NLAN model, the CKG graph model provides better connectivity and makes it possible to find the answers for the reachability and shortest path queries. While the node GeorgeW.Bush is unreachable from the node BillClinton in the NLAN model, it is not only reachable but also neatly presented as the SP-path (BillClinton, holdsPos?id=1, hasSuccessor, GeorgeW.Bush) in the CKG graph model. We will formally define the contextualized resource path and triple path in Section 4.2.3.
1.6 Our Contributions

In summary, my dissertation includes both theoretical and practical contributions.

1.6.1 Foundational contributions

We propose the singleton property data model and develop the foundations for this data model to provide the formal support for the representation, querying, reasoning, and traversal of contextualized knowledge graphs using Semantic Web technologies.

**A triple-based compact representation for CKGs.** The fundamental limitation of RDF data model is the lack of an efficient mechanism to represent triple about triple. We propose a principled approach for modeling RDF triple about triple by extending the current RDF data model with a new concept called singleton property. This concept plays the role of the triple identifier and allows RDF triple about triple to be represented in the form of triples.

**A formal model-theoretic semantics for CKGs.** Beside the compact representation using singleton property, we also formalize the semantics of the singleton properties and its relationships to the triple it represents. We extend the current RDF model-theoretic semantics to capture the semantics and provide the interpretation of the singleton properties at three levels: simple, RDF, and RDFS. The semantics we propose is totally compatible with the current semantics (Chapter 2).

**A sound and complete inference mechanism for CKGs.** The semantics we propose allows us to develop an inference mechanism based on the syntax and the context of SP triples. We derive a set of inference rules for validating and inferring new triples based on the SP syntax. We also derive different sets of context-based inference rules for provenance, temporal, and uncertainty. The SP representation also allows the tracking of inferred triples (Chapter 3).

**A graph-based formalism for CKGs.** We propose a contextualized graph model for the contextualized knowledge graphs using the SP representation. We formalize the RDF triples as a mathematical graph by combining the model theory and the graph theory into a hybrid RDF model-theoretic semantics. The unified semantics allows the RDF semantics for singleton properties to be leveraged in the CKG graph-based algorithms (Chapter 4).

1.6.2 Implementation

Beside the foundations, we contribute to the efficient implementation of the proposed model.

**Compatibility.** Our data model is compatible with the syntax and semantics of the current RDF data model. Therefore, the proposed model can be implemented and evaluated using the existing Semantic Web datasets, tools, and applications. Particularly, the SP datasets with triple-
compliant syntax can be loaded into both triple/quad stores and queried using SPARQL (Chapter 5).

**RDF-contextualizer tool implementation.** We have developed a tool to transform the existing datasets from named graph representation to SP representation. We have also implemented the proposed inference rules in this tool and used it for evaluating the performance on several real-world datasets. The results show that our mechanism can be practically implemented in the Semantic Web reasoners (Chapter 6).

**Contextualized KGs with SP representation.** We have implemented the SP model in several real-world datasets such as BKR, Yago2S, PubChem, DBPedia, and all Bio2RDF datasets (Chapter 6).

**A graph engine.** We have developed a new graph engine for implementing the proposed graph model for CKGs. We have also implemented this model in the existing query engine RDF-3X. In both engines, we have implemented the graph-based algorithms for contextual reachability and shortest path queries. We have also evaluated the performance of these algorithms in both engines (Chapter 7).

### 1.7 Outline

The rest of the dissertation is organized into two parts as follows.

Part I includes the following chapters describing the foundations for representing, reasoning, and traversing the contextualized knowledge graphs.

Chapter 2 starts with the discussion about the singleton property and the intuition behind this concept. We then formalize this concept using the model theory.

Chapter 3 describes the new concepts introduced in the singleton property model and formalizes them using the model-theoretic semantics. The extended semantics is utilized for developing new sets of inference rules which have been shown to respect the model theoretic semantics.

Chapter 4 proposes the new graph model for the SP representation and formalize this model by combining the RDF model-theoretic semantics with the graph theory.

Part II includes the following chapters providing the implementation information for the foundations described in Part I.

Chapter 5 provides two use cases for representing provenance and time in the BKR and Yago2S datasets. The evaluation of BKR queries compares the performance among existing approaches with the SP approach.

Chapter 6 describes our implementation for the tool RDF-contextualizer. We describe how
existing datasets can be transformed into SP representation using our tool. We also describe the experiments for evaluating the applications of the proposed inference rules on the real-world datasets.

Chapter 7 describes our algorithms and implementation for the proposed graph model. We also describe the experiments evaluating the graph algorithms for the shortest path and reachability queries.

Chapter 8 discusses the adoption for our proposed model.

Chapter 9 concludes this dissertation.
Part I

Foundations
2

Knowledge Representation for Contextualized Knowledge Graphs

This chapter starts with the discussion about the singleton property and the intuition behind this concept. We then formalize this concept using the model theory.

2.1 Singleton Property Concept

In this section, we will explain our intuition for the proposed approach and justify our design choices for the singleton property. Here we explain our rationale accounting for the novel perspective that leads to our approach.

2.1.1 Singleton property as unique key for statement within a context

Back to the motivating example we used in Chapter 1, the reification process represents a triple as a resource, which is an instance of the Statement class [Hayes and McBride 2004]. Reifying a statement requires two steps. The first step is to find a resource that uniquely identifies a statement. The second is to create assertions for that statement via that resource. The first step involves finding which resource among the three elements of a triple could fundamentally distinguish statements.

In the Semantic Web, anyone can create a statement. The same statements can be created in different datasets by different organizations. Therefore, we need to find a resource that can distinguish such two statements. Given that the statements may be the same, they may be associated with different contextual information when they are created. The information capturing the context when a statement is created could be helpful for identifying statements. Such contextual information of a statement could be described by various dimensions of meta knowledge, including the source
2.1. SINGLETON PROPERTY CONCEPT

Table 2.1: Singleton properties and their meta knowledge assertions for the same fact BobDylan isMarriedTo SaraLownds occurring in two documents

<table>
<thead>
<tr>
<th>No.</th>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>T₁</td>
<td>BobDylan</td>
<td>isMarriedTo?id=1</td>
<td>SaraLownds</td>
</tr>
<tr>
<td>T₂</td>
<td>isMarriedTo?id=1</td>
<td>singletonPropertyOf</td>
<td>isMarriedTo</td>
</tr>
<tr>
<td>T₃</td>
<td>isMarriedTo?id=1</td>
<td>hasSource</td>
<td>wk:Bob_Dylan</td>
</tr>
<tr>
<td>T₄</td>
<td>isMarriedTo?id=1</td>
<td>extractedOn</td>
<td>2009-06-07</td>
</tr>
<tr>
<td>T₅</td>
<td>BobDylan</td>
<td>isMarriedTo?id=2</td>
<td>SaraLownds</td>
</tr>
<tr>
<td>T₆</td>
<td>isMarriedTo?id=2</td>
<td>singletonPropertyOf</td>
<td>isMarriedTo</td>
</tr>
<tr>
<td>T₇</td>
<td>isMarriedTo?id=2</td>
<td>hasSource</td>
<td>wk: Sara_Dylan</td>
</tr>
<tr>
<td>T₈</td>
<td>isMarriedTo?id=2</td>
<td>extractedOn</td>
<td>2009-08-08</td>
</tr>
</tbody>
</table>

recording that statement, the time or place that statement occurs, and the certainty of the author about that statement. We can assume that a statement within a context is unique. Now the next question is, what can represent that uniqueness of a statement within a context? If the same statement is associated with different contexts, are they the same in nature? What remains the same? What becomes different?

From a philosophical point of view, we believe that the existence of two entities in the subject and the object of one statement is independent from the contexts creating that statement. Particularly, they already exist before the statement is created. For example, the existences of Bob Dylan and Sara Lownds do not depend on their marriage, and obviously they also exist before they marry each other. While creating a new statement, what we actually do is connect two existing entities and establish a new relationship between them. Therefore, the contextual information in establishing a new relationship can play the role of a key for any statement. We can manifest that key by creating a new property instance that represents the newly established relationship associated with a context and enforces it to be unique. We call it singleton property. The singleton concept is taken from set theory. A singleton set has only one element.

We define a singleton property as a unique property instance representing a newly established relationship between two existing entities in one particular context.

For example, a new relationship is established for Bob Dylan and Sara Lownds according to two Wiki pages. We can consider each Wiki page as a context associated with the new relationship. Note that here we merely give examples of context and leave the questions of how exactly context is described and how to identify it for data publishers because those are subjective to them. As a result,
2.1. SINGLETON PROPERTY CONCEPT

we can create two singleton properties \texttt{isMarriedTo?id=1} and \texttt{isMarriedTo?id=2} to represent the new relationships associated with these two contexts. The statements asserting the new relationships become:

\begin{itemize}
  \item $T_1$: BobDylan isMarriedTo?id=1 SaraLownds, and
  \item $T_5$: BobDylan isMarriedTo?id=2 SaraLownds.
\end{itemize}

Obviously, the number of such singleton properties would be as enormous as the number of facts and contexts in any real RDF datasets. We need to provide a mechanism to cluster them into groups for higher level abstraction. Such a mechanism allows us to group similar singleton properties into a more general one. We observe that, although statements are fundamentally distinguishable based on their context, they do share common characteristics in their nature which are captured by generic properties. The relationship between singleton and generic properties can be considered from two different perspectives: the singleton property is either a sub property, or an instance of the generic property.

\textbf{Sub property.} Singleton property can be considered as a specialization, or sub property of a generic property in one particular context. In this case, if we create one singleton property for each fact via \texttt{rdfs:subPropertyOf}, the number of singleton property nodes below the generic property in the property hierarchy would become enormous. For example, YAGO has 23,770 facts using the property \texttt{isMarriedTo}. A schema with such a large amount of child nodes in the property hierarchy of \texttt{rdfs:subPropertyOf} is not desirable.

\textbf{Property instance.} In this view, while generic properties are properties whose extension contains a set of entity pairs, each singleton property is unique to one particular entity pair. Intuitively, we can consider singleton properties as instances of generic properties. In that sense, a singleton property is interconnected to its generic property via \texttt{rdf:type}. However, the property \texttt{rdf:type} as a generic property may also have its own instances. For example, since YAGO contains 9,019,948 facts using \texttt{rdf:type}, a triple like this may cause ambiguity: \texttt{type?id=1 rdf:type rdf:type}.

Considering the nature of the relationship from both perspectives, we invent a new property, \texttt{singletonPropertyOf} to connect singleton properties with their generic property. The extension of a generic property contains the set of singleton property instances created in all contexts. In the example described in Table 2.1 we use \texttt{singletonPropertyOf} in both $T_2$ and $T_6$.

\begin{itemize}
  \item $T_2$: isMarriedTo?id=1 singletonPropertyOf isMarriedTo
  \item $T_6$: isMarriedTo?id=2 singletonPropertyOf isMarriedTo.
\end{itemize}

We will provide further explanations of how singleton properties can be interpreted in RDF and RDFS in Section 2.2.

\footnote{http://www.mpi-inf.mpg.de/yago-naga/yago/statistics.html}
2.1. SINGLETON PROPERTY CONCEPT

2.1.2 Asserting meta knowledge for triples

Here we demonstrate how to assert metadata for a statement, such as provenance, time, location, or certainty. Please note that we are not attempting to model complex contextual information involving these four dimensions for a statement because context modeling is out of the scope of this dissertation. We also note that meta properties used in the examples such as `hasSource`, `extractedOn`, `hasStart`, `hasEnd`, `tookPlaceAt`, and `hasScore`, are only for demonstration. While adopting this approach, one may want to use vocabularies of meta knowledge recommended by W3C such as PROV [Lebo et al. 2012] or OWLTime [Hobbs and Pan 2006] for enhancing the interoperability with other Semantic Web knowledge bases and applications.

**Provenance.** Provenance of a statement explains the origin of that statement [Moreau 2010; Simmhan et al. 2005]. It includes many kinds of metadata for answering questions such as the ones listed in Table 1.1. For example, the triples $T_1$ and $T_2$ are extracted from the Wiki page of Bob Dylan and Sara Lownds. We can assert the provenance of two triples using the properties `hasSource` and `extractedOn` as follows:

\[
T_3: \text{isMarriedTo}\ ?id=1 \text{ hasSource wk:Bob_Dylan} \\
T_4: \text{isMarriedTo}\ ?id=1 \text{ extractedOn 2009-06-07} \\
T_7: \text{isMarriedTo}\ ?id=2 \text{ hasSource wk:Sara_Dylan} \\
T_8: \text{isMarriedTo}\ ?id=2 \text{ extractedOn 2009-08-08}
\]

**Time.** The validity of a statement may be associated with a specific time or a time span. For example, a person is born at one specific time, and a marriage between two persons may last for one period of time. Here we represent the time span of the marriage between Bob Dylan and Sara Lownds using `hasStart` and `hasEnd`:

\[
\text{isMarriedTo}\ ?id=1 \text{ hasStart 1965-11-22} . \\
\text{isMarriedTo}\ ?id=1 \text{ hasEnd 1977-06-29} .
\]

**Location.** A statement may be associated with a spatial dimension. For example, the Wiki page of Sara Lownds stated that the marriage of Bob Dylan and Sara Lownds took place at Mineola, Long Island. We assert this meta knowledge for `isMarriedTo?id=2` as follows:

\[
\text{isMarriedTo}\ ?id=2 \text{ tookPlaceAt Mineola} .
\]

**Certainty.** The certainty of a statement reflects the confidence of the authors while creating that statement. For example, if the confidence score of the tool extracting the statement $T_2$ is 0.7, we can represent it as follows:

\[
\text{isMarriedTo}\ ?id=2 \text{ hasScore 0.7} .
\]

From the assertions created for provenance, time, location and certainty above, we observe that they share the same triple pattern, which is `singleton-property meta-property meta-value`. In our
2.1. SINGLETON PROPERTY CONCEPT

Table 2.2: Singleton graph pattern asserting meta knowledge for data triple \((s,p,o)\)

<table>
<thead>
<tr>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>p?id=i</td>
<td>singletonPropertyOf</td>
<td>p</td>
</tr>
<tr>
<td>s</td>
<td>p?id = i</td>
<td>o</td>
</tr>
<tr>
<td>p?id=i</td>
<td>mp</td>
<td>mv</td>
</tr>
</tbody>
</table>

example, meta properties are \texttt{hasStart}, \texttt{hasEnd}, \texttt{hasSource}, \texttt{hasScore}, and \texttt{tookPlaceAt}. We can generalize this pattern for representing all dimensions of meta knowledge as follows.

\textbf{Singleton Graph Pattern.} In general, given a fact \((s, p, o)\), let \texttt{p?id=i} be the singleton property representing this fact in one particular context, \texttt{mp} be the meta property, \texttt{mv} be the value of meta property, the set of triples forming a singleton graph pattern asserting meta knowledge for this fact is provided in Table 2.2. We will use this singleton graph pattern for querying meta knowledge in Section 5.

2.1.3 Enforcing the singleton-ness of property instances

If the property \texttt{isMarriedTo?id=1} is asserted in another triple such as \texttt{BarackObama isMarriedTo?id=1 MichelleObama}, this together with the existing assertion \texttt{isMarriedTo?id=1 hasStart 1965-11-22}, would imply the marriage date of the Obamas to be 1965-11-22, which is not true. In order to avoid this, we need to ensure the singleton property \texttt{isMarriedTo?id=1} occurs as a property in only one triple.

This constraint has to be enforced for all URIs of singleton property instances. Data publishers may combine their URI prefix, the generic property name and the timestamp when the instance is created into the URI of a singleton property to make it unique. However, there are still cases where two instances may share the same URI. Therefore, data publishers may employ the Universally Unique Identifier (UUID) [Leach et al. 2005], which is also supported by SPARQL and various programming languages, to ensure the singleton-ness of their property instances. The validation of this uniqueness constraint is straightforward, by counting the number of triple occurrences per singleton property. As the current RDF syntax does not allow blank nodes as properties, we do not represent singleton properties as blank nodes, although one advantage of using blank nodes in the property is providing the completeness for deduction rules [Mallea et al. 2011].
2.2 Model-Theoretic Semantics

This section explains how the singleton property can fit well with the existing formal semantics. We reuse the model-theoretic semantics described in [Hitzler et al. 2011] with three levels of interpretation: simple, RDF and RDFS. For each interpretation we add additional criteria for supporting the singleton property. While we explain the new vocabulary elements in detail, elements without further explanation remain as they are in the original model-theoretic semantics described in [Hitzler et al. 2011].

Given a vocabulary $V$, the original simple interpretation $I$ consists of:

- $IR$, a non-empty set of resources, alternatively called domain or universe of discourse of $I$,
- $IP$, the set of generic properties of $I$,
- $I_{EXT}$, a function assigning to each property a set of pairs from $IR$
  \[ I_{EXT} : IP \rightarrow 2^{IR \times IR} \] where $I_{EXT}(p)$ is called the extension of property $p$,
- $IS$, a function, mapping URIs from $V$ into the union set of $IR$ and $IP$,
- $IL$, a function from the typed literals from $V$ into the set of resources $IR$,
- $LV$, a subset of $IR$, called the set of literal values.

We define $IPs$ as a set of singleton properties and $I_{S,EXT}(p_s)$ as the function mapping a singleton property into a pair of resources.

**Simple interpretation** of vocabulary $V$ is an original simple interpretation $I$ of the vocabulary $V \cup V_{SIM}$ that satisfies the additional criteria:

- $IPs$, called the set of singleton properties of $I$, as a subset of $IR$,
- $I_{S,EXT}(p_s)$, the function mapping a singleton property to a pair of resources. $I_{S,EXT} : IPs \rightarrow IR \times IR$.

Note that the mapping function $I_{S,EXT}$ is not a one-to-one mapping; multiple singleton properties may be mapped to the same pair of entities.

**RDF interpretation** of a vocabulary $V$ is a simple interpretation $I$ of the vocabulary $V \cup V_{RDF}$ that satisfies the criteria from the original RDF interpretation and the following criteria:

- $x_s \in IPs$ if \( \langle x_s, rdf: \text{SingletonProperty}\rangle \in I_{EXT} \)
  \( (rdf: type) \), a singleton property $x_s$ is an instance of class SingletonProperty if they are interconnected by the property rdf:type.
Table 2.3: Singleton property approach representing facts and their temporal assertions

<table>
<thead>
<tr>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>BobDylan</td>
<td>isMarriedTo</td>
<td>SaraLownds</td>
</tr>
<tr>
<td>BobDylan</td>
<td>isMarriedTo?id=1</td>
<td>SaraLownds</td>
</tr>
<tr>
<td>isMarriedTo?id=1</td>
<td>rdf:singletonPropertyOf</td>
<td>isMarriedTo</td>
</tr>
<tr>
<td>isMarriedTo?id=1</td>
<td>hasStart</td>
<td>1965-11-22</td>
</tr>
<tr>
<td>isMarriedTo?id=1</td>
<td>hasEnd</td>
<td>1977-06-29</td>
</tr>
<tr>
<td>BobDylan</td>
<td>isMarriedTo</td>
<td>CarolDennis</td>
</tr>
<tr>
<td>BobDylan</td>
<td>isMarriedTo?id=2</td>
<td>CarolDennis</td>
</tr>
<tr>
<td>isMarriedTo?id=2</td>
<td>rdf:singletonPropertyOf</td>
<td>isMarriedTo</td>
</tr>
<tr>
<td>isMarriedTo?id=2</td>
<td>hasStart</td>
<td>1986-06-##</td>
</tr>
<tr>
<td>isMarriedTo?id=2</td>
<td>hasEnd</td>
<td>1992-10-##</td>
</tr>
</tbody>
</table>

- \( x_s \in IPs \) if \( \langle x_s, x^T \rangle \in I_{EXT}(\text{rdf: singletonPropertyOf}^T) \), \( x \in IP \). A singleton property \( x_s \) is an instance of a generic property \( x \) if they are interconnected by the property rdf:singletonPropertyOf, where \( x \) is called a \textit{generic property}. Since the singletonPropertyOf is defined here, we use rdf:singletonPropertyOf from now on.

- if \( x_s \in IPs \) then \( \exists! \langle u, v \rangle : \langle u, v \rangle = I_{S,EXT}(x_s^T) \), and \( u, v \in IR \). This enforces the singleton-ness for the property instances.

Given the set of triples with singleton properties and their temporal assertions in Table 2.3, let \( V_{EX} \) be the vocabulary consisting of all the names of subjects, predicates and objects in those triples, the RDF interpretation of the vocabulary \( V_{EX} \) is provided in Table 3.1.2.

In the RDFS interpretation, we will reuse the function 
\[ I_{CEXT} : IR \to 2^{IR} \text{ where } I_{CEXT}(y) \text{ is called (class) extension of } y, I_{CEXT}(y) = \{ x \mid \forall x \in IR : \langle x, y \rangle \in I_{EXT}(\text{rdf: type}^T) \}\]

**RDFS interpretation** of a vocabulary \( V \) is an RDF interpretation \( \mathcal{I} \) of the vocabulary \( V \cup V_{RDF S} \) that satisfies criteria from the original RDFS interpretation and the following criteria:

- \( \langle \text{rdf: SingletonProperty}^T, \text{rdfs: Class}^T \rangle \in I_{EXT} \)
  \( \langle \text{rdf: type}^T \rangle \).

\( \text{rdf:SingletonProperty} \) is defined as a class. The extension of \( \text{rdf:SingletonProperty} \) is the set \( IPs \) of all singleton properties, or
Table 2.4: RDF interpretation for the vocabulary $V_{EX}$ from Table 2.3

<table>
<thead>
<tr>
<th>$I_S$</th>
<th>$\rightarrow$</th>
</tr>
</thead>
<tbody>
<tr>
<td>BobDylan</td>
<td>$\alpha$</td>
</tr>
<tr>
<td>SaraLownds</td>
<td>$\beta$</td>
</tr>
<tr>
<td>CarolDennis</td>
<td>$\gamma$</td>
</tr>
<tr>
<td>$\text{isMarriedTo}$</td>
<td>$\delta$</td>
</tr>
<tr>
<td>$\text{isMarriedTo?id=1}$</td>
<td>$\theta$</td>
</tr>
<tr>
<td>$\text{isMarriedTo?id=2}$</td>
<td>$\lambda$</td>
</tr>
<tr>
<td>$\text{hasStart}$</td>
<td>$\sigma$</td>
</tr>
<tr>
<td>$\text{hasEnd}$</td>
<td>$\phi$</td>
</tr>
</tbody>
</table>

| $IR$                      | $\{ \alpha, \beta, \gamma, \delta, \theta, \lambda \}$ |
| $IP$                      | $\{ \delta, \theta, \lambda, \sigma, \phi \}$       |

| $I_{EXT}$                 | $\theta \mapsto \{ \langle \alpha, \beta \rangle \}$ |
|                          | $\lambda \mapsto \{ \langle \alpha, \gamma \rangle \}$ |
|                          | $\sigma \mapsto \{ \langle \theta, 1965-11-22 \rangle, \langle \lambda, 1986-06-## \rangle \}$ |
|                          | $\phi \mapsto \{ \langle \theta, 1977-06-29 \rangle, \langle \lambda, 1992-10-## \rangle \}$ |
| $\text{rdf:singletonPropertyOf}$ | $\mapsto \{ \langle \theta, \delta \rangle, \langle \lambda, \delta \rangle \}$ |
| $\delta$                 | $\mapsto \{ \langle \alpha, \beta \rangle, \langle \alpha, \gamma \rangle \}$ |

| $IPs$                     | $\{ \theta, \lambda \}$ |
| $I_{S,EXT}$               | $\theta \mapsto \langle \alpha, \beta \rangle$ |
|                          | $\lambda \mapsto \langle \alpha, \gamma \rangle$ |
2.3 RDF 1.1 Singleton Property Model-Theoretic Semantics

\[ IPs = I_{CEXT}(\text{rdf} : \text{SingletonProperty}^T). \]

- \( \langle \text{rdf} : \text{SingletonProperty}^T, \text{rdfs} : \text{Resource}^T \rangle \in I_{EXT} \)
  \( (\text{rdfs} : \text{subClassOf}^T) \), this causes \( IPs \subset IR \),
  every singleton property is an RDF resource.

