Contextual knowledge representation and reasoning on knowledge graphs

Aljalbout, Sahar

UNIVERSITÉ DE GENÈVE
Département d'Informatique

FACULTÉ DES SCIENCES
Prof. Didier Buchs
Prof. Gilles Falquet(co-director)

Contextual knowledge representation and reasoning on knowledge graphs.

THÈSE

présentée à la Faculté de Sciences de l'Université de Genève
pour obtenir le grade de Docteur ès Sciences, mention Informatique

par

Sahar Aljalbout

de

Liban

Thèse N° 5607

GENÈVE
2021

UNIVERSITÉ
DE GENÈVE

FACULTÉ DES SCIENCES

DOCTORAT ÈS SCIENCES, MENTION INFORMATIQUE

Thèse de Madame Sahar ALJALBOUT

intitulée :

«Contextual Knowledge Representation and Reasoning on Knowledge Graphs.»

La Faculté des sciences, sur le préavis de Monsieur D. BUCHS, professeur ordinaire et directeur de thèse (Département d'informatique), Monsieur G. FALQUET, professeur associé et codirecteur de thèse (Faculté d'économie et de Management, GSEM), Madame G. DI MARZO SERUGENDO, professeure ordinaire (Faculté des sciences de la Société, CUI), Monsieur A. HOGAN, professeur (Departamento de Ciencias de la Computacion, Facultad de Ciencas Fisicas y Matematicas - Universidad de Chile Santiago du Chile, Chile), autorise l'impression de la présente thèse, sans exprimer d'opinion sur les propositions qui y sont énoncées.

Genève, le 15 octobre 2021

Thèse - 5607 -

[illegible]

Le Doyen

N.B. - La thèse doit porter la déclaration précédente et remplir les conditions énumérées dans les "Informations relatives aux thèses de doctorat à l'Université de Genève".

Acknowledgements

This dissertation concludes a long, fulfilling journey that formed me as a researcher. I am happy I was surrounded by amazing people who encouraged me and supported me in reaching my goals.

First and foremost, I want to express my deepest appreciation to my professors, Prof. Gilles Falquet and Prof. Didier Buchs. Thank you for granting me the opportunity to do my Ph.D at the University of Geneva. Your guidance, insights, and support were vital during these years.

I strongly acknowledge the depth of the evaluation, insightful feedback, corrections, and comments from the jury member, Prof. Aidan Hogan. I am likewise thankful to Prof. Giovanna di Marzo for her corrections and feedback on my manuscript.

To my colleagues in the knowledge engineering group and the CUI, thank you for making this a memorable and enjoyable stay.

Many thanks to my friends. Thank you for being there through the long process of writing and completing my dissertation.

Finally, I could not have completed this academic step without the support and love of my family. I dedicate this dissertation to you!

Abstract

Dealing with context-sensitive knowledge has been challenging in Artificial Intelligence in general and the Semantic Web in particular. The standard Semantic Web languages (RDF/RDFS/OWL) and reasoning tools do not explicitly consider the contextual dimension of knowledge, i.e., a particular statement (RDF triple) is valid only in certain circumstances known as contexts. With the rise of modern knowledge graphs like Wikidata (known as a property graph), the topic has gained increasing popularity. Property graphs are graph structures rich with contextual knowledge. For instance, Wikidata contains thousands of attribute-value pairs (i.e., qualifiers) to enrich its statements with information about time, space, causality, provenance, etc.

This thesis studies contextual knowledge representation and reasoning on 1) ontologies and 2) property graphs. First, in the context of ontologies, we present OWL^C, an OWL extension for contextual reasoning. The target is to contextualize OWL axioms with instances and classes of contexts. Next, the applicability of this extension on modern knowledge graphs like Wikidata is examined. This study led to the second and core part of the thesis: reasoning on property graphs. This part starts by describing OWL-based languages' limitations in the context of property graphs. Then, a logical framework that formalizes the impact of Wikidata qualifiers in reasoning is presented. More concretely, a contextual model that handles different categories of qualifiers is designed and formalized in a many-sorted logic. Each sort represents a qualifier category and is a high-level abstraction of several Wikidata qualifiers. Functions and axioms in a specific module of an algebraic specification are designed to define the test and combination operations specific to each qualifier category. Finally, we use this approach to formalize the rules corresponding to the ontological properties and some property constraints of Wikidata and present a corresponding rule-based reasoner with a prototype implementation.

Résumé

Le traitement des connaissances contextuelles a constitué un défi pour l'intelligence artificielle en général et pour le web sémantique en particulier. Les langages standard du web sémantique (RDF/RDFS/OWL) et les outils de raisonnement ne prennent pas explicitement en compte la dimension contextuelle de la connaissance, c'est-à-dire qu'une déclaration particulière (triple RDF) n'est valable que dans certaines circonstances connues sous le nom de contexte. Avec l'essor des graphes de connaissances modernes comme Wikidata (connu sous le nom de graphe de propriétés), le sujet a gagné en popularité. Les graphes de propriétés sont des structures de graphes riches en connaissances contextuelles. Par exemple, Wikidata contient des milliers de paires attribut-valeur (c'est-à-dire des qualificateurs) pour enrichir ses déclarations avec des informations sur le temps, l'espace, la causalité, la provenance, etc.

Cette thèse étudie la représentation des connaissances contextuelles et le raisonnement contextuel sur 1) les ontologies et 2) les graphes de propriétés. Tout d'abord, pour les ontologies, nous présentons OWL^C, une extension d'OWL pour le raisonnement contextuel. L'objectif est de contextualiser les axiomes OWL avec des instances et des classes de contextes. Ensuite, l'applicabilité de cette extension sur les graphes de connaissances modernes comme Wikidata est examinée. Cette étude a conduit à la deuxième partie de la thèse : le raisonnement sur les graphes de propriétés. Cette partie commence par décrire les limitations des langages basés sur OWL dans le contexte des graphes de propriétés. Ensuite, un cadre logique qui formalise l'impact des qualificateurs de Wikidata dans le raisonnement est présenté. Plus concrètement, un modèle contextuel qui gère différentes catégories de qualificateurs est conçu et formalisé dans une logique sortée (à plusieurs sortes). Chaque sorte représente une catégorie de qualificateur et est une abstraction de haut niveau de plusieurs qualificateurs de Wikidata. Les fonctions et axiomes d'un module spécifique d'une spécification algébrique sont conçus pour définir les opérations de test et de combinaison spécifiques à chaque catégorie de qualificateurs. Enfin, nous utilisons cette approche pour formaliser les règles correspondantes aux propriétés ontologiques et à certaines contraintes de propriété de Wikidata et présentons un raisonneur à base de règles correspondant avec un prototype d'implémentation.

Table of contents

List of figures

List of tables

Chapter 1

Introduction

Chapter Contents

1.1 Introduction

Dealing with context-sensitive knowledge has been challenging in artificial intelligence (AI) in general and the Semantic Web in particular. AI applications require modeling, retrieving, and reasoning on contextual knowledge. Contextual knowledge refers to knowledge that is true only in specific contexts (i.e., circumstances). Despite the frequent use of the term "context" in contextual representation and reasoning, the notion of context remains vague. It differs from one application to another, from one author to another. Although everyone seems to have an intuitive idea about context, there is not much formalization. Recently, research in context-aware knowledge representation and reasoning has become more relevant in the Semantic Web and intelligent systems.

Unfortunately, the standard Semantic Web languages (RDF/RDFS/OWL) and reasoning tools do not explicitly consider the contextual dimension of knowledge. For example, there is no straightforward way to indicate that a particular statement (RDF triple) is valid only in a specific context. Furthermore, it is possible to neither restrict the validity of an axiom to a particular context nor add contextual information that specifies the cause or the source of the statement. With the rise of knowledge graphs (KGs) and their relevance in industrial and academic fields, the topic has increasingly gained in popularity. Knowledge graphs, such as Wikidata (also known as a property graphs), are rich in contextual knowledge. They contain millions of attribute-value pairs that enrich statements with information about time, space, causality, provenance, etc. However, they still lack the languages and tools for contextual reasoning.

Despite the oldness of this topic, many issues with context representation and reasoning remain unsolved. Through this thesis, the following problems and research directions were identified and addressed:

- Formalizing the semantics of the term "context": despite its frequent use in contextual representation and reasoning, the term "context" remains highly polysemous. Its meaning differs from one application to another, from one author to another. In the Semantic Web field, the term "context" is sometimes indifferently used to refer to statements on statements, annotations, meta-knowledge, etc. Moreover, these meanings are rarely explicitly defined, compared, or classified.

- Contexts in ontologies: academics have proposed several works, most theoretical and rarely used in practice. Most importantly, there is a lack of tools that handle contexts based on solid theoretical grounds and principles for context-aware ontologies, KG, and application design.

- Contexts in property graphs: this direction of research has been less explored. The richness of property graphs in particular (and knowledge graphs in general) with contextual information make them more appealing to be studied by a context practitioner. Extensive research should be done on formalizing and reasoning on the context semantics in property graphs. We focus our work along this direction.

1.2 Scope of the work and research questions

This thesis presents models for reasoning on ontologies and property graphs. The critical concept that is technically explicated and utilized in this thesis is that of context. What is a context? It is refined transcendently, starting from ontologies and moving to property graphs. Through the thesis, we study the following:

- The different context definitions developed in the fields of knowledge representation and the Semantic Web, such as description logics and the Web Ontology Language, are explored.
- In ontologies: the two-dimensional approach of contextual reasoning is explored. The inclusion of validity contexts in DL ontology is studied. The thesis mainly shows how to apply it on the OWL RL profile by including contexts in the inference rules.
- In property graphs: we study the simultaneous consideration of several categories of contexts in the reasoning on large property graphs. Notably, the thesis answers the questions asked by the Wikidata reasoning group on the use of Wikidata's qualifiers[1] in reasoning. These qualifiers' impact on formalizing the semantics of Wikidata is also explored. We propose a convenient entailment regime.

These problems are approached using a logic-based approach. We use logics only to formalize the semantics of property graphs. We set ourselves and propose answers to the following research questions:

- **RQ1** What are the obstacles encountered when implementing practical contextual reasoning for ontologies on the SW ? How can validity contexts be added in an OWL2 profile?
- **RQ2** To what extent can the Semantic Web languages serve the purpose of context representation and reasoning on property graphs?
- **RQ3** What types of context exist in property graphs such as Wikidata? How can they be used in contextual reasoning on property graphs?
- **RQ4** How can this logical formalism be used to extract/infer implicit contextual knowledge from these graphs?

[1] https://www.wikidata.org/wiki/Wikidata:WikiProject_Reasoning/Use_cases

1.3 Contributions and the outline of the thesis

The content of this thesis is divided into eight chapters, where the particular contributions are highlighted as follows. Chapter 2 presents background information and basic knowledge of the field of knowledge representation and reasoning. In chapter 3, related works in contextual reasoning are analyzed, starting from theories in artificial intelligence, through logical approaches, to practical works in the field of the Semantics Web. Chapter 4 presents the first contribution of the work: a contextual extension of OWL for contextual reasoning with validity contexts. The application of this extension on a property graph like Wikidata was studied, where we focus on the limitations of OWL-based languages on property graphs. This experiment led to a new contextual language for property graphs, formalized and operationalized in chapters 5 to 8. Chapter 5 explores the Wikidata data model, its types of contexts represented by qualifiers, and its non-formalized conceptual properties and constraints. In chapter 6, a categorization of Wikidata's qualifiers is proposed, based on which the semantics of Wikidata's qualifiers is formalized using a many-sorted logic. This logic is backed by an algebraic specification used to represent the behaviors of qualifiers in reasoning. The algebraic specification is presented and discussed in chapter 7. This chapter also formalizes the conceptual and constraint properties. Chapter 8 discusses the implementation: the proposed theory is operationalized with an implementation of the rules and constraints. Further, the methodology to follow to apply the many-sorted theory on a knowledge graph is described. In addition, the reasoning results on different use cases from Wikidata are presented. Finally, chapter 9 concludes the thesis by summarizing the contributions and opening the door for future works.

Chapter 2

Background

Chapter Contents

2.1 Knowledge representation in the Semantic Web

The Semantic Web is an initiative for turning the Web into a global, distributed repository of machine-understandable information. The idea of adding semantics to web resources was recognized after the publication of the famous 2001 Scientific American article by Tim Berners-Lee, Jim Hendler, and Ora Lassila, where they describe a vision of the Semantic Web .

> The Semantic Web is not a separate Web but an extension of the current one, in which information is given well-defined meaning, better-enabling computers and people to work in cooperation. (Berners-Lee et al., 2001).

The vision of machine understandability is achieved by well-defined knowledge representation formalisms for publishing information on the Web, particularly the Resource Description Framework (RDF), RDF Schema (RDFS), and the family of Web Ontology Languages (OWL).

2.1.1 RDF

RDF (Miller, 1998) stands for Resource Description Framework and is a standard for knowledge representation and interchange, developed and agreed upon by the World Wide Web Consortium (W3C). Being a robust technology for modeling data, RDF provides a foundation for publishing and linking these data. Initially, it was a part of the Semantic Web stack, but it has currently been used for representing high-quality connected data. Representing data in RDF, in an integrated way, allows identifying, disambiguating, and interconnecting information using machines and various systems to be read, analyzed, and acted upon.

Technically speaking, RDF is a standardized data model based on directed edge-labeled graphs that represent knowledge as triples consisting of a subject, predicate, and an object. Different syntaxes exist among them the Turtle syntax that expresses triples as simple patterns that are easily readable for both humans and machines. Take the following example:

```
:Elizabeth   :knows   :Richard
```

This triple consists of the three parts separated by white spaces: first, the subject :Elizabeth, second, the predicate :knows, and third, the object :Richard, with a final period at the end. The three identifiers are opaque for a machine. They are considered a meaningless string of characters like any other. There is actually not much semantics to it. However, if IRIs are used as identifiers for each part, then each concept is uniquely identified and consequently receives a well-defined interpretation. For instance,
http://dbpedia.org/resource/Elizabeth_Taylor

http://xmlns.com/foaf/0.1/knows
http://dbpedia.org/resource/Richard_Burton
In the above example, the identifiers have been replaced by IRIs that respectively correspond to, Elizabeth Taylor, the relation "knowing," and Richard Burton. In this way, the meaning is constructed; a concept is uniquely identified by one or more IRIs, and a machine can then interpret statements about this concept by simply matching its IRI. Now, if the machine is aware that the above IRI identifies Elizabeth Taylor, consequently, it can easily determine that the triple is a statement about this person. Furthermore, if it is aware of the predicate "knows," it can determine that the triple means "Elizabeth Taylor knows somebody." Indeed, the comprehension of Richard Burton IRI implies that the machine can "understand" the triple: Elizabeth has a "knows" relation to Richard or simply "Elizabeth knows Richard" in the human language.

Since IRIs appear extensively in RDF, a Turtle syntax has been provided to make it easier and shorter to use them by utilizing an abbreviated syntax:

```
@prefix dbp: <http://dbpedia.org/resource/>.
@prefix  foaf: <http://xmlns.com/foaf/0.1/>.
dbp:Elizabeth_Taylor foaf:knows  dbp:Richard_Burton
```

Prefixes are used for recurring parts of IRIs. This saves space and makes the document of triples clearer. If the machine does not have any knowledge about the IRIs, it can simply dereference them. By using the dereferencing concept (i.e., using the HTTP protocol to retrieve a representation of the RDF resource), a machine can discover the meaning of the concept in terms of its relation to other concepts.

Besides all that, RDF defines different types of nodes, including i) Internationalized Resource Identifiers (IRIs), which allow for the global identification of entities on the Web; ii) literals, which allow for representing strings and other datatype values (integers, dates, etc.); and iii) blank nodes, which are anonymous nodes that are not assigned an identifier.

2.1.2 RDFS

RDF does not make assumptions about any particular application domain, nor does it define the semantics of any domain. To specify these semantics, a developer or user of RDF must define what those vocabularies mean in terms of a set of basic domain independent structures defined by RDF Schema (RDFS). RDFS (Brickley and Guha) describes such classes and properties and indicates which classes and properties are expected to be used together.

In RDFS, a class is any resource having an rdf:type [1] property whose object is the resource rdfs:Class. Therefore, a Human class would be described by assigning the class a URI, say

[1] https://www.w3.org/TR/rdf-schema/#ch_type

ex:Human [2], and describing that resource with an rdf:type property whose object is the resource rdfs:Class.

ex:Human rdf:type rdfs:Class.

Hierarchical constructions can also be illustrated using the *subclass* property. To illustrate the fact that man is a subclass of the human class and woman is a subclass of the human class, we use the following representation:

ex:Man rdfs:subClassOf ex:Human.

ex:Woman rdfs:subClassOf ex:Human .

The meaning of the rdfs:subClassOf relationship is that any instance of class ex:Man or ex:Woman is also an instance of class ex:Human. Thus, if resource ex:Gilles is an instance of ex:Man, then, based on the declared rdfs:subClassOf relationship, RDF software written to understand the RDF Schema vocabulary can infer the additional information that ex: Gilles is also an instance of ex:Human.

2.1.3 OWL

OWL, which stands for the Web Ontology Language, is a more powerful standardized language at the expressive level then RDF et RDFS, which is specially designed to represent ontologies in the Semantic Web (which will be explained later) that allows to easily create, share, and exchange knowledge in the Semantic Web. OWL is derived from the language DAML + OIL. It covers most features of the DAML + OIL language and renames most of its primitives (McGuinness et al., 2002). OWL is equipped with axioms for class hierarchies. Suppose you want to express the fact that the Person class is equivalent to Human class. Using the equivalentClass construct of OWL.

```
ex:Person rdf:type owl:Class.
ex:Human rdf:type owl:Class.
ex:Person owl:equivalentClass ex:Human.
```

Or if you want to declare that the two classes are disjoint

```
ex:Person rdf:type owl:Class.
ex:Human rdf:type owl:Class.
ex:Person owl:disjointWith ex:Human.
```

In addition, OWL is equipped with a set of constructs that allows expressing more expressive facts than using RDFS. For instance, you may want to create a class Student, which is the union of bachelor's, master's and PhD students.

[2]using ex: to stand for the URIref http://www.example.org/schemas/humans, which is used as the prefix for URIrefs from example.org's vocabulary

```
:Student owl:equivalentClass [
a owl:Class ;
owl:unionOf (:BachelorStudent :MasterStudent :PhDStudent) ] .
```

2.2 Knowledge graphs

The use of the term "knowledge graph" became common after Google started an initiative of the same name in 2012, although the term has been used in the literature since at least 1973 (Schneider, 1973). There is currently no clear definition of the term "knowledge graph"; it is sometimes used as a synonym for ontology. One common interpretation is that a knowledge graph represents a collection of interlinked descriptions of entities (e.g., real-world objects, events, situations, or abstract concepts). Unlike ontologies, knowledge graphs often contain large volumes of factual information with **less** formal semantics. Knowledge graphs represent information in a similar way to human intelligence. Data is stored in graphs that mimic the intuitive way humans understand information rather than it being stored into a relational database format. The renewed interest in graph-based knowledge representation has been triggered by the modern rise of artificial intelligence (AI) in research and applications. Knowledge graphs (KGs) have become essential components for intelligent assistants such as Apple's Siri and Amazon's Alexa (Haase et al., 2017), for the question-answering of modern search engines such as Google and Microsoft Bing, and for new expert systems in the style of IBM's Watson (Ferrucci et al., 2010). A variety of knowledge graphs are found in the industry (e.g., at Google and Facebook), on the Web (e.g., Wikidata (Vrande, 2014)), and in research as well (e.g., YAGO2 (Hoffart et al., 2013) and Bio2RDF (Belleau et al., 2008)). What makes them different from plain graphs is their enriched structure that includes additional annotations to provide contextual information for every edge or node. They are characterized by the following properties (Krötzsch, 2017):

1. Normalization: Information is decomposed into small units, interpreted as edges of some form of graph.
2. Connectivity: Knowledge is represented by the relationships between these units.
3. Context: Data is enriched with contextual information to record aspects such as temporal validity, provenance, trustworthiness, or other side conditions and details.

Current graph-based knowledge representation formalisms suffer from an inability to handle contextual information in the form of sets of attribute–value pairs.

Unfortunately, the "knowledge graphs" approached in much of the research are different from the real KGs used in applications and industries. For instance, in academia, researchers often employ small, simplified excerpts of KGs for benchmarking. Technically speaking, KG embeddings often conceive graphs as sets of “triplets” — a simplified form of RDF triples —

and n-ary relations, compound structures, quantities, and annotations are poorly supported (Krötzsch, 2019). However, real KGs used in applications are very large. In addition, they are often based on annotated directed graph models, where directed, labeled edges are enriched with additional context information (validity time, data source, trustworthiness, etc.). Related data models are used in Wikidata as in the widely used property graph data model, where graph edges are conceived as complex data objects with additional key-value pairs.