- if \( \langle x_s, x \rangle \in I_{EXT}(\text{rdf} : \text{singletonPropertyOf}^T) \), \( x_s \in IPs \), and \( x \in IP \), then \( I_{S,EXT}(x_s) \in I_{EXT}(x) \).
  \( I_{EXT}(x) \) is called property extension of the generic property \( x \).
  The set of singleton properties connected to that property via rdf:singletonPropertyOf is a sub set of the property extension of its generic property.

- Let \( \langle x_s, x \rangle \in I_{EXT}(\text{rdf} : \text{singletonPropertyOf}^T) \), and \( \langle x, y \rangle \in I_{EXT}(\text{rdfs} : \text{domain}) \), if \( \langle u, v \rangle \in I_{S,EXT}(x_s) \), then \( u \in I_{CEXT}(y) \) where \( I_{CEXT}(y) \) is the class extension of \( y \).
  A singleton property shares domain with its generic property.

- Let \( \langle x_s, x \rangle \in I_{EXT}(\text{rdf} : \text{singletonPropertyOf}^T) \), and \( \langle x, y \rangle \in I_{EXT}(\text{rdfs} : \text{range}^T) \), if \( \langle u, v \rangle = I_{S,EXT}(x_s) \), then \( v \in I_{CEXT}(y) \).
  A singleton property also shares range with its generic property.

Since the singleton property semantics \cite{Nguyen et al. 2014} was developed with regards to RDF 1.0 \cite{Hayes and McBride 2004}, here we revise it to make it compatible with RDF 1.1 \cite{Hayes and Patel-Schneider}.

In RDF 1.1 \cite{Hayes and Patel-Schneider}, the simple interpretation \( I \) is a structure consisting of the following elements. \( IR \) is a non-empty set of resources, alternatively called domain or universe of discourse of \( I \). \( IP \) is the set of properties of \( I \). The mapping function \( I_{EXT} \) assigns each property a set of pairs from \( IR \). \( I_{EXT} : IP \rightarrow 2^{IR \times IR} \) where \( I_{EXT}(p) \) is called the extension of property \( p \).

\( IS \) maps IRIs into \( IR \cup IP \). \( IL \) maps literals into \( IR \).

In RDF 1.1 \cite{Hayes and Patel-Schneider}, the interpretation \( I \) is also treated as a function from expressions to elements of the universe and truth values. It satisfies the following semantic conditions for ground graphs. If \( e \) is a literal then \( I(e) = IL(e) \). If \( e \) is an IRI then \( I(e) = IS(e) \). If \( e \) is a ground triple \( spo \), then \( I(e) = \text{true} \) if \( I(p) \) is in IP and the pair \( (I(s), I(o)) \) is in \( I_{EXT}(I(p)) \), otherwise \( I(e) = \text{false} \). If \( e \) is a ground RDF graph then \( I(e) = \text{false} \) if \( I(e') = \text{false} \) for some triple \( e' \) in \( e \), otherwise \( I(e) = \text{true} \).

According to \cite{Nguyen et al. 2014}, the RDF interpretation \( I \) recognizing singleton properties satisfies the semantic conditions in \cite{Hayes and Patel-Schneider} and the following ones. \( IPs \), called
the set of singleton properties of I, is a subset of IP. The singleton mapping function \( I_{S,EXT}(p_s) \) assigns each singleton property to a pair of resources. \( I_{S,EXT} : IPs \rightarrow IR \times IR \). A singleton property \( p_s \in IPs \) iff \( \langle p_s, I(rdf:SingletonProperty) \rangle \in I_{EXT}(I(rdf:type)) \). If \( p_s \in IPs \) then \( \exists! \langle u, v \rangle : I_{S,EXT}(p_s) = \langle u, v \rangle \), and \( u, v \in IR \). Therefore, \( I_{EXT}(p_s) = \{ \langle u, v \rangle | I_{S,EXT}(p_s) = \langle u, v \rangle \} \), or \( I_{EXT}(p_s) = \{ I_{S,EXT}(p_s) \} \). This enforces the singleton-ness for the singleton property triples. For the example at hand, we have

\[
I_{S,EXT}(I(type?id=1)) = \langle I(chadHurley), I(youtubeEmp) \rangle
\]

\[
I_{EXT}(I(type?id=1)) = \{ I(chadHurley), I(youtubeEmp) \}. \]

As the singleton property \( type#1 \) is also a property, its property extension is a singleton set, which has only one element.

In order to enforce the singleton-ness on the URIs of singleton properties, they can be generated by appending the unique number or string to the URIs of their generic property.

The RDFS interpretation \( I \) recognizing singleton properties satisfies the semantic conditions in [Hayes and Patel-Schneider; Nguyen et al. 2014].

The syntax and semantics of singleton properties described here are sufficient to allow us to represent contextualized statements in the machine-processable form. However, it is unable to address the questions discussed in Section 1. Therefore, to enable the reasoning support for the RDF singleton property model, we describe how we can extend this formalism in Chapter 3.

### 2.4 Related Work

Many approaches have been proposed to address the problem of representing and querying statements about statements. We can divide these approaches into three main categories based on their form: triples [Sahoo et al. 2011; Sahoo et al. 2010], quadruples [Carroll et al. 2005; Flouris et al. 2009; Straccia et al. 2010] and quintuples [Schueler et al. 2008] based on the number of elements in the structure each approach employs. Each approach reflects one perspective on how meta knowledge for triples could add elements into tuples. We will discuss the contribution of each approach, and how our approach fits into the scheme.

**Triples.** Representing different dimensions of meta knowledge for triples using RDF triples in order to retain the compatibility and interoperability with existing Semantic Web knowledge bases, tools, languages and methods is the main goal of this approach. The reification approach [Manola et al. 2004; Hayes and McBride 2004] allows meta knowledge to be asserted from reified statements.

Sahoo et al. [Sahoo et al. 2010] propose the PaCE approach for representing the provenance of a triple by creating different instances for its subject, property and object for different contexts and asserting provenance for those instances. The source of the triple is derived from the source of its
individual components.

**Quadruples.** In the reification approach, we need to create statement instances and indicate the subject, property, and object for those instances. Intuitively, this verbosity can be avoided by adding one more element into a triple to make it a quadruple. Named graph [Carroll et al. 2005] and other work on top of named graph such as [Flouris et al. 2009] [Pediaditis et al. 2009] follow the approach, using the fourth element to represent the provenance of a set of triples. Although technically we can restrict a named graph to a single triple and use it to assert meta knowledge for that triple, it does not naturally serve this purpose because originally the named graph was designed for representing provenance and trust of a set of triples.

Straccia et al. [Straccia et al. 2010] also annotate the RDF triple using meta knowledge such as temporal and certainty for RDF triples. We classify this approach under quadruples because it annotates every RDF triple with an annotation term. It proposes a new algebraic structure with well-defined operators for manipulating meta information. This approach is followed up with the RDFS reasoning supported by [Damásio and Ferreira 2011]. Our approach differs from this approach in that we leverage RDF triples for the representation of meta knowledge assertions, allowing them to be queried and entailed using existing languages and tools, while this approach does not.

**Quintuples.** The RDF+ approach [Schueler et al. 2008] defines the abstract syntax of RDF+ statement as a quintuple by extending the named graph quad with a statement identifier. The statement identifier is used as the subject of the meta knowledge assertion, which is an RDF triple. Since the formal semantics is defined in RDF+, mappings from RDF to RDF+ and vice versa have to be made. Additionally, the SPARQL syntax and semantics have to be extended to support querying of RDF+. First, while a statement identifier is defined in the RDF+ statement which is a quintuple, our approach represents singleton property instances in RDF triples. As a result, our approach does not need any mapping while the RDF+ does. Secondly, our approach does not require any extension to the syntax or semantics of SPARQL because it is completely compatible with SPARQL.
3

Reasoning for Contextualized Knowledge Graphs

This chapter describes the new concepts for the singleton property model and formalizes them in the model-theoretic semantics. The extended semantics is utilized for deriving new sets of inference rules.

3.0.1 Introduction

The singleton property (SP) \cite{Nguyen et al. 2014} is a specific property instance that represents a unique relationship under a specific context. For example, given the contextualized statement $C_1$ “(chadHurley, type, youtubeEmp): [2005, 2010] and score 0.8” in Table 1.3, the singleton property model represents it with 5 triples $T_1, T_2, T_3, T_4$, and $T_5$ in Table 3.1. The singleton property $\text{type?id=1}$ uniquely represents the $\text{type}$ relationship between $\text{chadHurley}$ and $\text{youtubeEmp}$ in the triple $T_1$. The property $\text{sing}$ (singletonPropertyOf) connects the singleton property $\text{type?id=1}$ to the property $\text{type}$ in the triple $T_2$. The singleton property $\text{type?id=1}$ can be asserted with contextual information (time [2005, 2010] and confidence score 0.8) about the relationships as shown in the triples $T_3, T_4$, and $T_5$.

Comparing the representations from Tables 1.3 and 1.8, we observe several questions that have not yet been addressed by the model theory of \cite{Nguyen et al. 2014}. Here we analyze some of the questions and motivate our approach with the examples at hand.

Validating singletonPropertyOf. Although we \cite{Nguyen et al. 2014} created the property singletonPropertyOf to connect the property $\text{type?id=1}$ and the property $\text{type}$, we did not formally define the semantics of this property. What should be the domain and range of this property? What is the relationship between the triple $T_1$ (chadHurley, type?id=1, youtubeEmp) and the orig-
Table 3.1: Singleton property model representation for the contextualized knowledge base from Table 1.3 (sp: subPropertyOf, and sc: subClassOf)

<table>
<thead>
<tr>
<th>Triple</th>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>chadHurley</td>
<td>type?id=1</td>
<td>youtubeEmp</td>
</tr>
<tr>
<td>T2</td>
<td>type?id=1</td>
<td>singletonPropertyOf</td>
<td>type</td>
</tr>
<tr>
<td>T3</td>
<td>type?id=1</td>
<td>from</td>
<td>2005</td>
</tr>
<tr>
<td>T4</td>
<td>type?id=1</td>
<td>to</td>
<td>2010</td>
</tr>
<tr>
<td>T5</td>
<td>type?id=1</td>
<td>score</td>
<td>0.8</td>
</tr>
<tr>
<td>T6</td>
<td>youtubeEmp</td>
<td>sc?id=2</td>
<td>googleEmp</td>
</tr>
<tr>
<td>T7</td>
<td>sc?id=2</td>
<td>singletonPropertyOf</td>
<td>sc</td>
</tr>
<tr>
<td>T8</td>
<td>sc?id=2</td>
<td>from</td>
<td>2006</td>
</tr>
<tr>
<td>T9</td>
<td>sc?id=2</td>
<td>to</td>
<td>2017</td>
</tr>
<tr>
<td>T10</td>
<td>sc?id=2</td>
<td>score</td>
<td>0.9</td>
</tr>
<tr>
<td>T11</td>
<td>chadHurley</td>
<td>ceo?id=3</td>
<td>youtube</td>
</tr>
<tr>
<td>T12</td>
<td>ceo?id=3</td>
<td>singletonPropertyOf</td>
<td>ceo</td>
</tr>
<tr>
<td>T13</td>
<td>ceo?id=3</td>
<td>from</td>
<td>2005</td>
</tr>
<tr>
<td>T14</td>
<td>ceo?id=3</td>
<td>to</td>
<td>2010</td>
</tr>
<tr>
<td>T15</td>
<td>ceo?id=3</td>
<td>score</td>
<td>0.7</td>
</tr>
<tr>
<td>T16</td>
<td>ceo</td>
<td>sp?id=4</td>
<td>worksFor</td>
</tr>
<tr>
<td>T17</td>
<td>sp?id=4</td>
<td>singletonPropertyOf</td>
<td>sp</td>
</tr>
<tr>
<td>T18</td>
<td>sp?id=4</td>
<td>derivedFrom</td>
<td>work</td>
</tr>
<tr>
<td>T19</td>
<td>sp?id=4</td>
<td>score</td>
<td>1.0</td>
</tr>
<tr>
<td>T20</td>
<td>worksFor</td>
<td>sp?id=5</td>
<td>member</td>
</tr>
<tr>
<td>T21</td>
<td>sp?id=5</td>
<td>singletonPropertyOf</td>
<td>sp</td>
</tr>
<tr>
<td>T22</td>
<td>sp?id=5</td>
<td>derivedFrom</td>
<td>work</td>
</tr>
<tr>
<td>T23</td>
<td>sp?id=5</td>
<td>score</td>
<td>1.0</td>
</tr>
<tr>
<td>T24</td>
<td>ceo</td>
<td>domain?id=6</td>
<td>Person</td>
</tr>
<tr>
<td>T25</td>
<td>domain?id=6</td>
<td>singletonPropertyOf</td>
<td>domain</td>
</tr>
<tr>
<td>T26</td>
<td>domain?id=6</td>
<td>derivedFrom</td>
<td>work</td>
</tr>
<tr>
<td>T27</td>
<td>domain?id=6</td>
<td>score</td>
<td>1.0</td>
</tr>
<tr>
<td>T28</td>
<td>ceo</td>
<td>range?id=7</td>
<td>Company</td>
</tr>
<tr>
<td>T29</td>
<td>range?id=7</td>
<td>singletonPropertyOf</td>
<td>range</td>
</tr>
<tr>
<td>T30</td>
<td>range?id=7</td>
<td>derivedFrom</td>
<td>work</td>
</tr>
<tr>
<td>T31</td>
<td>range?id=7</td>
<td>score</td>
<td>1.0</td>
</tr>
</tbody>
</table>
inal triple (chadHurley, type, youtubeEmp)? What do they share in common and what are the differences between them?

**Hierarchy inferences.** Given a class hierarchy, what kinds of inferences can be performed for the singleton triples? For example, the original triples (chadHurley, type, youtubeEmp) and (youtubeEmp, sc, googleEmp) infer the new triple (chadHurley, type, googleEmp). What do their corresponding singleton triples \( T_1 \) (chadHurley, type?id=1, youtubeEmp) and \( T_6 \) (youtubeEmp, sc?id=2, googleEmp) entail? Similarly, for the property hierarchy inference, the original triples (ceo, sp, worksFor) and (worksFor, sp, member) entail the new triple (ceo, sp, member). What do the singleton triples (ceo, sp?id=4, worksFor) and (worksFor, sp?id=5, member) entail?

**Contextual inferences.** Given the two contextualized statements

\[ C_1 \quad \text{“(chadHurley, type, youtubeEmp) : [2005,2010]”} \quad \text{and} \quad C_2 \quad \text{“(youtubeEmp, sc, googleEmp) : [2006,2011]”}, \]

we can entail the new contextualized triple (chadHurley, type, googleEmp):[2006,2010] using the inference rules for the annotated RDF from Zimmermann et al. 2012 Udrea et al. 2010. Since these two contextualized statements are represented using the set of SP triples \( T_1 \) to \( T_{10} \), what can be concluded from this set of SP triples?

In the dissertation, we investigate the questions above and propose a principled approach to derive the inference rules for the singleton property model. We identify the new concepts and formalize their semantic associations with singleton properties in a model theory (Section 3.1). Based on the semantic associations, we develop a set of inference rules with their proofs derived from the model theory (Section 3.2). We applied the inference rules from the annotated RDF work Zimmermann et al. 2012 into the SP representation of RDFS rules to derive the set of contextual rules (for time-interval and fuzzy). The formalism allows the proposed reasoning mechanism to be implemented in Semantic Web reasoners and allow the contextualized KBs to be reasoned with via these reasoners.

Our reasoning mechanism does not limit the support to only contextualized KBs with SP representation. Any contextualized KBs available in other representations such as the named graph or RDF reification can also be reasoned with our mechanism. These KBs need to be transformed from non-SP to SP representation. To support such a transformation, we need to develop a tool that transforms the syntax and adds explicit semantic relationships between the statements and their context. This tool, which is described in Chapter 6, allows existing contextualized KBs such as Bio2RDF with the named graph provenance to be transformed and inferred with our mechanism.
3.1 Conceptual Modeling

Here we motivate the need for new concepts and then formalize them in terms of RDF and RDFS interpretations. The new concepts and their formalization will be leveraged to derive the proofs for the inference rules in the next section.

3.1.1 Concepts and Mapping Functions

Motivating example. For the contextualized statement $C_3$ that “(chadHurley, ceo, youtube) : [2005,2010] and score 0.7”, we can represent it using the list of singleton property triples \{$T_{11}, T_{12}, T_{13}, T_{14}, T_{15}$\} as described in Table 3.1.

\begin{align*}
T_{11}: & \text{chadHurley ceo?id=3 youtube .} \\
T_{12}: & \text{ceo?id=3 singletonPropertyOf ceo .}
\end{align*}

If we decontextualize the statement $C_3$, we have the decontextualized statement (chadHurley, ceo, youtube), represented in the triple $T_1$ as follows.

\begin{align*}
T_1: & \text{chadHurley ceo youtube .}
\end{align*}

The predicate ceo of the triple $T_1$ is connected to the singleton property ceo?id=3 via the predicate singletonPropertyOf in the triple $T_{12}$.

We observe that the singleton property ceo?id=3 and the triple $T_{11}$ have been formalized by the class SingletonProperty and the singleton property extension $I_{S,EXT}$, respectively. However, the property ceo, the triple $T_{12}$, and the relationship between the two triples $T_{11}$ and $T_1$ have not yet been formalized. Therefore, we here formalize them.

First, we formalize the semantics of the property ceo.

Generic/Decontextualized Property. From the triples $T_1, T_{11}$, and $T_{12}$, we observe that the property ceo differs from the singleton property in that the former is decontextualized while the latter is contextualized. However, this difference has not been captured. Moreover, this property ceo differs from other regular properties such as from in that it has contextualized property instances as in $T_{12}$ and it can be decontextualized as in $T_1$. To capture this distinction, we propose a class GenericProperty for all the decontextualized properties. Alternatively, to distinguish the generic property from the singleton property, generic property and singleton property are also called decontextualized property and contextualized property, respectively.

Why do we need to distinguish the property types based on their association with context? Regular triples or triples describing triples are all represented in the form of triples. Without annotations specifying the types of the triples, it is ambiguous to identify which triple is associated/dissociated with a context, or even context-agnostic.
3.1. CONCEPTUAL MODELING

Second, we formalize the semantics of the triple \( T_{12} \) by defining a new mapping function between the generic properties and singleton properties. We reuse the model-theoretic semantics from Section 2.3.

**Definition 1.** Given an interpretation \( I \) on the vocabulary \( V_{RDF} \), a generic instance function \( I_G \) is a binary relation that maps a generic property to a set of its singleton properties. Formally, let \( IP_g \subseteq IP \) be the set of generic properties. We define the function \( I_G : IP_g \rightarrow 2^{IP_s} \) such that \( I_G(p_g) = \{ p_s \mid \langle p_s, p_g \rangle \in I_{EXT}(I(rdf:singletonPropertyOf)) \} \).

For example, \( ceo \) is a generic property and it has a singleton property \( ceo?id=3 \), then \( I_G(I(ceo)) = \{ I(ceo?id=3) \} \).

Third, we formalize the relationship between the generic triple \( T_1 \) and the singleton triple \( T_{11} \).

**Generic/Decontextualized Triple.** The triple having a generic property is called generic triple, or decontextualized triple. In our example, \( T_1 \) is a generic/decontextualized triple. One generic or decontextualized triple can be derived from multiple singleton triples. Therefore, we can also consider a singleton triple as a triple instance of a generic triple under one specific context.

Here we define a new mapping function called generic property extension to capture this semantics.

**Definition 2.** Given an interpretation \( I \), a generic mapping function \( I_{G,EXT} \) is a binary relation that maps a generic property to a set of pairs of resources. Formally, \( I_{G,EXT} : IP_g \rightarrow 2^{IR \times IR} \) such that \( I_{G,EXT}(p_g) = \{ I_{S,EXT}(p_s) \mid p_s \in I_G(p_g) \} \).

From the example at hand,
\[
I_{G,EXT}(I(ceo)) = \{ \langle I(chadHurley), I(youtube) \rangle \}.
\]

Here we extend the RDF interpretation from Section 2.3 by incorporating the new concepts and mapping functions defined above.

### 3.1.2 Extended RDF Interpretation

Let \( V_{RDF} \) be the vocabulary of RDF level. Besides the original terms, the vocabulary \( V_{RDF} \) also contains the terms SingletonProperty, GenericProperty, singletonPropertyOf for their semantics to be interpreted in the model theory. Since they are defined and interpreted at the RDF level, we use the prefix rdf for their URIs.

**Model-theoretic semantics.** Given a vocabulary \( V \), the extended RDF interpretation \( I \) on \( V \cup V_{RDF} \) satisfies the criteria described in Section 2.3 and the following elements and semantic conditions.
3.1. CONCEPTUAL MODELING

1. $IP_g$, called the set of generic properties of $I$, is a subset of $IP$, $IP_s \cap IP_g = \emptyset$. Note that the set $IP_g$ is not the complement of the set $IP_s$ because of the set of regular properties $IP_r = IP \setminus (IP_g \cup IP_s)$. A regular property is neither a singleton nor a generic property.

2. Definition of a generic property:
   
   \[ x_g \in IP_g \iff (x_g, I(rdf:GenericProperty)) \in I_{EXT}(I(rdf:type)). \]

3. $IG$, a function assigning a generic property to a set of singleton properties as defined in Definition 1.
   
   \[ IG(x_g) = \{ x_s \mid (x_s, x_g) \in I_{EXT}(I(rdf:singletonPropertyOf)) \}. \]

4. Generic mapping extension: $IG_{EXT}(x_g)$ is called a generic mapping extension of the generic property $x_g$ (Definition 2).
   
   \[ IG_{EXT}(x_g) = \{ IS_{EXT}(x_s) \mid x_s \in IG(x_g) \}. \]

3.1.3 Extended RDFS Interpretation

Here we extend the formal semantics at the RDFS level by adding the semantic conditions.

The RDFS interpretation $I$ on the vocabulary $V_{RDFS}$ recognizing singleton property reasoning satisfies the semantic criteria described in Section 3.1.2 and 2.3, and the following additional criteria.

1. Class $rdf:GenericProperty$:

   \[ \langle I(rdf:GenericProperty), I(rdfs:Class) \rangle \in I_{EXT}(I(rdf:type)). \]

2. Every generic property is a resource:

   \[ \langle I(rdf:GenericProperty), I(rdfs:Resource) \rangle \in I_{EXT}(I(rdfs:subClassOf)), \text{this causes } IP_g \subseteq IR. \]

3. \( \langle I(rdfs:singletonPropertyOf), I(rdf:SingletonProperty) \rangle \in I_{EXT}(I(rdfs:domain)) \). This condition is easily derived from Definition 1.

4. \( \langle I(rdfs:singletonPropertyOf), I(rdf:GenericProperty) \rangle \in I_{EXT}(I(rdfs:range)) \). This condition is also easily derived from Definition 1.