2.2.1 Data graphs

At the foundation of any knowledge graph is the principle of first applying a graph abstraction to the data, resulting in an initial data graph (Hogan et al., 2020). In this section, a selection of graph-structured data models are presented. Using the tourism board example in (Hogan et al., 2020), the differences between the difference types are illustrated.

a) Directed edge-labeled graphs It is characterized by the following two structures:

- A set of nodes. Nodes are used to represent entities in the case of knowledge graphs.
- A set of directed labeled edges between those nodes. They are used to represent (binary) relations between those entities.

Figure 2.1 represents data about the names, types, start and end date times, and venues for events. The $EID15$ node has a name, three types, and a start and end time. Representing incomplete information requires simply omitting a particular edge. A standardized data model based on directed edge-labeled graphs is the Resource Description Framework [3].

b) Property graphs Property graphs are a fundamental type of knowledge graphs. The idea behind a property graph model is to offer additional flexibility when modeling data as a graph. For instance, property–value pairs and a label can be associated with both nodes and edges. Figure 2.2 depicts the companies offering flights between Santiago and Arica. For the node Santiago, we can find two attributes value pairs (country = chile, capital = true). We find property graphs mostly used in graph databases, such as Neo4j.

c) Graph dataset A graph dataset is a set of named graphs and a default graph. Each named graph is constituted by a graph ID and a graph. The default graph, in contrast, is a graph without an ID and is referenced "by default" if a graph ID is not specified. Fig 2.3 shows an example of a graph dataset with two named graphs – one for events and one for routes – and a default graph containing meta-data about the named graphs. Note that graph names can also be used as nodes in a graph. In addition, nodes and edges can be repeated

[3] https://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/

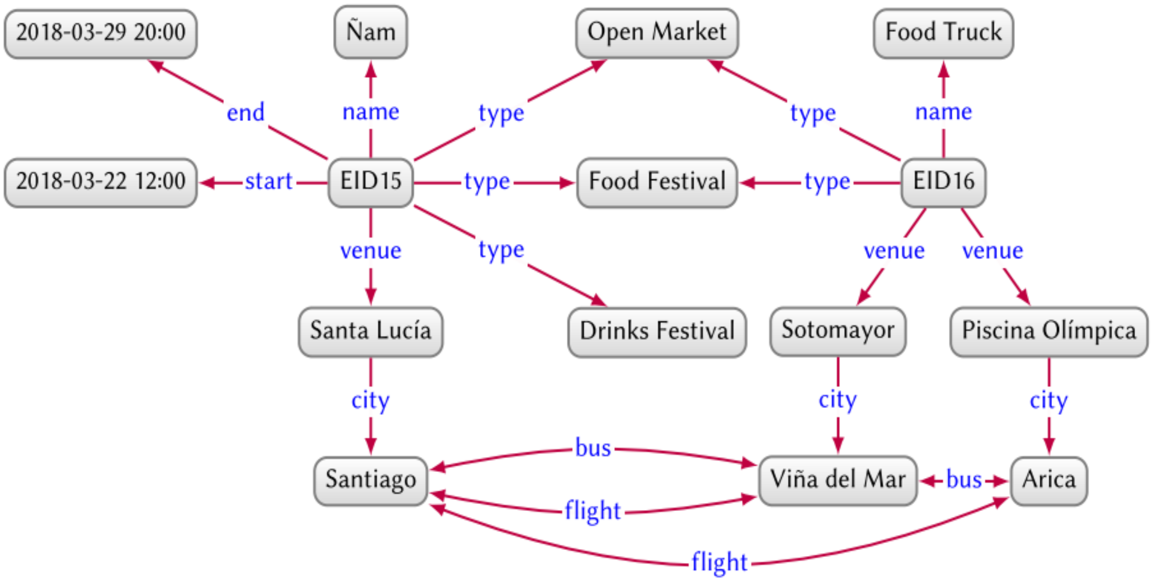


Fig. 2.1 Directed edge-labeled graph describing events and their venues (taken from Hogan et al. (2020))

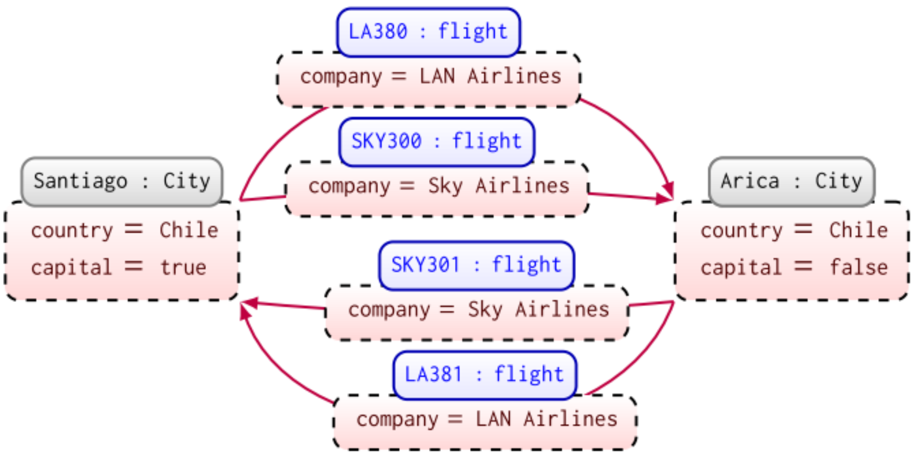


Fig. 2.2 Property graph representing companies offering flights between Santiago and Arica (taken from Hogan et al. (2020))

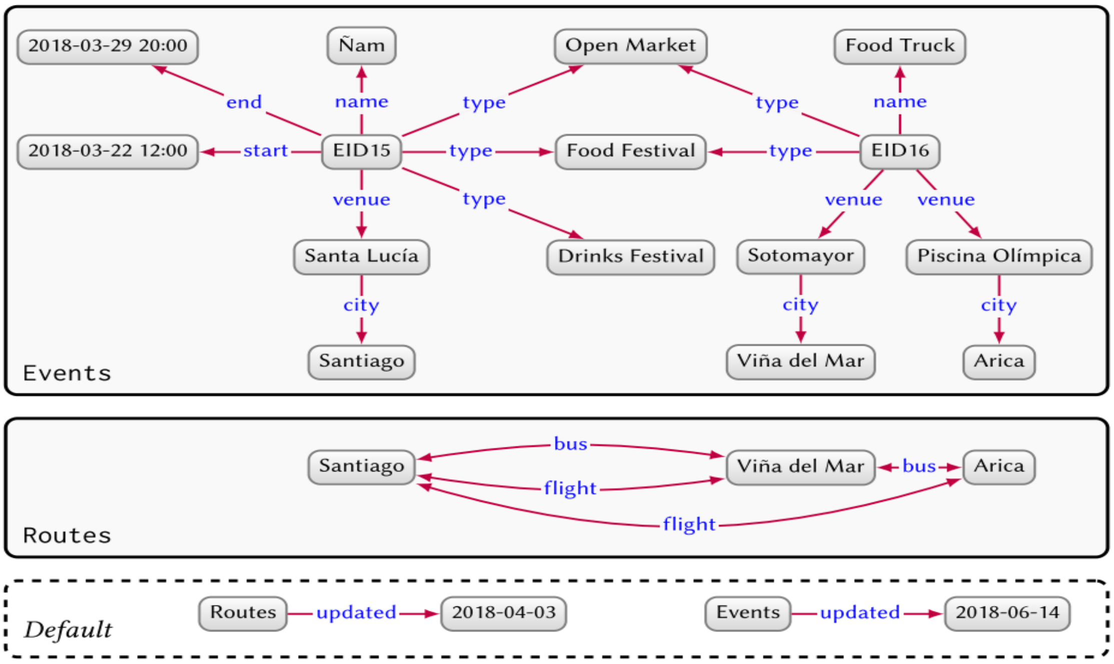


Fig. 2.3 Graph dataset with two named graphs and a default graph describing events and routes (taken from Hogan et al. (2020))

across graphs. Hence, the same node in different graphs will typically refer to the same entity, allowing data integration on that entity when merging graphs.

The previous three models are popular examples of graph representations. However, there might be other graph data models like nested graphs (Angles and Gutierrez, 2008) or complex nodes that may contain individual edges (Angles and Gutierrez, 2008) etc.

2.3 Ontologies

The word “ontology” has different meanings across different communities. The term stems from the philosophical study of ontology (i.e., specifically the branch of philosophy that explains the nature and structure of “reality”). Aristotle analyzed this subject in his Metaphysics and defined Ontology (in capital) as the science of “being qua being,” or more clearly the study of attributes that belong to things because of their very nature (Guarino et al., 2009).

From a computational context, people, organizations, and software systems must communicate among themselves. However, each one uses different concepts and structures due to different backgrounds. This results in a lack of shared understanding that leads to poor communication and limits the interoperability and the potential of reuse and sharing. All this leads to a wasted effort reinventing the wheel (Uschold et al., 1996). For all these reasons and more, an ontology comes into play. An ontology is a formal representation of what terms mean within a given domain. It is used as a countable noun ("an ontology," with lowercase initial) in the computational context. Computational ontologies are used to formally model the structure of a system or more concretely the relevant entities and relations that emerge from its observation, and which are actually useful for our purposes. For example, this system can be a company with all its employees and their relationships. Ontologies are considered the backbone of the Semantic Web since they provide information that machines can understand.

The utility and usefulness of an ontology depends on the level of agreement on what it defines, how detailed it is, and how broadly it is adopted. There are many ontological languages including first-order logic, description logics, etc. The most popular ontology language used in practice is the Web Ontology Language (OWL)[4], which is an application of description logics. It is recommended by the W3C and compatible with RDF graphs as well.

2.3.1 Core components

The key idea behind ontologies is that there are individuals that are instances of classes, which have attributes and are related to each other by relations.

Individuals Also known as instances, they are the primary, "ground level" components of an ontology. The individuals in an ontology represent concrete or abstract objects such as *Albert Einstein*, *Barack Obama*, *Eiffel Tower*, *Lake Léman*, etc.

Properties They are also known as relations. We can assert (binary) relations (using triples) between individuals using edges. Properties between individuals in an ontology specify how individuals are related to other individuals. They usually have a domain and range. We may define a pair of properties to be equivalent, inverse, or disjoint. A particular property may be further defined to denote a transitive, symmetric, asymmetric, reflexive, or irreflexive relation. We can also define the multiplicity of the relation denoted by properties, based on being functional (many-to-one) or inverse-functional (one-to-many), etc.

Classes The basic idea of conceptualization is to create classes of individuals that have common characteristics. Class hierarchies can also be modeled using a sub-class relation. OWL supports sub-classes, and many additional features, for defining and making claims about classes.

Axioms Axioms are statements that say what is true in the domain. They are usually of two types: the first type of axiom is used to define classes definitions (i.e., what the class is about) and inclusions (i.e., a class included in another one); the second type is about class instances, or more concretely individuals belonging to a certain class.

[4]https://www.w3.org/TR/2012/REC-owl2-primer-20121211/

Table 2.1 DL constructors and semantics

Name	syntax	Semantics
Top concept	$\top$	$\Delta^{\mathcal{I}}$
Bottom concept	$\bot$	$\emptyset$
Intersection	$C \sqcap D$	$\{x \in \Delta^{\mathcal{I}} \| x \in C^{\mathcal{I}} \cap D^{\mathcal{I}}\}$
Union	$C \sqcup D$	$\{x \in \Delta^{\mathcal{I}} \| x \in C^{\mathcal{I}} \cup D^{\mathcal{I}}\}$
Existential Restriction	$\exists R.C$	$x \in \Delta^{\mathcal{I}} \| \exists y : (x,y) \in (r)^{\mathcal{I}} \wedge y \in (C)^{\mathcal{I}}$
Universal Restriction	$\forall R.C$	$x \in \Delta^{\mathcal{I}} \| \forall y : (x,y) \in (r)^{\mathcal{I}} \longrightarrow y \in (C)^{\mathcal{I}}$
Complement	$\neg C$	$\{x \in \Delta^{\mathcal{I}} \| x \notin C^{\mathcal{I}}$

2.4 Description logics

Description logics (DLs) were initially introduced as a way to formalize the meaning of frames and semantic networks (Sowa, 2014). They are a family of knowledge representation languages used in ontological modeling because they provide one of the main underpinnings of OWL, although they have been used in knowledge representation long before ontological modeling.

DLs are restricted fragments of first-order logic (FOL) that permit decidable reasoning tasks, such as entailment checking. However, over time, the focus shifted dramatically to studying the computational properties of various fragments.

A DL language L is defined over a vocabulary $\Sigma = (N_C, N_R, N_I)$, where $N_C = \{A, B, ...\}$ is a countably infinite set of concept names, $N_R = \{r, s, ...\}$ a set of role names, and $N_I = \{a, b, ...\}$ a set of individual names. The semantics of L is given through interpretations $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, where $\Delta^{\mathcal{I}}$ is a non-empty domain of individuals, and $\cdot^{\mathcal{I}}$ is an interpretation function, which fixes the meaning of the vocabulary by mapping $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$, for every $A \in N_C$, $r^I \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$, for every $r \in N_R$, and $a^I \in \Delta^{\mathcal{I}}$, for every $a \in N_I$. The grammar of L is defined relative to the given DL. Particular logics in the DL family allow different collections of constructors for expressing complex concepts, roles, and axioms. The interpretation function is extended over the complex expressions according to the fixed conditions associated with each constructor. Tables 2.1 and 2.2 present $\mathcal{ALC}$[5] the syntax and semantics of the constructors and axioms. In the table, C, D are concept names, $r, s \in N_R$, and $a, b \in N_I$.

Building blocks: The axioms are usually divided into three building blocks:

- ABox: referring to axioms asserting facts such as concept assertion and role assertion
- TBox: referring to terminological knowledge such as concept inclusion and concept equivalence

[5]which is the prototypical DL Attributive Concept Language with Complements

Table 2.2 DL axioms and semantics

Name	syntax	Semantics
Concept Assertion	$C(a)$	$a^{\mathcal{I}} \in C^{\mathcal{I}}$
Role Assertion	$r(a,b)$	$(a^{\mathcal{I}}, b^{\mathcal{I}}) \in r^{\mathcal{I}}$
Concept inclusion	$C \sqsubseteq D$	$C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$
Role inclusion	$r \sqsubseteq s$	$r^{\mathcal{I}} \subseteq s^{\mathcal{I}}$
Domain Restriction	$dom(r) \sqsubseteq C$	$\{x \in \Delta^{\mathcal{I}} \mid \exists y : (x,y) \in (r)^{\mathcal{I}}\} \subseteq C^{\mathcal{I}}$
Range Restriction	$ran(r) \sqsubseteq C$	$\{y \in \Delta^{\mathcal{I}} \mid \exists x : (x,y) \in (r)^{\mathcal{I}}\} \subseteq C^{\mathcal{I}}$

- RBox: referring to modeling relationships between roles with axioms such as role inclusion and role equivalence

DL fragments: Many different description logics have been introduced in the literature. Typically, they can be characterized by the types of constructors and axioms that they allow. SROIQ is currently one of the most expressive DLs commonly considered. The axioms of SROIQ are of the following basic forms:

- ABox: $C(a) \quad R(a,a) \quad a \approx a \quad a$ different a
- TBox: $C \sqsubseteq C \quad C \equiv C$
- RBox: $R \sqsubseteq R \quad R \equiv R \quad R \circ R \sqsubseteq R \quad Disjoint(R,R)$

a stands for any individual specification, C denotes a class specification, and R represents a role specification. To obtain lightweight DLs, restrictions must be added on the expressivity. The three main approaches for obtaining lightweight DLs are EL, DLP, and DL-Lite [6]. A complete list of DL languages and their properties can be found here.

2.4.1 Relationship between DL and OWL

OWL is one of the most important applications of description logics today. The main building blocks of OWL are very similar to those of DLs. However, some differences are noted in the terminology such as concepts being called classes and roles being called properties. From a historical point of view, OWL has also been conceived as an extension to RDF. However, the formal semantics of RDF is different from that of DLs. Extending the RDF semantics to OWL improves the compatibility between the two, but it also makes reasoning undecidable. For this reason, we can find both styles of formal semantics for OWL: the Direct Semantics based on DLs and the RDF-Based Semantics. In this section, the main focus is therefore on the Direct Semantics of OWL. The OWL 2 web page provides the direct model-theoretic semantics for OWL 2, which is compatible with the description logic SROIQ.

[6]which also correspond to language fragments OWL EL, OWL RL and OWL QL of the Web Ontology Language.

There are few differences between OWL DL under the Direct Semantics and SROIQ. Syntactically, OWL provides considerably more operators, although they are logically redundant, including, e.g., the domain and range constructs of a property (Krötzsch et al., 2014). OWL also includes some expressive features like support for datatypes and datatype literals.

2.4.2 Open world assumption and non-unique name assumption

We distinguish between the open world assumption (OWA) and the closed world assumption (CWA). The OWA assumes incomplete information by default, while the CWA assumes that if we don't have the information then it is false. Consider the example, "Can pigs fly?" Let us assume that we do not have an answer for this question in our knowledge base. In the CWA, because our knowledge base does not contain the answer, we assume it to be false. However, in the OWA, unless we have a statement (or we can infer) “pigs can/cannot fly,” we return “don't know”. OWL is based on the OWA.

On the other hand, normally, if two things have different names (IDs), they are, by default, different. We call this the unique name assumption (UNA). However, have a look at the names in the following list:

$$\{Women, Femmes, Dames, Frauen, Donne\}$$

They all refer to the term "women" but in different languages. We can state that they may refer to the same individual or instance by using the non-unique name assumption (NUNA). This is the case of OWL. It does not follow the UNA; thus, we can say that two objects are the same (i.e., using owl: sameAs) or are different (i.e., using owl:differentFrom).

2.5 Semantics and entailments

In this section, the difference between the model-based and rule-based semantics is shown. To further clarify the explanation, the cases of RDFS and description logics are discussed.

2.5.1 Model-based semantics:

We define an interpretation of a data graph as composed of two elements: a domain graph and a mapping from the terms of the data graph to those of the domain graph. Thus, for each term in a triple, we define the notion of interpretation as a function that:

- associates IRIs and blank nodes to domain objects,
- associates literals to values in a datatype domain, and
- associates the interpretation of properties to binary relations over domain objects (extensions).

We consider a graph interpretation to be true (with respect to a given graph) if it represents every edge and node of the graph and satisfies some semantic conditions (e.g., the extension of the interpretation of *rdfs:subClassOf* is a transitive relation). A model is a true interpretation. Models are defined such that they capture the intended meaning of the vocabulary. If a triple α is true in a model M, then we write M $\models \alpha$.

An entailment relation X – representing the notion of logical consequence – is usually defined between two graphs. We say that a graph G1 X-entails a graph G2 if each true X-interpretation of G1 is also a true X-interpretation of G2. (X is a notation such as RDF, RDFS, OWL, ...). For example, let G1 be the following graph:

```
ex:Picasso   rdf:type          ex:Painter.
ex:Painter   rdfs:subClassOf   ex:Artist.
ex:Artist    rdfs:subClassOf   ex:Person.
```

and G2 :

```
ex:Picasso   rdf:type          ex:Painter.
ex:Painter   rdfs:subClassOf   ex:Artist.
ex:Artist    rdfs:subClassOf   ex:Person.
ex:Painter   rdfs:subClassOf   ex:Person.
```

We say that G1 RDFS-entails G2 because each true interpretation of G1 is a true interpretation of G2. This is possible due to the semantic conditions imposed by the RDFS semantics on rdfs:subClassOf (e.g., the extension of the interpretation of rdfs:subClassOf is a transitive relation — some axiomatic triples).

On the other hand, description logics are also defined using a model-theoretic semantics in tables 2.1 and 2.2. Expressive DLs support complex entailments involving existentials, universals, counting, etc. In a DL knowledge base, something is true if it is a logical consequence of the axioms (i.e., if it belongs to every model of the axioms), whereas it is false if its negation is true. Otherwise, it is unknown. To decide whether an ontology is consistent, entailments are reduced to satisfiablity (Horrocks and Patel-Schneider, 2003). Thereafter, methods such as tableau can be used to check satisfiability, cautiously constructing models by completing them along similar lines to the materialisation strategy but additionally branching models in the case of disjunction, introducing new elements to represent existentials, etc. If any model is successfully "completed," the process concludes that the original definitions are satisfiable (Hogan et al., 2020).

2.5.2 Logical rule-based semantics

A logic could also be defined using only inferences rules (i.e., there is no need to define a model-theoretic semantics). In this case, the entailment is what we can do with the inference rules. OWL 2 RL is a typical example. It is a trade between expressivity and applicability. Applications that require scalable reasoning without sacrificing too much expressive power may use OWL RL. This trade is achieved by defining a syntactic subset of OWL 2 that could be implemented using rule-based technologies. An example of a union rule used in OWL RL is as follows:

```
T(?c, owl:unionOf, ?x) and
LIST[?x, ?c1, ..., ?cn] and
T(?y, rdf:type, ?ci )
----------------------------
T(?y, rdf:type, ?c),
```

where the predicate T generalizes triples. $T(s, p, o)$ represents a generalized RDF triple with the subject s, predicate p, and the object o. The rules are given as universally quantified first-order implications.

In addition, we can define a logic with a model-theoretic semantics like OWL and then extend it with a rule-based language like SWRL. The reason behind it is that OWL 2 is not able to express all type of relations. For example, it cannot express the relation "child of married parents." This is because there is no way in OWL 2 to express a relation between individuals with which an individual has relations. To overcome this, the expressivity of OWL has been extended by adding SWRL (Semantic Web Rule Language) rules (Horrocks et al., 2004) to an ontology. Consider the following example in OWL:

```
SymmetricObjectProperty(:hasSpouse)
ObjectPropertyAssertion(:hasSpouse :Tom :Katherine)
ObjectPropertyAssertion(:hasParent :Karl :Tom )
ObjectPropertyAssertion(:hasParent :Karl :Katherine )
```

We can add an SWRL rule saying that an individual X from the Person class, which has parents Y and Z such that Y has spouse Z, belongs to a new class ChildOfMarriedParents. Such a rule is best described in the Protege[7] syntax:

```
Person(?x), hasParent(?x, ?y), hasParent(?x, ?z), hasSpouse(?y, ?z)
-> ChildOfMarriedParent(?x)
```

2.6 Rule-based engines using SPARQL-based languages

Besides the logical inference rules discussed in the previous sections, domain rules could be used to enhance the reasoning. This section presents some of the SPARQL-based languages

[7]https://protege.stanford.edu

that are used in rule-based languages nowadays and in the rest of the manuscript as well. SPARQL is the standard query language and protocol for RDF triplestores and Linked Open Data. It stands for “SPARQL Protocol and RDF Query Language.” It was designed and endorsed by the W3C. SPARQL 1.0 was acknowledged by W3C on January 15, 2008, as an official recommendation. The main idea is to query for pattern matching. The following is a simple pattern to be used for a SPARQL query:

```
prefix definitions
select output variables
[ from graph ]
where { basic graph pattern }
```

The syntax is similar to turtle with variables.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX : <http://cui.unige.ch/geo/>
SELECT ?x
WHERE {?x :near :Florence. }
```

The above query retrieves the entities that are near Florence. The variable ?x will be bound to the subjects of triples with a predicate :near and object :Florence.

2.6.1 SPIN

Also known as SPARQL Inferencing notation [8], SPIN is a W3C Member Submission to represent SPARQL rules and constraints on Semantic Web models. Recall that SPARQL (Pérez et al., 2009) is an RDF query language, i.e., a semantic query language for knowledge bases to retrieve and manipulate data stored in the Resource Description Framework format.

2.6.2 SHACL

SHACL[9] was influenced by SPIN and can be seen as the legitimate successor. SHACL, which stands for "Shapes Constraint Language," is a language for validating RDF graphs against a set of conditions. The conditions are provided as shapes and other constructs expressed in the form of an RDF graph. We call these RDF graphs "shapes graphs" in SHACL, and the RDF graphs that are validated against shapes are called "data graphs". Take the following example:

Shapes Graphs

```
ex:PersonShape
    a  sh:NodeShape;
    sh:targetNode  ex: Alice.
```

[8]https://spinrdf.org
[9]https://www.w3.org/TR/shacl/

Data Graph