5. If $\langle x, y \rangle \in I_{EXT}(I(rdfs:subPropertyOf))$, then $I_{EXT}(x) \subseteq I_{EXT}(y)$, $IG(x) \subseteq IG(y)$, and $IG_{EXT}(x) \subseteq IG_{EXT}(y)$. We put more constraints on the property hierarchy to include the new extension mappings.
3.2 SP Inference Rules

Here we propose a set of inference rules based on the semantics formalized in the RDF and RDFS interpretations from Section 3.1.

3.2.1 Validating rdf:singletonPropertyOf

**Motivation.** We proposed the use of the property `singletonPropertyOf` in [Nguyen et al. 2014] but did not formalize the connection. They did not put constraints on the domain and range of this property for them to be validated. We address this issue by adding constraints on the use of this `rdf:singletonPropertyOf` property.

**Proposition 1.** *(Infer singleton and generic property)*

Given an interpretation $I$, if $\langle x_s, x_g \rangle \in I_{\text{EXT}}(I(\text{rdf:singletonPropertyOf}))$ then $x_s \in \text{IPs}$ and $x_g \in \text{IPg}$.

\[
\frac{x_s \text{ rdf:singletonPropertyOf } x_g}{x_s \text{ rdf:type SingletonProperty}} \quad \text{(sp-1)}
\]

\[
\frac{x_s \text{ rdf:singletonPropertyOf } x_g}{x_g \text{ rdf:type GenericProperty}} \quad \text{(sp-2)}
\]

**Proof.** This can be trivially derived from their definition and from the domain and range of `rdf:singletonPropertyOf` defined in the extended RDFS interpretation.

**Proposition 2.** *(rdf:singletonPropertyOf is irreflexive)*

Given an interpretation $I$, if $x_s \in \text{IPs}$ and $x_g \in \text{IPg}$ then $(x_s, x_s) \notin I_{\text{EXT}}(I(\text{rdf:singletonPropertyOf}))$ and $(x_g, x_g) \notin I_{\text{EXT}}(I(\text{rdf:singletonPropertyOf}))$.

**Proof.** Assume that $\langle x_s, x_s \rangle \in I_{\text{EXT}}(I(\text{rdf:singletonPropertyOf}))$, then $x_s \in \text{IPg}$. If $\langle x_g, x_g \rangle \in I_{\text{EXT}}(I(\text{rdf:singletonPropertyOf}))$, then $x_g \in \text{IPs}$. Since $x_s \in \text{IPs}$ and $x_s \in \text{IPg}$, $x_g \in \text{IPs}$ and $x_g \in \text{IPg}$, we have $\text{IPs} \cap \text{IPg} = \{x_s, x_g\}$.

This can not happen because $\text{IPs} \cap \text{IPg} = \emptyset$.

Note that `rdf:singletonPropertyOf` $\notin \text{IPs}$. It is because if `rdf:singletonPropertyOf` $\in \text{IPs}$, its occurrences as a property in multiple triples will violate the singleton-ness definition of the singleton property.
3.2. SP INFEERENCE RULES

3.2.2 Inferring Generic/Decontextualized Triple

As discussed in the motivating example of Section 3.1, here we formalize the relationship between the contextualized triples and their decontextualized triples.

**Proposition 3.** *(Generic triple derivation)*

Given an interpretation \( I \), if \( I_{S,\text{EXT}}(x_s) = \langle u, v \rangle \) and \( \langle x_s, x_g \rangle \in I_{\text{EXT}}(I(\text{rdf:singletonPropertyOf})) \)

\[
\frac{x_s \text{ rdf:singletonPropertyOf } x_g \quad u \quad x_s \quad v}{u \quad x_g \quad v} \quad (\text{sp-3})
\]

**Proof.** \( \langle x_s, x_g \rangle \in I_{\text{EXT}}(I(\text{rdf:singletonPropertyOf})) \) implies (i): \( I_{S,\text{EXT}}(x_s) \in I_{G,\text{EXT}}(x_g) \). The combination of \( I_{S,\text{EXT}}(x_s) = \langle u, v \rangle \) and (i) implies \( \langle u, v \rangle \in I_{G,\text{EXT}}(x_g) \). This shows how the generic triple \( u \ x_g \ v \) obtained from \( \langle u, v \rangle \in I_{G,\text{EXT}}(x_g) \) can be derived from its singleton graph pattern.

For the example at hand, the triples \( T_{11} \) (chadHurley, ceo?id=3, youtube) and \( T_{12} \) (ceo?id=3, singletonPropertyOf, ceo) derives the generic or decontextualized triple \( T_1 \) (chadHurley, ceo, youtube).

The set of triples in Table 3.1 derive the following generic triples.

- \( G_1 \): chadHurley type youtubeEmp .
- \( G_2 \): youtubeEmp sc googleEmp .
- \( G_3 \): chadHurley ceo youtube .
- \( G_4 \): ceo sp worksFor .
- \( G_5 \): worksFor sp member .
- \( G_6 \): ceo domain Person .
- \( G_7 \): ceo range Company .

3.2.3 Inferring via Property Hierarchy: Inheritance Rules

**Proposition 4.** *(Property hierarchy)*

Given an interpretation \( I \), if \( \langle x_s, x \rangle \in I_{\text{EXT}}(I(\text{rdf:singletonPropertyOf})) \), and \( \langle x, y \rangle \in I_{\text{EXT}}(I(\text{rdfs:subPropertyOf})) \), then \( \langle x_s, y \rangle \in I_{\text{EXT}}(I(\text{rdf:singletonPropertyOf})) \).

\[
\frac{x_s \text{ rdf:singletonPropertyOf } x \quad x \text{ rdf:s:subPropertyOf } y \quad x_s \text{ rdf:singletonPropertyOf } y}{x_s \text{ rdf:singletonPropertyOf } y} \quad (\text{sp-4})
\]

**Proof.** \( \langle x_s, x \rangle \in I_{\text{EXT}}(I(\text{singletonPropertyOf})) \) implies (ii): \( x_s \in I_G(x) \).
\[ \langle x, y \rangle \in I_{EXT}(I(\text{subPropertyOf})) \text{ implies (iii): } I_G(x) \subseteq I_G(y) \text{ according to RDFS condition (5). (ii) and (iii) derive (iv): } x_s \in I_G(y). \] In other words, \( x_s \) is a singleton property of \( y \), or \( \langle x_s, y \rangle \in I_{EXT}(I(\text{rdf:SingletonPropertyOf})) \).

For the example at hand, the triples \( G_4 \) (Section 3.2.2) and \( T_{11} \) infer the new triple (ceo?id=3, singletonPropertyOf, worksFor).

Furthermore, we derived the domain and range of a singleton property from its generic property in \cite{Nguyen2014}. Here we add the two rules derived from their RDFS interpretation.

\[
\frac{u \cdot x \cdot s \cdot v . \ x \cdot s \ \text{rdf:singletonPropertyOf} \ x \cdot g \ . \ x \cdot g \ \text{rdfs:domain} \ y}{u \ \text{rdf:type} \ y} \quad \text{(sp-5)}
\]

\[
\frac{u \cdot x \cdot s \cdot v . \ x \cdot s \ \text{rdf:singletonPropertyOf} \ x \cdot g \ . \ x \cdot g \ \text{rdfs:range} \ y}{v \ \text{rdf:type} \ y} \quad \text{(sp-6)}
\]

### 3.2.4 Inferring via SP Triple Chain

Back to the motivating example in Table 1.3, we have the two schema triples \( C_1 \) “(chadHurley, type, youtubeEmp): [2005, 2010] and score 0.8” and \( C_2 \) “(youtubeEmp, sc, googleEmp): [2006, 2011] and score 0.9” with contextual information associated with them. They are represented using the set of SP triples \( T_1 \) to \( T_{10} \) in Table 3.1.

From the triples (chadHurley, type, youtubeEmp) and (youtubeEmp, sc, googleEmp), we can derive the new triple (chadHurley, type, googleEmp): [2006, 2010]. How do we derive such a triple from the set of SP triples \( T_1 \) to \( T_{10} \)?

Let us consider the following triples.

\begin{align*}
T_1: & \quad \text{chadHurley} \quad \text{type?id=1} \quad \text{youtubeEmp} \quad . \\
T_2: & \quad \text{type?id=1} \quad \text{singletonPropertyOf} \quad \text{type} \quad . \\
T_6: & \quad \text{youtubeEmp} \quad \text{sc?id=2} \quad \text{googleEmp} \quad . \\
T_7: & \quad \text{sc?id=2} \quad \text{singletonPropertyOf} \quad \text{sc} \quad . \\
\end{align*}

Applying the rule sp-3 to \( T_1 \) and \( T_2 \), we obtain the triple \( G_1 \). Applying the rule sp-3 to \( T_6 \) and \( T_7 \), we obtain the triple \( G_2 \).

\begin{align*}
G_1: & \quad \text{chadHurley} \quad \text{type} \quad \text{youtubeEmp} \quad . \\
G_2: & \quad \text{youtubeEmp} \quad \text{sc} \quad \text{googleEmp} \quad .
\end{align*}

From \( G_1 \) and \( G_2 \), we derive the new relationship (chadHurley, type, googleEmp) from the original singleton properties \( \text{type?id=1} \) and \( \text{sc?id=2} \).

Let \( \text{type?id=9} \) be the singleton property of type representing the unique context, the new relationship can be represented as \( \exists ! \text{type?id=9} \): (chadHurley, type?id=9, googleEmp), (type?id=9, sing, type). From the original triples represented by the two singleton properties \( \text{type?id=1} \) and \( \text{sc?id=2} \), we obtain the new relationship represented by the singleton property \( \text{type?id=9} \). We capture the
provenance of this singleton property by the two triples (type?id=9, derivedFrom, type?id=1) and (type?id=9, derivedFrom, sc?id=2). We represent the inference process as:

```
chadHurley type?id=1 youtubeEmp . type?id=1 sing type .
youtubeEmp sc?id=2 googleEmp . sc?id=2 sing sc .
```

\( \exists! \text{type?id}=9: \text{chadHurley type?id}=9 \text{googleEmp . type?id}=9 \text{sing type} . \)

```
type?id=9 derivedFrom type?id=1 . type?id=9 derivedFrom sc?id=2 .
```

We generalize this pattern to an inference rule as follows. Let A, B, C, and D be meta-variables, and \( p_i, p_j, p_k \) denote the singleton property of the generic property \( p \).

\[
\begin{array}{c}
A \text{type}_i B . \text{type}_i \text{sing type} . \\
B \text{sc}_j C . \text{sc}_j \text{sing sc} . \\
\exists! \text{type}_k: A \text{type}_k C . \text{type}_k \text{sing type} . \\
\text{type}_k \text{derivedFrom type}_i . \text{type}_k \text{derivedFrom sc}_j .
\end{array}
\]

(\text{sptc-1})

For each RDFS inference rule involving subclass, sub-property, domain, and range, since the premises are associated with contexts using the SP representation, we use the SP triple chain to represent them. Here we represent other RDFS rules with the SP triple chain.

\[
\begin{array}{c}
A \text{sc}_i B . \text{sc}_i \text{sing sc} . \\
B \text{sc}_j C . \text{sc}_j \text{sing sc} . \\
\exists! \text{sc}_k: A \text{sc}_k C . \text{sc}_k \text{sing sc} . \\
\text{sc}_k \text{derivedFrom sc}_i . \text{sc}_k \text{derivedFrom sc}_j .
\end{array}
\]

(\text{sptc-2})

\[
\begin{array}{c}
A \text{sp}_i B . \text{sp}_i \text{sing sp} . \\
B \text{sp}_j C . \text{sp}_j \text{sing sp} . \\
\exists! \text{sp}_k: A \text{sp}_k C . \text{sp}_k \text{sing sp} . \\
\text{sp}_k \text{derivedFrom sp}_i . \text{sp}_k \text{derivedFrom sp}_j .
\end{array}
\]

(\text{sptc-3})

\[
\begin{array}{c}
A \text{D}_i B . \text{D}_i \text{sing D} . \\
D \text{dom}_j C . \text{dom}_j \text{sing dom} . \\
\exists! \text{type}_k: A \text{type}_k C . \text{type}_k \text{sing type} . \\
\text{type}_k \text{derivedFrom D}_i . \text{type}_k \text{derivedFrom dom}_j .
\end{array}
\]

(\text{sptc-4})
3.3. CONTEXT-BASED INFERENCE RULES

\[ A \text{ D}_i B \leftrightarrow \text{ D}_i \text{ sing D} . \]
\[ D \text{ ran}_j C \leftrightarrow \text{ ran}_j \text{ sing ran} . \]

\[ \exists \text{ type}_k : B \text{ type}_k C \leftrightarrow \text{ type}_k \text{ sing type} . \]
\[ \text{type}_k \text{ derivedFrom D}_i \leftrightarrow \text{ type}_k \text{ derivedFrom ran}_j . \]

**Proof.** These rules can be proved using the same process explained in the above examples with a sequence of steps: (1) applying the rule sp-3 on the premises to derive the generic triples; (2) applying the corresponding RDFS rule on the generic triples, (3) creating a singleton property representing the unique context, and (4) asserting the provenance for the new singleton property.

### 3.3 Context-based Inference Rules

Zimmermann et al. [Zimmermann et al. 2012] have developed a sound and complete derivation system for the annotated RDF (or aRDF [Udrea et al. 2010]) using a lattice of annotation values. Their inference rules can be represented by reification or by using the named graph. Here we represent their rules using the singleton property model.

Let \( A, B, C, D, X, \) and \( Y \) be meta-variables, and \( L \) be the partially ordered set of annotation values. The generalized inference rule is in the form

\[(X, A, Y): v_1, (X, A, Y): v_2 \]
\[(X, A, Y): v_1 \oplus v_2 \]
\[(X, A, Y): v_1 \otimes v_2 \]

The operations \( \oplus \) (conjunctive) and \( \otimes \) (disjunctive) are defined as least upper bound or greatest lower bound based on the type of annotation values.

Here we consider two types of annotation values: time interval and fuzzy. We will represent the annotated triples within the RDFS inference rules in the form of SP triples. The annotations are in bold. For the triples sharing the same subject, we use the Turtle syntax to shorten them.

#### 3.3.1 Temporal Inference Rules

Let \( L_{time} \) be the set of time intervals, \([t_1, t_2], [t_3, t_4], \) and \([t_5, t_6] \in L_{time} \) such that: \([t_5, t_6] = [t_1, t_2] \cap [t_3, t_4] \), or \( t_5 = \max(t_1, t_3), t_6 = \min(t_2, t_4) \). The conjunctive operation \( \otimes \) is \( \cap \) in this case.

We rewrite the RDFS rules (subclass, sub-property, domain, range) with time-interval annotations from [Zimmermann et al. 2012] using the SP triple chain as follows.
3.3. CONTEXT-BASED INFERENCE RULES

\[ 3.3.2 \text{ Fuzzy Inference Rules} \]

\begin{align*}
A \text{ type}_i B &. \text{ type}_i \text{ sing type} . \text{ type}_i \text{ from } t_1 & \text{ to } t_2 . \\
B \text{ sc}_j C &. \text{ sc}_j \text{ sing sc} . \text{ sc}_j \text{ from } t_3 & \text{ to } t_4 . \tag{spt-1}
\end{align*}

\[ \exists! \text{ type}_k: A \text{ type}_k C . \text{ type}_k \text{ sing type} . \]
\[ \text{ type}_k \text{ from } t_5 & \text{ to } t_6 . \]

\begin{align*}
& \text{ type}_k \text{ derivedFrom type}_i . \text{ type}_k \text{ derivedFrom sc}_j . \\
& A \text{ sc}_i B . \text{ sc}_i \text{ sing sc} . \text{ sc}_i \text{ from } t_1 & \text{ to } t_2 . \\
& B \text{ sc}_j C . \text{ sc}_j \text{ sing sc} . \text{ sc}_j \text{ from } t_3 & \text{ to } t_4 . \tag{spt-2}
\end{align*}

\[ \exists! \text{ sc}_k: A \text{ sc}_k C . \text{ sc}_k \text{ sing sc} . \]
\[ \text{ sc}_k \text{ from } t_5 & \text{ to } t_6 . \]

\begin{align*}
& \text{ sc}_k \text{ derivedFrom sc}_i . \text{ sc}_k \text{ derivedFrom sc}_j . \\
& A \text{ sp}_i B . \text{ sp}_i \text{ sing sp} . \text{ sp}_i \text{ from } t_1 & \text{ to } t_2 . \\
& B \text{ sp}_j C . \text{ sp}_j \text{ sing sp} . \text{ sp}_j \text{ from } t_3 & \text{ to } t_4 . \tag{spt-3}
\end{align*}

\[ \exists! \text{ sp}_k: A \text{ sp}_k C . \text{ sp}_k \text{ sing sp} . \]
\[ \text{ sp}_k \text{ from } t_5 & \text{ to } t_6 . \]

\begin{align*}
& \text{ sp}_k \text{ derivedFrom sp}_i . \text{ sp}_k \text{ derivedFrom sp}_j . \\
& A \text{ D}_i B . \text{ D}_i \text{ sing D} . \text{ D}_i \text{ from } t_1 & \text{ to } t_2 . \\
& D \text{ dom}_j C . \text{ dom}_j \text{ sing dom} . \text{ dom}_j \text{ from } t_3 & \text{ to } t_4 . \tag{spt-4}
\end{align*}

\[ \exists! \text{ type}_k: A \text{ type}_k C . \text{ type}_k \text{ sing type} . \]
\[ \text{ type}_k \text{ from } t_5 & \text{ to } t_6 . \]

\begin{align*}
& \text{ type}_k \text{ derivedFrom D}_i . \text{ type}_k \text{ derivedFrom dom}_j . \\
& A \text{ D}_i B . \text{ D}_i \text{ sing D} . \text{ D}_i \text{ from } t_1 & \text{ to } t_2 . \\
& D \text{ ran}_j C . \text{ ran}_j \text{ sing ran} . \text{ ran}_j \text{ from } t_3 & \text{ to } t_4 . \tag{spt-5}
\end{align*}

\[ \exists! \text{ type}_k: B \text{ type}_k C . \text{ type}_k \text{ sing type} . \]
\[ \text{ type}_k \text{ from } t_5 & \text{ to } t_6 . \]

\begin{align*}
& \text{ type}_k \text{ derivedFrom D}_i . \text{ type}_k \text{ derivedFrom ran}_j .
\end{align*}

3.3.2 Fuzzy Inference Rules

Let \( L_{\text{fuzzy}} = [0, 1] \) be the set of annotation values for the fuzzy setting, \( c_1, c_2, c_3 \in [0, 1]: c_3 = c_1 * c_2 \).
We rewrite the RDFS rules (subclass, sub-property, domain, range) with fuzzy annotations from [Zimmermann et al. 2012] using the SP triple chain as follows.

\[ A \text{ type}_i B \ . \text{ type}_i \text{ sing type} \cdot \text{ type}_i \text{ score } c_1 . \]

\[ B \text{ sc}_j C \ . \text{ sc}_j \text{ sing sc} \cdot \text{ sc}_j \text{ score } c_2 . \]  

\[ \exists! \text{ type}_k: A \text{ type}_k C \cdot \text{ type}_k \text{ sing type} \cdot \text{ type}_k \text{ score } c_3 . \]  

\text{type}_k \text{ derivedFrom } \text{ type}_i \cdot \text{ type}_k \text{ derivedFrom } \text{ sc}_j .

For example, this rule could be applied to the set of SP triples \( T_1 \) to \( T_{10} \).

\[ A \text{ sc}_i B \ . \text{ sc}_i \text{ sing sc} \cdot \text{ sc}_i \text{ score } c_1 . \]

\[ B \text{ sc}_j C \ . \text{ sc}_j \text{ sing sc} \cdot \text{ sc}_j \text{ score } c_2 . \]  

\[ \exists! \text{ sc}_k: A \text{ sc}_k C \cdot \text{ sc}_k \text{ sing sc} \cdot \text{ sc}_k \text{ score } c_3 . \]  

\text{sc}_k \text{ derivedFrom } \text{ sc}_i \cdot \text{ sc}_k \text{ derivedFrom } \text{ sc}_j .

For example, this rule could be applied to the set of SP triples \( T_{16} \) to \( T_{23} \).

\[ A \text{ sp}_i B \ . \text{ sp}_i \text{ sing sp} \cdot \text{ sp}_i \text{ score } c_1 . \]

\[ B \text{ sp}_j C \ . \text{ sp}_j \text{ sing sp} \cdot \text{ sp}_j \text{ score } c_2 . \]  

\[ \exists! \text{ sp}_k: A \text{ sp}_k C \cdot \text{ sp}_k \text{ sing sp} \cdot \text{ sp}_k \text{ score } c_3 . \]  

\text{sp}_k \text{ derivedFrom } \text{ sp}_i \cdot \text{ sp}_k \text{ derivedFrom } \text{ sp}_j .

For example, this rule could be applied to the set of SP triples \( T_{16} \) to \( T_{23} \).

\[ A \text{ D}_i B \ . \text{ D}_i \text{ sing D} \cdot \text{ D}_i \text{ score } c_1 . \]

\[ D \text{ dom}_j C \ . \text{ dom}_j \text{ sing dom} \cdot \text{ dom}_j \text{ score } c_2 . \]  

\[ \exists! \text{ type}_k: A \text{ type}_k C \cdot \text{ type}_k \text{ sing type} \cdot \text{ type}_k \text{ score } c_3 . \]  

\text{type}_k \text{ derivedFrom } \text{ D}_i \cdot \text{ type}_k \text{ derivedFrom } \text{ dom}_j .

This rule could be applied to the set of SP triples \( \{T_{16}, T_{17}, T_{18}, T_{19}, T_{24}, T_{25}, T_{26}, T_{27}\} \).

\[ A \text{ D}_i B \ . \text{ D}_i \text{ sing D} \cdot \text{ D}_i \text{ score } c_1 . \]

\[ D \text{ ran}_j C \ . \text{ ran}_j \text{ sing ran} \cdot \text{ ran}_j \text{ score } c_2 . \]  

\[ \exists! \text{ type}_k: B \text{ type}_k C \cdot \text{ type}_k \text{ sing type} \cdot \text{ type}_k \text{ score } c_3 . \]  

\text{type}_k \text{ derivedFrom } \text{ D}_i \cdot \text{ type}_k \text{ derivedFrom } \text{ ran}_j .

This rule could be applied to the set of SP triples \( \{T_{16}, T_{17}, T_{18}, T_{19}, T_{28}, T_{29}, T_{30}, T_{31}\} \).
3.3. CONTEXT-BASED INFERENCE RULES

3.3.3 Multi-annotation Inference Rules

Let $L_{time}$ be the set of time intervals, $[t_1, t_2]$, $[t_3, t_4]$, and $[t_5, t_6] \in L_{time}$ such that: $[t_5, t_6] = [t_1, t_2] \cap [t_3, t_4]$, or $t_5 = \max(t_1, t_3)$, $t_6 = \min(t_2, t_4)$.

Let $L_{fuzzy} = [0, 1]$ be the set of annotation values for the fuzzy setting, $c_1, c_2, c_3 \in [0, 1]$: $c_3 = c_1 \ast c_2$.