```
ex:Alice a  ex:Person
ex:Bob a  ex:Person
```

In the shape graph, we are defining a person shape that is a node shape and has target node Alice; this means that if we have a data graph, we will target the individual Alice.

Chapter 3

State of the Art

Chapter Contents

3.1 Introduction

This chapter presents the most illustrative types of research in the area of contextual knowledge representation and reasoning. The goal is to answer the two following questions:

- What are the contextual description logics that we have so far?
- Is there any usable framework for contextual knowledge representation and reasoning in the Semantic Web?

To answer these questions, the state of the art is divided into two sections:

- Section 2 presents scientific papers that discuss the theories of contexts and the history of how they were introduced to artificial intelligence and logic. Subsequently, research works proposing contextual description logics are analyzed. We note that these works are usually applied to ontologies. The last subsection is dedicated to presenting the few logical approaches for contextual reasoning on property graphs.
- Section 3 presents different methods and techniques used to encode contexts in the triple language of RDF. It also presents some frameworks for contextual RDF/RDFS reasoning.

Through the chapter, we describe the semantics and structure of the term *context* for each work, which might vary slightly from one scientific paper to another.

3.2 Contexts in symbolic artificial intelligence and logics

3.2.1 Introducing contexts to artificial intelligence:

Although the notion of contexts has been around for a long time, there is no consensus on the semantics of a context. The problem of formalizing and operationalizing contexts for use in AI applications has a relatively long history and has been addressed by numerous researchers in a plethora of works. In the late 1980s, J. McCarthy proposed to formalize contexts as a crucial step toward solving the problem of generality. McCarthy (1987) wrote, "When we take the logical approach to AI, lack of generality shows up in that the axioms we devise to express common-sense knowledge are too restricted in their applicability for a general common sense database [. . .]. Whenever we write an axiom, a critic can say that the axiom is true only in a certain context. With a little ingenuity, the critic can usually devise a more general context in which the precise form of the axiom doesn't hold."

The great appeal of McCarthy's paradigm stems from the simplicity and intuitiveness of the following three points:

- **Contexts should be treated as first-class objects** A context is anything that can be represented by a first-order term and used in a statement of the form $ist(c, \phi)$, saying that formula ϕ is true (ist) in context c, e.g., $ist(Hamlet, `Hamlet\ is\ a\ prince.')$ (Klarman, 2017). By adopting a strictly formal view on contexts, one can bypass unproductive debates on what they really are and instead take them as primitives underlying practical models of contextual reasoning.

- **Contexts have properties and can be described.** As first-order objects, contexts can be described in a first-order language. This allows for describing them generically through quantified formulas such as $\forall x(C(x) \rightarrow ist(x, \phi))$, expressing that ϕ is true in every context of type C, e.g., $\forall x(barbershop(x) \rightarrow ist(x, `Main\ service\ is\ a\ haircut.'))$ (Klarman, 2017).

- **Contexts are organized in relational structures.** Contexts are entered and then exited, accessed from other contexts or transcended to broader ones. Formally, this can be captured by allowing nestings of the form $ist(c, ist(d, \phi))$, e.g., $ist(France, ist(capital, `Thecity\ river\ is\ Seine.'))$ (Klarman, 2017).

In line with McCarthy's work, an annotated logic was introduced by Subrahmanian (1987). Later on, it was developed by Kifer and Subrahmanian (1992), annotations were extended to allow variables and functions, and it was argued that such logics can be used to provide a formal semantics for rule-based expert systems with uncertainty. They introduced a new semantics for such programs based on ideals of lattices.

In 1993, in his PhD dissertation and under McCarthy's supervision, Guha proposed a first formalization of context along the lines suggested in (McCarthy, 1993). Guha presented a formal semantics for the formula $ist(c, p)$. Among several important concepts, he discussed a) the notion of context structure and vocabulary, b) the distinction between grammaticality (expressions that are well-formed in a sort of universal language) and meaningfulness (expressions that have a meaning in a given context), c) the notion of lifting axioms (namely axioms relating the truth of formulas in different contexts). On top of that, he tackled several applications and techniques of context-based problem-solving techniques (e.g., lift-and-solve).

In the same year, Giunchiglia (1993) argued that the problem of locality was the main reason behind the formalization of context, especially the problem of modeling reasoning that uses only a subset of what reasoners know about the world. The key point is that people do not use all their knowledge while solving a problem on a given occasion. Instead, they construct a "local theory" and then use it as if it contains all pertinent facts about the problem.

In 1994, Giunchiglia and Serafini (1994) proposed multi-context systems (MCS) as a proof-theoretic framework for contextual reasoning with bridge rules. They are a special kind of inference rules whose premise and conclusion hold in different contexts.

In 2001, Ghidini and Giunchiglia (2001) proposed local modal semantics as a model-theoretic framework for contextual reasoning. It is characterized by the principle of locality (reasoning always happens in a context) in addition to the principle of compatibility (there can be relationships between the reasoning process in different contexts).

As we can see, all the aforementioned papers seek to formalize contexts and operationalize them. These research works were published before the era of the Semantic Web, but will form later a strong basis for many description logics and Semantic Web frameworks.

3.2.2 Contexts in description logics

With the rise of the Semantic Web that uses description logics as a backbone for its OWL language, researchers started to explore the possibilities of including contexts in description logics and OWL. We present the following highly cited works:

C-OWL is a scientific paper by Bouquet et al. (2003a) and is one of the earliest works to include contexts in OWL. The authors considered that an ontology is built to be shared, while a context is built to be kept local. By local, they mean "not shared." They considered ontologies to be shared because they are best used in applications where there's an intention to use and manage common representations, whereas contexts are used in applications where the problem is the use and management of local, non globalized representations. The motivations behind C-OWL include i) the need to keep track of the source and target ontology of a specific piece of information; ii) giving up the hypothesis that all ontologies are interpreted in a single global domain; and iii) most importantly, establishing context mappings, which means the possibility of stating that two elements (concepts, roles, individuals) of two ontologies, although (extensionally) different, are contextually related. Based on that, they defined contextual ontology as an ontology whose contents are kept local (and therefore not shared with other ontologies) and put in relation with the contents of other ontologies via explicit mappings. They define the C-OWL syntax by taking the OWL syntax and by adding "bridge rules". These rules allow to relate, at both the syntactic and semantic level, concepts, roles, and individuals in different ontologies. A set of bridge rules between two ontologies is called a context mapping. The bridge rules they propose in the paper were first put forth by Bouquet et al. (2003b). An advantage of this approach is that it provides interoperability between local ontologies when different local ontologies cannot be integrated or merged because of the mutual inconsistency of their information. However, finding mappings between local ontologies may not be that easy because of the lack of common vocabularies.

Next, we present a work called **two-dimensional description logics of context** by Klarman and Gutiérrez-Basulto (2011a). The authors study a natural extension of DLs, in the style of two-dimensional modal logics, which support the declarative modeling of viewpoints as contexts, in the sense of McCarthy, and their semantic interoperability. The results relate to the two following problems:

1. How to extend DLs to support the representation of inherently contextualized knowledge?
2. How to use knowledge from coexisting classical DL ontologies while respecting its context-specific scope?

The framework provided as a solution is introduced in two steps: 1) simple interoperability system and 2) abstract interoperability systems.

Simple interoperability system (SIS): It involves a fixed number of explicitly named contexts and an object language, which extends a DL with special context operators. The object language permits the representation of

$$\langle c \rangle\, C \quad \langle c \rangle\, r \quad \langle c \rangle\, a$$

The operator $\langle c \rangle$ "imports" the meaning of the bounded expression from the context denoted by name c, to the context of occurrence. The resulting expressions include a concept (C), a role (r), and an individual name(a) of the object language, in a context c.

SIS could be used to represent a collection of ontologies where each ontology represents knowledge relevant to a certain context. A mapping is used to provide alignment between the concepts of these ontologies. For instance, two ontologies that might overlap:
In the ontology O_c, we have:

$$\mathsf{Staff} \sqsubseteq \exists \mathsf{isEmployed.Company}$$

$$\mathsf{Staff(J.Smith)}$$

In the ontology O_d, we have:

$$\mathsf{Employee} \sqsubseteq \exists\ \mathsf{employedIn}.\top$$

$$\mathsf{Employee(JohnSmith)}$$

The two ontologies might be integrated by the mean of constraints:

$$\langle \mathsf{c} \rangle\ \mathsf{Staff} \equiv \langle \mathsf{d} \rangle\ \mathsf{Employee}$$

$$\langle \mathsf{c} \rangle\ \mathsf{isEmployed} \sqsubseteq \langle \mathsf{d} \rangle\ \mathsf{employedIn}$$

$$\langle \mathsf{c} \rangle\ \{\mathsf{J.Smith}\} \equiv \langle \mathsf{d} \rangle\ \{\mathsf{JohnSmith}\}$$

These mappings behave like bridge rules in DDL lifting information from one context to another.

The complexity of reasoning in the simple interoperability language is the same as that in description logics language.

Abstract interoperability system: The approach is generalized to account for a possibly infinite domain of contexts and include a lightweight metalanguage for describing them. The metalanguage consists of a set of concept names, the top concept $\top$, and a set of individual names. The axioms of the metalanguage are formulas:

$$A \sqsubseteq B \quad A(c)$$

where A and B are concepts, and c is an individual.
The object language is closed under the constructors of two concept-forming operators:

$$\langle A \rangle\, C \quad [A]C$$

where A is a concept of the metalanguage, and C a concept of the object language. Informally, the concept $\langle A \rangle\, C$ denotes all objects that are C in some context of type A, whereas $[A]C$ objects that are C in all such contexts. An abstract interoperability system is a pair $K = (C, O)$, where C is a set of axioms of the metalanguage, and O is a set of formulas:

$$c : \phi \quad A : \phi$$

where ϕ is an axiom of the object language, c is a context instance, and A is a concept of the metalanguage. A formula $A : \phi$ states that the axiom ϕ must hold in all contexts of type A.

Interestingly, reasoning in the abstract interoperability system is not significantly harder than reasoning in the underlying DLs. The complexity of entailment is *Ptime* when the fragment is $\mathcal{EL}$, $EXPTIME\!-\!complete$ when $\mathcal{ALC}$, $NEXPTIME\!-\!complete$ when $\mathcal{ALCO}$.

Expressive metalanguages. In this case, the metalanguage is the standard DL. It makes sense to use it in cases where the description of knowledge requires not only concept tags but also properties. This is the case when describing provenance (authorship, date, place, relationships to other sources, etc.). The complexity depends on the fragments employed in the two languages; see Table 3.1 where the columns illustrate the complexities of the metalanguages and the lines are the complexities of the object language. We use the expressive metalanguage as a basis for our work on OWL^C in the next chapter. Therefore, the syntax and semantics are directly presented in chapter 4.

Table 3.1 Complexity of reasoning with the expressive metalanguages: the rows represent the fragment of the core language and the columns represent the fragment of the meta-language

	$\mathcal{EL}$	$\mathcal{ALC}, \mathcal{ALCO}$
$\mathcal{EL}$	*Ptime*	$EXPTIME - hard$
$\mathcal{ALC}$	$EXPTIME - complete$	$NEXPTIME - complete$
$\mathcal{ALCO}$	$NEXPTIME - complete$	$NEXPTIME - complete$

In another work of Klarman and Gutiérrez-Basulto (2011a) published in the same year, Klarman adds two accessibility constructors.

$$\langle r.C \rangle \, D \quad [r.C]D$$

where r a relation, C is a concept from the context language, and D is a concept from the object language. The first formula denotes all objects that are D in some context of type C accessible from the current one through r. The second formula denotes all objects that are D in some context of type C accessible from the current one through r.

Another work that uses the two-dimensional approaches is by Bozzato and Serafini (2013b). They use a DL language for the representation of the context language. We will develop this work in detail within the part of the conceptual framework. This is because this work is an extension of a previous work that goes into the conceptual framework section, and we would like to present them together.

3.2.3 The shift for contextual reasoning on modern knowledge graphs

The search for a suitable logical formalism that handles contexts started with the rise of knowledge graphs like Wikidata. The shift from description logics to extended first-order logics was noticed. Research done in this direction is limited. We cite the following papers.

Attributed description logic by Krötzsch et al. (2018) enriches DL concepts and roles with finite sets of atribute-values pairs called annotations. It allows concept inclusions as well to express constraints on annotations. In addition, every fact can occur with one or more finite annotation sets. Syntactically, the approach attaches annotation variables to role and concept names within axioms. It adds pre-conditions that restrict the sets that these variables may represent. For example, the following DL axiom

$$\mathsf{Z} : \lfloor \mathsf{pos} : \mathsf{fellow} \rfloor \qquad \exists \mathsf{employer@Z.cern} \sqsubseteq \mathsf{CernFellow@[start} : \mathsf{Z.start,\ end} : \mathsf{Z.end]}$$

where Z is implicitly universally quantified. This means that everybody employed by CERN as a fellow is in the class $CernFellow$. This class is annotated with the same start and end date as the employment.

The second interesting work is called **logic on MARS** Marx et al. (2017b). The authors offer a formalisation of a generalized notion of property graphs, called multi-attributed relational structures (MARS), and introduce a matching knowledge representation formalism, multi-attributed predicate logic (MAPL). The formalism could be applied to multi-attributed knowledge graphs like Wikidata. Wikidata supports attributed statements but goes beyond property graphs by allowing attributes with multiple values. A decidable fragment of MAPL was then proposed. The fragment is characterized by several useful expressive features including the class of formulae used to express conditions on attribute-value sets. They are called specifiers. For example, the formulae

$$[\mathsf{start}{:}{*},\ \mathsf{loc}{:}\mathsf{Montreal}](\mathsf{U})$$

states that U is an annotation that contains 0 or more values for attribute start, the value Montreal for attribute loc, and no other attributes. A specifier could be closed or open. Closed specifiers are like the previous one and are used to describe annotations that only use certain expected attributes. Open specifiers however characterize the set of all supersets of a given annotation. For example, for the open specifier,

$$\lfloor \mathsf{start}{:}{+},\ \mathsf{loc}{:}\mathsf{Montreal} \rfloor$$

We should have one or more values for the attribute start, the value *Montreal* for the attribute *loc*, and possibly other attributes. An eligible interpretation is the following:

$$\{start : 1964, loc : Montreal, end : 1974\}^{\mathcal{I}} \in \lfloor start : +, loc : Montreal\rfloor^{\mathcal{I}}$$

Specifiers can be built from open and closed specifiers using the operators $\cap, \quad \cup \quad , and \quad \setminus$ The fragment also includes functions. In addition, they introduce a rule-based fragment of MAPL, where rules are allowed to contain arbitrary specifiers. While still giving high expressivity, this is sufficient to obtain decidability for fact entailment.

Marx and Krötzsch (2017) implement later SQID as a browser application that integrates data obtained from multiple sources such as Wikidata's live SPARQL query service (https://query.wikidata.org), full data dumps analyzed offline, Wikidata's Web API, and other Wikimedia sources. A couple of years later, Patel-Schneider and Martin (2020) extended MARS with Wikidata datatypes. Their work supports the explicit expression of Wikidata's ontological axioms and provides a means for handling them. In an extended technical report (Martin and Patel-Schneider, 2020), they also explain how they support the expression of nearly all current Wikidata property constraints, as well as a variety of other constraints.

3.3 Contexts in the Semantic Web

In this section, we move from the theoretical level to the practical one. We analyze scientific papers that propose techniques of context implementation, encoding, or prototypes. They are divided in two categories: the first one pertains to techniques for encoding contexts in RDF, and the second one is about contextual frameworks. However, before, we share the following properties, which according to Bozzato et al. (2012) should be considered to enable the explicit representation and reasoning about contexts.

Property 1: Encapsulation. A typical example is the "context as a box" metaphor. The underlying vision of a context through this metaphor is a “box” whose boundaries are given by a set of contextual attributes.

Property 2: Explicit meta-knowledge. Knowledge about contexts should be explicitly represented. We call this knowledge meta-knowledge.

Property 3: Separation of meta-knowledge and object knowledge. This property implies that one can immediately identify which statements belong to object knowledge and which belong to meta-knowledge.

Property 4: Relations between contexts. The literature on contextual reasoning has dedicated significant attention to possible relations between contexts, including the context coverage relation (Benerecetti et al., 2000) (Benerecetti et al., 2002) (Lenat, 1998), which enables organizing the contexts and the neighboring context (in space) or consecutive contexts (in time).

Property 5: Contextual reasoning. Reasoning should also take into the account the contextual meta-knowledge and the relations between contexts. Hence, the reasoner should be able to access and process the meta-knowledge, and it must further consider the relations between contexts in order to reason with such a constraint.

Property 6: Locality of Knowledge. In each context, we should be able to state axioms with local effect, and by that, we mean that they do not affect other contexts and are not affected by other contexts.

Property 7: Knowledge lifting. Property 6 should not imply opacity. If needed, knowledge from one context should be accessible to another context. This propagation of knowledge between contexts is known as knowledge lifting (Benerecetti et al., 2000).

Property 8: Overlapping and varying domains. Objects can be present in multiple contexts, but that doesn't mean that they should be present in all contexts.

Property 9: Complexity invariance. The complexity of reasoning should not be increased due to the contextual layer.

3.3.1 Techniques to encode context in RDF

The problem with including contextual annotation in RDF is with the RDF predicates. They are binary predicates, and adding contextual knowledge needs an n-ary predicate. The approaches of the state of the art are divided into two main categories:

Extending the data model and/or the semantics of RDF: The triple data structure could be extended by adding a fourth element to each triple, intended to express the context: RDF+ (Dividino et al., 2009), RDF* (Hartig and Thompson, 2014), Singleton Properties (Nguyen et al., 2014), Annotated RDF (Udrea et al., 2010)

Using design patterns: They could be categorized along three axes:

- The contextual index co is attached to the statement $R(a, b)$, and thus $R(a, b)$ holds for co such as RDF reification (Berners-Lee et al., 2001). However, such methods did not support explicit reasoning about time or uncertainty or combinations thereof and were not shown to work in practice.

- The contextual index co is attached to the predicate $R(a, b, co)$. An example of this representation is the situation pattern (Gangemi and Mika, 2003). One advantage is being able to talk about assertions as (reifying) individuals, but the disadvantage is being unable to use them as properties. Another example is the n-ary model (Noy et al., 2006).

- The contextual index co is attached to the object terms $R(a@co, b@co)$, where co is the contextual-slice of the thing named (Welty, 2010) (Welty et al., 2006). The drawback of this method is that it introduces many contextualized individuals that causes object proliferation.

RDF quadruples: adding contexts to triples forming quadruples such as N-Quads [1], Named graphs (Carroll et al., 2005), nanopublications (Groth et al., 2010), and Hoganification (Hogan, 2018).

[1]https://www.w3.org/TR/n-quads/

3.3.2 Conceptual frameworks

In the following, some of the relevant frameworks of contextual knowledge representation and reasoning are presented.

Annotated RDF aRDF (for short) (Udrea et al., 2010) was one of the first mechanisms for integrating RDF and metalevel statements about RDF in a practically usable and theoretically unifying framework. By metalevel statements, the authors refer to the uncertainty, provenance, temporal validity that are usually used to annotate RDF statements after their extraction from textual sources. They assumed the existence of a partially ordered set $(A, \preceq)$, where elements of A are called annotations, and $\preceq$ is a partial ordering on A. They further assumed that A has a unique bottom element. For example, $A_{time} = \mathbf{N}$ could be the set of all non-negative integers (denoting time points) with the usual "less than or equals" ordering on it. Note that A should be a complete lattice [2]. Based on that, they defined an annotated RDF theory (aRDF-theory for short) as a finite set of triples $(r, p : a, v)$, where r is a resource name, p is a property name, a $\in$ A, and v is a value (which could also be a resource name) in dom(p). aRDF provides a semantics for RDF augmented with a partially ordered set. In particular, new partially ordered sets can be plugged into aRDF depending upon the application, and there will be an immediate set of results that apply to that domain. The definition of an aRDF-interpretation is the natural one expected from annotated logic (Kifer and Subrahmanian, 1992). However, two differences are noted here. First, annotated logic assumes that A is a complete lower semilattice while aRDF only requires that it be a partial order.

Joseph and Serafini (2011) introduce a framework for contextual knowledge called **simple reasoning for contextualized RDF knowledge** representation and reasoning based on current RDF(S) standards. In this framework called a contextualized knowledge base (CKB), each statement is qualified with a set of contextual parameters. Statements qualified with the same parameters are grouped as a unique graph that constitutes a context. We call such contextual parameters dimensions. Hence, a context defined over the n dimensions $\{D_i\} 1 \leq i \leq n$, is a triple $\langle C, d(C), Graph(C)\rangle$, where:

1. C is the context identifier,
2. d(C) = $\langle d_1, ..., d_n\rangle$, with each d_i being the context dimensions, and
3. Graph(C) is an RDF graph.

An important component of a CKB is a knowledge base called meta-knowledge where knowledge about dimensions and how their values are related to context identifiers are defined. In addition, they introduce the relation "cover" that provides a structure among

[2]A partially ordered set (X, $\leq$) is a complete lattice iff (i) every subset of X has a unique greatest lower bound and (ii) every directed subset of X has a unique least upper bound. A set Y $\subseteq$ X is directed iff for all y_1, y_2 $\in$ Y, there is an x $\in$ X such that $y_1 \leq$ x and $y_2 \leq$ x

the values of a dimension. For instance, take the location dimension, whose values are geographic locations, e.g., Milan, Florence, Italy, etc. The relation $\prec_{Location}$ represents the coverage between geographical regions, for instance, Milan $\prec_{Location}$ Italy, Florence $\prec_{Location}$ Italy. They also introduce the mechanism of qualification of a symbol w.r.t. a (set of) contextual dimensions. For instance, in a broader context about sport, the qualified symbols $\text{Winner}_{2010,uefa-champions-legue}$ and $\text{Winner}_{2010,french-open-tennis}$ are used to denote Winner in the two different contexts: (2010,uefa-champions-legue) and (2010,french-open-tennis).

Reasoning in CKR is performed at two distinct layers: the meta-level and the object level. Meta-level reasoning is performed in the meta-knowledge, and its main objective is to infer the coverage relation between contexts. Object level reasoning is performed within and across the set of contexts, and its main objective is to infer knowledge within a context and propagate knowledge across contexts connected via coverage relation. They provide a very interesting set of rules including the domain expansion and contraction rules. The latter refers to the set of rules that allow propagating triples from broader contexts into narrower contexts whenever the property of the triple is qualified with dimensions smaller than or equal to those of the narrower context, i.e., to derive $\text{d} : \text{C}_d(a)$ from $\text{e} : \text{C}_d(a)$ given $\text{d} \prec \text{e}$.

The problem in CKR is that contexts are organized hierarchically according to a broader–narrower relation and the knowledge propagation across contexts is limited among hierarchically related contexts. In several applications, this structure is too restrictive, as they require for a more flexible and scalable framework for representing and reasoning about contextual knowledge. Therefore, Bozzato and Serafini (2013a) extend the work by presenting an evolution of the original CKR (based on OWL RL) based on generalizing the contextual structure and the knowledge propagation. This extension is known as **materialization calculus for contexts in the Semantic Web**. In the contextual structure, hierarchical contextual attributes are replaced with a generic graph structure specified by a DL knowledge base [3], containing individual names to denote contexts (i.e., contexts IDs), identifiers for contextual attributes values, and binary relations that allow specifying the relations and contextual attributes for each context. This generalization also enables the introduction of context classes, prototypical contexts that represent homogeneous classes of contexts. The context class definition language is a DL language in which role restriction operations ($\forall R.C$, ...) can only refer to class expressions of the form $\{a\}$, where a is a constant (attribute value). To generalize knowledge propagation, they introduce the "eval" operator, a primitive that allows referring to the extension of a concept (or role) in another set of contexts. For instance, the subsumption $c1 : eval(A, C) \sqsubseteq A$ can be used to state that the extension of concept A in contexts of type C is contained in the extension of A in c1. For example, if John likes all Indian restaurants in Trento, then the extension of the concept GoodRestaurant in the context of John's preferences contains the extension of IndianRestaurant in the context

[3]meta-knowledge is not limited by a fixed structure like in Joseph and Serafini (2011) but can be defined as a full DL knowledge base.

of tourism in Trento. It can be formalized by adding the following axiom to the context of John's preferences: John: $\mathsf{eval}(\mathsf{IndianRestaurant}, \{\mathsf{trento\ tourism}\}) \sqsubseteq \mathsf{GoodRestaurant}$. The *eval* operator can be seen as generalization of the notion of qualified symbols to generic object expressions and context classes. Compared to Klarman and Gutiérrez-Basulto (2011a), which also uses a two-dimensional approach, contextual modal operators introduced based on this logic allow defining references to other contexts as the eval operator in CKR. Apparently, $eval(C, D)$ is equivalent to $\langle D\rangle C$ in Klarman. However, there is no equivalent to the construct $\langle r.D\rangle C$ of Klarman. This seems to limit the expressive power of the language in terms of knowledge lifting. Finally, they proposed a sound and complete materialization calculus for reasoning over the new CKR definition and outlined its implementation based on SPARQL and a concrete syntax in RDF with named graphs.

Finally, to evaluate the CKR framework, Bozzato et al. (2013), in an experimental paper, show the benefits of adopting this framework versus using standard SW primitives. In their paper, they make the difference between "contextual knowledge" and in what sense it is different from annotating knowledge with some metadata (e.g., time and space, provenance, or certainty degree). The difference lies in the fact that a context is a meaningful unit that contains a set of facts, which jointly describe a portion of a structured domain to a certain level of approximation. For their experiment, the case of the FIFA World Cup was used. The comparison considers the three key aspects of engineering and exploiting knowledge: (i) simplicity and expressivity of the (formal) language; (ii) the compactness of the representation; and (iii) the efficiency of reasoning. As for (i), they show that context-based language enables the construction of simpler and more intuitive models, while the RDF/OWL "flat" model presents practical limitations in modeling cross-contextual knowldge. With regard to (ii), they show that the contextualized model is more compact than the OWL based model. Finally for (iii), query answering in the context-based model outperforms in most of the cases performances on the flat model.

Straccia et al. (2010) describe a generic framework for representing and reasoning with annotated Semantic Web data. They present the extension to RDF toward generic annotations. This extension includes two operators: $\oplus$ and $\otimes$.

- They use $\oplus$ to combine information about the same statement. For instance, for a temporal context: from $\gamma : [2000; 2006]$ and $\gamma : [2003; 2008]$, they infer $\gamma : [2000; 2008]$, as [2000,2008] $=$ [2000,2006] $\cup$ [2003,2008]. Here, $\cup$ plays the role of $\oplus$. In the fuzzy context, from $\gamma : 0.7$ and $\gamma : 0.6$, we infer $\gamma : 0.7$, as $0.7 = \max(0.7,0.6)$ (here, max plays the role of $\oplus$).

- They use $\otimes$ to model the "conjunction" of information. For instance, for a temporal context: from (a, subclassOf, b):[2000,2006] and (b, subclassOf, c): [2003,2008], the inference is (a, subclassOf, c): [2003,2006], as [2003,2006] $=$ [2000,2006] $\cap$ [2003,2008]; here, $\cap$ plays the role of $\otimes$. An example from the fuzzy context: from (a, subclassOf,b):

0.7 and (b, subclassOf, c): 0.6, the inference is (a, subclassOf, c): 0.42, as $0.42 = 0.7 \cdot 0.6$ (here, $\cdot$ plays the role of $\otimes$).

A deductive system is then presented. The schema of the rules is the same for any annotation domain (only support for the domain dependent $\oplus$ and $\otimes$ operations has to be provided) and are thus amenable to an easy implementation.

The paper was later extended (Zimmermann et al., 2012b) to present a detailed and systematic approach for combining multiple annotation domains into a new single complex domain. The problem with this approach is that the annotations are dealt independently from each other. This problem is made explicit when one observes the unexpected consequences of the $\oplus$ operator on such a combination:

$$(SkypeCollab, subclassOf, EbayCollab) : \langle [2005, 2009], 1 \rangle$$

$$\langle (SkypeCollab, subclassOf, EbayCollab) : [2009, 2011], 0.3 \rangle$$

Applying the point-wise operation $\oplus$, this leads to the conclusion:

$$(SkypeCollab, subclassOf, EbayCollab) : \langle [2005, 2011], 1 \rangle.$$

This defies the intuition that between 2005 and 2009, Skype collaborators were also Ebay employees (collaborate to degree 1), but from 2009 to 2011, Skype collaborators were Ebay collaborators to the degree 0.3. We argue that the fuzzy value itself has a duration, so that the temporal interval corresponds more to an annotation of a quadruple. Hence, they propose using sets of pairs of primitive annotations, exemplified as follows: $(SkypeCollab; sc; EbayCollab) : \langle [2005, 2009], 1 \rangle \ ; \ \langle [2009, 2011], 0.3 \rangle$.

Finally, they propose a normalization algorithm based on two main operations:(i) saturate: increases the size of a set of pairs of annotations by adding any redundant pairs that "result from the application of $\oplus$ and $\otimes$ to values existing in the initial pairs" and (ii) reduce: takes the output of the saturation step and removes "subsumed" pairs.

This work (Straccia et al., 2010) with its extended version (Zimmermann et al., 2012a) differs from the two-dimensional approaches mentioned previously, in the way they consider a context. Klarman and Serafini regard a context as a meaningful unit in which some statements are true; however, here, it is more about annotating a statement with time, provenance, and confidence. Nevertheless, these two approaches overlap in some cases, e.g., when annotations represent validity contexts such as validity time.

Temporal frameworks In the last part of this section, we show that there are specialized frameworks of contextual representation where the context can be time, for example. Temporal knowledge representation is an old AI topic. Temporal facts are referred to as fluents (Welty et al., 2006). Fluents are instances of relations whose validity is a function of time. There are different approaches to translate this notion of fluents into the Semantic Web world, including the approaches we cited previously to include context in RDF. However, the earliest approach of the W3C has been to favor event entities. For example, the birth of a person can be represented by an event entity such as *birth42*. Entities that participate in this event — person, date, location, etc. -– are then linked by relations to this event entity. The drawback of this approach is that one has to decide a priori which relations are represented as binary relations with standard RDF triples and which relations should be cast into event entities with additional annotations.

Other RDF extensions for temporal knowledge include the work of Gutierrez et al. (2006). They introduced a temporal semantics for RDF, where time is modeled as a label on RDF triples, giving each triple a validity time. The following notation is used:

$$(a, b, c) : [t]$$

The expression $(a, b, c) : [t_1, t_2]$ is a notation for $\{(a, b, c) : [t] \mid t_1 \leq t \leq t_2\}$. They also present the semantics for temporal RDF graphs, a syntax to incorporate temporality into standard RDF graphs. In addition, they present an inference system for temporal RDF graphs, complexity bounds showing that entailment in temporal RDF graphs does not yield extra asymptotic complexity with respect to and sketch a temporal query language for RDF.

The need to represent temporal knowledge within RDF and OWL has also been discussed (Motik, 2010). He presents a general framework for temporal reasoning on RDF with any entailment relation (RDFS-like entailment defined by rules or semantic conditions or OWL-like entailment defined by a model-based over axioms expressed in RDF).

3.4 Toward a benchmark of contextual reasoning