We rewrite the RDFS rules (subclass, sub-property, domain, range) with multiple annotations from [Zimmermann et al. 2012] using the SP triple chain as follows.

$$A \text{type}_i B . \text{type}_i \text{sing type}_i \text{from } t_1 \text{ to } t_2; \text{score } c_1 .$$

$$B \text{sc}_j C . \text{sc}_j \text{sing sc}_j \text{from } t_3 \text{ to } t_4; \text{score } c_2 .$$

$$\exists! \text{type}_k: A \text{type}_k C . \text{type}_k \text{sing type}_k \text{from } t_5 \text{ to } t_6; \text{score } c_3 .$$

$$\text{type}_k \text{derivedFrom type}_i . \text{type}_k \text{derivedFrom sc}_j .$$

$$A \text{sc}_i B . \text{sc}_i \text{sing sc}_i \text{from } t_1 \text{ to } t_2; \text{score } c_1 .$$

$$B \text{sc}_j C . \text{sc}_j \text{sing sc}_j \text{from } t_3 \text{ to } t_4; \text{score } c_2 .$$

$$\exists! \text{sc}_k: A \text{sc}_k C . \text{sc}_k \text{sing sc}_k \text{from } t_5 \text{ to } t_6; \text{score } c_4 .$$

$$\text{sc}_k \text{derivedFrom sc}_i . \text{sc}_k \text{derivedFrom sc}_j .$$

$$A \text{sp}_i B . \text{sp}_i \text{sing sp}_i \text{from } t_1 \text{ to } t_2; \text{score } c_1 .$$

$$B \text{sp}_j C . \text{sp}_j \text{sing sp}_j \text{from } t_3 \text{ to } t_4; \text{score } c_2 .$$

$$\exists! \text{sp}_k: A \text{sp}_k C . \text{sp}_k \text{sing sp}_k \text{from } t_5 \text{ to } t_6; \text{score } c_3 .$$

$$\text{sp}_k \text{derivedFrom sp}_i . \text{sp}_k \text{derivedFrom sp}_j .$$

$$A \text{D}_i B . \text{D}_i \text{sing D}_i \text{from } t_1 \text{ to } t_2; \text{score } c_1 .$$

$$D \text{dom}_j C . \text{dom}_j \text{sing dom}_j \text{from } t_3 \text{ to } t_4; \text{score } c_2 .$$

$$\exists! \text{type}_k: A \text{type}_k C . \text{type}_k \text{sing type}_k \text{from } t_5 \text{ to } t_6; \text{score } c_3 .$$

$$\text{type}_k \text{derivedFrom D}_i . \text{type}_k \text{derivedFrom dom}_j .$$
3.4 OWL 2

From the RDF-based semantics of OWL 2 Full [Schneider et al. 2012], we consider the semantic conditions of the OWL classes and properties that are relevant to singleton properties. These semantic conditions belong to the two categories: logical characteristics of the properties, and relations to other properties. We tighten the semantic conditions of these OWL classes and properties to make sure they are valid in the extended semantics, by enforcing more constraints on the generic property and singleton property extensions. The semantic conditions of these properties must be satisfied in the interpretations extended with singleton property semantics.

Let $V_p$ be the vocabulary of OWL classes and properties relevant to singleton properties: $V_p = \{\text{FunctionalProperty, InverseFunctionalProperty, ReflexiveProperty, IrreflexiveProperty, SymmetricProperty, AsymmetricProperty, TransitiveProperty, inverseOf, equivalentOf}\}$.

Let $p_s, p'_s,$ and $p''_s$ be the singleton properties of the generic property $p$, then $p_s \in I_G(p)$, $p'_s \in I_G(p)$, $p''_s \in I_G(p)$. We define the OWL 2 RDF-based interpretation as follows.

**OWL 2 RDF-based interpretation** of a vocabulary $V$ is an RDFS interpretation $I$ of the vocabulary $V \cup V_{OWL}$ that satisfies criteria from the OWL interpretation [Schneider et al. 2012] and the following semantic conditions:

- **Functional property.** If a property is functional, then at most one distinct value can be assigned to any given individual via this property.

  A property $p$ is an instance of owl:FunctionalProperty iff $\forall x, y_1, y_2$:

  1. $p \in IP, \langle x, y_1 \rangle \in I_{EXT}(p), \langle x, y_2 \rangle \in I_{EXT}(p)$ implies $y_1 = y_2$,

  2. $p \in IP_g, \langle x, y_1 \rangle \in I_{G,EXT}(p), \langle x, y_2 \rangle \in I_{G,EXT}(p)$ implies $y_1 = y_2$,

  3. $\forall p_s \in I_G(p), \langle x, y_1 \rangle = I_{S,EXT}(p_s), \langle x, y_2 \rangle = I_{S,EXT}(p_s)$ implies $y_1 = y_2$. 
• **Inverse functional property.** An inverse functional property can be regarded as a “key” property, i.e., no two different individuals can be assigned the same value via this property.

A property $p$ is an instance of owl:InverseFunctionalProperty iff $\forall x_1, x_2, y:\n
(1) \ p \in IP, \langle x_1, y \rangle \in I_{EXT}(p), \langle x_2, y \rangle \in I_{EXT}(p) \text{ implies } x_1 = x_2,\n
(2) \ p \in IPg, \langle x_1, y \rangle \in I_{G,EXT}(p), \langle x_2, y \rangle \in I_{G,EXT}(p) \text{ implies } x_1 = x_2,\n
(3) \ \forall p_s \in IG(p), \langle x_1, y \rangle = I_{S,EXT}(p_s), \langle x_2, y \rangle = I_{S,EXT}(p_s) \text{ implies } x_1 = x_2.\n
• **Reflexive property.** A reflexive property relates every individual in the universe to itself.

A property $p$ is an instance of the class owl:ReflexiveProperty iff $\forall x:\n
(1) \ p \in IP, \langle x, x \rangle \in I_{EXT}(p),\n
(2) \ p \in IPg, \langle x, x \rangle \in I_{G,EXT}(p),\n
(3) \ \forall p_s \in IG(p), \langle x, x \rangle = I_{S,EXT}(p_s).\n
• **Irreflexive property.** An irreflexive property does not relate any individual to itself.

A property $p$ is an instance of the class owl:IrreflexiveProperty iff $\forall x:\n
(1) \ p \in IP, \langle x, x \rangle \notin I_{EXT}(p),\n
(2) \ p \in IPg, \langle x, x \rangle \notin I_{G,EXT}(p),\n
(3) \ \forall p_s \in IG(p), \langle x, x \rangle \neq I_{S,EXT}(p_s).\n
• **Symmetric property.** If two individuals are related by a symmetric property, then this property also relates them reversely.

A property $p$ is an instance of the class owl:SymmetricProperty iff $\forall x, y:\n
(1) \ p \in IP, \langle x, y \rangle \in I_{EXT}(p) \text{ implies } \langle y, x \rangle \in I_{EXT}(p),\n
(2) \ p \in IPg, \langle x, y \rangle \in I_{G,EXT}(p) \text{ implies } \langle y, x \rangle \in I_{G,EXT}(p),\n
(3) \ \forall p_s \in IG(p), \langle x, y \rangle = I_{S,EXT}(p_s) \text{ implies } \exists p_s' \in IG(p), \langle y, x \rangle = I_{S,EXT}(p_s').\n
• **Asymmetric property.** If two individuals are related by an asymmetric property, then this property never relates them reversely.

A property $p$ is an instance of the class owl:AsymmetricProperty iff $\forall x, y:\n
(1) \ p \in IP, \langle x, y \rangle \in I_{EXT}(p) \text{ implies } \langle y, x \rangle \notin I_{EXT}(p),\n
(2) \ p \in IPg, \langle x, y \rangle \in I_{G,EXT}(p) \text{ implies } \langle y, x \rangle \notin I_{G,EXT}(p),\n
(3) \ \forall p_s \in IG(p), \langle x, y \rangle = I_{S,EXT}(p_s) \text{ implies } \nexists p_s' \in IG(p), \langle y, x \rangle = I_{S,EXT}(p_s).
• **Transitive property.** A transitive property that relates an individual \(a\) to an individual \(b\), and individual \(b\) to an individual \(c\), also relates \(a\) to \(c\).

A property \(p\) is an instance of the class owl:TransitiveProperty iff \(\forall x, y, z:\)

1. \(p \in IP, \langle x, y \rangle \in I_{EXT}(p), \langle y, z \rangle \in I_{EXT}(p)\) implies \(\langle x, z \rangle \in I_{EXT}(p),\)
2. \(p \in IPg, \langle x, y \rangle \in I_{G,EXT}(p), \langle y, z \rangle \in I_{G,EXT}(p)\) implies \(\langle x, z \rangle \in I_{G,EXT}(p),\)
3. \(\forall p_s, p'_s \in I_G(p), \langle x, y \rangle = I_{S,EXT}(p_s), \langle y, z \rangle = I_{S,EXT}(p'_s)\) implies \(\exists p''_s \in I_G(p), \langle x, z \rangle = I_{S,EXT}(p''_s).\)

• **Inverse property.** The inverse of a given property is the corresponding property with subject and object swapped for each property assertion built from it.

\((p_1, p_2) \in I_{EXT}(\text{owl:inverseOf } I)\) iff

1. \(\forall p_1, p_2 \in IP, I_{EXT}(p_1) = \{\langle x, y \rangle | \langle y, x \rangle \in I_{EXT}(p_2)\},\)
2. \(\forall p_1, p_2 \in IPg, I_{G,EXT}(p_1) = \{\langle x, y \rangle | \langle y, x \rangle \in I_{G,EXT}(p_2)\},\)
3. \(\forall p_s \in I_G(p_1), \exists p'_s \in I_G(p_2), I_{S,EXT}(p_s) = \langle x, y \rangle, I_{S,EXT}(p'_s) = \langle y, x \rangle.\)

• **Equivalent property.** Two equivalent properties share the same property extension.

\((p_1, p_2) \in I_{EXT}(\text{owl:equivalentOf } I)\) iff

1. \(\forall p_1, p_2 \in IP : I_{EXT}(p_1) = I_{EXT}(p_2),\)
2. \(\forall p_1, p_2 \in IPg : I_{G,EXT}(p_1) = I_{G,EXT}(p_2),\)
3. \(\forall p_s \in I_G(p_1), \exists p'_s \in I_G(p_2) : I_{S,EXT}(p_s) = I_{S,EXT}(p'_s).\)

The rule rdfs-sp-5 can easily be drived by combining the two rules rdfs-sp-3 and rdf-sp-2. Similar to the property rdfs:subPropertyOf, here we also provide the rules for the owl:equivalentOf.

\[
\begin{align*}
\text{u} \text{ rdf:singletonPropertyOf } x . & \quad (\text{owl-sp-1}) \\
\text{x} \text{ owl:equivalentOf } y . & \\
\text{u} \text{ rdf:singletonPropertyOf } y . & \\
\text{x} \text{ rdf:type GenericProperty} . & \\
\text{x} \text{ owl:equivalentOf } y . & \\
y \text{ rdf:type GenericProperty} . & \quad (\text{owl-sp-2}) \\
\text{u} \text{ rdf:singletonPropertyOf } x . & \\
\text{x} \text{ owl:equivalentOf } y . & \\
y \text{ rdf:type GenericProperty} . & \quad (\text{owl-sp-3})
\end{align*}
\]

If \(x\) owl:equivalentOf \(y\) then

1. \(x\) rdfs:subPropertyOf \(y\) and
2. \(y\) rdfs:subPropertyOf \(x\).

The above three owl-sp rules can be derived easily by combing this rule and the rules rdfs-sp-3, rdfs-sp-4, and rdfs-sp-5, respectively.
3.5 Related Work

Several work relevant in the broader scope of reasoning with contexts have been proposed. Joseph et al. [Joseph and Serafini 2011] use the named graph to group the triples sharing the same context together and apply the reasoning per context while our work focuses on a variety of contexts per triple. The 4d-fluents [Welty et al. 2006] pre-defines ontology specifically made for time. It has not been generalized for any kind of context, while SP can represent multiple kinds of context on the go without predefining ontologies. Plus, it is worse than reification as it takes two more temporal objects and 6 additional triples. CKR by Homola et al. [Serafini and Homola 2012] is developed for OWL 2 DL. These approaches have been implemented and evaluated in several projects [Fu et al. 2015; Hernández et al. 2015; Hernández et al. 2016].

The stream reasoning [Nguyen and Siberski 2013] where the temporal dimension is not represented directly in RDF may benefit from our work as it allows the temporal dimension to be incorporated within the RDF syntax. The temporal RDF [Gutierrez et al. 2005] incorporates temporal reasoning into RDF using reification. The annotated RDF [Udrea et al. 2010] and the general inference scheme for it [Zimmermann et al. 2012] used a partially ordered set of annotation values.
Chapter 4 proposes the new graph model for the SP representation and formalize this model by combining the RDF model-theoretic semantics with the graph theory.

### 4.1 Introduction

As the adoption of Semantic Web grows, a large number of datasets have been generated and made publicly available, for example, as in Linked Open Data. Due to the graph nature of these datasets, many graph-based algorithms for RDF datasets have been developed for query processing [Hartig and Heese 2007; Zeng et al. 2013], graph matching [Bröcheler et al. 2009; Cheng et al. 2008], semantic associations [Anyanwu et al. 2005; Anyanwu and Sheth 2003], path computation [Heim et al. 2009; Przyjaciel-Zablocki et al. 2012], and centrality measures [Cheng et al. 2011; Zhang et al. 2007].

These existing graph algorithms employ the abstract graph model called Node-Labeled Arc-Node (NLAN), currently recommended by W3C [Cyganiak et al. 2014].

In this NLAN graph, the subject and object of a triple are mapped to two nodes of a graph, and the predicate is mapped to a directed labeled arc connecting the subject and the object nodes of that graph. Although these straightforward mappings work for simple triples at the instance level, challenges may arise where predicates also play the role of the subject in other triples as in Figure 4.1.

Particularly, in the first triple, the predicate is mapped to a labeled arc connecting the two subject/object nodes of the subgraph G1 representing the first triple. When the predicate from the first triple becomes the subject or the object of the second triple, it is mapped to another node of
the subgraph \( G_2 \) representing the second triple. As a result, a predicate is mapped to both a labeled arc in the subgraph \( G_1 \) and a node within the subgraph \( G_2 \). Despite the fact that the predicate is shared between the two triples, the resulting graph is comprised of two disconnected subgraphs \( G_1 \) and \( G_2 \), making it impossible to traverse between the subgraphs. The object 2 is unreachable from subject 1. In addition, it is unfortunate for the NLAN graph model that such a scenario is common as RDF syntax allows us to make assertions about any resources, including predicates. Here we present two such scenarios: schema triples and singleton property triples. We will take the singleton property triples forward to motivate our work.

<table>
<thead>
<tr>
<th>Subject 1</th>
<th>Predicate 1</th>
<th>Object 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Predicate 2</td>
<td>Object 2</td>
</tr>
</tbody>
</table>

Figure 4.1: Subgraphs \( G_1 \) and \( G_2 \) are disconnected in the Node-LabeledArc-Node diagram (NLAN).

**Schema triples.** One common scenario where a predicate becomes the subject or object of another triple is through the RDF schema triples. A schema triple may describe a property using many other different properties such as \( \text{rdfs:subPropertyOf} \), \( \text{rdf:type} \), \( \text{rdfs:domain} \), or \( \text{rdfs:range} \). For example, the predicates \( \text{hasFamilyName} \) and \( \text{hasGivenName} \) are asserted as sub properties of the predicate \( \text{rdfs:label} \), and hence become the subjects in these triples:

\[
\text{hasFamilyName rdfs:subPropertyOf rdfs:label .}
\]

\[
\text{hasGivenName rdfs:subPropertyOf rdfs:label .}
\]

The Yago2S-SP dataset [Nguyen et al. 2014] contains 938 properties like these. We can also find schema triples in many other common datasets, such as DBPedia.

**Singleton Property triples.** Another scenario comes from the RDF datasets created by a recent work [Nguyen et al. 2014] where singleton properties are used to describe RDF statements. A simple approach to make assertions about an RDF statement is through reification. Reification creates an instance of class \( \text{rdf:Statement} \) to represent a statement and asserts metadata about this statement through this instance. Recall that although this approach is intuitive, reifying one statement requires at least four triples, making it less attractive. Instead of reifying a statement, Nguyen et al. [Nguyen et al. 2014] propose a new approach called the singleton property that is more efficient. Each singleton property uniquely representing a statement can be annotated with different kinds of metadata such as provenance, time, and location describing that statement. Therefore, the singleton properties also become subjects and objects of other meta triples. Meta triples are the
triples that describe other triples through the use of singleton properties. While singleton properties enrich RDF datasets with different kinds of metadata for the triples they represent, the metadata subgraph is disconnected from the triple they describe. Traversing the NLAN graphs created by this approach also becomes a challenge for the RDF graph model.

Being unable to traverse among triples in the scenarios described above due to the limited connectivity of the NLAN graph is a critical issue, because it limits the capability of answering reachability and shortest path queries on RDF datasets. A reachability query verifies if a path exists between any two nodes in a graph. A shortest path query returns the shortest distance in terms of the number of edges between any two nodes in a graph if the path exists. In the NLAN graph model, both query types require traversing the graph from subject node to object node of a triple. Traversing from one node in subgraph G1 to another node in subgraph G2 in this way will not find any result because they do not share any node in common. The only resource these two triples share is the predicate of the first triple, which is also the subject of the second triple. Therefore, the ability to connect the triples represented in G1 with the corresponding triples in G2 (schema triples or meta triples represented through the singleton property) is desirable. Such connectivity strengthens the robustness of the graph model.

Next, we will take one sample set of RDF triples represented by the singleton property approach as our motivating example. We will analyze the limitations of the existing work through this motivating example and demonstrate our approach to address the challenges.

### 4.1.1 Motivating Example

Consider the example with the facts that Bill Clinton is succeeded by George W. Bush as the president of the United States, and is succeeded by Frank White as the Governor of Arkansas. If we represent the facts as follows, these relationships are unclear and incomplete because the graph does not represent the political position context in which Bill Clinton is succeeded.

\[
\text{BillClinton hasSuccessor GeorgeWBush .} \\
\text{BillClinton hasSuccessor FrankWhite .}
\]

Instead, we propose to use the singleton property approach to represent the facts as shown in Table 1.7 Here we consider the set of triples shown in Table 1.1 which will be used as a running example throughout the chapter. Indeed, our example is motivated by the facts from the Yago2S-SP dataset, which will be used in the evaluation described in Section 7.3. We create our example in the way that makes it intuitive to readers, and for readability, we eliminate all the URI prefixes. For each political position of Bill Clinton, we create a singleton property to capture the information related to that context. Particularly, we create two singleton properties holdsPos?id=1 and holdsPos?id=2.
4.1. INTRODUCTION

Table 4.1: Example triples using singleton properties to represent the facts about the American politician Bill Clinton and his successors

<table>
<thead>
<tr>
<th>No</th>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>BillClinton</td>
<td>holdsPos?id=1</td>
<td>U.S.President</td>
</tr>
<tr>
<td>T2</td>
<td>BillClinton</td>
<td>holdsPos?id=2</td>
<td>ArkansasGovernor</td>
</tr>
<tr>
<td>T3</td>
<td>holdsPos?id=1</td>
<td>hasSuccessor</td>
<td>GeorgeWBush</td>
</tr>
<tr>
<td>T4</td>
<td>holdsPos?id=2</td>
<td>hasSuccessor</td>
<td>FrankWhite</td>
</tr>
</tbody>
</table>

Then we attach the predicate hasSuccessor to the two singleton properties to represent the 3-ary relationship among the politician, the position and the successor.

We will analyze how such a set of triples can be represented in the form of a graph by two existing approaches: the NLAN diagram [Cyganiak et al. 2014] and the bipartite (BI) model [Hayes and Gutierrez 2004].

A triple may be represented as a NLAN diagram as explained in the W3C RDF 1.1 Concepts and Syntax [Cyganiak et al. 2014]. The set of triples in Table 4.1 are represented as a NLAN diagram in Figure 4.2. We observe two problems with this modeling when dealing with such a set of triples.

First and more importantly, the NLAN resulting graph is not precisely a mathematical graph. In Figure 4.2, the RDF term holdsPos?id=1 is mapped to the predicate arc of triple $T_1$, and to the subject node in $T_3$, $T_4$. Mapping the same RDF term holdsPos?id=1 to more than one different mathematical object makes the object being referred to ambiguous. No mapping function satisfies this because a function must map any source object to only one target. For this reason, this NLAN model cannot be compatible with the formal semantics that is defined by several mapping functions.

The second serious problem is that the disconnectedness between subgraphs limits the possibility to traverse between the triples that are connected through the predicates. In the example at hand, the NLAN resulting graph is comprised of three disconnected subgraphs as shown in Figure 4.2.
Although the two predicates \texttt{holdsPos?id=1} and \texttt{holdsPos?id=2} are shared between the triples from two subgraphs, no connectivity between two subgraphs can be found in Figure 4.2. As a consequence, although Bill Clinton is succeeded by George W. Bush, no path can be found from Bill Clinton to George W. Bush. From this perspective, the bipartite model provides better graph connectivity than the NLAN model.

The bipartite (BI) model proposed by Hayes [Hayes and Gutierrez 2004] does not encounter these problems of the NLAN model. It represents all subjects, predicates, and objects as nodes. It creates an auxiliary node and links this node to the subject, the predicate, and the object of this triple via three additional arcs. However, the cost incurred in this approach with one extra node and three extra arcs for every triple is too high.

### 4.1.2 Our approach

In this chapter, we propose a new formal graph model CKG which represents RDF triples in a labeled directed multigraph with triple nodes. Similar to the bipartite model, all subjects, predicates, and objects are mapped to nodes of a CKG graph. This model, however, differs from the bipartite model in that it adds one pair of directed edges (subject-predicate, predicate-object) to directly connect three nodes of the same triple. Furthermore, this approach differs from the NLAN diagram in that the predicates are mapped to nodes instead of labeled edges.

As traversing the NLAN graph starts from a node and explores its adjacent nodes, predicates as arcs never get explored. In the CKG model, predicates are mapped to nodes in the graph. As they are adjacent to the subjects, they are explored after the subject. The objects now become adjacent to the predicates and get explored after the predicates. Traversing the CKG graph this way starting from the node \texttt{Subject 1} will reach the node \texttt{Object 2} after 3 hops, as shown in Figure 4.3.

![Figure 4.3](image)

Figure 4.3: Subgraphs G1 and G2 from Figure 4.1 are now connected in a CKG graph with labeled directed multigraph with triple nodes.

The CKG graph representation of the example from Table 4.1 is illustrated in Figure 4.4. For the set of six triples in Table 4.1 we map all subjects, predicates, and objects to the set of ten nodes. We link every three nodes representing the same triple by one pair of directed edges. For instance,
the three nodes \{BillClinton, holdsPos?id=1, U.S.President\} are connected by the pair of edges \(e_1^T, e_1^T\) to form the triple \(T_1\).

![CKG graph with labeled directed multigraph with triple nodes for Example 4.](image)

Figure 4.4: CKG graph with labeled directed multigraph with triple nodes for Example 4.

Compared to the NLAN model, the CKG graph model provides better connectivity and makes it possible to find the answers for the reachability and shortest path queries. While the node GeorgeW.Bush is unreachable from the node BillClinton in the NLAN model, it is not only reachable but also neatly presented as the resource path \((BillClinton, holdsPos?id=1, hasSuccessor, GeorgeW.Bush)\) in the CKG model. We will formally define resource path and triple path in Section 4.2.3.

In terms of practical impact, we show that compared to the NLAN model, our CKG model does not introduce cost to the system in terms of space even though it conceptually adds two edges to connect (subject-predicate) and (predicate-object) in each triple. These initial and terminal edges that allow us to disambiguate the nodes forming each triple in the abstract model are unnecessary in the physical model. It is because the triples are already stored separately in the indices. We use the same indices for traversing in both models. In terms of execution time, traversing the NLAN graph is supposed to be faster than traversing the CKG graph because exploration of the predicate nodes is bypassed. However, that extra exploration in the CKG graph allows for additional paths that are impossible to find in the NLAN graph. We will report the extra time taken in traversing the CKG graph through the shortest path algorithm in Section 7.3.

We summarize our contributions in this chapter:

- We develop a formal graph model (CKG) for RDF. For traversing the new graph, we define two types of paths, resource path and triple path with a new traversal algorithm (in Section 4.2).
- We rewrite the model-theoretic formal semantics for RDF with three levels of interpretation (simple, RDF, and RDFS) based on the CKG model (Section 4.3) and demonstrate the graph entailments using RDFS deduction rules (Section 4.3.4), which the NLAN and BI models do not support.
We implement a new engine called GraphKE and extend the RDF-3X to support the CKG graph model (Section 4.2.1). We implement and evaluate the empirical performance of the reachability and shortest path queries on both engines (Section 7.3) to demonstrate that our formal model is technically viable for both new and existing systems.

We discuss future work in Section 4.5.

### 4.2 Formalizing Graph Model

We demonstrate how to represent any RDF triples in the CKG model. We start with the step-by-step modeling of simple facts in Section 4.2.1, then formalize the CKG model in Section 4.2.2.

#### 4.2.1 Representing RDF Graph

Referring back to the motivating example in Section 4.1.1, the NLAN diagram represents the triple

\[ T_3: \text{holdsPos?id=1 hasSuccessor GeorgeWBush} \]

by mapping the predicate hasSuccessor to an arc which connects the two nodes holdsPos?id=1 and GeorgeWBush. We argue that, just as subjects and objects are mapped to nodes, predicates should also be mapped to nodes since they are all instances of the same class rdf:Resource. The next question is “How do we link those three nodes to form a triple?”

Existing models use either labeled edges (NLAN) or one auxiliary node plus three labeled edges (BI). We will approach the question from another perspective. Although the predicate is mapped to a node, this predicate node differs from subject or object nodes because of its linking role. We need to represent this link without causing a discrepancy like the one found in the NLAN model.

Here we have three nodes to form a triple. However, every edge in the graph can only connect two nodes. Instead of adopting a complicated solution like using a hyperedge, how about using two regular edges? Obviously, two directed edges can connect the three nodes into two pairs as illustrated in Figure 4.5 (a). However, if we add another triple, say

\[ T_4: \text{holdsPos?id=2 hasSuccessor FrankWhite} \]

then an ambiguity arises in figuring out which two edges form the triple as in Figure 4.5 (b). The next question is “How do we tie these two edges together to represent that they are part of the same triple?”

Since we have two edges, we call the first edge between subject and predicate the initial edge, and we call the second edge between predicate and object the terminal edge. We create a mapping that maps every initial edge to its terminal edge. The pair of edges for a given triple \( i \) is denoted as \( (e^I_i, e^T_i) \), where \( e^I_i \) is the initial edge and \( e^T_i \) is the terminal edge. It follows that \( (e^I_3, e^T_3) \) corresponds to \( T_3 \) and \( (e^I_4, e^T_4) \) to \( T_4 \). The complete CKG graph for \( T_3 \) and \( T_4 \) is illustrated in Figure 4.5 (c).
4.2. FORMALIZING GRAPH MODEL

Next, we will formalize the CKG model for any set of RDF triples.

4.2.2 Formalizing RDF Graph

Formally, any RDF dataset is a set of RDF triples. Let \( T = \{t_0, t_1, \ldots, t_n\} \) be a set of triples on the vocabulary \( V \), and let \( G_{RDF} \) be the labeled directed multigraph with triple nodes (CKG) of \( T \).

We will first create the graph \( G_{RDF} \) from \( T \) and then regenerate the set of original triples \( T \) from \( G_{RDF} \).

Proposition 4.2.1. (Forward transformation). Any set of RDF triples can be transformed into a labeled directed multigraph with triple nodes \( G_{RDF} \).

Proof. Let \( V \) be the set of RDF terms in \( T \). Let \( N \) and \( E \) be the set of nodes and the set of directed edges in the graph \( G_{RDF} \), respectively.

The bijective function \( \mu : V \to N \) maps an RDF term in \( V \) to a node in \( N \). Let \( t_i \) be a triple in \( T \), \( t_i = (s_i, p_i, o_i) \in T \) with \( 0 \leq i \leq n \). Let \( N_i \subseteq N \) such that \( N_i = \{n_{s_i}, n_{p_i}, n_{o_i} | n_{s_i} = \mu(s_i), n_{p_i} = \mu(p_i), n_{o_i} = \mu(o_i)\} \), then \( N = \bigcup_{i=0}^{n} N_i \).

The function \( \epsilon : E \to N \times N \) is defined to map every edge in \( E \) to an ordered-pair of nodes. Let \( e_i^1, e_i^2 \in E : \epsilon(e_i^1) = (n_{s_i}, n_{p_i}) \) and \( \epsilon(e_i^2) = (n_{p_i}, n_{o_i}) \).

The bijective function \( \tau : E \to E \) maps an initial edge to a terminal edge of the same triple. Then \( \tau(e_i^1) = e_i^2 \). Let \( E_i \subseteq E \) be the set of two edges representing \( t_i \), \( E_i = \{e_i^1, e_i^2\} \), and \( E = \bigcup_{i=0}^{n} E_i \).

Therefore, \( G_i = (N_i, E_i, \epsilon, \tau, \mu) \) is the labeled directed graph with triple nodes of the triple \( t_i \).

Finally, with \( N = \bigcup_{i=0}^{n} N_i \) and \( E = \bigcup_{i=0}^{n} E_i \), the graph \( G_{RDF} = (N, E, \epsilon, \tau, \mu) \) is a labeled...
4.2. FORMALIZING GRAPH MODEL

A directed multigraph with triple nodes for all of the triples in $T$.

**Proposition 4.2.2.** (Backward transformation). Given the graph $G_{RDF}(N, E, \epsilon, \tau, \mu)$ transformed by Proposition 4.2.1, a set of RDF triples can be derived from $G_{RDF}$.

**Proof.** Let $e_i^t$ be any edge in $E$ with $0 \leq i \leq n$, the corresponding terminal edge of $e_i^t$ in $E$ is $e_i^T = \tau(e_i^t)$. From this pair of edges $e_i^t$ and $e_i^T$, we find the nodes connected by the two edges by using the $\epsilon$ function. Let $n_{si}, n_{pi}, n_{oi} \in N$ such that $(n_{si}, n_{pi}) = \epsilon(e_i^t)$, $(n_{pi}, n_{oi}) = \epsilon(e_i^T)$.

Let $\mu^{-1}$ be the reverse function of $\mu$, then $\mu^{-1} : V \rightarrow N$ returns the RDF term mapped to a graph node. Let $s_i, p_i, o_i \in V$ such that $s_i = \mu^{-1}(n_{si}), p_i = \mu^{-1}(n_{pi})$ and $o_i = \mu^{-1}(n_{oi})$. The three nodes form the original triple $t_i = (s_i, p_i, o_i)$. The set $T$ of all RDF triples $t_i$ derived from $G_{RDF}$ is as follows:

$T = \{ t_i | \forall i : t_i = (s_i, p_i, o_i) \}$.

**Proposition 4.2.3.** The size of the graph $G_{RDF}$ on the set of triples $T$ is $2|T|$.

**Proof.** The size of the graph $G_{RDF}$ is the number of edges needed to form all of the triples in $T$. For each triple $t = (s, p, o) \in T$, we need 2 edges to form the triple. Therefore, as the number of triples in $T$ is $|T|$, we need $2|T|$ edges.

### 4.2.3 Traversing RDF Graph

A triple path is defined as a sequence of triple subgraphs where two adjacent triple subgraphs share one common node, that the subject node of the later triple subgraph is also the object or predicate node of the previous triple subgraph. In the given CKG graph $G_{RDF}$, the triple path $tp = (t_i, t_{i+1}, ..., t_j)$ where $t_k = (n_{sk}, n_{pk}, n_{ok})$ with $i \leq k \leq j$ and $t_{k+1} = (n_{s(k+1)}, n_{p(k+1)}, n_{o(k+1)})$ satisfy one of following conditions: (1) $n_{s(k+1)} = n_{pk}$ or (2) $n_{s(k+1)} = n_{ok}$.

For example, $(T_1, T_2)$ from Table 4.3 is a triple path because two triples share the node holdsPos?id=1.

A resource path is defined as a sequence of nodes such that (1) every two adjacent nodes are connected by an edge, (2) every three nodes connected by a pair of initial and terminal edges should form a triple. In the given CKG graph $G_{RDF}$, the resource path $(n_0, n_1, ..., n_k)$ satisfies the following conditions:

1. $(n_i, n_{i+1}) = \epsilon(e_i) \in E$ for all $0 \leq i \leq k - 1$
2. for every edge $e_i^T = (n_i, n_{i+1}) \in E$ and $\exists e_i^t \in E$ such that $\tau(e_i^t) = e_i^T$,
   - if $\exists e_{i-1}^t \in E$ such that $e_{i-1}^t = (n_{i-1}, n_i) \in E$ and $\tau(e_{i-1}^t) = e_i^T$, then $\tau(e_{i-1}^t) = e_i^T$, or $e_{i-1}^T = e_i^T$. 

4.3. FORMAL SEMANTICS

The second condition excludes the case where the two initial and terminal edges from different triples are adjacent to each other in a path. From the motivating example, the path (BillClinton, holdsPos?id=1, hasSuccessor, GeorgeWBush) satisfies these conditions and hence, it is a resource path. We will describe the implementation of the shortest resource path in Section 7.1 and its evaluation in Section 7.3.

4.3 Formal Semantics

This section explains how a labeled directed multigraph with triple nodes, or CKG graph, can be exploited as an underlying model for RDF(S) formal semantics. Similar to the model-theoretic semantics described in [Hitzler et al. 2011; Nguyen et al. 2014], we also represent three levels of interpretation: simple, RDF, and RDFS. However, we define new functions for modeling the underlying labeled directed multigraphs and use them to redefine class/property extension functions.

4.3.1 Simple interpretation

The simple interpretation \( \mathcal{I} \) of the vocabulary \( V \) based on the LDM model consists of:

- \( \mathcal{I}N \), a non-empty set of nodes in \( \mathcal{I} \),
- \( \mathcal{I}E \), a set of directed edges in \( \mathcal{I} \),
- \( \mathcal{I}_E : \mathcal{I}E \to \mathcal{I}N \times \mathcal{I}N \), mapping each edge to an ordered pair of nodes. This function may be many-to-one because a multigraph allows multiple edges between two nodes.
- \( \mathcal{I}_T : \mathcal{I}E \to \mathcal{I}E \), mapping an initial edge to a terminal edge of the same triple. This is a bijective function, and that the reverse function \( \mathcal{I}^{-1}_T : \mathcal{I}E \to \mathcal{I}E \) maps a terminal edge to its initial edge.
- \( \mathcal{I}L \), a set of distinct labels to be assigned to the edges in \( \mathcal{I}E \),
- \( \mathcal{I}_EL : \mathcal{I}L \to \mathcal{I}E \), a labeling function, mapping labels from \( \mathcal{I}L \) into the set \( \mathcal{I}E \) of edges. All labeling functions are also bijective.
- \( \mathcal{I}S : V \to \mathcal{I}N \), a labeling function, mapping URIs from \( V \) into \( \mathcal{I}N \),
- \( \mathcal{I}LV \), a set of literal values in \( \mathcal{I}N \),
- \( \mathcal{I}_L : V \to \mathcal{I}N \), a labeling function, mapping literal values from \( V \) to \( \mathcal{I}N \).

Let \( \mathcal{I} \) be the interpretation function that maps all the URIs and literals in \( V \) to the set of nodes in \( \mathcal{I}N \). A ground triple \( (s, p, o) \) is assigned true if all \( s, p, o \in V \) and \( \exists e_1, e_2 \in \mathcal{I}E : \mathcal{I}_E(e_1) = (s^\mathcal{I}, p^\mathcal{I}), \mathcal{I}_E(e_2) = (p^\mathcal{I}, o^\mathcal{I}) \), and \( \mathcal{I}_T(e_1) = e_2 \).
4.3. FORMAL SEMANTICS

The simple interpretation in [Hitzler et al. 2011] contains an extension function that maps a property to a set of resource pairs. In this simple interpretation, we define a set of new functions, \( \mathcal{I}_E \) and \( \mathcal{I}_T \), for representing nodes and edges of the underlying labeled directed multigraphs with triple nodes. To make this interpretation compatible with the existing ones, we incorporate the existing criteria for generic properties and singleton properties into this interpretation. The simple interpretation \( \mathcal{I} \) also satisfies the following criteria:

- **IP**, a set of generic property nodes, which is also a subset of \( \mathcal{I} \), \( IP \subset \mathcal{I} \).
- **\( I_{EXT} \)**, a function assigning to each property node a set of pairs from \( \mathcal{I} \). \( I_{EXT} : IP \rightarrow 2^{\mathcal{I} \times \mathcal{I}} \) where \( I_{EXT}(p) = \{ \langle s, o \rangle \mid \exists e_1, e_2 \in \mathcal{I} : \mathcal{I}_E(e_1) = (s, p), \mathcal{I}_E(e_2) = (p, o), \text{ and } \mathcal{I}_T(e_1) = e_2 \} \).
- **\( I_{S,EXT} \)**, a function mapping a singleton property to a pair of resources. \( I_{S,EXT} : IS \rightarrow \mathcal{I} \). Particularly, \( I_{S,EXT}(p) = (s, o) \) such that \( \exists e_1, e_2 \in \mathcal{I} : \mathcal{I}_E(e_1) = (s, ps), \mathcal{I}_E(e_2) = (ps, o), \text{ and } \mathcal{I}_T(e_1) = e_2 \).

4.3.2 RDF interpretation

The RDF interpretation of a vocabulary \( V \) is a simple interpretation \( \mathcal{I} \) of the vocabulary \( V \cup V_{RDF} \) that satisfies the following criteria:

- **\( p \in IP \)** if \( \exists e_1, e_2 \in \mathcal{I} : \mathcal{I}_E(e_1) = (p, \text{rdf:type}^\mathcal{I}), \mathcal{I}_E(e_2) = (\text{rdf:type}^\mathcal{I}, \text{rdf:Property}^\mathcal{I}), \text{ and } \mathcal{I}_T(e_1) = e_2 \). A generic property is an instance of the class rdf:Property.
- **\( ps \in ISPs \)** if \( \exists e_1, e_2 \in \mathcal{I} : \mathcal{I}_E(e_1) = (ps, \text{rdf:type}^\mathcal{I}), \mathcal{I}_E(e_2) = (\text{rdf:type}^\mathcal{I}, \text{rdf:SingletonProperty}^\mathcal{I}), \text{ and } \mathcal{I}_T(e_1) = e_2 \). Every singleton property is an instance of the class rdf:SingletonProperty.
- **\( ps \in ISPs \)** if \( \exists e_1, e_2 \in \mathcal{I} : \mathcal{I}_E(e_1) = (ps, \text{rdf:singletonPropertyOf}^\mathcal{I}), \mathcal{I}_E(e_2) = (\text{rdf:singletonPropertyOf}^\mathcal{I}, p), \text{ and } \mathcal{I}_T(e_1) = e_2 \). A singleton property is connected to a generic property via the rdf:singletonPropertyOf.
- if \( ps \in ISPs \) then \( \exists (e_1, e_2) : \mathcal{I}_E(e_1) = (s, ps), \mathcal{I}_E(e_2) = (p, o), \mathcal{I}_T(e_1) = e_2 \), with \( s, o \in \mathcal{I} \) and \( e_1, e_2 \in \mathcal{I} \). This ensures only one occurrence of a singleton property as a predicate of a triple.
In the RDFS interpretation, we define the function $I_{CEXT}$ of the vocabulary $V$ is an RDF interpretation $I$ of the vocabulary $V \cup V_{RDF} \cup V_{RDFS}$ that satisfies the following criteria:

- $I_{CEXT} : IP \to 2^{IN}$, a function assigning to each class a set of nodes from $IN$. $I_{CEXT}(c)$ is called the class extension of $c$. Particularly, $I_{CEXT}(c) = \{ s | s \in IN, \exists e_1, e_2 \in IE : I_{e}(e_1) = (s, rdf:type^T), I_{e}(e_2) = (rdf:type^T, c), and I_T(e_1) = e_2 \}$.

- if $\exists e_1, e_2, e_3, e_4 \in IE : I_T(e_1) = e_2, I_T(e_3) = e_4$, $I_{e}(e_1) = (x, rdfs:domain^T), I_{e}(e_2) = (rdfs:domain^T, y), I_{e}(e_3) = (u, x), I_{e}(e_4) = (x, v)$, then $\exists e_5, e_6 \in IE : I_T(e_5) = e_6$, $I_{e}(e_5) = (u, rdf:type^T), I_{e}(e_6) = (rdf:type^T, y)$.

- if one class is a domain of a property, then the class extension includes all subjects in the same triples with the property.

- if $\exists e_1, e_2, e_3, e_4 \in IE : I_T(e_1) = e_2, I_T(e_3) = e_4$, $I_{e}(e_1) = (x, rdfs:range^T), I_{e}(e_2) = (rdfs:range^T, y), I_{e}(e_3) = (u, x), I_{e}(e_4) = (x, v)$, then $\exists e_5, e_6 \in IE : I_T(e_5) = e_6$, $I_{e}(e_5) = (v, rdf:type^T), I_{e}(e_6) = (rdf:type^T, y)$.

The class range of a property includes all objects in the same triples with the property.

- if $\exists e_1, e_2 \in IE : I_T(e_1) = e_2$, $I_{e}(e_1) = (x, rdfs:subPropertyOf^T)$,
4.3. FORMAL SEMANTICS

and \( I_E(e_2) = (\text{rdfs:subPropertyOf}^T, y) \), then \( x, y \in IP \) and \( I_{EXT}(x) \subseteq I_{EXT}(y) \). The extension of a property is a subset of the extension of its super property.

- if \( \exists e_1, e_2 \in IE: I_T(e_1) = e_2 \),
  \( I_E(e_1) = (x, \text{rdfs:subClassOf}^T) \),
  and \( I_E(e_2) = (\text{rdfs:subClassOf}^T, y) \),
  then \( I_{CEXT}(x) \subseteq I_{CEXT}(y) \). The extension of a class is a subset of its super class extension.

4.3.4 RDFS Entailments

Here we present how the CKG graph-based semantics described in Section 4.3 can derive other graphs using RDFS rules. We demonstrate the entailments by using three rules: rdfs5, rdfs7, and rdfs9 from [Hitzler et al. 2011]. We choose these rules since they are commonly used for reasoning with class and property hierarchy. Other rules can also be applied in the same way. Let \( G \) and \( G' \) be the two CKG graphs.

- rdfs5: if \((u, \text{rdfs:subPropertyOf}, v)\) and 
  \((v, \text{rdfs:subPropertyOf}, x)\)
  then \((u, \text{rdfs:subPropertyOf}, x)\). This rule states that the \text{rdfs:subPropertyOf} property is transitive.

  if \( \exists e_1, e_2, e_3, e_4 \in IE \) in \( G \) : \( I_T(e_1) = e_2, I_T(e_3) = e_4 \),
  \( I_E(e_1) = (u, \text{rdfs:subPropertyOf}^T) \),
  \( I_E(e_2) = (\text{rdfs:subPropertyOf}^T, v) \),
  \( I_E(e_3) = (v, \text{rdfs:subPropertyOf}^T) \),
  \( I_E(e_4) = (\text{rdfs:subPropertyOf}^T, x) \),
  then \( \exists e_5, e_6 \notin IE : I_T(e_5) = e_6 \),
  \( I_E(e_5) = (u, \text{rdfs:subPropertyOf}^T) \),
  \( I_E(e_6) = (\text{rdfs:subPropertyOf}^T, x) \). The graph \( G \) entails \( G' : IE' = IE \cup \{e_5, e_6\} \).

- rdfs7: if \((a, \text{rdfs:subPropertyOf}, b)\) and \((u, a, y)\) then \((u, b, y)\).

All resources interlinked by a property are interlinked by its super property.

if \( \exists e_1, e_2, e_3, e_4 \in IE \) in \( G \) : \( I_T(e_1) = e_2 \),
\( I_E(e_1) = (a, \text{rdfs:subPropertyOf}^T) \),
\( I_E(e_2) = (\text{rdfs:subPropertyOf}^T, b) \),
\( I_E(e_3) = (u, a), I_E(e_4) = (a, y) \) and \( I_T(e_3) = e_4 \), then \( \exists e_5, e_6 \notin IE : I_T(e_5) = e_6, I_E(e_5) = (u, b), I_E(e_6) = (b, y) \), and the graph \( G \) entails \( G' : IE' = IE \cup \{e_5, e_6\} \).
4.4 RELATED WORK

- rdfs9: if \((v, \text{rdf:type}, u)\) and \((u, \text{rdfs:subClassOf}, x)\) then \((v, \text{rdf:type}, x)\).

A resource as a member of one class is also a member of its superclass.

\[
\begin{align*}
\text{if } \exists e_1, e_2, e_3, e_4 \in IE \text{ in } G : \mathcal{I}_T(e_1) = e_2, \mathcal{I}_T(e_3) = e_4, \\
\mathcal{I}_\mathcal{E}(e_1) &= (u, \text{rdfs:subClassOf}^I), \\
\mathcal{I}_\mathcal{E}(e_2) &= (\text{rdfs:subClassOf}^I, v), \\
\mathcal{I}_\mathcal{E}(e_3) &= (v, \text{rdfs:subClassOf}^I), \\
\mathcal{I}_\mathcal{E}(e_4) &= (\text{rdfs:subClassOf}^I, x) \\
\text{then } \exists e_5, e_6 \notin IE : \mathcal{I}_T(e_5) = e_6, \\
\mathcal{I}_\mathcal{E}(e_5) &= (u, \text{rdfs:subClassOf}^I), \\
\mathcal{I}_\mathcal{E}(e_6) &= (\text{rdfs:subClassOf}^I, x), \\
\text{and the graph } G \text{ entails } G' : IE' = IE \cup \{e_5, e_6\}.
\end{align*}
\]

4.4 Related Work

Although a Semantic Web knowledge base can be called a knowledge graph, the syntax and semantics of the RDF data model are not formally defined as a graph. The abstract graph model is the diagram mapping the subject and the object of a triple to two nodes and the predicate is mapped to a directed labeled arc connecting the subject and the object nodes of that graph. This diagram abstracts the RDF triples to be a node-labeled arc-node graph (NLAN), currently recommended by W3C [Cyganiak et al. 2014].

Due to the graph nature of these datasets, many graph-based algorithms for RDF datasets have been developed for query processing [Hartig and Heese 2007; Zeng et al. 2013], graph matching [Brocheler et al. 2009; Cheng et al. 2008], semantic associations [Anyanwu et al. 2005; Anyanwu and Sheth 2003], path computing [Heim et al. 2009; Przyjaciel-Zablocki et al. 2012], centrality [Cheng et al. 2011; Zhang et al. 2007], etc. These existing graph algorithms employ the abstract graph model Node-Labeled Arc-Node (NLAN).

In order to incorporate contexts into the facts, the reification approach can be used. To treat the reified triples as a graph, Hayes et al. proposed the bipartite (BI) model [Hayes and Gutierrez 2004]. It represents all subjects, predicates, and objects as nodes. It creates an auxiliary node for each triple and links this node to the subject, the predicate, and the object of this triple via three additional arcs. Therefore, the cost incurred in this approach with one extra node and three extra arcs for every triple is too high. The resulting knowledge base is a contextualized knowledge hypergraph.
4.5 Future Work

**Future work.** A document of the CKG model that fully supports the latest specifications of RDF syntax, semantics, and deduction rules is necessary (as we skipped some due to space constraints). We believe that many applications in the Semantic Web areas such as graph databases, knowledge representation, graph theory, and logics could benefit from our CKG graph model.