The field of contextual knowledge representation and reasoning on Semantic Web languages and ontologies has been extensively explored by researchers. Several paradigms and languages have been proposed to add context awareness to the Semantic Web. However, today, if you have a project that needs context representation or contextual reasoning, there is no standard to follow nor methodology to apply (we only find a benchmark tentative (Homola et al., 2010)). In addition and more importantly, the term "context" seems vague. It can be used indifferently to refer to many things, thus creating more confusion. Table 3.2 shows that some state-of-the-art works are more adapted to particular contexts and not all, say time and provenance. The discussed works are compared in a state-of-the-art chapter. While doing so, it was noted that there is no benchmark or a point of reference against which previous works can be compared. To have a complete view of the above works, we highlight some criteria that we found relevant in most of the contextual reasoning applications, and then we compare the above works with these criteria. These criteria comprise both theoretical and practical criteria. The theoretical criteria are inspired by Guhas' thesis (Guha, 1991), and the practical ones are inspired by practical problems that we encountered in a digital humanity project (Aljalbout and Falquet, 2017). These criteria can possibly form the basis for a benchmark of contextual knowledge representation tools.

1. **Theoretical or practical work:** Is the proposed approach only a theoretical one, such as a formal language, reasoning algorithm, or calculus, or does it have a concrete implementation, such as a reasoner or a set of rules an interested user can utilize?

2. **Context for what:** Are we adding contexts to ontologies, to a language such as RDFS or a less formal structure like a property graph?

3. **Context type:** Most people use the term "context" indifferently to define annotations, ontologies, validity contexts, etc. With the "context type" criteria, we would like to refine the type of context used in each work. In other words: what do the authors refer to as a context?

4. **Context expressivity and structure:** What is unique about this contextual approach? What is the structure used for representing context? Do they use a particular language for contexts (RDFS, DL, etc.)? Or maybe they are based on McCarthy's three postulates? Is there any mathematical structure applied to contexts, such as lattices?

5. **Context lifting:** As mentioned previously in the state of the art, Guha discussed the concept of context lifting, which involves axioms relating the truth of formulas in different contexts.

6. **Complexity:** We want to compare the complexity of each of these works for the reasoning or semantics.

7. **Usability:** We want to study the usability of this work. One problem encountered in contextual knowledge representation and reasoning is the lack of guidance and references for possible implementations or applications. We want to check if there is a straightforward methodology to help users utilize the framework.

Research title	**Theoretical (T) or Practical (P)**	**Context for what**	**Context type**	**Context structure**	**Lifting**
C-OWL	T	ontologies	each ontology is a context	defined by the bridge rules	context mapping
Klarman	T	DL	objects (McCarthy)	DL ontology	the constructors [] ($\langle\rangle$ D)
Annotated RDF	T	RDF	time, space etc.	lattices on contexts	–
Simple reasoning for contextualized KG	P	RDF(S)	contexts as a box	lattices (borader/ narrower)	qualifier symbols
Materialization calculus	P	RDF(S)	context as a box	simple DL: cxt classes, subclass rel	eval operator + $\sqsubseteq$
Straccia et al.	P	RDFS	time	lattice $\oplus$ $\otimes$	no

Table 3.2 Comparison of context semantics between different works.

3.5 Observations and conclusion

The utility of context-dependent knowledge has been highlighted through various published works before the rise of the Semantic Web and until today. Nevertheless, the following are noted: of the different research directions adopted in DL, most are theoretical (i.e., not implemented or tested on actual use cases). In addition, the semantics and granularity of contexts differ from one work to another. From a semantic web perspective, contextual reasoning with OWL axioms has been less explored. Also, a recognized practical framework and guidelines for contextual reasoning are yet to be developed. Furthermore, research on modern graphs remains premature. However, many aspects can be researched, notably the interpretations of contexts and their impact on reasoning. In this thesis, we started by proposing a contextual OWL language intended for practical use. While applying it to Wikidata, the incompatibility of DL-based languages with modern knowledge graphs was identified. Thus, a model that formalizes contextual knowledge and contexts within a many-sorted logic was designed.

Chapter 4

OWL^C: an OWL extension for reasoning in validity contexts

Chapter Contents

4.1 Motivation

Many research papers have been written on adding context awareness to semantic web applications. However, to date, there has not been a clear standard to follow for contextual reasoning on ontologies.

This part of the thesis has been prompted by work on a digital humanities project. The linguists wanted to construct a semantic knowledge base with time as a context. For a new context practitioner, the state of the art of contextual knowledge representation and reasoning seems vague and unclear. Of late, the focus is on providing techniques to encode contexts in RDF (Giménez-García et al. (2017), Nguyen et al. (2014), Hartig and Thompson (2014), etc.). However, the confusion becomes clearer when moving to contextual reasoning. Different research proposals deal with this topic, moving from RDFS to DL. Each work discusses a different aspect of reasoning, and none provide a straightforward guideline for practical reasoning.

In addition, the term "context" is used indifferently to refer to many things of different granularities, leading to more confusion. Therefore, we developed OWL^C – a two-dimensional contextual web ontology language. It is based on the semantics of the two-dimensional DLs of Klarman and Gutiérrez-Basulto (2011a). We see OWL^C as an OWL extension for reasoning with validity contexts. The target is to have scalable reasoning similar to OWL RL. Therefore, the work should not be seen as theoretical work but as a guideline for the practical implementation of contexts. The contributions of this chapter are as follows:

- The meaning of the term "context" is refined by introducing two fundamental concepts: validity contexts and additional contexts (non-validity contexts).
- The OWL RL rules are redefined for reasoning with context instances and classes.
- Optimization techniques on contextual rules are proposed using a contextual construct and set of contexts.

At the end of this chapter, the compatibility of this profile with modern knowledge graphs is discussed, and it is further explored in the upcoming chapter.

Due to the resemblance in their names, OWL^C could be compared directly to C-OWL, but they are very different. In C-OWL, the idea is to establish context mappings between two elements (concepts, roles, individuals) of two ontologies. Although (extensionally) different, the two elements could be contextually related. For instance, they both refer to the same object in the world. The work is also different from that of Zimmermann et al. (2012a), where the authors focus on annotating context with time, provenance, and fuzziness. Treating all of these annotations the same way, they don't distinguish between contexts that specify the validity domain of a triple and those that provide additional information about a triple. This distinction is clarified in the next section.

4.2 Clarifying the term *context*

OWL^C contains contextual rules that only apply to validity contexts. Therefore, the notion of validity context is clarified at the beginning of this chapter by differentiating it from additional contexts as inspired by Patel-Schneider (2018).

- Validity contexts: They can affect the fact by enhancing its meaning or limiting its purpose to a given context. Fluents (Welty et al., 2006) illustrates validity contexts. A fluent property is a temporal property whose object is subject to change over time. The notion of validity in a context implies that a statement is not valid everywhere, so there is an abstract space where we can evaluate the statement at each "point." A context of validity is a region of this space. Temporal validity is generally a temporal region of one or more time intervals.

- Additional contexts: They supplement a fact with additional elements that do not modify its meaning. As a result, the fact is more precisely described with the additional context but is sufficiently clear without it. A typical example is the publication context that provides information about the provenance of the triple as a reference to support the claim.

The following are the examples of the two contexts:

(Barack Obama, president of, United States)
[time : *[2009-2017]*, provenance: : *Wikipedia*]

The following proposition taken from Wikipedia states that Barack Obama was the president of the United States from 2009 to 2017. (Barack Obama, presidentOf, United States) is the proposition. *(time: [2009-2017])* and *(provenance: Wikipedia)* is the context. Time is the validity context, without which the truth of Barack Obama's presidency is incomplete. The presence or absence of the validity time interferes with the semantics of the proposition. If the time is not indicated, we have the impression that he is THE president; however, if indicated, the semantics is restricted to the indicated validity time. Provenance is an additional context (non-validity context), and adding it to the original proposition does not interfere with the semantics. Rather, it only adds to it.

In the rest of the chapter, we work only on the validity context.

4.3 Background: a two-dimensional logics for contexts

This section presents the two-dimensional description logic of Klarman and Gutiérrez-Basulto (2011b). The framework is derived from two foundations as mentioned by Klarman (2013) – conceptually, from McCarthy's theory of formalizing contexts (McCarthy, 1987) and formally, from two-dimensional DLs (Wolter and Zakharyaschev, 1999), (Baader and Laux, 1994). This section will serve as the basis for the OWL^C profile that will be built later. In (Klarman and Gutiérrez-Basulto, 2011a), the authors offer different variants of the two-dimensional approach. We use the expressive meta-language that employs a natural DL for the contextual language.

A two-dimensional DL for context consists of two dimensions.

- The core dimension used to represent contextual object knowledge such as contextual classes, contextual properties and contextual axioms
- The context dimension used to represent contexts that are considered first-class citizens.

Formally speaking, the signature (or vocabulary) is a pair of DL signatures $(\langle N_C, N_R, N_I\rangle, \langle N_{KC}, N_{KR}, N_{KI}\rangle)$ where:

- N_C (resp. N_{KC}) is a set of domain (resp. context) concept names,
- N_R (N_{KR}) is a set of domain (context) role names, and
- N_I (N_{KI}) is a set of domain (context) individuals' names.

4.3.1 The context language

The context language is a description logic from the $\mathcal{ALC}$ family. Let us take the example of validity context that is composed of many dimensions, such as temporal validity, spatial validity, etc. We will use the term "characteristics" instead of "dimensions" to avoid any confusion with the two-dimensions of the language. Using the context language, the context is represented by an object (we use k usually) in the contextual formula of the core language. The characteristics of this context are explicitly illustrated using properties within the context language. For example:

$$\begin{aligned}
&k : convictedOf(John, Murder)\\
&\wedge \quad time(k, kt)\\
&\wedge \quad startYear(kt, 2009)\\
&\wedge \quad endYear(kt, 2017)\\
&\wedge \quad location(k, Geneva)
\end{aligned}$$

states that John was convicted of a murder within a validity context $k \in N_{KI}$. This validity context is composed of a validity time ([2009–2017]) and validity space (Geneva) that are represented using the properties $startYear$, $endYear$, and $location$ respectively.

4.3.2 The core language

The core language is a description logic from the $\mathcal{ALC}$ family. An axiom expression of the core language is either an expression of the form $K : \phi$, where K is either an individual context name (in N_{KI}) or a concept expression over the context signature $\langle N_{KC}, N_{KR}, N_{KI}\rangle$. Such an expression states that the axiom ϕ holds in the specified context or in all contexts of the specified context concept. ϕ can be:

1. a concept axiom ($C \sqsubseteq D$, $C \equiv D$, $C\ disjoint\ D$)

   $$\mathsf{1969 : CanVote \sqsubseteq Aged21orMore}$$

   states that the axiom $CanVote \sqsubseteq Aged21orMore$ holds in the temporal context *1969*.

2. a class or role assertion ($C(a)$, $R(a, b)$) defined on the core signature with contextual concept and role expressions

   $$\mathsf{1857 : Professor(Saussure)}$$

   states that Saussure was a professor during 1857.

A contextual interpretation is a pair of interpretations $\mathcal{M} = (\mathcal{I}, \mathcal{J})$, where $\mathcal{I} = (\Delta, \cdot^{\mathcal{I}[\cdot]})$ is the core interpretation, $\mathcal{J} = (\Omega, \cdot^{\mathcal{J}})$ is the context interpretation, and $\Delta \cap \Omega = \emptyset$. $\cdot^{\mathcal{I}[\cdot]}$ is a family of interpretation functions, one for each context $k \in \Omega$. $.^{\mathcal{J}}$ is the (non-contextual) interpretation function of every context in the context language. The interpretation of the class constructors of the core language is straightforward. Table 4.1 contains the semantics of core language basic class constructors.

CDL syntax	Semantics (Interpretation in context k)
$C_1 \sqcap ... \sqcap C_n$	$C_1^{\mathcal{I}[k]} \cap ... \cap C_n^{\mathcal{I}[k]}$
$C_1 \sqcup ... \sqcup C_n$	$C_1^{\mathcal{I}[k]} \cup ... \cup C_n^{\mathcal{I}[k]}$
$\neg C$	$(\neg C)^{\mathcal{I}[k]} = \Delta^{\mathcal{I}[k]} \setminus C^{I[k]}$
$\exists(R.C)$	$\{x \vert \exists y : (x, y) \in (R)^{\mathcal{I}[k]} and \quad y \in (C)^{\mathcal{I}[k]}\}$
$\forall(R.C)$	$\{x \vert \forall y : (x, y) \in (R)^{\mathcal{I}[k]} \to y \in (C)^{\mathcal{I}[k]}\}$

Table 4.1 Core language direct model-theoretic semantics

A contextual axiom $K : \phi$ is satisfied by an interpretation $\mathcal{M}$ if in every context k that belongs to the interpretation of K, the interpretation in k of the concepts, roles and individuals that appear in ϕ satisfy the axiom condition

- $\mathcal{M} \models K : C \sqsubseteq D$ iff $\forall k \in K^{\mathcal{J}} : C^{\mathcal{I}[k]} \subseteq D^{\mathcal{I}[k]}$, where $C \in N_C$ and $D \in N_C$
- $M \models K : C(a)$ iff $\forall k \in K^{\mathcal{J}} : C(a)^{\mathcal{I}(k)}$, where $C \in N_C$ and $a \in N_I$
- $M \models K : R(a, b)$ iff $\forall k \in K^{\mathcal{J}} : R(a, b)^{\mathcal{I}(k)}$, where $R \in N_R$, $a \in N_I$ and $b \in N_I$

(if K is not a concept expression but a context individual name k, $K^{\mathcal{J}}$ designates the singleton $\{k^{\mathcal{J}}\}$ in the above expressions).

4.3.3 The core-context interaction language

The interaction between the two languages is carried out using special operators. Table 4.2 provides the semantics of the context-based concept-forming operators. Examples:

CDL	Semantics
$\langle K \rangle\, C$	$\{x \in \Delta \mid \exists y \in K^{\mathcal{J}} : x \in C^{\mathcal{I}[y]}\}$
$[K]C$	$\{x \in \Delta \mid \forall y \in K^{\mathcal{J}} \rightarrow x \in C^{\mathcal{I}[y]}\}$

Table 4.2 Semantics of the contexts-based concept forming operators.

- $\langle AsianCountry \rangle\, Professor$: the individuals that belong to the class $Professor$ in some context of type $AsianCountry$
- $[EuropeanCountry]Professor$: the individuals that belong to the class $Professor$ in all contexts of type $EuropeanCountry$

4.3.4 Context lifting

Context lifting is the process of defining the data transfer rules across contexts to allow knowledge propagation. Klarman does not talk about lifting explicitly, but he does the following: in Klarman and Gutiérrez-Basulto (2011a), the authors define

$$\langle r.C \rangle\, D \quad [r.C]D$$

where r is a relation, C is a concept from the context language, and D is a concept from the object language. The first formula denotes all objects that are D in some contexts of type C accessible from the current one through r. The second formula denotes all objects that are D in all contexts of type C accessible from the current one through r. The semantics are denoted in table 4.3

CDL	Semantics: interpretation in k
$\langle r.K \rangle\, C$	$\{x \in \Delta \mid \exists j \in K^{\mathcal{J}} : (k, j) \in r^{\mathcal{J}} \wedge x \in C^{\mathcal{I}[j]}\}$
$[r.K]C$	$\{x \in \Delta \mid \forall j \in K^{\mathcal{J}} : (k, j) \in r^{\mathcal{J}} \rightarrow x \in C^{\mathcal{I}[j]}\}$

Table 4.3 Semantics of the context-based concept-forming operators.

4.4 OWL^C: an extension for contextual reasoning in validity contexts

Anyone familiar with OWL would probably be familiar with the three profiles of OWL2. They are subsets of OWL DL and are highly attractive for many applications. Reasoning algorithms for the OWL profiles, compared with OWL DL, show better performance and are usually easier to implement. Among those profiles, we consider OWL RL [1] as a profile aimed at applications that require scalable reasoning without sacrificing too much expressive power. OWL^C (Aljalbout et al., 2019b) (Aljalbout et al., 2019a) is an extension of OWL RL for contextual reasoning. It is based on the semantics of Klarman and Gutiérrez-Basulto (2011b). OWL^C is regarded as an extension aimed at applications that require scalable reasoning without sacrificing too much expressive power. This is achieved by restricting the use of constructs to a certain syntactic position, similar to OWL 2 RL.

4.4.1 Mapping OWL^C to RDF and to ternary/quaternary predicates

The inference rules of OWL2 RL are defined on the representation of OWL2 in RDF, as defined in OWL2 Mapping to RDF graphs. We do the same for OWL^C, but to do that, we extend the mapping. In OWL2 Mapping to RDF graphs, the mapping is represented using a T, but to avoid confusing it with the T we use later for the rules, we use τ. τ represents the mapping from an OWL^C ontology O to an RDF graph $\tau(O)$. The TANN operator should be adapted to the representation of OWL^C axioms, but it is not presented since annotations play no role in reasoning.

We extended OWL RL mapping to represent the contextual constructs of OWL^C, as described in Table 4.4. To keep this representation compact, we utilized quadruples (that can then be represented using patterns for n-ary relations (Noy et al., 2006) or directly in RDF-star (Hartig and Thompson, 2014) or another technique from the ones discussed in chapter 2).

Element E of the language	**Triples and quadruples in $\tau(E)$**	**Main node of $\tau(E)$**
$\langle K\rangle D$	`_:x owlc:onClass` $\tau(D)$ `_:x owlc:inSomeContextOf` $\tau(K)$	`_:x`
$[K]D$	`_:x owlc:onClass` $\tau(D)$ `_:x owlc:inAllContextOf` $\tau(K)$	`_:x`
$K : C(a)$	$\tau(a)$ `rdf:type` $\tau(C)$ $\tau(K)$	
$K : R(a,b)$	$\tau(a)$ $\tau(R)$ $\tau(b)$ $\tau(K)$	
$K : C \sqsubseteq D$	$\tau(C)$ `rdfs:subClassOf` $\tau(D)$ $\tau(K)$	

[1] https://www.w3.org/TR/owl2-profiles/#Feature_Overview_3

$K : R \sqsubseteq S$	$\tau(R)$ `rdfs:subPropertyOf` $\tau(S)$ $\tau(K)$	
$\langle r.K \rangle D$	`_:x owlc:onClass` $\tau(D)$ `_:x owlc:inSomeContextOf` $\tau(K)$ `_:x owlc:linkedThrough` $\tau(r)$	`_:x`
$[r.K]D$	`_:x owlc:onClass` $\tau(D)$ `_:x owlc:inAllContextOf` $\tau(K)$ `_:x owlc:linkedThrough` $\tau(r)$	`_:x`

Table 4.4 Mapping OWL^C contextual constructs and axioms to triples and quadruples

For example, the OWL^C axiom

$$K : \exists R.C \sqsubseteq D$$

will be represented by a quadruple and two triples:

C `rdfs:subClassOf _:x` K .
`_:x owl:someValuesFrom` D .
`_:x owl:onProperty` R .

In the following, the contextualized axioms are presented. In the original version of OWL2 RL, the rules are given as universally quantified first-order implications over a ternary predicate T. This predicate generalizes an RDF triple thus, *T(s,p,o)* represents a generalized RDF triple in which bnodes and literals are allowed in all positions[2]; with the subject s, predicate p, and the object o. Variables in the implications are preceded with a question mark. The rules are based on the triples of the standard RDF representation (mapping) of an OWL ontology [3].

In OWL^C, the $T(?s, ?p, ?o)$ predicate is used to represent the context language statements since they are not contextualized. For the core language, we use a quaternary predicate $Q(s, p, o, k)$, where s is the subject, p is the predicate, o is the object, and k is the context or the context class for which the predicate holds. To enable representing non-contextual statements in the core language using the Q() predicate, we introduce a class `owlc:Context` that contains all contexts. Hence, a statement that is valid in every context, (i.e., non-contextual) can be represented as $Q(?x, ?p, ?y,$ `owlc:Context`$)$.

4.4.2 Semantics of context rules

The context rule that will be used for the hierarchical reasoning on contexts in the core rules is the following:

[2]https://www.w3.org/TR/owl2-profiles/#OWL_2_RL
[3]https://www.w3.org/TR/owl2-mapping-to-rdf/

$$\mathsf{T}(?k, \texttt{rdf:type}; ?kc) \wedge \quad \mathsf{Q}(?x, ?p, ?y, ?kc) \\ \longrightarrow \mathsf{Q}(?x, ?p, ?y, ?k)$$

This rule states that if a statement is true in a context class kc, then it is true in every context k of this class.

$$\mathsf{T}(?ks, \texttt{rdfs:subClassOf}; ?kc) \wedge \mathsf{Q}(?x, ?p, ?y, ?kc) \\ \longrightarrow \mathsf{Q}(?x, ?p, ?y, ?ks)$$

This rule declares that if a statement is true in a context class kc, then it is true in every sub(context)class ks of kc.

To ensure that all the contexts are members of class `owlc:Context`, the following two rules are added:

$$\mathsf{T}(k, \texttt{rdf:type}, \texttt{owl:Individual}) \wedge \quad \mathsf{Q}(s, p, o, k) \\ \longrightarrow \mathsf{T}(k, \texttt{rdf:type}, \texttt{owlc:Context})$$

$$\mathsf{T}(k, \texttt{rdf:type}, \texttt{owl:Class}) \wedge \quad \mathsf{Q}(s, p, o, k) \\ \longrightarrow \mathsf{T}(k, \texttt{rdfs:subClassOf}, \texttt{owlc:Context})$$

4.4.3 Semantics of the classes

In this subsection, the inference rules of the class constructors are presented. k is used to refer to a context instance (T(?k, `rdf:type`, `owl:Individual`)) or a context class (T(?k, `rdf:type`, `owl:Class`)). We omit these triples in the table.

	IF	**THEN**

cls-com	Q($?c_1$, `owl:complementOf`, $?c_2$, `owlc:Context`)
	Q($?x$, `rdf:type`, $?c_1$, $?k$)
	Q($?x$, `rdf:type`, $?c_2$, $?k$)
		false

cls-int1	Q($?c$, `owl:intersectionOf`, $?x$, `owlc:Context`)
	LIST[?x, $?c_1$, ..., $?c_n$]
	Q($?y$, `rdf:type`, $?c_1$, $?k$)
	Q($?y$, `rdf:type`, $?c_2$, $?k$)
	...
	Q($?y$, `rdf:type`, $?c_n$, $?k$)
		Q($?y$, `rdf:type`, $?c$, $?k$)

cls-int2	Q(?c, `owl:intersectionOf`, ?x, `owlc:Context`) LIST[$?x$, $?c_1$, ..., $?c_n$] Q($?y$, `rdf:type`, $?c$, $?k$)	Q($?y$, `rdf:type`, $?c_1$, $?k$) Q($?y$, `rdf:type`, $?c_2$, $?k$) ... Q($?y$, `rdf:type`, $?c_n$, $?k$)
cls-uni	Q($?c$, `owl:unionOf`, $?x$, `owlc:Context`) LIST[$?x$, $?c_1$, ..., $?c_n$] Q($?y$, `rdf:type`, $?c_i$, $?k$) (one rule for each c_i)	Q($?y$, `rdf:type`, $?c$, $?k$)
cls-svf1-1	Q($?x$, `owl:someValuesFrom`, $?y$, `owlc:Context`) Q($?x$, `owl:onProperty`, $?p$, `owlc:Context`) Q($?u$, $?p$, $?v$, $?k$) Q($?v$, `rdf:type`, $?y$, $?k$)	Q($?u$, `rdf:type`, $?x$, $?k$)
cls-avf-1	Q($?x$, `owl:allValuesFrom`, $?y$, `owlc:Context`) Q($?x$, `owl:onProperty`, $?p$, `owlc:Context`) Q($?u$, `rdf:type`, $?x$, $?k$) Q($?u$, $?p$, $?v$, $?k$)	Q($?v$, `rdf:type`, $?y$, $?k$)

Table 4.5 OWL^C: Entailment rules for the class semantics

We proceed in a similar way with rules containing cardinalities.

4.4.4 Semantics of class axioms

The semantics of class axioms are presented in Table 4.6. All axioms of OWL RL are contextualized straightforwardly.

	IF	**THEN**
cax-sco	Q($?c$, `rdfs:subClassOf`, $?d$, $?k$) Q($?x$, `rdf:type`, $?c$, $?k$)	Q($?x$, `rdf:type`, $?d$, $?k$)
cax-eqc1	Q($?c_1$, `owl:equivalentClass`, $?c_2$, ?k) Q($?x$, `rdf:type`, $?c_1$, $?k$)	Q($?x$, `rdf:type`, $?c_2$, $?k$)

cax-eqc2	Q($?c_1$, `owl:equivalentClass`, $?c_2$, $?k$)
	Q($?x$, `rdf:type`, $?c_2$, $?k$)
		Q($?x$, `rdf:type`, $?c_1$, $?k$)

cax-dw	Q($?c_1$, `owl:disjointWith`, $?c_2$, $?k$)
	Q($?x$, `rdf:type`, $?c_1$, $?k$)
	Q($?x$, `rdf:type`, $?c_2$, $?k$)
		false

cax-adc	$?x$, `rdf:type`, `owl:AllDisjointClasses`, $?k$)
	Q($?x$, `owl:members`, $?y$, `owlc:Context`)
	LIST[$?y$, $?c_1$, ..., $?c_n$]
	Q($?z$, `rdf:type`, $?c_i$, $?k$)
	Q($?z$, `rdf:type`, $?c_j$, $?k$)
		false

Table 4.6 OWL^C: Entailment rules for class axioms

4.4.5 The semantics of equality

There have been some works in the "sameAs" state of the art that treats the contextuality of owl:sameAs, including the following works Beek et al. (2016) and Batchelor et al. (2014). The authors use the term "context" to refer to the set of properties that have the same values for the two linked entities. Based on this definition, weaker types of identities can be represented by using subsets of properties with respect to which two resources can be considered the same. The definition of context is not the same one used in OWL^C, where contexts are used to represent only validity contexts.

	IF	**THEN**
eq-ref	Q($?s$, $?p$, $?o$, $?k$)	Q($?s$, `owl:sameAs`, $?s$, $?k$) Q(?p, `owl:sameAs`, ?p, ?k) Q(?o, `owl:sameAs`, ?o, ?k)
eq-sym	Q($?x$, `owl:sameAs`, $?y$, $?k$)	Q($?y$, `owl:sameAs`, $?x$, $?k$)

eq-trans	Q($?x$, `owl:sameAs`, $?y$, $?k$)
	Q($?y$, `owl:sameAs`, $?z$, $?k$)
		Q($?x$, `owl:sameAs`, $?z$, $?k$)

eq-rep-s	Q($?s$, `owl:sameAs`, $?s'$, $?k$)
	Q($?s$, $?p$, $?o$, $?k$)
		Q($?s'$, $?p$, $?o$, $?k$)

eq-rep-p	Q(?p, `owl:sameAs`, ?p', ?k) Q(?s, ?p, ?o, ?k)	Q($?s$, $?p'$, $?o$, $?k$)

eq-rep-o	Q($?o$, `owl:sameAs`, $?o'$, $?k$)
	Q($?s$, $?p$, $?o$, $?k$)
		Q($?s$, $?p$, $?o'$, $?k$)

eq-diff1	Q(?x, `owl:sameAs`, ?y, ?k)
	Q(?x, `owl:differentFrom`, ?y, ?k)
		false

eq-diff2	Q(?x, `rdf:type`, `owl:AllDifferent`, ?k)
	Q(?x, `owl:members`, ?y, `owlc:Context`)
	LIST[?y, ?z_1, ..., ?z_n]
	Q(?z_i, `owl:sameAs`, ?z_j, ?k)
		false
		
		one rule for each $1 \leq i < j \leq n$

eq-diff3	Q(?x, `rdf:type`, `owl:AllDifferent`, ?k)
	Q(?x, `owl:distinctMembers`, ?y, `owlc:Context`)
	LIST[?y, ?z_1, ..., ?z_n]
	Q(?z_i, `owl:sameAs`, ?z_j, ?k)
		false
		
		one rule for each $1 \leq i < j \leq n$

Table 4.7 OWL^C: Entailment rules for equality

Forcing the rigidity of the sameAs property Although the semantics of OWL^C makes no hypothesis about the interpretation of individuals, some users might want to restrict rigid interpretations in which an individual has the same interpretation in every context, i.e., $a^{\mathcal{I}(i)} = a^{\mathcal{I}(j)}$ for any individual a and any pair of contexts i, j. In this situation, starting with OWL2 semantics that defines $(a^I, b^I) \in sameAs^I \Leftrightarrow a^I = b^I$, it is easy to see that sameAs is also rigid. Therefore, if some OWL^C user considers the interpretation of individuals rigid, they must force the rigidity of sameAs by adding the following rule:

$$\mathrm{Q}(x,\ \texttt{sameAs},\ y,\ k)$$
$$\longrightarrow \mathrm{Q}(x,\ \texttt{sameAs},\ y,\ \texttt{owlc:Context})$$

This means that if a sameAs is true in one context, then it is true in all contexts.

4.4.6 Semantics of properties axioms

The semantics of property axioms is presented in table 4.8. All the axioms are contextualized except the prp-ap rule for annotations.

	IF	**THEN**
prp-ap		Q(ap, `rdf:type`, `owl:AnnotationProperty`, `owlc:context`)

prp-dom	Q(?p, `rdfs:domain`, ?c, ?k)
	Q(?x, ?p, ?y, ?k)
		Q(?x, `rdf:type`, ?c, ?k)

prp-rng	Q(?p, `rdfs:range`, ?c, ?k)
	Q(?x, ?p, ?y, ?k)
		Q(?y, `rdf:type`, ?c, ?k)

prp-fp	Q(?p, `rdf:type`, `owl:FunctionalProperty`, ?k)
	Q(?x, ?p, ?y_1, ?k)
	Q(?x, ?p, ?y_2, ?k)
		Q(?y_1, `owl:sameAs`, ?y_2, ?k)

prp-ifp	Q(?p, `rdf:type`, `owl:InverseFunctionalProperty`, ?k)
	Q(?x_1, ?p, ?y, ?k)
	Q(?x_2, ?p, ?y, ?k)
		Q(?x_1, `owl:sameAs`, ?x_2, ?k)

prp-irp	Q(?p, `rdf:type`, `owl:IrreflexiveProperty`, ?k)
	Q(?x, ?p, ?x, ?k)
		false

prp-symp	Q(?p, `rdf:type`, `owl:SymmetricProperty`, ?k)
	Q(?x, ?p, ?y, ?k)
		Q(?y, ?p, ?x, ?k)

prp-asyp	Q(?p, `rdf:type`, `owl:AsymmetricProperty`, ?k)
	Q(?x, ?p, ?y, ?k)
	Q(?y, ?p, ?x, ?k)
		false

prp-trp	Q(?p, `rdf:type`, `owl:TransitiveProperty`, ?k)
	Q(?x, ?p, ?y, ?k)
	Q(?y, ?p, ?z, ?k)
		Q(?x, ?p, ?z, ?k)

prp-spo1	Q(?p_1, `rdfs:subPropertyOf`, ?p_2, ?k)
	Q(?x, ?p_1, ?y, ?k)
		Q(?x, ?p_2, ?y, ?k)

prp-spo2	Q(?p, `owl:propertyChainAxiom`, ?x, `owlc:Context`)
	LIST[?x, ?p_1, ..., ?p_n]
	Q(?u_1, ?p_1, ?u_2, ?k)
	Q(?u_2, ?p_2, ?u_3, ?k)
	...
	Q(?u_n, ?p_n, ?u_{n+1}, ?k)
		Q(?u_1, ?p, ?u_{n+1}, ?k)

prp-eqp1	Q(?p_1, `owl:equivalentProperty`, ?p_2, ?k)
	Q(?x, ?p_1, ?y, ?k)
		Q(?x, ?p_2, ?y, ?k)

prp-eqp2	Q($?p_1$, `owl:equivalentProperty`, $?p_2$, $?k$)
	Q($?x$, $?p_2$, $?y$, $?k$)
		Q($?x$, $?p_1$, $?y$, $?k$)

prp-pdw	Q($?p_1$, `owl:propertyDisjointWith`, $?p_2$, $?k$)
	Q($?x$, $?p_1$, $?y$, $?k$)
	Q($?x$, $?p_2$, $?y$, $?k$)
		false

prp-adp	Q($?x$, `rdf:type`, `owl:AllDisjointProperties`, $?k$)
	Q($?x$, `owl:members`, $?y$, `owlc:Context`)
	LIST[$?y$, $?p_1$, ..., $?p_n$]
	Q($?u$, $?p_i$, $?v$, $?k$)
	Q($?u$, $?p_j$, $?v$, $?k$)
		false

prp-inv1	Q($?p_1$, `owl:inverseOf`, $?p_2$, $?k$)
	Q($?x$, $?p_1$, $?y$, $?k$)
		Q($?y$, $?p_2$, $?x$, $?k$)

prp-inv2	Q($?p_1$, `owl:inverseOf`, $?p_2$, $?k$)
	Q($?x$, $?p_2$, $?y$, $?k$)
		Q($?y$, $?p_1$, $?x$, $?k$)

prp-key	Q($?c$, `owl:hasKey`, $?u$, $?k$)
	LIST[$?u$, $?p_1$, ..., $?p_n$]
	Q($?x$, `rdf:type`, $?c$, $?k$)
	Q($?x$, $?p_1$, $?z_1$, $?k$)
	...
	Q($?x$, $?p_n$, $?z_n$, $?k$)
	Q($?y$, `rdf:type`, $?c$, $?k$)
	Q($?y$, $?p_1$, $?z_1$,$?k$)
	...
	Q($?y$, $?p_n$, $?z_n$, $?k$)
		Q($?x$, `owl:sameAs`, $?y$, $?k$)

prp-npa1	Q($?x$, `owl:sourceIndividual`, $?i_1$, $?k$)
	Q($?x$, `owl:assertionProperty`, $?p$, $?k$)
	Q($?x$, `owl:targetIndividual`, $?i_2$, $?k$)
	Q($?i_1$, $?p$, $?i_2$, $?k$)
		false

prp-npa2	Q($?x$, `owl:sourceIndividual`, $?i$, $?k$)
	Q($?x$, `owl:assertionProperty`, $?p$, $?k$)
	Q($?x$, `owl:targetValue`, $?lt$, $?k$)
	Q($?i$, $?p$, $?lt$, $?k$)
		false

Table 4.8 OWL^C: Entailment rules for property axioms

4.4.7 Semantics of datatypes

The semantics of the datatype properties are the same as that in OWL RL.

4.4.8 Semantics of schema vocabulary

Table 4.9 provides the semantics of the schema vocabulary. We don't contextualize the scm-cls rule.

	IF	**THEN**

scm-cls	Q($?c$, `rdf:type`, `owl:Class`, `owlc:Context`)	Q($?c$, `rdfs:subClassOf`, $?c$, `owlc:Context`)
		Q($?c$, `owl:equivalentClass`, $?c$, `owlc:Context`)
		Q($?c$, `rdfs:subClassOf`, `owl:Thing`, `owlc:Context`)
		Q(`owl:Nothing`, `rdfs:subClassOf`, $?c$, `owlc:Context`)

scm-sco	Q($?c_1$, `rdfs:subClassOf`, $?c_2$, $?k$)
	Q($?c_2$, `rdfs:subClassOf`, $?c_3$, $?k$)
		Q($?c_1$, `rdfs:subClassOf`, $?c_3$, $?k$)

scm-eqc1	Q($?c_1$, `owl:equivalentClass`, $?c_2$, $?k$)	Q($?c_1$, `rdfs:subClassOf`, $?c_2$, $?k$)
		Q($?c_2$, `rdfs:subClassOf`, $?c_1$, $?k$)

scm-eqc2	Q($?c_1$, `rdfs:subClassOf`, $?c_2$, $?k$)
	Q($?c_2$, `rdfs:subClassOf`, $?c_1$, $?k$)
		Q($?c_1$, `owl:equivalentClass`, $?c_2$, $?k$)

scm-op	Q($?p$, `rdf:type`, `owl:ObjectProperty`, $?k$)	Q($?p$, `rdfs:subPropertyOf`, $?p$, $?k$)
		Q($?p$, `owl:equivalentProperty`, $?p$, $?k$)

scm-dp	Q($?p$, `rdf:type`, `owl:DatatypeProperty`, $?k$)	Q($?p$, `rdfs:subPropertyOf`, $?p$, $?k$)
		Q($?p$, `owl:equivalentProperty`, $?p$, $?k$)

scm-spo	Q($?p_1$, `rdfs:subPropertyOf`, $?p_2$, $?k$)
	Q($?p_2$, `rdfs:subPropertyOf`, $?p_3$, $?k$)
		Q($?p_1$, `rdfs:subPropertyOf`, $?p_3$, $?k$)

scm-eqp1	Q($?p_1$, `owl:equivalentProperty`, $?p_2$, $?k$)	Q($?p_1$, `rdfs:subPropertyOf`, $?p_2$, $?k$)
		Q($?p_2$, `rdfs:subPropertyOf`, $?p_1$, $?k$)

scm-eqp2	Q($?p_1$, `rdfs:subPropertyOf`, $?p_2$, $?k$)
	Q($?p_2$, `rdfs:subPropertyOf`, $?p_1$, $?k$)
		Q($?p_1$, `owl:equivalentProperty`, $?p_2$, $?k$)

scm-dom1	Q($?p$, `rdfs:domain`, $?c_1$, $?k$)
	Q($?c_1$, `rdfs:subClassOf`, $?c_2$, $?k$)
		Q($?p$, `rdfs:domain`, $?c_2$, $?k$)

scm-dom2	Q($?p_2$, `rdfs:domain`, $?c$, $?k$)
	Q($?p_1$, `rdfs:subPropertyOf`, $?p_2$, $?k$)
		Q($?p_1$, `rdfs:domain`, $?c$, $?k$)

scm-rng1	Q($?p$, `rdfs:range`, $?c_1$, $?k$)
	Q($?c_1$, `rdfs:subClassOf`, $?c_2$, $?k$)
		Q($?p$, `rdfs:range`, $?c_2$), $?k$)

scm-rng2	Q($?p_2$, `rdfs:range`, $?c$, $?k$)
	Q($?p_1$, `rdfs:subPropertyOf`, $?p_2$, $?k$)
		Q($?p_1$, `rdfs:range`, $?c$, $?k$)

scm-hv	T($?c_1$, `owl:hasValue`, $?i$, `owlc:Context`)
	Q($?c_1$, `owl:onProperty`, $?p_1$, `owlc:Context`)
	Q($?c_2$, `owl:hasValue`, $?i$, `owlc:Context`)
	Q($?c_2$, `owl:onProperty`, $?p_2$, `owlc:Context`)
	Q($?p_1$, `rdfs:subPropertyOf`, $?p_2$, $?k$)
		Q($?c_1$, `rdfs:subClassOf`, $?c_2$, $?k$)

scm-svf1	Q($?c_1$, `owl:someValuesFrom`, $?y_1$, `owlc:Context`)
	Q($?c_1$, `owl:onProperty`, $?p$, `owlc:Context`)
	Q($?c_2$, `owl:someValuesFrom`, $?y_2$, `owlc:Context`)
	Q($?c_2$, `owl:onProperty`, $?p$, `owlc:Context`)
	Q($?y_1$, `rdfs:subClassOf`, $?y_2$, $?k$)
		Q($?c_1$, `rdfs:subClassOf`, $?c_2$, $?k$)

scm-svf2	Q($?c_1$, `owl:someValuesFrom`, $?y$, `owlc:Context`)
	Q($?c_1$, `owl:onProperty`, $?p_1$, `owlc:Context`)
	Q($?c_2$, `owl:someValuesFrom`, $?y$, `owlc:Context`)
	Q($?c_2$, `owl:onProperty`, $?p_2$, `owlc:Context`)
	Q($?p_1$, `rdfs:subPropertyOf`, $?p_2$, $?k$)
		Q($?c_1$, `rdfs:subClassOf`, $?c_2$, $?k$)

scm-avf1	Q($?c_1$, `owl:allValuesFrom`, $?y_1$, `owlc:Context`)
	Q($?c_1$, `owl:onProperty`, $?p$, `owlc:Context`)
	Q($?c_2$, `owl:allValuesFrom`, $?y_2$, `owlc:Context`)
	Q($?c_2$, `owl:onProperty`, $?p$, `owlc:Context`)
	Q($?y_1$, `rdfs:subClassOf`, $?y_2$, ?k)
		Q($?c_1$, `rdfs:subClassOf`, $?c_2$, $?k$)

scm-avf2	Q($?c_1$, `owl:allValuesFrom`, $?y$, `owlc:Context`)
	Q($?c_1$, `owl:onProperty`, $?p_1$, `owlc:Context`)
	Q($?c_2$, `owl:allValuesFrom`, $?y$, `owlc:Context`)
	Q($?c_2$, `owl:onProperty`, $?p_2$, `owlc:Context`)
	Q($?p_1$, `rdfs:subPropertyOf`, $?p_2$, $?k$)
		Q($?c_2$, `rdfs:subClassOf`, $?c_1$, $?k$)

scm-int	Q($?c$, `owl:intersectionOf`, $?x$, `owlc:Context`) LIST[$?x$, $?c_1$, ..., $?c_n$]	Q($?c$, `rdfs:subClassOf`, ?c$_i$, `owlc:Context`), $i = 1, n$
scm-uni	Q(?c, `owl:unionOf`, ?x, `owlc:Context`) LIST[?x, ?c$_1$, ..., ?c_n]	Q(?c$_i$, `rdfs:subClassOf`, ?c, `owlc:Context`) $i = 1, n$

Table 4.9 OWL^C: Entailment rules for the schema vocabulary

4.4.9 Interaction between core/context and restrictions

Table 4.10 presents the rules of the interaction between the two languages. Syntactic restrictions are applied to the new constructors; an existential contextual restriction ($\langle C\rangle\, D$, $\langle C\rangle\, R$, $\langle R.C\rangle\, D$, $\langle R.C\rangle\, S$) may only appear on the left-hand side of a subclass axiom, whereas a universal contextual restriction ($[C]D$, $[C]R$, $[R.C]D$, $[R.C]S$) may only appear on the right-hand side.

	IF	**THEN**

cxt-svf
$E = (\langle C\rangle\, D)$
$k : D(x)$
$C(k)$
$\rightarrow$
$\top : E(x)$
	Q(?e, `owlc:onClass`, ?d, `owlc:Context`)
	Q(?e, `owlc:inSomeContextOf`, ?c, `owlc:Context`)
	Q(?x, `rdf:type`, ?d, ?k)
	T(?k, `rdf:type`, ?c)
		Q(?x, `rdf:type`, ?e, ?`owlc:Context`)

cxt-avf
$E = ([C]D)$
$E(x)$
$C(k)$
$\rightarrow$
$k : D(x)$
	Q(?e, `owlc:onClass` ?d, `owlc:Context`)
	Q(?e, `owlc:inAllContextOf`, ?c, `owlc:Context`)
	Q(?x, `rdf:type`, ?e, `owlc:Context`)
	T(?k, `rdf:type`, ?c)
		Q(?x, `rdf:type`, ?d, ?k)

cxt-svf-k
$E = (\langle r.C\rangle\, D)$
$z : D(x)$
$C(z)$
$r(y, z)$
$\rightarrow$
$y : E(x)$
	Q(?e, `owlc:onClass`, ?d, `owlc:Context`)
	Q(?e, `owlc:inSomeContextOf`, ?c, `owlc:Context`)
	Q(?e, `owlc:linkedThrough`, ?r, `owlc:Context`)
	Q(?x, `rdf:type`, ?d, ?z)
	T(?z, `rdf:type` ?c)
	T(?y, ?r ?z)
		Q(?x, `rdf:type`, ?e, ?y)

cxt-avf-k
$E = ([r.C]\, D)$
$y : E(x)$
$r(y, z)$
$C(z)$
$\rightarrow$
$z : D(x)$
	Q(?e, `owlc:onClass`, ?d, `owlc:Context`)
	Q(?e, `owlc:inAllContextOf`, ?c, `owlc:Context`)
	Q(?e, `owlc:linkedThrough` ?r, `owlc:Context`)
	Q(?x, `rdf:type`, ?e, ?y)
	T(?y ?r ?z)
	T(?z, `rdf:type`, ?c)
		Q(?x, `rdf:type`, ?d, ?z)

Table 4.10 OWL^C : entailment rules for the context-based concept-forming operators

Example. Consider the OWL^C class definition

$$\mathsf{FormerPresident} = \langle \mathsf{PastPresidentialTerm} \rangle \mathsf{President}$$

(A former president is someone who is president in some context that is a past presidential term)

If the ABox contains the facts

$$\mathsf{1933\text{-}1945} : \mathsf{President}(\mathsf{Roosvelt})$$

$$\mathsf{PastPresidentialTerm}(\mathsf{1933\text{-}1944})$$

the application of the existential rule ctxt-svf will infer

$$\mathsf{owlc{:}Context} : \mathsf{FormerPresident}(\mathsf{Roosvelt})$$

(Roosevelt is a former president, in every context)

4.5 Optimizing reasoning

The rules presented in the previous section are complete for instance checking, i.e., every fact of the form $k : C(a)$ or $k : r(a, b)$ that is a logical consequence of the ontology can be inferred by applying the rules (see appendix A for a formal statement and proof). However, the inference mechanism may generate a considerable amount of useless facts. Consider, for instance, the axiom

$$k_1 : \exists R.C \sqsubseteq D$$

When mapped to triples and quadruples, it will generate

```
    _:x rdfs:subClassOf :D :k1,
    _:x owl:someValueFrom :C,
    _:x owl:onProperty :R
```

If the Abox contains the quadruples

```
  :a1 :R :b1 :k1, :b1 rdf:type :C :k1,
  :a2 :R :b2 :k2, :b2 rdf:type :C :k2,
  :a3 :R :b3 :k3, :b3 rdf:type :C :k3
```

The application of rule cls-svf1 will generate

```
  :a1 rdf:type _:x :k1,
  :a2 rdf:type _:x :k2,
  :a3 rdf:type _:x :k3
```

But the rule cax-sco will only use the first of these assertions to generate

```
  :a1 rdf:type :D :k1
```

Since the axiom is valid only in context `k1`, it was totally useless to generate type assertions in contexts `k2` and `k3`.

One way to mitigate this problem is by introducing a context restriction operator $C|_K$, which is defined by

$$(C|_K)^{\mathcal{I}[k]} = \begin{cases} C^{\mathcal{I}[k]} & \text{if } k \in K^{\mathcal{J}} \\ \emptyset & \text{else} \end{cases}$$

In other words, $C|_K$ is non-empty only in those contexts that belong to K.

This constructor enjoys the following properties: $(C \sqcup D)|_K = C|_K \sqcup D|_K$, $(C \sqcap D)|_K = C|_K \sqcap D|_K$, $(\exists R.C)|_K = \exists R|_K.C|_K$, ...

Moreover, we have the equivalence

$$K : C \sqsubseteq D \equiv K : C|_K \sqsubseteq D|_K$$

and the tautology

$$\top^C : C|_K \sqsubseteq C$$

Using theses properties, for any formula that is in a negative normal form (all negations appear in front of an atomic subexpression made of a class symbol), it is possible to "push" all the $|_K$ constructors inward in front of C or $\neg C$ expressions.

An expression $C|_K$ can be mapped to the FOL subexpression $K(k) \wedge C(x,k)$. Further, $R|_K$ can be mapped to $K(k) \wedge R(x,y,k)$. This leads to the inference rules (for individual contexts).

IF	THEN

T($?cr$, `owlc:restrictionOfClass`, $?c$)
T($?cr$, `owlc:onContext`, $?k$)
Q($?x$, `rdf:type`, $?c$, $?k$)
	Q($?x$, `rdf:type`, $?cr$, $?k$)

T($?pr$, `owlc:restrictionOfProperty`, $?p$)
T($?pr$, `owlc:onContext`, $?k$)
Q($?x$, $?p$, $?y$, $?k$)
	Q($?x$, $?pr$, $?y$, $?k$)

T($?cr$, `owlc:restrictionOfClass`, $?c$)
T($?cr$, `owlc:onContext`, $?k$)
Q($?x$, `rdf:type`, $?cr$, $?k$)
	Q($?x$, `rdf:type`, $?c$, $?k$)

T($?pr$, `owlc:restrictionOfProperty`, $?p$)
T($?pr$, `owlc:onContext`, $?k$)
Q($?x$, $?pr$, $?y$, $?k$)
	Q($?x$, $?p$, $?y$, $?k$)

Using the above equivalences, the above axiom

$$k1 : \exists R.C \sqsubseteq D$$

can be transformed to

$$k1 : \exists R|_{k1}.C|_{k1} \sqsubseteq D|_{k1},$$

and mapped to the triples

```
_:x rdfs:subClassOf _:dr,
_:x owl:someValueFrom _:cr,
_:x  owl:onProperty _:rr,
_:cr owlc:restrictionOfClass :C,
_:cr owlc:onContext :k1,
_:rr owlc:restrictionOfProperty :R,
_:rr owlc:onContext :k1,
_:dr owlc:restrictionOfClass :D,
_:dr owlc:onContext :k1
```

With the ABox of the above example, the rules will only generate these quadruples:

```
:a1 :rr :b1 :k1,
:b1 rdf:type :cr :k1,
:a1 rdf:type :dr :k1,
:a1 rdf:type :D :k1.
```

4.5.1 Optimizing with context sets

When the TBox contains many contexts, for instance, if the temporal granularity is the year and we want to represent the period $[1850, 1910]$, we must introduce 61 contexts. Even worse, if we reduce the granularity to a day, it would give rise to thousands of contexts. To avoid an explosion of fact generation, we can use a compact representation $Q(?s, ?p, ?o, ?K)$, where K represents a (possibly infinite) set of contexts. To do that, the following is introduced:

- an abstract syntax, based on T predicates, to represent a contexts set. For a purely temporal context this could typically be :

```
T(k, rdf:type owlc:TemporalContextSet)
T(k, start, 1908)
T(k, end, 1924)
T(k, increment, 1)
```

- the "intersect ([K$_1$, ..., K$_n$], K)" function that calculates the intersection of two or more context (K$_1$, ..., K$_n$) sets and output K, which is the result of the intersection. The intersection must be efficiently computable (typically in constant time). We also introduce the notempty(K), a Boolean function that will be used to test if the result of the intersection is true or false.

Table 4.11 presents the rules of the class constructors using context sets.

	IF	THEN

cls-com	T($?c_1$, owl:complementOf, $?c_2$)
	Q($?x$, rdf:type, $?c_1$, $?K_1$)
	Q($?x$, rdf:type, $?c_2$, $?K_2$)
	intersect($[?K_1, ?K_2], ?K$)
	notempty($?K$)
		false

cls-int1	T($?c$, owl:intersectionOf, $?x$)
	LIST[?x, $?c_1$, ..., $?c_n$]
	Q($?y$, rdf:type, $?c_1$, $?K_1$)
	Q($?y$, rdf:type, $?c_2$, $?K_2$)
	...
	Q($?y$, rdf:type, $?c_n$, $?K_n$)
	intersect($[?K_1, \ldots, ?K_n], ?K$)
	notempty($?K$)
		Q($?y$, rdf:type, $?c$,$?K$)

cls-int2	T(?c, owl:intersectionOf, ?x)
	LIST[?x, $?c_1$, ..., $?c_n$]
	Q($?y$, rdf:type, $?c$, $?K$)
		Q($?y$, rdf:type, $?c_1$, $?K$)
		Q($?y$, rdf:type, $?c_2$, $?K$)
		...
		Q($?y$, rdf:type, $?c_n$, $?K$)

cls-uni	T($?c$, owl:unionOf, $?x$)
	LIST[$?x$, $?c_1$, ..., $?c_n$]
	Q($?y$, rdf:type, $?c_i$, $?K_i$)
		Q($?y$, rdf:type, $?c$, $?K_i$)
		
		one rule for each c_i

cls-svf1-1
$\exists R.C$
	T($?x$, owl:someValuesFrom, $?y$)
	T($?x$, owl:onProperty, $?p$)
	Q($?u$, $?p$, $?v$, $?K_1$)
	Q($?v$, rdf:type, $?y$, $?K_2$)
	intersect($[?K_1, ?K_2], ?K$))
	notempty($?K$)
		Q($?u$, rdf:type, $?x$, $?K$)

cls-avf-1
$\forall R.C$
	T($?x$, owl:allValuesFrom, $?y$)
	T($?x$, owl:onProperty, $?p$)
	Q($?u$, rdf:type, $?x$, $?K_1$)
	Q($?u$, $?p$, $?v$, $?K_2$)
	intersect($[?K_1, ?K_2], ?K$)
	notempty($?K$)
		Q($?v$, rdf:type, $?y$, $?K$)

Table 4.11 OWL^C: Entailment rules for the core language with context sets

4.6 Complexity

Each rule application generates one or several new facts of the form $Q(s, p, o, k)$ (or false) where s, p, o, and k are entities (URI, literal, or blank nodes) that already exist in the (asserted) ontology O. No rule creates new entities (URI, literal). Therefore, the maximum number of new Q(...) that can be generated in the reasoning (rule application) process is $N \times N \times N \times M$, where N is the number of entities that appear in the core axioms of O, and M is the number of entities that appear in the context axioms of O.

However, note that the rule-reasoning process is complete (for a form of instance checking) only if some conditions are satisfied (as stated in the completeness theorem in appendix A). Moreover, problems such as class expression satisfiability, class expression subsumption, and instance checking for OWL2-RL are PTIME-complete. Since OWLC "includes" OWL2-RL, these problems are at least PTIME-complete in OWL^C.

4.7 OWL^C on property graphs: problems and restrictions

As mentioned previously, OWL^C was prompted by a project in digital humanities. The rise of knowledge graphs and discussions with researchers in the field turned our focus to available graph datasets such as Wikidata. What made it more appealing is: 1) it being an available dataset and 2) its richness with contextual knowledge. At that time, researchers investigated the compatibility of DL and OWL with modern knowledge graphs (Krötzsch and Thost, 2016), which made the idea of applying OWL^C on Wikidata more appealing.

The big problem when trying to apply OWL^C on Wikidata is the incompatibility of Wikidata's structure with OWL^C. We will see in the next chapter that Wikidata is extremely rich with contexts known as qualifiers. These contexts are not restricted to validity contexts that make their representation and processing through OWL^C impossible. Now, OWL^C can certainly be applied to a subset of Wikidata that contains only validity contexts, but this means dropping the rich contextual structure of Wikidata. In addition, Wikidata has many levels of the instance of property (which are similar to rdf: type in RDF) (a, instance of, b) (b, instance of, c). These levels are not tolerated in DL-based languages. Besides that, Wikidata is a knowledge graph containing contexts without any semantics. There is no semantics at all in Wikidata.

Since we are interested in contextual reasoning and graph-based structures that are rich with contexts, Wikidata was used. At this stage of work, we took a step back and asked ourselves: Do DL-based languages adequately represent the semantics of knowledge graphs such as Wikidata? Having a semantic web background, we were so focused on DL and semantic web languages (RDFS, OWL), asking the question: Are they a good fit for these emerging data structures? We opened the door for a new contribution to this thesis with these questions, which are illustrated in the upcoming four chapters. Chapter 5 analyzes Wikidata in detail, its incompatibility with DL-based languages, and sets the goal of the work. In chapters 6 and 7, the semantics of Wikidata and its qualifiers (and hence knowledge graphs) are formalized with a contextual model that uses a many-sorted logic. These sorts are used to represent the different contexts in Wikidata. Finally, chapter 8 discusses implementation details.

Chapter 5

Contextual reasoning on knowledge graphs: the Wikidata use case

Chapter Contents

5.1 Introduction

This chapter analyzes knowledge graph properties and discusses reasoning. It mainly discusses the case of Wikidata, a knowledge graph (more precisely, a property graph) lacking formal semantics. Its compatibility is analyzed with current knowledge representation formalisms, and ultimately, the research questions are formalized.

5.2 Reasoning challenges on knowledge graphs

Reasoning over knowledge graphs (notably property graphs) poses several unique challenges:

C1 Some knowledge graphs are enormous (i.e., Wikidata); hence, they need highly scalable reasoning techniques.

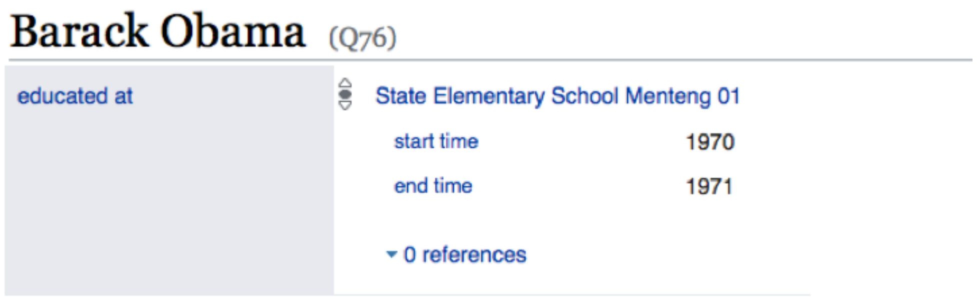
C2 Knowledge graphs are not represented with ontological/semantic web languages such as RDFS/ OWL (i.e., Wikidata is published in JSON). Compared with OWL, much data published in Wikidata violates the strict syntactic fundamentals of OWL. For instance, in Wikidata, sequences of *instance of* properties *((x instanceOf y)(y instanceOf w))* can be found.

C3 Knowledge graphs are evolving; on Wikidata, people add and remove information.

C4 Knowledge graphs use a data model that is different from those of RDFS and OWL. RDFS and OWL are based on labeled graphs (i.e., the labels correspond to binary relations). Knowledge graphs are generally labeled graphs with attributes/qualifiers attached to the edges. The edge labels correspond to ternary or n-ary relations. In terms of reasoning, they would benefit from a logical language to formalize the hidden semantics.

5.3 A Wikidata description

Wikidata is a free, collaboratively edited multilingual knowledge graph. It aims to curate and represent the factual information of Wikipedia (across all languages) in an interoperable, machine-readable format. Previously, initiatives like DBpedia (Bizer et al., 2009) and YAGO (Hoffart et al., 2013) have generated essential knowledge bases simply by applying custom extraction frameworks over Wikipedia. The problem is that the ad-hoc way used to embed structured data in Wikipedia limits the amount of data that can be extracted. Similarly, the results may not always be coherent when information is gathered from multiple articles or multiple language versions of Wikipedia. Redundant facts must be manually curated and updated by human editors. Therefore, by allowing human editors to collaboratively add, edit, and curate a centralized, structured knowledge base, Wikidata should have one consistent



Barack Obama (Q76)

educated at	State Elementary School Menteng 01	
	start time	1970
	end time	1971
	▾ 0 references	

Fig. 5.1 Statement from the page of Barack Obama (This figure is constructed using screenshots from the Wikidata page of Barack Obama (Wik, a))

version of factual data. Wikidata was launched in October 2012, and as of January 2021, it stores information about almost 93,512,660 items. The content of Wikidata is used in other applications and on the Web. For example, the Academy Awards portal of the major German newspaper FAZ online[1] uses Wikidata to generate interactive query views on specific subjects, and Apple's Siri search engine uses Wikidata for question-answering.

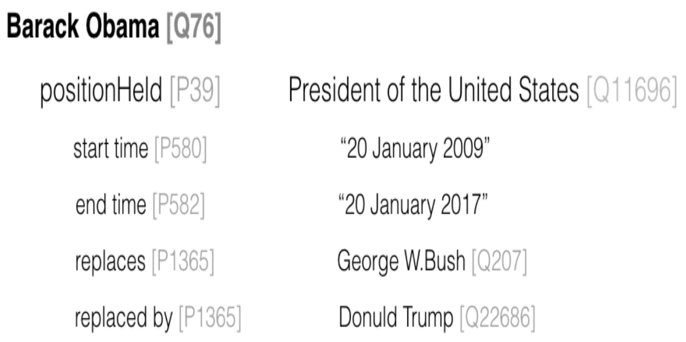
Wikidata uses the term "statement" [2] to refer to a property-value pair (with optional qualifiers [3]), augmented by references and a rank. Fig 5.1 shows an example statement taken from the page on Barack Obama. The main part of the statement can be read as a directed edge: Barack Obama educated at State Elementary School Menteng 01. The statement is annotated with attribute-value pairs that include classical "meta-data" such as validity time and references. Interestingly, Wikidata allows for using the same attribute in the main part of the graph as a property, or for adding contextual/additional information. Furthermore, Wikidata allows users to create new properties and make statements about them. Wikidata contains specific details (Krötzsch and Thost, 2016) that are worth mentioning:

1. The same directed relationship may occur several times with different annotations, e.g., in the case of Elizabeth Taylor, who was married to Richard Burton several times.

2. Wikidata uses an underlying conceptual (informal) vocabulary whose "ontological vocabulary" is subclass of (P279) and instance of (P31), in addition to symmetric properties and other types of property constraints.

3. The order of statements and statement annotations is not relevant.

[1]http://www.faz.net/aktuell/feuilleton/kino/academy-awards-die-oscar-gewinner-auf-einen-blick-12820119.html5
[2]https://www.wikidata.org/wiki/Wikidata:Glossary
[3]Wikidata use the term qualifiers to refer to attributes



Barack Obama [Q76]

positionHeld [P39]	President of the United States [Q11696]
start time [P580]	"20 January 2009"
end time [P582]	"20 January 2017"
replaces [P1365]	George W.Bush [Q207]
replaced by [P1365]	Donuld Trump [Q22686]

Fig. 5.2 Raw Wikidata format

5.4 Wikidata data model

The factual information of Wikidata reminds us of the RDF data model, where the main data item (entity) can be viewed as the subject of a triple and the attribute–value pairs associated with that item can be mapped naturally to predicates and objects associated with the subject. Qualifications are a set of attribute-value pairs or qualifiers-values. We use the term "qualifiers" in the rest of the manuscript. Qualifiers are properties such as start time, end time, etc., whose values may scope the statement's validity and/or provide additional information. Besides, statements are often associated with one or more references that support the claims and with a rank. The role of the rank is to provide a mechanism for annotating multiple values for a statement. Fig. 5.2 provides a sample statement from Wikidata describing the entity Barack Obama. It contains a primary relation, with Barack Obama as subject, position held as property, and President of the United States of America as value. The qualifiers are start time, end time, replaces, and replaced by. Among the identifiers shown in grey, those starting with Q refer to entities, and those starting with P refer to properties. These identifiers map to IRIs, where information about that entity or relationship can be found. All entities and relationships are also associated with labels; the English version is shown for readability. Values of properties may also be datatype literals, as exemplified with the dates.

Hernández et al. (2015) propose to view Wikidata conceptually in terms of two tables: the first table contains quads of the form (s, p, o, i), where (s, p, o) is a primary relation, and i is an identifier for that statement; the second table contains the following:

1. primary relations that can never be qualified (e.g., labels) and thus need not be identified

2. triples of the form (i, q, v) (e.g., where q stands for qualifier, and v stands for value) that specify the qualifiers associated to a statement

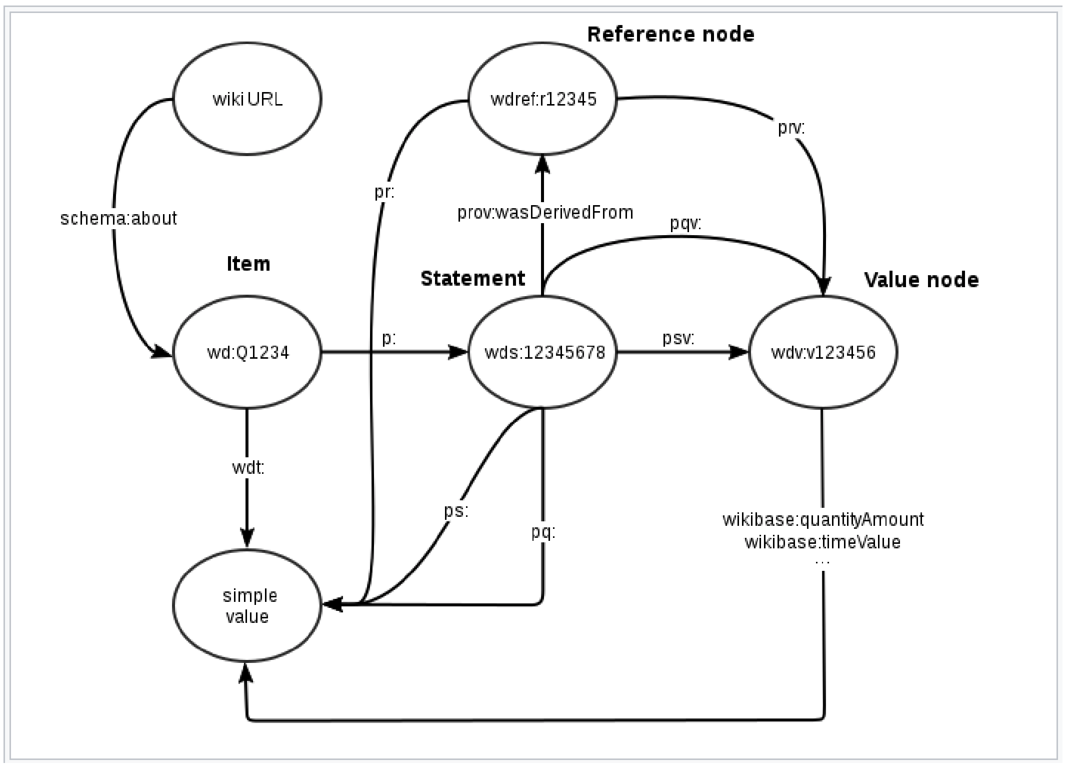


Fig. 5.3 Wikidata data model (This figure is a screenshot taken from https://www.mediawiki.org/wiki/Wikibase/Indexing/RDF_Dump_Format)

3. triples of the form (v, x, y) that further describe the properties of qualifier values

Notice that Wikidata is internally stored in a document-centric form using JSON. To represent Wikidata in RDF while capturing meta-information such as qualifiers and references, we need some way in RDF to describe the Wikidata statements. This method should be compatible with the existing semantic web standards and tools and should not consider the domain of triple annotation as fixed in any way. Fig.5.3 shows the RDF representation of the Wikidata data model based on the n-ary pattern. It was chosen because it was the only model with built-in support for SPARQL property paths (Hernández et al., 2015). Different prefixes[4] are used in this model. For instance, the prefixes p: and ps: are used to select statements. they can be used with the pq: prefix to access meta-information through the qualifiers while wdt: focuses more entities.

5.5 Reasoning in Wikidata: use cases

Wikidata is an interesting project. Although it encodes various forms of contextual information, it lacks formal semantics not just for contexts but in general. Hence, no reasoning is performed on Wikidata. The need for well-founded reasoning capabilities in Wikidata has been noted and lightly documented in a WikiProject[5]. In this section, two reasoning questions are raised:

- How to reason with Wikidata qualifiers? As mentioned in the previous sections, Wikidata contains many qualifiers. Dealing with the qualifiers while reasoning is one of the issues raised by the Wikidata community[6]: should they be copied, ignored, or do they deserve special treatment? Some people talk about ignoring qualifiers, but some qualifiers like the validity time cannot be ignore. What if we can find a mechanism for reasoning that takes qualifiers into consideration?

- How can the Wikidata "ontological properties " and "constraints" be axiomatized? In fact, Wikidata contains some properties similar to RDF(S)[7] and many constraints similar to RDFS and OWL constraints [8].

[4]https://www.wikidata.org/wiki/EntitySchema:E49
[5]https://www.wikidata.org/wiki/Wikidata:WikiProject_Reasoning/Use_cases
[6]https://www.wikidata.org/wiki/Wikidata:WikiProject_Reasoning
[7]https://www.wikidata.org/wiki/Wikidata:Relation_between_properties_in_RDF_and_in_Wikidata
[8]https://www.wikidata.org/wiki/Wikidata:WikiProject_property_constraints

5.5.1 How to reason with Wikidata qualifiers?

This section discusses two examples raised and inspired by the Wikidata community contributors. The examples are presented in the following format:

$$\text{(subject, predicate, object)}$$
$$[\text{qualifier}_1 : \textit{values}, ..., \text{qualifier}_n : \textit{values}]$$

Example 1. The first example is about "nationalities," specifically when people belong to a political country whose organization may change over time (e.g., a kingdom may become a republic, leading to a new country).

(Harry Secombe, has country of citizenship, the United kingdom of great Britain and Ireland)
[end time : *12 April 1927*, end cause : *Royal and parliamentary acts of 1927*]

(Harry Secombe, has country of citizenship, the United kingdom)

The Wikidata reasoning group criticizes the redundancy of the fact. We add to this the imprecision of the second statement; it can be seen that it is not annotated with qualifiers. However, Wikidata also contains this statement:

(The united Kingdom of great Britain and Ireland, is followed by, the United Kingdom)
[point in time : *12/4/1927*, has cause : *Royal and parliamentary acts of 1927*].

Using the first, second, and third statements, we can infer a new statement that is more precise than the second one: (Harry Secombe, has country of citizenship, the United Kingdom) [start time: *13 April 1927*, cause: *Royal and parliamentary acts of 1927].* We need the second statement because Ireland separated from the United Kingdom of Great Britain and Ireland, and therefore some UK citizens became Irish citizens. However, no theory mentions how to practically deal with qualifiers in reasoning. There exists many other people in that same case. Moreover, we find an example about merging countries like the German reunification.

Example 2. Another example is "the offices of heads of government." We find the following statements in Wikidata:

(Cyprus, office held by head of state, President of Cyprus)
(Cyprus, head of government, Dimitris Christofias) [end time : *28 February 2013)*]

It should be possible to infer (Dimitris Christofias, position held, President of Cyprus). The original statement is accompanied by a validity time that can be copied to the inference[9].

[9]check this link for more information https://www.wikidata.org/wiki/Wikidata:WikiProject_Reasoning/Use_cases#Usecase_description

5.5.2 Ontological properties and constraints

Although we have used the term "ontological," the notion of ontological properties is not explicitly defined in Wikidata. By the term, we refer to properties and entities used to conceptualize a domain.

Ontological properties: Wikidata contains some ontological properties, which include the instance of and subclass of. The instance of property is similar to the rdf:type property in RDF. We can find sequences of instance of properties (e.g., two statements having instance of as a property and the value of the first one is the subject of the second one) such as

$$(s, \textsf{instance of}, o)(o, \textsf{instance of}, r)$$

, where $(s, \textsf{instance of}, o)$ represents the first statement and $(o, \textsf{instance of}, r)$ the second statement. Both of them are qualified with qualifiers. Other notable sequences are as follows:

$$(s, \textsf{instance of}, o)(o, \textsf{subclass of}, r)$$

where $(s, \textsf{instance of}, o)$ represents the first statement and $(o, \textsf{subclass of}, r)$ the second statement with the object o of the first statement being the subject of the second statement. Both are enriched with qualifiers. The same question arises: What do we do with qualifiers when inferring a new statement?

Constraints Wikidata uses constraints to express some restrictions or characterizations of properties, such as the symmetry of a property or the domain and range of a property. Despite their usefulness, they are represented and processed in an incomplete, ad-hoc fashion. For instance, despite each property being declared and documented reasonably clearly, this documentation does not fully express their meaning and thus does not provide a precise or logical foundation for constraint-checking implementations. In the following, we present some examples:

- Symmetric properties: To declare that a property is symmetric, we should use the $Template : Constraint : Symmetric$. For example, by saying (spouse, property constraint, symmetric constraint), we declare that spouse is a symmetric property. However, despite being symmetric, there is no process to operationalize the symmetry of the property (i.e., to derive the same property between the object and subject)

  Example 3. Let us discuss the following example:

$$(\textsf{Douglas Adams, has spouse, Jane Belson})$$
$$[\textsf{start time} : \textit{25/11/1991}, \textsf{end cause} : \textit{death}].$$

The statement is also supported with one reference. Since spouse is a symmetric property, we should have as an inference that (Jane Belson, has spouse, Douglas Adam). Two issues arise in this case: i) the lack of an inference process for the symmetry and ii) what do we do with the qualifiers or the other contextual information? Should they be copied?

Example 4. On the Wikidata community page, a contributer gives the following example showing that copying the qualifiers is not enough:

(United States of America, diplomatic relation, Afghanistan)[
subject of : *Afghanistan–United States relations*,
diplomatic mission sent : *Embassy of the United States, Kabul*]

The property *diplomatic relation* is symmetric. Therefore, we should be able to infer (Afghanistan, diplomatic relation, United States of America) + (subject of: *Afghanistan–United States relations)* as qualifier but without the value *Embassy of the United States, Kabul* for the qualifier diplomatic mission sent (there might be another appropriate value for this qualifier, but we cannot know this from the statement on United States of America).

- Transitive properties: there is no explicit transitivity constraint in Wikidata. However, it contains some "ontological properties" that are implicitly transitive, such as the property subclass of. A list of qualifiers usually accompanies the subclass of property. It is however unclear as to what to do with these qualifiers when applying the transitivity.

5.6 Ontological modeling for Wikidata?

Having a semantic web background, we studied the benefits of applying knowledge representation technologies on Wikidata. As seen in the previous sections, Wikidata is a data management platform without an explicit ontology. However, the community uses some properties to capture "ontological" information, such as instance of and subclass of inspired by RDFS. Furthermore, some Wikidata properties are classified to express ontological information, such as the property spouse, which is declared as a symmetric property.

Although this "ontological" data is used in Wikidata, it ultimately remains data since it does not get special treatment. However, it would be desirable to axiomatize its semantics using a suitable ontology language, e.g., to declare that instances of symmetric properties are indeed symmetric. Therefore, it is natural to consider description logics (DLs) as good candidates for a logical approach to knowledge graphs. This is because DLs can provide an ontological layer that describes concepts and relationships used in the KG. However, the applicability of standard non-contextual DL languages is somewhat limited due to the context-aware nature of most knowledge graphs. Moreover, contrary to RDFS, DLs cannot represent facts such as $(A\ \mathsf{instance\ of}\ B), (B\ \mathsf{instance\ of}\ C)$, so a DL representation would necessarily be a limited view of the KG. Technically speaking, two limitations of DLs that restrict their usability for KG can be identified:

Limitation 1: Most DLs have the tree model property, and the modeling of contextual reasoning is very limited. For instance, consider the following axiom:

> *every person who was born in a city that was a German city at that time was born in Germany*

corresponds to the expression

$$\forall x, c, t.bornIn(x, c, t) \wedge GermanCity(c, t) \rightarrow BornInGermany(x)$$

which cannot be represented in the usual DLs. Its translation in a form that uses only unary and binary predicates is

$$\begin{aligned}&\forall b, x, c, t.birthOf(b, x) \wedge birthIn(b, c) \wedge birthAtTime(b, t)\\&\wedge GermanCityAtTime(c, t) \rightarrow BornInGermany(x)\end{aligned}$$

that has cyclic models. Hence, it cannot be expressed in DLs that have the tree model property.

Limitation 2: Wikidata contains the statement that *Taylor was married to Burton starting from 1964 and ending 1974.* This can be represented with several facts (Krötzsch, 2017):

$$spouse_1(s, taylor), \quad spouse_2(s, burton), \quad start(s, 1964) \quad end(s, 1974)$$

The claim that spouse is symmetric implicitly involves the assertion that Burton is also married to Taylor with the same start and end date. This asserts the existence of another inferred statement, which could be written as follows:

$$\forall x, y_1, y_2, y_3, y_4.spouse_1(x, y_1) \wedge spouse_2(x, y_2) \wedge start(x, y_3) \wedge end(x, y_4)$$

$$\rightarrow \exists v.spouse_2(v, y_1) \wedge spouse_1(v, y_2) \wedge start(v, y_3) \wedge end(v, y_4)$$

DLs cannot express this statement since they allow only one universally quantified variable to occur in both the premise and conclusion in axioms that have existential quantifiers.

Besides the two limitations, OWL contains many properties and class axioms far more complicated than what is needed in Wikidata. A subset of some of them is enough.

5.6.1 OWL^C for Wikidata

What limits the use of OWL^C on Wikidata is the following:

- The domains of context and object knowledge are separated in OWL^C, unlike in the case of Wikidata.
- OWL^C can be applied only to validity contexts. Wikidata contains validity and non-validity contexts such as causality. Hence, OWL^C can be applied only to a subset of Wikidata that deals with validity contexts.
- Wikidata needs something different from OWL^C. There is more implicit knowledge hidden in Wikidata that cannot be captured with the semantics of OWL^C such as the semantics of the qualifiers, the diverse constraints, and others.

OWL^C can be used on Wikidata only after removing everything that does not fit into the DL and removing qualifiers that do not represent validity contexts. However, all this means changing the rich structure of Wikidata.

5.7 Problem statement and research questions

This chapter described Wikidata and showed its incompatibility with existing reasoning formalisms. Providing a reliable construction of the implicit meanings in Wikidata requires a formal theory and its implementation. Without this, different users and tool builders determine and implement the meaning on their own, which often produces different, non-compatible results. Based on the previous sections, the following research questions are formulated:

Q1. How can the massive number of qualifiers in Wikidata be interpreted from a logical point of view?
Q2. What is an appropriate entailment regime for reasoning on Wikidata?

Chapter 6

Many-sorted logics and algebraic specification for reasoning on knowledge graphs

Chapter Contents

6.1 Introduction

Wikidata's data model supports attributed statements where the attributes are known as qualifiers and can have multiple values. Marx et al. (2017a) defines it as a generalized property graph as well as a multi-attributed graph and observes that "in spite of the huge practical significance of these data models ..., there is practically no support for using such data in knowledge representation." This observation imposes the challenge of constructing a logic in which a logical formulation improves the under-specified (formalized) parts of Wikidata. In this thesis, two under-specified issues are tackled:

1. formalizing the behavior and semantics of qualifiers
2. formalizing the ontological properties and constraints of Wikidata

It is right that Wikidata is just data, but many parts of Wikidata are documented and well discussed as if it is more than data. A formal theory would unify and standardize the implicit meaning of this documentation. One way of providing a common meaning is via a logic and a service that implements reasoning in the logic. Therefore, the contribution of this work is a many-sorted logic (MSL) that provides a decent logical view of the data in Wikidata. The many-sorted logic we propose:

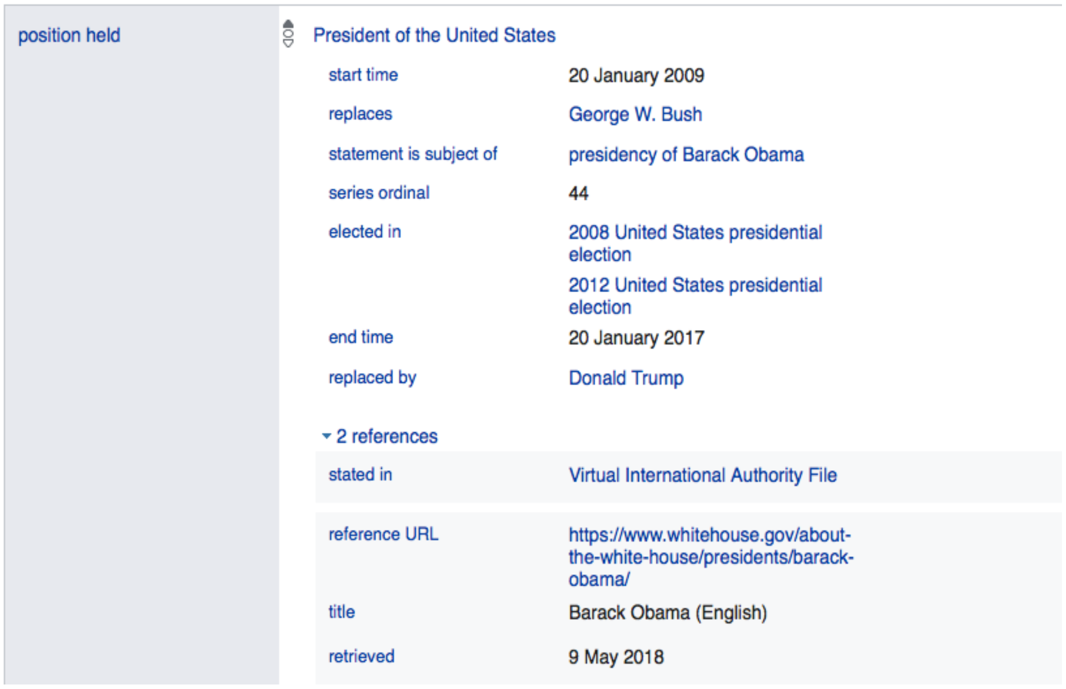
1. uses sorts to formalize the different types of qualifiers. The characteristics of each sort and the instructions on how it will be processed in reasoning are formalized using (parametrized) modules in an algebraic specification, and
2. formalizes the ontological properties and constraints of Wikidata and specifies the behavior of qualifiers in these properties.

In this chapter, the Wikidata qualifiers are analyzed and an abstraction of the massive qualifiers in Wikidata is proposed through many-sorted logic. Subsequently, in chapter 9, the algebraic specification is specified and then used to formalize and characterize inference rules.

This work is different from the state of the art of contextual reasoning on modern knowledge graphs, which to the best of our knowledge is constituted of two dependant work (Marx et al., 2017a), (Patel-Schneider and Martin, 2020) (see chapter 4, section 2.3, for more information). The difference is that we want to handle each type of attribute uniformly so that each rule would describe, for example, how the Wikidata temporal qualifiers are processed.

6.2 Analyzing wikidata qualifiers

Wikidata uses qualifiers to further describe or refine the value of a given property in a statement. Fig 6.1 shows the details of the `position held` property illustrating the presidency of Barack Obama, the president of the United States. This refinement is illustrated as follows:



position held	President of the United States	
	start time	20 January 2009
	replaces	George W. Bush
	statement is subject of	presidency of Barack Obama
	series ordinal	44
	elected in	2008 United States presidential election
		2012 United States presidential election
	end time	20 January 2017
	replaced by	Donald Trump
	2 references	
	stated in	Virtual International Authority File
	reference URL	https://www.whitehouse.gov/about-the-white-house/presidents/barack-obama/
	title	Barack Obama (English)
	retrieved	9 May 2018

Fig. 6.1 Qualifiers of the position Held property (This figure is a screenshot taken from the Wikidata page of Barack Obama (Wik, a))

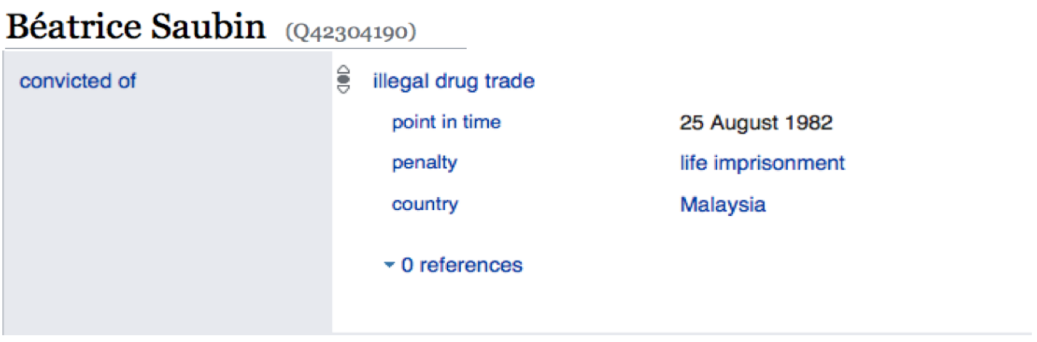
1. The validity time of the statement: from **20 January 2008** until **20 January 2017**
2. The order of the presidents: The president who came before (George Bush) and after (Donald Trump)
3. The provenance illustrated by a set of qualifiers (stated in, archive date, reference URL, etc.)
4. Other annotations adding extra details to the property

Previous works like Marx et al. (2017b) treat the value of qualifiers' properties as opaque. However, this work seeks to treat qualifiers by type. Wikidata provides a simple categorization of qualifiers [1]. The two categories are restrictive and non-restrictive qualifiers. However, no formal semantics is provided. By exploring all the qualifiers of Wikidata, the following five categories were identified. Other categories may be detected, but this work limits categories to these five.

6.2.1 Validity contexts

They provide information about the validity of a particular statement and affect the semantics of the statement by restricting its truth to a specific context. For instance, when saying that

[1] https://www.wikidata.org/wiki/Wikidata:List_of_properties/Wikidata_qualifier



Béatrice Saubin (Q42304190)

convicted of	illegal drug trade	
	point in time	25 August 1982
	penalty	life imprisonment
	country	Malaysia
	0 references	

Fig. 6.2 Validity space (This figure is constructed using screenshots taken from the Wikidata page of Beatrice Saubin (Wik, b))

Barack Obama is the president of the United States from 20 January 2009 till 20 January 2017, the validity of his presidency is restricted to the temporal interval [20/1/2009, 20/1/2017].

Validity contexts are not restricted to time; we also have validity space such as fig. 6.2. For example, according to Wikidata, Beatrice Saubin was convicted of illegal drug trade in Malaysia. Hence, the validity of her conviction is restricted to Malaysia, which makes Malaysia a validity space. As can be seen, validity contexts can be constituted of many dimensions, such as time ([20/1/2009, 20/1/2017]) and space (Malaysia). We can also find the jurisdiction used as a validity context.

We observe a frequent use of validity context qualifiers. Table 6.1 shows the three validity context categories explicitly: the blue color represents validity time, grey indicates validity space, and white represents the "applies to" context. For each qualifier, its ID, its Wikidata classification [2], data type, and its prominence [3]. The prominence results of this table and the others are computed on 05/08/2021 using Wikidata query service.

[2]Wikidata use this page https://www.wikidata.org/wiki/Wikidata:List_of_properties/Wikidata_qualifier to provide a classification of the qualifiers that include: 1) non-restrictive qualifier: a qualifier that does not restrict or modify the referent of the subject but merely adds additional information to the statement; 2) a qualifier that restricts or modifies the referent of the subject, without which the statement may be inaccurate or meaningless; 3) a Wikidata property used as "depicts" (P180) qualifier on Commons; and 4) other.

[3]the prominence is generated using a SPARQL query on Wikidata query service