Adopting this CKG model and the singleton property approach would allow more well-connected knowledge graphs that are aware of temporal, spatial, and provenance to be represented in RDF. For example, it is directly applicable to publishing, querying, and browsing the Linked Open Data (LOD). Well-connected knowledge graphs would be amenable to graph-based algorithms, which could be applied to both RDF data and schema triples. Moreover, standard algorithms well-studied from graph theory could be directly applied to these well-formalized graphs, while also leveraging the RDF(S) semantics. We are also interested in studying the possibility of performing graph entailments directly on the real graph structures.
Part II

Implementation
5

Querying Contextualized Knowledge Graphs

Chapter 5 provides two use cases for representing provenance and time in the BKR and Yago2S datasets. The evaluation of BKR queries compares the performance among existing approaches with the SP.

5.1 Using Singleton Property in Existing Knowledge Bases

5.1.1 BKR and Provenance

The Biomedical Knowledge Repository (BKR) is an extensive knowledge base that integrates biomedical knowledge from multiple sources while tracking their provenance using a unified provenance framework [Sahoo et al. 2011; Sahoo et al. 2010]. A triple in the BKR may be extracted from PubMed articles and is associated with a confidence score from its extraction tool. Given a triple \((s, p, o)\) extracted from PMID?id=1 with confidence score 0.3, and from PMID?id=2 with confidence score 0.8, the current representation of provenance of a triple with PaCE [Sahoo et al. 2010] provided in Table 5.1 is unable to represent.

The basic idea of PaCE is to create one instance of subject, property, and object per context, and asserting the source of those instances. PaCE offered three flavors: minimalist (C1), intermediate (C2) and exhaustive (C3). The source of the triple is inferred from the common source of subject (C1), subject - property (C2), and subject - property - object (C3) instances. Among the three flavors, C1 was proven to be better than the reification approach in terms of number of triples and query performance in [Sahoo et al. 2010]. However, this approach is limited in supporting different dimensions of meta knowledge because it can only represent the source of a triple. Here we have
5.1. USING SINGLETON PROPERTY IN EXISTING KNOWLEDGE BASES

at least two metadata associated with a triple, but it can only represent the source. For instance, if there exists another triple \((s, p', o')\) with a different confidence score 0.2 extracted from the \(\text{PMID?id}=1\), then this score cannot be represented correctly. Since \((s, p, o)\) and \((s, p', o')\) are from the same \(\text{PMID?id}=1\), the instance \(s\_\text{PMID?id}=1\) representing both triples in the \(\text{PMID?id}=1\) is used to assert the meta property \(\text{hasScore}: s\_\text{PMID?id}=1 \text{ hasScore } 0.3\). This automatically infers the confidence score of \((s, p', o')\) in \(\text{PMID?id}=1\) is 0.3, which is incorrect because its score is 0.2.

Using the Singleton Property approach, we can represent the complete metadata information as provided in Table 5.2. Moreover, if we need to represent more meta knowledge dimensions for the triple \((s, p, o)\), we can simply add assertions into the singleton properties \(p\_\text{id}=1\) and \(p\_\text{id}=2\).

**Provenance query.** Since the BKR integrates data from multiple sources, it is common to ask about the provenance of a triple, such as the sources, the publication date, and the confidence score. For example, one may query the sources of a triple that has a high confidence score (above 0.7). This query cannot be supported by PaCE approach because the confidence score is not present. Using the Singleton Property approach, and adopting the metadata query discussed in Section 5, we can create a query like the following:

```
SELECT ?source ?score
?pi derivedFrom ?source . ?pi hasScore ?score .
FILTER (?score > 0.7) }
```

In the next section, we provide a more thorough comparison in the performance of the singleton property approach to existing approaches.

5.1.2 YAGO2 and Temporal-Spatial Enhancement

While YAGO [Suchanek et al. 2007] provides an extensive collection of factual triples extracted from Wiki and other sources, YAGO2 [Hoffart et al. 2012] enhances this knowledge base with temporal and spatial information for those factual triples. This knowledge base becomes aware of times and places and, hence, is capable of answering more complex queries involving such metadata.

Here we reuse the example from [Hoffart et al. 2012] to demonstrate the requirements of representing meta knowledge in YAGO2. We put the set of facts from the example into Table 5.3. YAGO2 uses fact identifiers to represent the facts, and asserts the occurring time and place of the facts by using their fact identifiers as subjects of the meta assertions. It also provides a SPARQL-like query language which allows it to incorporate fact identifiers in the query pattern. Here we propose to replace the fact identifier by the singleton property in representing a statement and asserting its temporal and spatial information. This would enable interoperability between this dataset with
### 5.1. Using Singleton Property in Existing Knowledge Bases

Table 5.1: PaCE approach for \((s, p, o)\) with meta knowledge \((\text{PMID?id}=1, 0.3)\) and \((\text{PMID?id}=2, 0.8)\)

<table>
<thead>
<tr>
<th>Subject</th>
<th>Property</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>s_PMID?id=1</td>
<td>rdf:type</td>
<td>s</td>
</tr>
<tr>
<td>p_PMID?id=1</td>
<td>rdf:type</td>
<td>p</td>
</tr>
<tr>
<td>o_PMID?id=1</td>
<td>rdf:type</td>
<td>o</td>
</tr>
<tr>
<td>s_PMID?id=1</td>
<td>p_PMID?id=1</td>
<td>o_PMID?id=1</td>
</tr>
<tr>
<td>s_PMID?id=1</td>
<td>derivedFrom</td>
<td>PMID?id=1</td>
</tr>
<tr>
<td>p_PMID?id=1</td>
<td>derivedFrom</td>
<td>PMID?id=1</td>
</tr>
<tr>
<td>o_PMID?id=1</td>
<td>derivedFrom</td>
<td>PMID?id=1</td>
</tr>
<tr>
<td>s_PMID?id=2</td>
<td>rdf:type</td>
<td>s</td>
</tr>
<tr>
<td>p_PMID?id=2</td>
<td>rdf:type</td>
<td>p</td>
</tr>
<tr>
<td>o_PMID?id=2</td>
<td>rdf:type</td>
<td>o</td>
</tr>
<tr>
<td>s_PMID?id=2</td>
<td>p_PMID?id=2</td>
<td>o_PMID?id=2</td>
</tr>
<tr>
<td>s_PMID?id=2</td>
<td>derivedFrom</td>
<td>PMID?id=2</td>
</tr>
<tr>
<td>p_PMID?id=2</td>
<td>derivedFrom</td>
<td>PMID?id=2</td>
</tr>
<tr>
<td>o_PMID?id=2</td>
<td>derivedFrom</td>
<td>PMID?id=2</td>
</tr>
</tbody>
</table>

Table 5.2: Singleton Property approach for \((s, p, o)\) with meta knowledge \((\text{PMID?id}=1, 0.3)\) and \((\text{PMID?id}=2, 0.8)\)

<table>
<thead>
<tr>
<th>Subject</th>
<th>Property</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>p?id=1</td>
<td>rdf:singletonPropertyOf</td>
<td>p</td>
</tr>
<tr>
<td>s</td>
<td>p?id=1</td>
<td>o</td>
</tr>
<tr>
<td>p?id=1</td>
<td>derivedFrom</td>
<td>PMID?id=1</td>
</tr>
<tr>
<td>p?id=1</td>
<td>hasScore</td>
<td>0.3</td>
</tr>
<tr>
<td>p?id=2</td>
<td>rdf:singletonPropertyOf</td>
<td>p</td>
</tr>
<tr>
<td>s</td>
<td>p?id=2</td>
<td>o</td>
</tr>
<tr>
<td>p?id=2</td>
<td>derivedFrom</td>
<td>PMID?id=2</td>
</tr>
<tr>
<td>p?id=2</td>
<td>hasScore</td>
<td>0.8</td>
</tr>
</tbody>
</table>
5.1. USING SINGLETON PROPERTY IN EXISTING KNOWLEGE BASES

Table 5.3: Overall statistics of the SP-YAGO2 dataset

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of triples</td>
<td>292,166,376</td>
</tr>
<tr>
<td>Number of generic props</td>
<td>83</td>
</tr>
<tr>
<td>Number of singleton props</td>
<td>62,643,969</td>
</tr>
</tbody>
</table>

Table 5.4: Sample meta properties in SP-YAGO2 including temporal, spatial and provenance

<table>
<thead>
<tr>
<th>Generic property</th>
<th># of singleton properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>extractionSource</td>
<td>32,598,374</td>
</tr>
<tr>
<td>isLocatedIn</td>
<td>1,262,563</td>
</tr>
<tr>
<td>hasLongitude</td>
<td>393,717</td>
</tr>
<tr>
<td>hasLatitude</td>
<td>393,250</td>
</tr>
<tr>
<td>occursSince</td>
<td>553,116</td>
</tr>
<tr>
<td>occursUntil</td>
<td>337,116</td>
</tr>
<tr>
<td>wasBornOnDate</td>
<td>804,816</td>
</tr>
</tbody>
</table>

other RDF datasets and allow them to be queried using standard query language. This RDF representation is compatible with existing RDF datasets and interoperable with other Semantic Web applications that use existing standards such as SPARQL. We do not attempt to compare the query performance or expressiveness between SPARQL and SPARQL-like language used in YAGO2.

The YAGO2 is available in the RDF Turtle format[^1]. However, the link between the triple identifier and the triple itself doesn’t exist. We created a new version of YAGO2 using the singleton property to link the triples and their identifiers. The fact identifiers in commented lines become the property of the fact. The statistics of the SP-YAGO2S version are provided in Table 5.3 and Table 5.4.

**Temporal-spatial query** in the YAGO can be specified in its query language SPARQL-like. For example, for finding concerts that took place near San Francisco, one may create a SPARQL-like query as follows:

5.1. **USING SINGLETON PROPERTY IN EXISTING KNOWLEDGE BASES**

Table 5.5: YAGO2 uses fact ID for representing fact and asserting meta knowledge

<table>
<thead>
<tr>
<th>Id</th>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>GratefulDead</td>
<td>performed</td>
<td>TheClosingOfWinterland</td>
</tr>
<tr>
<td>#2</td>
<td>#1</td>
<td>occursIn</td>
<td>SanFrancisco</td>
</tr>
<tr>
<td>#3</td>
<td>#1</td>
<td>occursOn</td>
<td>1978-12-31</td>
</tr>
</tbody>
</table>

Table 5.6: Singleton property replaces fact ID in asserting meta knowledge

<table>
<thead>
<tr>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>performed?id=1</td>
<td>singletonPropertyOf</td>
<td>performed</td>
</tr>
<tr>
<td>GratefulDead</td>
<td>performed?id=1</td>
<td>TheClosingOfWinterland</td>
</tr>
<tr>
<td>performed?id=1</td>
<td>occursIn</td>
<td>SanFrancisco</td>
</tr>
<tr>
<td>performed?id=1</td>
<td>occursOnDate</td>
<td>1978-12-31</td>
</tr>
</tbody>
</table>

?id: ?s performed ?o .
?id occursIn ?l .
?l hasGeoCoordinates ?g .
SanFrancisco hasGeoCoordinates ?sf .
?g near ?sf .

We may also create an equivalent SPARQL query using the singleton property approach as follows:

?performed_sp rdf:singletonPropertyOf performed .
?s ?performed_sp ?o .
?performed_sp occursIn ?l .
?l hasGeoCoordinates ?g .
SanFrancisco hasGeoCoordinates ?sf .
?g near ?sf .

Given that **near** is a proximate predicate, it may need to be elaborated in the graph pattern of SPARQL query.
5.2 Experiments

In this section, we report the experiments comparing five approaches including the singleton property (denoted by \( \text{SP} \)), standard RDF reification (\( \text{R} \)) and the three flavors of PaCE (\( \text{C1} \), \( \text{C2} \), and \( \text{C3} \)). We will repeatedly use these important notions (\( \text{SP} \), \( \text{R} \), \( \text{C1} \), \( \text{C2} \), and \( \text{C3} \)) in the entire section.

A benchmarking dataset with SPARQL queries for metadata at triple level is not yet available. Therefore, in these experiments, we used the BKR dataset previously described in Section 5.1.1.

For evaluating the query performance, we used two sets of queries: set A and set B. The set A is obtained from the experiments conducted in [Sahoo et al. 2010]. Since all 5 queries of set A include one block of provenance-specific triple patterns related to one data triple, one may wonder how the approaches perform with longer queries. Therefore, we created set B with longer queries, where the lengths of data triple patterns range from 1 to 3. Although the lengths of data triple patterns look small, their corresponding SPARQL query patterns is relatively long, up to 21 triple patterns.

The comparison is based on three quantitative criteria: number of triples, query length and query execution time. Section 5.2.1 describes the comparison based on the number of triples in the five flavors of the same BKR dataset, in detail. Section 5.2.2 and 5.2.3 describe the query experiments.

The datasets and queries used in the experiments are provided for reproducing the experiments\(^2\).

5.2.1 RDF Datasets

In addition to four different RDF datasets from four representation approaches (\( \text{C1} \), \( \text{C2} \), \( \text{C3} \) and \( \text{R} \)) implemented by Sahoo et al. in [Sahoo et al. 2010], we created another dataset SP for our singleton property approach. Instead of reporting the total and provenance-specific triples as in [Sahoo et al. 2010], here we analyzed the number of triples in detail based on the triple pattern, whether it is for data triples, meta triples or statement handling triples. This analysis would be useful for understanding how each type of triples would contribute to the total number of triples when the data input increases.

We classified all triples into three main categories: data triples, metadata triples and triple handlers.

**Data triples** are original triples without any metadata association. The BKR dataset has approximately 23M triples without provenance information. With the provenance information, the number of distinct data triples is 33M because if the same triple occurs in two different articles, it is counted as two data triples. Therefore, we have 33M data triples in the BKR. We denote \( n = 33M \)

for later use in the total number of triples for each dataset.

**Triple handlers** are created to represent data triples as individual resources. Particularly, they are statement instances reified by four triple patterns from R, singleton properties from SP, subject instances from C1, subject-property instances from C2, and subject-property-object instances for C3. While the R approach needs $4n = 112M$ triples to represent statement instances, the SP approach needs only $n = 33M$ triples. The C1 approach needs only $22M$ triples because it contains duplicate subject instances in the same source. In the worst case, C1 approach would need $n$ triples if all the triples do not share any metadata values.

**Metadata triples** are additional triples created by each approach in order to attach metadata into triple handlers. Both R and SP need $n = 33M$ triples for this category because one meta property is asserted for each singleton property and statement instance. Again, in the case of C1, within $22M$ triples representing triple handlers, only $16M$ subject instances asserted the `derives_from` information; the remain $6M$ are for declaring the type of subject instances.

We present the number of triples of each category in Figure 5.1. The total number of triples in SP and R datasets is $3n$ and $6n$, respectively. The sizes of C1, C2 and C3 are application-specific and do not depend only on the number of data triples as the SP and R.

![Figure 5.1: Number of triples in million of 3 categories contributing to the total number of triples.](image)

**Discussion.** The size of SP dataset is half of the size of R dataset and is comparable to the sizes of C2 and C3. The C1 dataset is approximately 30% smaller than the SP dataset due to duplicated triples. However, if the BKR is extended to support provenance at a finer-grain level, such as at sentence level, this C1 approach would lose the size advantage and its size will become the same as the size of SP.
5.2. EXPERIMENTS

5.2.2 Query Set A

In this experiment, we repeated the query experiments performed in [Sahoo et al. 2010]. We reused the sets of queries in [Sahoo et al. 2010] for evaluating the performance among four representational approaches (C1, C2, C3, and R). In order to compare the performance of these four approaches with our singleton property approach, we created one more equivalent set of queries SP. Therefore, the set A has 5 sets of queries in total. We used Virtuoso Open Source 6.1.7 on a Linux server with 8GB RAM for this experiment. Each query run starts with a cold cache. The set of queries are run in two phases.

In the first phase, each query is evaluated for fixed values. Figure 5.2 presents the average of the last 5 of a total of 20 runs. In the second phase, each query is executed with a set of 100 random values. The set of those 100 queries are run 5 times and Figure 5.3 presents the average of 100 queries in the last run. We eliminate the execution times of Q1 because they are too small (less than 1 msec) to be readable in the two charts.

Discussion. For the set of fixed values in phase 1, Figure 5.2 shows that all the SP queries are the fastest ones. For the set of 100 random values, Figure 5.3 shows that SP queries are faster than all others in Q3 and Q5, and also faster than the two approaches in Q2 and Q4. Therefore, we can conclude that for most of the queries in this experiment, the SP queries give better query performance than other approaches.
5.2. EXPERIMENTS

5.2.3 Query set B

In this experiment, we created a set of queries of varying path lengths. Particularly, we created three queries (Q1, Q2, and Q3), each query contains a path of 1, 2, and 3 data triple patterns respectively. After incorporating the metadata triples involving the source into their SPARQL queries, the total number of triple patterns vary among the five representation approaches. The sizes of their corresponding SPARQL queries are presented in Figure 5.4.

Figure 5.3: Query performance in msec.: 100 values.

Figure 5.4: Number of triple patterns within three queries Q1, Q2, and Q3 of query set B.

Since each triple pattern is translated to one query join operator in the query plan, queries with
shorter patterns tend to execute faster. Among all the approaches, the SP queries are the shortest one. Therefore, we expect the SP queries perform better than others in terms of query execution time. We ran the set of queries of each approach 3 times in cold cache and reported the average execution time in Figure 5.5. This experiment was performed on a Ubuntu 12.04 desktop with 4GB of RAM.

![Figure 5.5: Query performance in msec. of the set B.](image)

Discussion. While all the queries in the set A are executed in seconds or minutes, some of the queries in the set B take longer time. Particularly, the longest query of the C1 approach with 18 patterns is in hours, and that of the C2 and C3 approaches with 20 and 21 patterns is in days because of full index lookups in their query plan. On the other hand, the queries in the two approaches, SP and R, still execute in seconds, and the SP queries are little faster than the R queries.

5.2.4 Overall Discussion

Our experiments show that the SP approach gives a decent performance in terms of number of triples, query size and query execution time. Here we do not conclude that our approach is better than other approaches in all the cases. For the number of triples, the C1 approach is the most compact one in the case of BKR where multiple predications share the same source. However, this C1 approach will have the same size with the SP approach in the cases where the data triples do not share metadata values, such as at a finer-grained level of provenance (e.g. statement level instead of article level), or discrete values for the temporal, spatial and certainty properties. For the query performance, the SP queries give the best performance, which is expected and consistent with the
query length comparison. Since only default indexes were created, and no optimization was provided, this leaves a room for query optimization.


6

Tool Implementation:

RDF-contextualizer

Chapter 6 describes our implementation for the tool RDF-contextualizer. We describe how existing datasets can be transformed into SP representation using our tool. We also describe the experiments for evaluating the proposed inference rules on the real-world datasets.

6.1 Transforming Representation

As we discussed earlier in Chapter 1, knowledge bases like DBpedia and Bio2RDF represent the contextual information such as provenance in the form of a quad. Here we discuss how we transform them to the SP representation.

Named Graph. Given any quad in the form of \((s, p, o, g)\), we treat the predicate \(p\) as generic property and the named graph \(g\) as the specific context. We transform the quad to the singleton property representation by creating a singleton property \((sp_i, \text{singletonPropertyOf}, p)\) and asserting the singleton triple \((s, sp_i, o)\). We use the property \texttt{wasDerivedFrom} from the PROV ontology [Lebo et al. 2012] to represent the provenance of the triple \((sp_i, prov:wasDerivedFrom, g)\). The singleton property URIs are constructed by appending a unique string to the generic property URI, with an incremental counter for the unique number in the whole dataset.

Reification. Given any reification in the form of \((stmt\#1, \text{rdf:type}, \text{rdf:Statement}), (stmt\#1, \text{rdf:subject}, s), (stmt\#1, \text{rdf:predicate}, p), (stmt\#1, \text{rdf:object}, o),\) and \((stmt \#1, mp, mo)\), we treat the predicate \(p\) as a generic property. We transform the reification to the SP representation by creating a singleton property \((sp_i, \text{singletonPropertyOf}, p)\), asserting the singleton triple \((s, sp_i, o)\), and attaching the metadata \((sp_i, mp, mo)\).
6.2. EXPERIMENTS FOR SP INFERENCES

Implementation. We developed a Java 8 tool to transform any named graph or reification representation by extending the Jena RIOT API [jen] with high throughput parsers for parsing an input file from any RDF format and generating a stream of RDF quads. For each quad or triple stream, we created a pipeline of streams for converting each quad to the singleton property representation, shortening triples to Turtle format, and writing them to gzip files through buffer writers. As each stream is handled by a separate thread, we can utilize the CPU resources, especially the ones with multiple cores, by creating multiple threads for parsing multiple files concurrently. We validated the syntax of the output datasets by writing an analyzer to parse the output files and also generate the statistics reported in Section 7.3.

6.2 Experiments for SP Inferences

Computational complexity and ease of use are important factors due to the scalable and distributed nature of Semantic Web data [Guha et al. 2004]. Blowing up the number of inferred triples and the run time are the main concerns for reasoning with SP. Running all proposed syntax-based entailment rules on every singleton triple produces at least two more triples (sp-1 and sp-3). The number of inferred triples could go up to multi-billion with datasets like DBpedia and Bio2RDF. Therefore, we perform the experimental evaluation of the proposed rules on various real world datasets with varying sizes to provide the estimated numbers for the two quantitative measurements: number of inferred triples and run time.

6.2.1 Experiment Setup

We use a single server installed with Ubuntu 12.04. It has 24 cores, each core is an Intel Xeon CPU 2.60GHz. We use two disks, one 220 GB SSD for storing input datasets, and one 2.7 TB HDD disk for writing the output. This server has 256 GB of RAM, however, we limit 60GB for each Java program.

We downloaded and used the ontologies and RDF quad datasets from DBpedia [dbp] and four Bio2RDF [bio b] datasets including NCBI Genes, PharmGKB, CTD, and GO Annotations in our evaluation. We chose these quad datasets because they are large and widely-used with high impact in the community. For Bio2RDF datasets, we also downloaded the Bio2RDF mapping files [bio a].

We reported the number of RDF quads per dataset in Table 6.1. The dataset identifier is taken from the first 3 letters of its name. We observed that although there was no duplicate within each file, there were too many duplicate quads among the files within each dataset we downloaded. We also believe that the duplicates may be created on purpose by the authors. Each of these datasets
Table 6.1: Number of RDF quads and unique quads per dataset with NCBI Genes (NCB), DBpedia (DBP), PharmGKB (PHA), CTD, and GO Annotations (GOA)

<table>
<thead>
<tr>
<th>Dataset</th>
<th># of Quads</th>
<th># of Unique Quads</th>
</tr>
</thead>
<tbody>
<tr>
<td>NCB-NG</td>
<td>4,043,516,408</td>
<td>2,010,283,374</td>
</tr>
<tr>
<td>DBP-NG</td>
<td>1,039,275,891</td>
<td>784,508,538</td>
</tr>
<tr>
<td>CTD-NG</td>
<td>644,147,853</td>
<td>327,648,659</td>
</tr>
<tr>
<td>PHA-NG</td>
<td>462,682,871</td>
<td>339,058,720</td>
</tr>
<tr>
<td>GOA-NG</td>
<td>159,255,577</td>
<td>97,522,988</td>
</tr>
</tbody>
</table>

has a number of files and some files may share a number of RDF quads for a topic. For example, NCBI Genes dataset has one file for all genes belonging to one species. Therefore, we keep these datasets in the original version -Dup and created a new version -Unique for each dataset with all duplicates being removed for varying the sizes of the experimental datasets.

We then generated the singleton property version of each dataset by running our tool (briefly described in Section 6.1) with and without the -infer option. We implemented this -infer option for computing all inferred triples for the proposed rules. Our implementation is naive and we do not describe it here. We ran each dataset version at least 3 times and reported the average results in Section 6.2.2.

The tool rdf-contextualizer and the materials used in this paper are publicly available for reproducing the experiments.

### 6.2.2 Results

We consider three dimensions in our evaluation: number of triples, number of singleton triples, and run time. In all figures, the series *Reasoning* stands for running the tool with -infer option and *No-Reasoning* stands for running the tool without -infer option. Running the *NoReasoning* option provides the results for the baseline cost. The differences in the results between the *NoReasoning* and *Reasoning* versions are the extra cost estimated for the computation of the inferred triples. The results are reported in percentage based on the baseline cost.

**Number of Inferred Triples.** Figure 6.1's top chart shows the total number of triples of each dataset in four cases. In general, the *Reasoning* cases contain larger number of triples than the *NoReasoning* cases, from 66.7% to 96.67%. For example, the *Reasoning-Dup* version of NCBI Genes dataset has a larger number of triples than the *NoReasoning-Dup* version.
6.2. EXPERIMENTS FOR SP INFERENCES

Figure 6.1: Total number of triples (top), total number of singleton triples (middle), and run time (bottom) for each dataset: with vs. without reasoning.

genes, the largest dataset, contains about 8 billion inferred triples (66.67%) more than the version NoReasoning-Dup. Similarly, the Reasoning-Unique version of NCBI genes contains about 4 billion inferred triples more than the version NoReasoning-Unique. Some other datasets have their number of inferred triples higher than 66.67%. For example, the Reasoning-Unique version of CTD contains about 950 million inferred triples (96.67%) more than the NoReasoning-Unique version.
6.2. EXPERIMENTS FOR SP INFERENCES

(a) Run time vs. number of triples
(b) Run time vs. number of inferred triples.

Figure 6.2: Run time vs. number of triples (a) and run time vs. number of inferred triples (b) across datasets in two cases: with vs. without reasoning.

**Number of Inferred Singleton Triples.** Resulting from the inference rules involving property hierarchy in the schema, the number of inferred singleton triples varies across datasets as shown in Figure 6.1's middle chart since they have different schema. For example, since NCBI Genes dataset does not have property hierarchy, the number of singleton triples remains the same in both Reasoning and NoReasoning versions. Meanwhile, CTD dataset inferred 194 million singleton triples (30\% more) for the Reasoning-Dup version and 148 million singleton properties (45\% more) for the Reasoning-Unique version.

**Run Time.** Figure 6.1's bottom chart shows the run time execution for all four cases of each dataset. GO Annotations dataset took 11 minutes for 480 million triples of NoReasoning-Dup version and 13 minutes for 796 million triples of Reasoning-Dup version. In other words, it added 2 minutes to the overall process to infer 316 million triples. NCBI Genes, the largest dataset, took 232 minutes for 12 billion triples of NoReasoning-Dup version and 274 minutes for 20 billion triples of Reasoning-Dup version. In terms of percentage, the Reasoning-Dup versions of these datasets add 14-19\% run time for inferring 66.7-96.67\% of the total number of triples in the versions NoReasoning-Dup. It shows that the inferencing time is quite small and practical.

We plot the size of the datasets and the time execution in the same chart to show the correlation between them. Figure 6.2(a) shows that when the size of the datasets increases, the execution time for both Reasoning and NoReasoning case also increases almost linearly as the size of the dataset. This figure also shows that for the same number of triples, the Reasoning versions take shorter run time than the NoReasoning versions. This makes sense because for generating the same amount of triples, the time it takes for the NoReasoning versions to parse content from files and serialize the output to files is longer than the time it does for the Reasoning version to infer the triples. Figure
6.2. EXPERIMENTS FOR SP INFERENCES

6.2(b) plots the time execution and the number of inferred triples. It also shows the run time is linear to the number of inferred triples.

In summary, the proposed rules increase the number of inferred triples up to 96% and add up to 20% of the run time. This experiment shows the feasibility of the proposed rules as the number of triples is manageable. Furthermore, the estimated run time could be improved with optimization. Consequently, the proposed approach can be implemented in practical applications and tools.
7

Tool Implementation: GraphKE

Here we describe the implementation of the two engines in order to demonstrate the feasibility of practical application of our graph model. Since we are proposing a new graph model with a new way of traversing the graph, our aim is not to optimize the shortest path algorithm because there is a multitude of existing solutions for this purpose, such as [Angles and Gutierrez 2008]. Instead, our aim is to implement the two traversal algorithms from two graph models on the same system so that we can fairly evaluate the two graph models. We also explain how existing systems can adopt this model by making changes to their traversal algorithm.

7.1 Shortest Resource Path

We implemented Dijkstra’s algorithm on both NLAN and CKG graph models to find the shortest resource path as defined in Section 4.2.3. One difference between the two algorithms is that traversing the CKG graph visits the predicates and explores their neighbor nodes while traversing the NLAN does not.

While traversing the graph, we use an in-memory priority queue to store all the nodes to be visited and their distance to the source node. We also use a hash table visited to store the set of visited nodes. For each visited node, we keep track of the previous node-id and the shortest distance to the source node. Here we explain our traversal algorithms on both graph models.

Algorithm 1 starts with the source pushed into the priority queue $pqueue$. At each step, one specific node curid with its distance dis to the source is removed from the priority queue and explored. All the pairs $⟨\text{pred, obj}⟩$ are looked up from the hash index. Each $\text{pred}$ (with distance $\text{dis} + 1$ and curid as previous node) or $\text{obj}$ (with distance $\text{dis} + 2$ and $\text{pred}$ as previous node) will be updated in the hash table visited. If the $\text{pred}$ or $\text{obj}$ hasn’t been visited before, or its new distance is shorter than the one in the hash table, the hash table visited will be updated with...
Algorithm 1 Dijkstra’s algorithm on the CKG graph

1: procedure CKG_Dijkstra(source, target)
2:     distance ← 0
3:     pqueue ← source
4:     while (curid ← pqueue_pop(pqueue, dis)) ≠ 0 do
5:         if (curid = target) then
6:             return dis
7:         else
8:             get all the pairs (pred, obj) of curid
9:             for each pair (pred, obj) do
10:                 ret ← update_node(pred, dis + 1, curid)
11:                     if (ret = 1) then
12:                         pqueue_push(pqueue, pred, dis + 1)
13:                     end
14:                 ret ← update_node(obj, dis + 2, pred)
15:                     if (ret = 1) then
16:                         pqueue_push(pqueue, obj, dis + 2)
17:                     end
18:         end
19:     end
20: end
the new distance for that node, and the curid will also be pushed into the pqueue. This process is repeated until the target is found or the pqueue becomes empty, which means no path exists between the source and the target.

Algorithm 2 Dijkstra’s algorithm on the NLAN graph

```
1: procedure NLAN_Dijkstra(source, target)
2:   distance ← 0
3:   pqueue ← source
4:   while (curid ← pqueue_pop(pqueue, dis)) ≠ 0 do
5:     if (curid = target) then
6:       return dis
7:     else
8:       get all the pairs (pred, obj) of curid
9:       for each pair (pred, obj) do
10:          ret ← update_node(obj, dis + 1, curid)
11:          if (ret = 1) then
12:             pqueue_push(pqueue, obj, dis + 1)
13:          end
14:       end
15:     end
16:   end
```

Dijkstra’s algorithm 2 for the NLAN model differs from the Dijkstra’s algorithm 1 for the CKG model in that it does not visit the predicate as explained from lines 10-13 of Algorithm 1. Instead, it only traverses from the curid to the obj and the distance dis + 1. We can observe that the distance associated with the obj in this model is shorter than the one associated with the obj in the CKG model. That is because the NLAN model traverses from sub to obj in one hop, while the CKG model traverses from sub to obj in two hops, with pred in the middle.

7.2 GraphKE

We implemented GraphKE on top of Berkeley DB (BDB) key-value store in C language [Olson et al. 1999]. This implementation can also be adapted to other key-value stores.

Dictionaries. All the URIs and literals are mapped to internal identifiers using 8 bytes. We use odd numbers for literal identifiers and even numbers for the rest. As a literal cannot be subject
of any triple, it will never get explored when traversing the RDF graph. We mark them as a sink node with odd numbers. We leave the URIs and the literals as they are. We did not compress them although compressing the strings may help save space. We created two dictionaries using either the hash index or the B-tree index supported by BDB. From our initial evaluation we observed that the hash index performed better than the B-tree index. However, this may not be the case for other datasets. Therefore, we do not mandate use of the hash index for all datasets. The type of index to be used can be specified before loading data.

Data triples. Each triple is internally represented in the form of \( \langle \text{subject-id}, \text{predicate-id}, \text{object-id} \rangle \). We loaded the triples into a hash index with subject-id as the key and a pair of \( \langle \text{predicate-id}, \text{object-id} \rangle \) as the value. We also created an extra index with subject-id as the key and the number of \( \langle \text{predicate-id}, \text{object-id} \rangle \) pairs as the value.

Basic graph operator. In order to support the CKG graph traversal, given a node-id, we need to lookup all of the adjacent nodes. We use a database cursor to iterate through the data items to find all the pairs (predicate-id, object-id).

7.3 Evaluation

This section describes the empirical evaluation on our graph models through their shortest path algorithms described in Section 7.1.

We deployed the two engines GraphKE and RDF-3X into the same server running Ubuntu 12.04.4 LTS with 256 GB of RAM and 220 GB of SSD hard drive. Since RDF-3X is self-tuned, no configuration is necessary. For Berkeley DB, we set the cache size to 64GB in the configuration.

7.3.1 Dataset

We use the YAGO2S-SF\(^1\) dataset generated by Nguyen et al. \cite{Nguyen et al. 2014}. This dataset was chosen because it is suitable for our purpose with 62 million singleton properties representing the facts that are attached with different kinds of metadata. We load this dataset into the two engines, excluding the file WikipediaInfo.ttl. This file contains triples with the `linksTo` predicate which does not provide a meaningful relationship between the resources. After removing this file, the dataset contains 267,161,278 triples with 77,895,604 URI nodes and 31,110,161 literals. We loaded this dataset into both GraphKE and RDF-3X.

In order to evaluate our graph traversal algorithms, we need to run them with varying lengths. We are able to find resource paths of distances up to 139 in our experiments. Here we explain how

\(^1\)http://wiki.knoesis.org/index.php/Singleton_Property
Table 7.1: Number of politicians and pairs for each input group

<table>
<thead>
<tr>
<th>Group</th>
<th>No of politicians</th>
<th>No of pairs</th>
</tr>
</thead>
<tbody>
<tr>
<td>HR</td>
<td>51</td>
<td>2550</td>
</tr>
<tr>
<td>SS</td>
<td>34</td>
<td>1122</td>
</tr>
<tr>
<td>CS</td>
<td>22</td>
<td>462</td>
</tr>
</tbody>
</table>

we randomly generated different sets of source, target input pairs for the path algorithms from querying the Yago2S-SP dataset.

We observe that the singleton properties of the property holdsPoliticalPosition and the property hasSuccessor form very long paths. The basic pattern is similar to the \((T_1, T_2, T_3)\) of the motivating example, but the pattern is repeated multiple times and forms a much longer path.

```
BASE <http://yago-knowledge.org/resource/>
SELECT ?politician
WHERE {
  ?sp1 rdf:singletonPropertyOf
    <holdsPoliticalPosition> .
}
GROUP BY ?position
```

We run the SPARQL query above to find the set of politicians and we group these politicians by their political positions. We then choose three political positions that have a good number of politicians:

1. White House Chief of Staff (CS),
2. Secretary of State (SS), and
3. Speakers of the U.S. House of Representatives (HR). For each group of politicians, we generate a set of \(\langle\text{politician}_1, \text{politician}_2\rangle\) and \(\langle\text{politician}_2, \text{politician}_1\rangle\) pairs, and we use three sets for the evaluation of the reachability and shortest path queries.

### 7.3.2 Reachability Queries

For reachability queries, we adjusted Dijkstra’s algorithms described in Section 7.1. For every pair \(\langle\text{source}, \text{target}\rangle\), if the algorithm starts with the source and finishes exploring all the nodes in the priority queue without reaching the target, we report that the target is not reachable from the source.
Table 7.2: Number of reachable pairs (R) and time taken (T) per group in the CKG model and the NLAN model, ran in 5 threads in GraphKE

<table>
<thead>
<tr>
<th></th>
<th>CKG</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Pairs</td>
<td>R</td>
<td>T(s)</td>
<td>Avg</td>
<td>R</td>
<td>T(s)</td>
<td>Avg</td>
<td></td>
</tr>
<tr>
<td>HR</td>
<td>2550</td>
<td>579</td>
<td>14782</td>
<td>5.797</td>
<td>0</td>
<td>5592</td>
<td>2.19</td>
<td></td>
</tr>
<tr>
<td>SS</td>
<td>1122</td>
<td>94</td>
<td>7721</td>
<td>6.881</td>
<td>0</td>
<td>2419</td>
<td>2.155</td>
<td></td>
</tr>
<tr>
<td>CS</td>
<td>462</td>
<td>164</td>
<td>3230</td>
<td>7.186</td>
<td>0</td>
<td>885</td>
<td>1.915</td>
<td></td>
</tr>
<tr>
<td>All</td>
<td>4134</td>
<td>837</td>
<td>25733</td>
<td>6.23</td>
<td>0</td>
<td>8896</td>
<td>2.15</td>
<td></td>
</tr>
</tbody>
</table>

Given the three sets of input pairs generated in the manner explained above, we run both NLAN and CKG algorithms on the GraphKE to find the set of pairs that are reachable. Since we are dealing with thousands of pairs, we expedite the process by running the algorithms on multiple threads. Running the CKG with 1 thread took 9,848 seconds (2.73 hrs) to finish processing 1,122 input pairs in the group CS. When we ran it with 5 threads, it only took 3,230 seconds (54 minutes) and averaged 7.2 seconds per pair. We report the experiment with 5 threads in Table 7.2. Two observations are noted from this evaluation.

First, by traversing the CKG graph one can find hundreds of reachable pairs, while traversing the NLAN graph does not find any from all three input groups. For example, out of 462 pairs from 22 politicians holding the position of White House Chief of Staff, 164 pairs are reachable in the CKG. Overall, out of 4,134 pairs from 3 position groups, 837 pairs are reachable.

Second, the algorithm running on the CKG takes about 6 seconds on average, while the one running on the NLAN takes about 2 seconds. Running on the NLAN graph is 3 times faster than the algorithm running on the CKG graph. This result is straightforward and confirms our hypothesis that the NLAN should be faster because it only traverses a subset of nodes that are visited by the CKG. It does not traverse to the predicates and their neighbors. As a consequence, NLAN misses all of the paths connected through singleton properties as predicates, which is a big loss since this dataset contains about 67 million singleton properties.

7.3.3 Shortest Resource Path Queries

Getting the results from the reachability queries described above, we collected one subset of reachable pairs from each input group and used them as the input for the shortest path algorithms. We ran the shortest path algorithms on both GraphKE and RDF-3X in single-threaded mode. For each
7.3. EVALUATION

input pair, we printed out the shortest resource path and its associated triple path. We ran each set of reachable pairs three times on GraphKE and two times on RDF-3X, and calculated the average of these runs. Afterwards, since each reachable input group contains a number of paths sharing the same distance, we also obtained the average time for each shortest distance within each group, and reported them in Figures 7.1, 7.2, and 7.3 for three groups CS, SS, and HR, respectively. Table 7.3 summarizes the total time and average time taken in seconds in the CKG model running in 1 thread and 5 threads in GraphKE, and 1 thread in RDF-3X.

Figure 7.1: Average time taken per shortest distance within CS.

Figure 7.2: Average time taken per shortest distance within SS.
7.3. EVALUATION

Figure 7.3: Average time taken per shortest distance within HR.

Regarding our choice of reporting on RDF-3X in addition to GraphKE, we want to clarify that our focus is on demonstrating the feasibility of meeting practically useful application needs by extending an existing system, as well as developing a new system based on the new model we defined. Although we use the same plots to show the results for two alternatives, the purpose is not to directly compare the two.

**GraphKE.** Our new implementation on GraphKE running with 1 thread takes 353 sec (6 mins), 770 sec (13 mins), 2,688 (45 mins) for 94 (SS), 164 (CS), and 579 (HR) inputs, respectively. It takes less than 10 seconds for every input pair in CS, HR, and for input pairs with shortest distances up to 15 in SS. For the shortest distances from 15 to 139 in SS, it takes up to 30 seconds. On average, it takes 4.553 seconds for an input with 1 thread as shown in Table 7.3.

When we run the GraphKE with 5 threads, it finishes processing all input pairs in 2,582 seconds (44 minutes), compared to 3,811 seconds (64 minutes) in 1 thread. We observe that the time taken for each input pair increases in 5 threads, but the overall time taken for all input pairs reduces 30% compared to 1 thread.

**Findings.** While investigating the results from our algorithm, we found many paths that are interesting to us. We draw one sample path in Figure 7.4. This CKG graph illustrates the use of singleton properties in representing the n-ary relationship between the politician, the position, and the successor. The politicians are connected to their position and successor of that position by a set of singleton properties represented by the rounded squares.

This graph has resulted from running Dijkstra’s algorithm on the CKG graph model to find the
Table 7.3: Total time taken and average time taken in seconds in the CKG model running in 1 thread vs. 5 threads in GraphKE, and 1 thread in RDF-3X

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>GraphKE</th>
<th></th>
<th></th>
<th>RDF-3X</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Pairs</td>
<td>1T</td>
<td>Avg 1</td>
<td>5T</td>
<td>Avg 5</td>
<td>1T</td>
</tr>
<tr>
<td>HR</td>
<td>579</td>
<td>2688</td>
<td>4.64</td>
<td>1826</td>
<td>3.15</td>
<td>10627</td>
</tr>
<tr>
<td>SS</td>
<td>94</td>
<td>353</td>
<td>3.76</td>
<td>237</td>
<td>2.52</td>
<td>888</td>
</tr>
<tr>
<td>CS</td>
<td>164</td>
<td>770</td>
<td>4.695</td>
<td>519</td>
<td>3.17</td>
<td>2388</td>
</tr>
<tr>
<td>All</td>
<td>837</td>
<td>3811</td>
<td><strong>4.55</strong></td>
<td>2582</td>
<td><strong>3.08</strong></td>
<td>13903</td>
</tr>
</tbody>
</table>

Figure 7.4: The shortest resource path and its respective triple path from Andrew Card to Jacob Lew in Yago2S-SP. The resource path includes all the edges along the dashed line in the periphery of the figure, and the triple path includes all the triples where three nodes from a triple form a straight line. Traversing the resource path from start to end will find a sequence of five politicians (Andrew Card, Joshua Bolten, Rahm Emanuel, Pete Rouse, William M. Daley, Jacob Lew) in consecutive terms.

The shortest path from Andrew Card to Jacob Lew. The shortest distance is 15. Starting from Andrew Card, we follow the outer edges along the dashed line in 15 hops, we will find a sequence of politicians Joshua Bolten, Rahm Emanuel, Pete Rouse, William M. Daley and then we will reach Jacob Lew.
For every two politicians in the sequence, the former is succeeded by the latter.

7.3.4 Discussion

We defined a new formal graph model for RDF that does not suffer from limitations of current alternative. To demonstrate viability for practical implementation and use, we showed two implementations - GraphKE as a ground up implementation and adaptation of RDF-3X, for executing reachability and shortest path queries. Based on the experimental results in the Yago2S-SP dataset with 260 million triples, we believe that the CKG graph model can be effectively implemented in a practical system, either new or existing.

7.4 Related Work

Directly related to the formal graph model for RDF are the NLAN diagram, which is currently used in the W3C Recommendation documents, and the bipartite model. As explained in prior sections, our approach differs from the NLAN approach in that predicates are mapped to nodes instead of arcs, and the nodes within the same triple are interconnected by a pair of initial and terminal edges. Our model differs from the bipartite model in that we use a pair of directed edges, whereas the bipartite model uses one extra node and three extra edges to connect this node to subject, predicate, and object.

In a broader context, a comprehensive collection of graph database models are described in the survey paper by Angles and Gutierrez [Angles and Gutierrez 2008]. The formal foundation of these graph models varies based on the basic definition of a mathematical graph, such as directed or undirected, labeled or unlabeled, graph or hypergraph, and node or hypernode. These models differ from our model in that they all represent predicates as labeled arcs, while we map them to nodes. From this perspective, the intuition of our approach is similar to that of conceptual graphs (CGs) [Sowa 1998] which also models relations as nodes. Our approach, however, differs from the CGs in the mechanism to form a statement or formula. We map one relation to a single node and add constraints to bind every two edges in order to form a statement. CGs allow the same relation to be mapped to multiple nodes when the same relation occurs in multiple formulas.
8

Adoption of Singleton Property Model

This chapter proposes the use of singleton property model for building the Open Knowledge Network infrastructure.

8.1 Open Knowledge Network

White paper [1] for the OKN we discussed earlier states:

“Starting in July 2016, the Big Data Interagency Working Group (BD IWG) leadership has been involved in two meetings to discuss the viability, and possible first steps to creating a joint public/private open data network infrastructure, the Open Knowledge Network (OKN). The vision of OKN is to create an open knowledge graph of all known entities and their relationships, ranging from the macro (have there been unusual clusters of earthquakes in the US in the past six months?) to the micro (what is the best combination of chemotherapeutic drugs for a 56 y/o female with stage 3 glioblastoma and an FLT3 mutation but no symptoms of AML?). OKN is meant to be an inclusive, open, community activity resulting in a knowledge infrastructure that could facilitate and empower a host of applications and open new research avenues including how to create trustworthy knowledge networks/graphs.”[2]

The 3rd OKN workshop has laid out the requirements for the data model to support the creation of the OKN infrastructure as follows. The data model must use the triple format and be able represent the provenance, time, location, and trust of a triple also in the form of triples. The publishing model should require minimal cooperation among the publishers.

8.1. OPEN KNOWLEDGE NETWORK

Here we address the above requirements and present our proposal for the data model in Section 8.1.1. We then present the best practices for publishing knowledge to the OKN 8.1.2.

8.1.1 Data Model

The requirements for the data model of the Open Knowledge Network are similar to those for contextualized knowledge graphs. Both use triple format and the representation of contextual metadata is also in the form of triples. Therefore, we propose the use of the singleton property model for representing the open knowledge network. We will modify the SP syntax so that it can also meet the minimal cooperation among the publishers.

Here we consider the following publishing scenario: three publishers A, B, and C would like to publish their own datasets from Examples 1, 2, and 3 from Chapter 1.

For minimizing the cooperation among the publishers, we propose three principles for the construction of the singleton property’s URI: parameterization, identifier, and derivation.

**Parameterization.** The parameterization principle requires every triple identifier to be a parameterized URI with two parameters: \( ds \) for the dataset which the triple belongs to, and \( id \) for the unique identifier within the dataset. The parameter \( ds \) of the singleton property URI can replace the named graph to group a set of triples together. The combination of the two in the singleton property URI can guarantee the global uniqueness of the URI. It requires the cooperation among the publisher: every dataset should have a global identifier \( ds \), and each singleton property should have a local identifier \( id \) (local to the dataset). For example, for the singleton property \( \text{isMarriedTo}\?id=1 \), we add another parameter \( ds=ex1 \). The singleton property URI becomes \( \text{isMarriedTo}\?ds=ex1&id=1 \).

Back to the publishing scenario, each publisher will choose one unique name for their datasets. The publisher A, B, and C will use the name \( ds=ex1 \), \( ds=ex2 \), and \( ds=ex3 \), respectively. Each publisher will manage their \( id \) to be unique within their dataset. The OKN representation for the three datasets are provided in Tables 8.1, 8.3, and 8.2.

**Identifier.** The identifier principle requires every triple to have an identifier represented by the singleton property of the triple. Triple identifiers allow triples to be described and hence, they create links connecting triples within the OKN.

In Table 8.1, the triple \( T_1 \) has the triple identifier \( \text{isMarriedTo}\?ds=ex1&id=1 \). Let D be the publisher who would like to integrate this triple with their datasets and add location information to this triple \( T_1 \). The publisher D could add the new triple as follows.

\[
D_1: \text{isMarriedTo}\?ds=ex1&id=1 \ \text{happenedIn}\?ds=ex5&id=1 \ \text{Chicago}.
\]

On the one hand, the new triple \( D_1 \) adds location information to the triple \( T_1 \). On the other hand, it preserves the origin \( ds=ex5 \) of the new triple.
8.1. OPEN KNOWLEDGE NETWORK

Table 8.1: The compact OKN representation of singleton properties and their meta knowledge assertions for the fact BobDylan isMarriedTo SaraLownds occurring in two documents

<table>
<thead>
<tr>
<th>No</th>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_1$</td>
<td>BobDylan</td>
<td>isMarriedTo? $ds=ex1&amp;$id=1$</td>
<td>SaraLownds</td>
</tr>
<tr>
<td>$T_2$</td>
<td>isMarriedTo? $ds=ex1&amp;$id=1$</td>
<td>hasSource? $ds=ex1&amp;$id=2$</td>
<td>wk:Bob_Dylan</td>
</tr>
<tr>
<td>$T_3$</td>
<td>isMarriedTo? $ds=ex1&amp;$id=1$</td>
<td>extractedOn? $ds=ex1&amp;$id=3$</td>
<td>2009-06-07</td>
</tr>
<tr>
<td>$T_4$</td>
<td>BobDylan</td>
<td>isMarriedTo? $ds=ex1&amp;$id=4$</td>
<td>SaraLownds</td>
</tr>
<tr>
<td>$T_5$</td>
<td>isMarriedTo? $ds=ex1&amp;$id=4$</td>
<td>hasSource? $ds=ex1&amp;$id=5$</td>
<td>wk:Sara_Dylan</td>
</tr>
<tr>
<td>$T_6$</td>
<td>isMarriedTo? $ds=ex1&amp;$id=4$</td>
<td>extractedOn? $ds=ex1&amp;$id=6$</td>
<td>2009-08-08</td>
</tr>
</tbody>
</table>

**Derivation.** The derivation principle allows a generic property to be derived from its singleton property’s URI. Since the singleton property’s URI is created by appending the parameters to the regular predicates. Since the construction of the parameterized URI allows its generic property to be derived, it is unnecessary for this triple to be explicitly presented in the SP datasets of Tables 8.1, 8.3, and 8.2. Such a derivation makes the SP dataset have the same size with the regular RDF datasets while offering the expressiveness of representing contexts with RDF triples.

The triple $S_1$ can be derived on-the-go when needed.

$S_1: isMarriedTo?ds=ex1&id=1 rdf:singletonPropertyOf isMarriedTo.$

8.1.2 Best Practices for OKN Publishing

We combine four principles from Linked Data and three principles from OKN as follows.

1. Use URIs as names for things

2. Use HTTP URIs so that people can look up those names.

3. When someone looks up a URI, provide useful information, using the standards (RDF, SPARQL)

4. Include links to other URIs. so that they can discover more things.

5. Every triple has an identifier represented by the singleton property of the triple.

6. Every triple identifier to be a parameterized URI with two parameters: $ds$ for the dataset which the triple belongs to, and $id$ for the unique identifier within the dataset.

7. The generic property can be derived from its singleton property’s URI
Table 8.2: The compact OKN representation for the contextualized knowledge base from Example 2 (sp: subPropertyOf, and sc: subClassOf)

<table>
<thead>
<tr>
<th>Triple</th>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_1$</td>
<td>chadHurley</td>
<td>type? $ds=ex2&amp;id=1$</td>
<td>youtubeEmp</td>
</tr>
<tr>
<td>$T_2$</td>
<td></td>
<td>from? $ds=ex2&amp;id=2$</td>
<td>2005</td>
</tr>
<tr>
<td>$T_3$</td>
<td></td>
<td>to? $ds=ex2&amp;id=3$</td>
<td>2010</td>
</tr>
<tr>
<td>$T_4$</td>
<td></td>
<td>score? $ds=ex2&amp;id=4$</td>
<td>0.8</td>
</tr>
<tr>
<td>$T_5$</td>
<td>youtubeEmp</td>
<td>sc? $ds=ex2&amp;id=5$</td>
<td>googleEmp</td>
</tr>
<tr>
<td>$T_6$</td>
<td>sc? $ds=ex2&amp;id=5$</td>
<td>from? $ds=ex2&amp;id=6$</td>
<td>2006</td>
</tr>
<tr>
<td>$T_7$</td>
<td>sc? $ds=ex2&amp;id=5$</td>
<td>to? $ds=ex2&amp;id=7$</td>
<td>2017</td>
</tr>
<tr>
<td>$T_8$</td>
<td>sc? $ds=ex2&amp;id=5$</td>
<td>score? $ds=ex2&amp;id=8$</td>
<td>0.9</td>
</tr>
<tr>
<td>$T_9$</td>
<td>chadHurley</td>
<td>ceo? $ds=ex2&amp;id=9$</td>
<td>youtube</td>
</tr>
<tr>
<td>$T_{10}$</td>
<td>ceo? $ds=ex2&amp;id=9$</td>
<td>from? $ds=ex2&amp;id=10$</td>
<td>2005</td>
</tr>
<tr>
<td>$T_{11}$</td>
<td>ceo? $ds=ex2&amp;id=9$</td>
<td>to? $ds=ex2&amp;id=11$</td>
<td>2010</td>
</tr>
<tr>
<td>$T_{12}$</td>
<td>ceo? $ds=ex2&amp;id=9$</td>
<td>score? $ds=ex2&amp;id=12$</td>
<td>0.7</td>
</tr>
<tr>
<td>$T_{13}$</td>
<td>ceo</td>
<td>sp? $ds=ex2&amp;id=13$</td>
<td>worksFor</td>
</tr>
<tr>
<td>$T_{14}$</td>
<td>sp? $ds=ex2&amp;id=13$</td>
<td>derivedFrom? $ds=ex2&amp;id=14$</td>
<td>work</td>
</tr>
<tr>
<td>$T_{15}$</td>
<td>sp? $ds=ex2&amp;id=13$</td>
<td>score? $ds=ex2&amp;id=15$</td>
<td>1.0</td>
</tr>
<tr>
<td>$T_{16}$</td>
<td>worksFor</td>
<td>sp? $ds=ex2&amp;id=16$</td>
<td>member</td>
</tr>
<tr>
<td>$T_{17}$</td>
<td>sp? $ds=ex2&amp;id=16$</td>
<td>derivedFrom? $ds=ex2&amp;id=17$</td>
<td>work</td>
</tr>
<tr>
<td>$T_{18}$</td>
<td>sp? $ds=ex2&amp;id=16$</td>
<td>score? $ds=ex2&amp;id=18$</td>
<td>1.0</td>
</tr>
<tr>
<td>$T_{19}$</td>
<td>ceo</td>
<td>domain? $ds=ex2&amp;id=19$</td>
<td>Person</td>
</tr>
<tr>
<td>$T_{20}$</td>
<td>domain? $ds=ex2&amp;id=19$</td>
<td>derivedFrom? $ds=ex2&amp;id=20$</td>
<td>work</td>
</tr>
<tr>
<td>$T_{21}$</td>
<td>domain? $ds=ex2&amp;id=19$</td>
<td>score? $ds=ex2&amp;id=21$</td>
<td>1.0</td>
</tr>
<tr>
<td>$T_{22}$</td>
<td>ceo</td>
<td>range? $ds=ex2&amp;id=22$</td>
<td>Company</td>
</tr>
<tr>
<td>$T_{23}$</td>
<td>range? $ds=ex2&amp;id=22$</td>
<td>derivedFrom? $ds=ex2&amp;id=23$</td>
<td>work</td>
</tr>
<tr>
<td>$T_{24}$</td>
<td>range? $ds=ex2&amp;id=22$</td>
<td>score? $ds=ex2&amp;id=24$</td>
<td>1.0</td>
</tr>
</tbody>
</table>
Table 8.3: The compact OKN representation the n-ary relationship from Example 3

<table>
<thead>
<tr>
<th>No.</th>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>T₁</td>
<td>John</td>
<td>buys?ds=ex3&amp;id=1</td>
<td>LennyTheLion</td>
</tr>
<tr>
<td>T₂</td>
<td>buys?ds=ex3&amp;id=1 from?ds=ex3&amp;id=2</td>
<td>book.com</td>
<td></td>
</tr>
<tr>
<td>T₃</td>
<td>buys?ds=ex3&amp;id=1 hasPrice?ds=ex3&amp;id=3</td>
<td>$15</td>
<td></td>
</tr>
<tr>
<td>T₄</td>
<td>buys?ds=ex3&amp;id=1 hasPurpose?ds=ex3&amp;id=4</td>
<td>birthdayGift</td>
<td></td>
</tr>
</tbody>
</table>
9

Conclusion

My dissertation aims to support the below thesis statement.

It is possible to develop (1) a compact and formal representation, (2) a sound and complete inference mechanism, and (3) a model-theoretic graph formalism for contextualized knowledge graphs that can be efficiently implemented.

Here we briefly summarize how each point is supported throughout the dissertation.

(1) A compact and formal representation. We have presented the singleton property (SP) representation and its formalism for representing contextualized knowledge graphs in Chapter 2. We described how the singleton property can be used for identifying a triple and asserting various kinds of contexts for this triple in a contextualized knowledge graph. We then provided the model-theoretic semantics for interpreting the singleton properties at three interpretation levels: simple, RDF, and RDFS. For the compactness and efficient implementation, our SP representation has been evaluated by our group (in Chapter 5) and several other research groups with different datasets (as discussed in Chapter 8). The results showed that our SP model offers the most compact representation among the existing approaches. The SP full representation only takes two triples to assert contextual metadata for a triple. The compact SP representation we proposed for the OKN data model does not add any overhead compared to the original knowledge graphs while it has the expressiveness of incorporating contexts to every triple. For the query performance, although the evaluating triple stores have not been optimized for querying the SP representation yet, the SP performance is efficient in some triple stores like Virtuoso. We believe that when the triple stores are optimized for querying SP patterns, the query performance will be further improved. Therefore, our SP representation is compact, formalized by a model theory, and efficiently implemented for contextualized knowledge graphs.

(2) A sound and complete inference mechanism. We formally introduced the new concepts to enable the validation of SP triples and the inferences for the new triples in Chapter 3. We extended...
the SP model-theoretic semantics to include the new concepts. We derived the new set of inference rules and prove them using the extended model theory. We implemented the SP inference rules in the tool RDF-contextualizer and evaluated the performance of computing the inferred triples in Chapter 6. For the experiments, we developed the tool RDF-contextualizer to transform existing datasets (e.g. DBpedia, Bio2RDF) from the named graph, reification, and nanopub representation into our SP representation. These contextualized knowledge graphs have been used in the evaluation of computing the inferred triples. The results showed that the computation is linear. In other words, the inference rules can be efficiently implemented.

(3) A model-theoretic graph formalism. We have presented the formal contextualized knowledge graph model CKG with examples clearly demonstrated in Chapter 4. Our CKG model allows us to represent any set of RDF triples in a formal graph. It also allows us to develop an underlying graph model for the model-theoretic semantics of RDF(S) and the RDFS entailments. We have implemented the CKG graph model in the new GraphKE engine and in the existing triple store RDF-3X (described in Chapter 7). We have evaluated the empirical aspects of our contextualized graph model in the Yago2S-SP dataset to demonstrate its practical feasibility for handling real-world applications.
Appendices
BKR Query Sets

A.1 Query Set A

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX provenir: <http://knoesis.wright.edu/provenir/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
```

Listing A.1: SPARQL query prefixes

```
# Query 1:

SELECT ?st_s_inst ?st_p_inst ?st_o_inst
FROM <http://mor.nlm.nih.gov/bkr_instances_sp>
WHERE {
}
```

Listing A.2: SPARQL query for all statements for a given PMID

# Query 2: C0012963—STIMULATES—C0598981—2006-12-01—2008-12-01
A.1. QUERY SET A

![SPARQL query for all sources for a given statement (Lipoproteins AFFECTS Inflammatory cell)](http://mor.nlm.nih.gov/bkr_instances_sp)

```
SELECT ?source_inst
FROM <http://mor.nlm.nih.gov/bkr_instances_sp>
WHERE {
    meta:C0012963 ?st_p_inst meta:C0598981 .
}
```

Listing A.3: SPARQL query for all sources for a given statement (Lipoproteins AFFECTS Inflammatory cell)

# Query 3:

![SPARQL query for frequency by source for 1 statement](http://mor.nlm.nih.gov/bkr_instances_sp)

```
SELECT ?source_name count(?source_name)
FROM <http://mor.nlm.nih.gov/bkr_instances_sp>
FROM <http://mor.nlm.nih.gov/bkr_schema>
WHERE {
    graph <http://mor.nlm.nih.gov/bkr_instances_sp> {
        meta:C0006307 ?st_p_inst ?o .
    } .
    graph <http://mor.nlm.nih.gov/bkr_schema> {
    } .
}
```

Listing A.4: SPARQL query for frequency by source for 1 statement

# Query 4:

![SPARQL query for counting](http://mor.nlm.nih.gov/bkr_instances_sp)

```
SELECT count(*)
FROM <http://mor.nlm.nih.gov/bkr_instances_sp>
FROM <http://mor.nlm.nih.gov/bkr_metadata>
WHERE {
    graph <http://mor.nlm.nih.gov/bkr_instances_sp> {
        meta:%s ?st_p_inst ?o .
    }
}
```
# Query 5:

```
SELECT ?year count(*)
FROM <http://mor.nlm.nih.gov/bkr_instances_sp>
FROM <http://mor.nlm.nih.gov/bkr_schema>
FROM <http://mor.nlm.nih.gov/bkr_metadata>
where {
  {
    select (bif:year(?source_date)) as ?year
    WHERE {
      graph <http://mor.nlm.nih.gov/bkr_instances_sp> {
        meta:%s ?st_p_inst ?o .
        ?st_p_inst rdf:singletonPropertyOf sn:%s .
        FILTER (?o = meta:%s) .
      } .
      graph <http://mor.nlm.nih.gov/bkr_metadata> {
        ?source_inst dc:date ?source_date .
      }
    }
  }
  FILTER(?source_date >= xsd:date("%s")
    && ?source_date < xsd:date("%s")
  )
}
```
### A.2 Query Set B

# Query 1:

```sparql
select ?s ?p ?o
where { graph <http://data.nlm.nih.gov/bkr_instances_sp>{{
  ?s ?sp1 ?o .
  ?sp1 provenir:derives_from pubmed:10979521-INST .
}}
}
```

Listing A.7: SPARQL query 1 using SP representation

```sparql
select ?s ?p ?o
where { graph <http://data.nlm.nih.gov/bkr_instances_r>{{
  ?st rdf:subject ?s .
}}
}
```

Listing A.8: SPARQL query 1 using R representation

```sparql
select ?s ?p ?o
where { graph <http://data.nlm.nih.gov/bkr_instances_c1>{{
  ?s_inst rdf:type ?s .
  ?o_inst rdf:type ?o .
  ?s_inst provenir:derives_from pubmed:10979521-INST .
}}
```
Listing A.9: SPARQL query 1 using C1 representation

```
select ?s ?p ?o
where { graph <http://data.nlm.nih.gov/bkr_instances_c2>{
  ?s_inst rdf:type ?s .
  ?o_inst rdf:type ?o .
  ?s_inst provenir:derives_from pubmed:10979521-INST .
}}
```

Listing A.10: SPARQL query 1 using C2 representation

```
select ?s ?p ?o
where { graph <http://data.nlm.nih.gov/bkr_instances_c3>{
  ?s_inst rdf:type ?s .
  ?o_inst rdf:type ?o .
  ?s_inst provenir:derives_from pubmed:10979521-INST .
  ?o_inst provenir:derives_from pubmed:10979521-INST .
}}
```

Listing A.11: SPARQL query 1 using C3 representation

```
select ?o1 ?o2 ?pmid2
where { graph <http://data.nlm.nih.gov/bkr_instances_sp>{
  meta:C0543467 ?sp1 ?o1 .
  ?sp1 rdf:singletonPropertyOf sn:TREATS .
  ?sp1 provenir:derives_from pubmed:10979521-INST .
  }}
```
A.2. QUERY SET B

Listing A.12: SPARQL query 2 using SP representation

```sparql
select ?o1 ?o2 ?pmid2
where { graph <http://data.nlm.nih.gov/bkr_instances_r>{
  ?st1 provenir:derives_from pubmed:10979521-INST .

  ?o1 bkr_sn:CAUSES ?o2 .
}}
```

Listing A.13: SPARQL query 2 using R representation

```sparql
select ?o1 ?o2 ?pmid2
where { graph <http://data.nlm.nih.gov/bkr_instances_c1>{
  ?s_inst_1 ?p_inst_1 ?o_inst_1 .
  ?s_inst_1 rdf:type meta:C0543467 .
  ?p_inst_1 rdfs:subPropertyOf sn:TREATS .
  ?o_inst_1 rdf:type ?o1 .
  ?s_inst_1 provenir:derives_from pubmed:10979521-INST .
```
A.2. QUERY SET B

Listing A.14: SPARQL query 2 using C1 representation

```sparql
select ?o1 ?o2 ?pmid2
where { graph <http://data.nlm.nih.gov/bkr_instances_c2>{
  ?s_inst_1 ?p_inst_1 ?o_inst_1 .
  ?s_inst_1 rdf:type meta:C0543467 .
  ?p_inst_1 rdfs:subPropertyOf sn:TREATS .
  ?o_inst_1 rdf:type ?o1 .
  ?s_inst_1 provenir:derives_from pubmed:10979521-INST .
  ?p_inst_1 provenir:derives_from pubmed:10979521-INST .

  ?s_inst_2 rdf:type ?o1 .
  ?o_inst_2 rdf:type ?o2 .
  ?s_inst_2 provenir:derives_from ?pmid2 .
}}}
```

Listing A.15: SPARQL query 2 using C2 representation

```sparql
select ?o1 ?o2 ?pmid2
where { graph <http://data.nlm.nih.gov/bkr_instances_c3>{
  ?s_inst_1 ?p_inst_1 ?o_inst_1 .
  ?s_inst_1 rdf:type meta:C0543467 .
  ?p_inst_1 rdfs:subPropertyOf sn:TREATS .
  ?o_inst_1 rdf:type ?o1 .
  ?s_inst_1 provenir:derives_from pubmed:10979521-INST .

  ?s_inst_2 rdf:type ?o1 .
  ?o_inst_2 rdf:type ?o2 .
  ?s_inst_2 provenir:derives_from ?pmid2 .
}}}
```
A.2. QUERY SET B

Listing A.16: SPARQL query 2 using C3 representation

```sparql
select ?o1 ?o2 ?pmid2 ?o3 ?pmid3
where { graph <http://data.nlm.nih.gov/bkr_instances_sp>{
  meta:C0543467 ?sp1 ?o1 .
  ?sp1 rdf:singletonPropertyOf sn:TREATS .
  ?sp1 provenir:derives_from pubmed:10979521-INST .

  ?o1 ?sp2 ?o2 .

}}
; }
```

Listing A.17: SPARQL query 3 using SP representation

```sparql
select ?o1 ?o2 ?pmid2 ?o3 ?pmid3
where { graph <http://data.nlm.nih.gov/bkr_instances_r>{
```
\begin{verbatim}
?st1 provenir:derives_from pubmed:10979521-INST .

?o1 bkr_sn:CAUSES ?o2 .

}
\end{verbatim}

Listing A.18: SPARQL query 3 using R representation

\begin{verbatim}
select ?o1 ?o2 ?pmid2 ?o3 ?pmid3
where { graph <http://data.nlm.nih.gov/bkr_instances_c1>{
  ?s_inst_1 ?p_inst_1 ?o_inst_1 .
  ?s_inst_1 rdf:type meta:C0543467 .
  ?p_inst_1 rdfs:subPropertyOf sn:TREATS .
  ?o_inst_1 rdf:type ?o1 .
  ?s_inst_1 provenir:derives_from pubmed:10979521-INST .

  ?s_inst_2 rdf:type ?o1 .
}
\end{verbatim}
A.2. QUERY SET B

Listing A.19: SPARQL query 3 using C1 representation

```
select ?o1 ?o2 ?pmid2 ?o3 ?pmid3
where { graph <http://data.nlm.nih.gov/bkr_instances_c2>{
  ?s_inst_1 ?p_inst_1 ?o_inst_1 .
  ?s_inst_1 rdf:type meta:C0543467 .
  ?p_inst_1 rdfs:subPropertyOf sn:TREATS .
  ?o_inst_1 rdf:type ?o1 .
  ?s_inst_1 provenir:derives_from pubmed:10979521-INST .
  ?p_inst_1 provenir:derives_from pubmed:10979521-INST .

  ?s_inst_2 rdf:type ?o1 .
  ?o_inst_2 rdf:type ?o2 .
  ?s_inst_2 provenir:derives_from ?pmid2 .

  ?s_inst_3 rdf:type ?o2 .
}
```
A.2. QUERY SET B

Listing A.20: SPARQL query 3 using C2 representation

```sparql
select ?o1 ?o2 ?pmid2 ?o3 ?pmid3
where { graph <http://data.nlm.nih.gov/bkr_instances_c3>{
  ?s_inst_1 ?p_inst_1 ?o_inst_1 .
  ?s_inst_1 rdf:type meta:C0543467 .
  ?p_inst_1 rdfs:subPropertyOf sn:TREATS .
  ?o_inst_1 rdf:type ?o1 .
  ?s_inst_1 provenir:derives_from pubmed:10979521-INST .
  ?p_inst_1 provenir:derives_from pubmed:10979521-INST .
  ?o_inst_1 provenir:derives_from pubmed:10979521-INST .

  ?s_inst_2 rdf:type ?o1 .
  ?o_inst_2 rdf:type ?o2 .
  ?s_inst_2 provenir:derives_from ?pmid2 .

  ?s_inst_3 rdf:type ?o2 .
} } ;
```

Listing A.21: SPARQL query 3 using C3 representation

```sparql
select ?o1 ?o2 ?pmid2 ?o3 ?pmid3
where { graph <http://data.nlm.nih.gov/bkr_instances_c3>{
  ?s_inst_1 ?p_inst_1 ?o_inst_1 .
  ?s_inst_1 rdf:type meta:C0543467 .
  ?p_inst_1 rdfs:subPropertyOf sn:TREATS .
  ?o_inst_1 rdf:type ?o1 .
  ?s_inst_1 provenir:derives_from pubmed:10979521-INST .
  ?p_inst_1 provenir:derives_from pubmed:10979521-INST .
  ?o_inst_1 provenir:derives_from pubmed:10979521-INST .

  ?s_inst_2 rdf:type ?o1 .
  ?o_inst_2 rdf:type ?o2 .
  ?s_inst_2 provenir:derives_from ?pmid2 .

  ?s_inst_3 rdf:type ?o2 .
} } ;
```
References


A.2. QUERY SET B


A.2. QUERY SET B


A.2. QUERY SET B