```
Select ?s
where {
?s ?p1 ?r1.
?r1 pq:P1534 ?qualifierValue.
SERVICE wikibase:label {
bd:serviceParam wikibase:language "en".
} }
```

Qualifier name	ID	Wikidata classification	Data type	Prominence
valid in period	P1264	restrictive qualifier	item	781 020
start period	P3415	restrictive qualifier	item	376
end period	P3416	restrictive qualifier	item	180
start time	P580	unqualified	point in time	4 284 404(*)
end time	P582	unqualified	point in time	2 311 341(*)
valid in place	P3005	unqualified	point in time	42 896
country	P17	depicts	item	71 824
applies to part	P518	restrictive qualifier	item	800 949
applies to jurisdiction	P1001	restrictive qualifier	item	1 187 489
applies to name of item	P5168	restrictive qualifier	monolingual text	6 312
applies to people	P6001	restrictive qualifier	item	171
applies to name of value	P8338	restrictive qualifier	monolingual text	736

Table 6.1 The validity context qualifiers
(*) These qualifiers are taken from our 114GB (almost complete) downloaded version of the Wikidata 2020-11-09 dump on November 22, 2020

6.2.2 Annotations

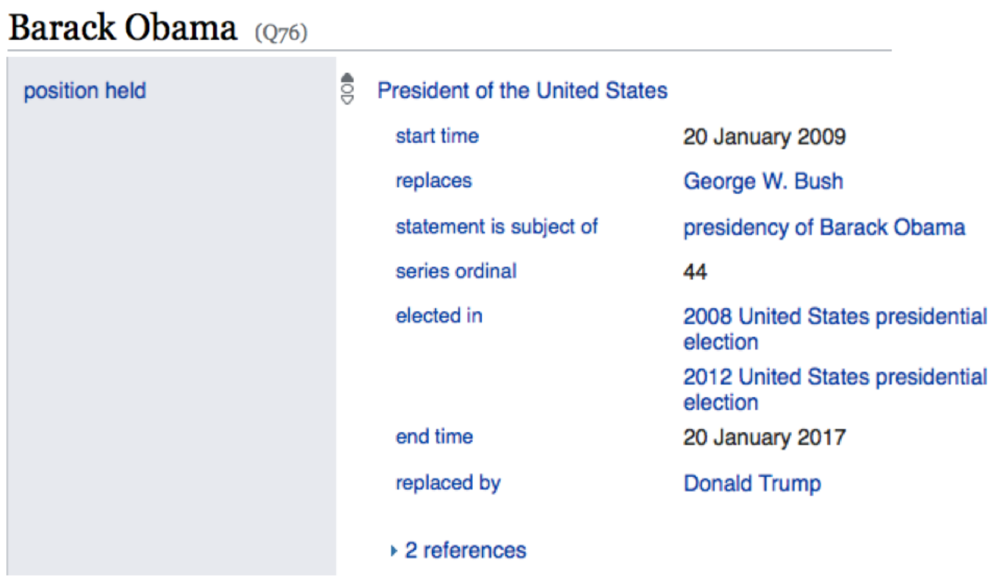
We use the term "annotations" to refer to qualifiers that supplement a fact with additional elements that do not modify its meaning. Thus, the fact is more precisely described, but it sufficiently clear without annotations. For instance,

Example 1.

(Alain de Benoist, educated at, University of Paris 1 Pantheon-Sorbonne)
[academic degree : *philosophy, sociology, history*]

academic degree is a qualifier used to provide additional information about the property educated at. This qualifier does not modify the semantics of the original statement; rather, it only adds information to it. Annotations in Wikidata can be divided into different subcategories.

- constraints annotations: used to define additional information on constraints such as exception to constraint (P2303), constraint status (P2316), separator (P4155), constraint scope (P4680), and constraint clarification (P6607).
- the of qualifier: stating that a statement applies within the scope of a particular item. This qualifier is a special one used for modeling purposes.



Barack Obama (Q76)

position held — President of the United States

start time	20 January 2009
replaces	George W. Bush
statement is subject of	presidency of Barack Obama
series ordinal	44
elected in	2008 United States presidential election
	2012 United States presidential election
end time	20 January 2017
replaced by	Donald Trump

▸ 2 references

Fig. 6.3 Sequence qualifiers: replaces and replaced by (This figure is constructed using screenshots taken from the Wikidata page of Barack Obama (Wik, a)

- others: attributes used to add more information about a particular topic such as hair color, academic degree, affiliation, etc.

The following three sections present non-validity qualifiers, or more clearly, qualifiers that do not affect the semantics of the original statement: sequence, causality, and provenance. However, since they deserve special treatment and have special reasoning rules, they are further explained in different sections and formally treated later in the logic and specification.

6.2.3 Sequence

A typical example of the sequence qualifiers is the Wikidata page of a politician. Figure 6.3 shows the Wikidata page of Barack Obama and especially the position held property. Among the qualifiers used, two sequence qualifiers can be spotted: replaces and replaced by. The latter is used to name the president who came after Obama. The former is used to name the president who came before him. Sequence qualifiers behave like pointers referring to the element that comes before the subject and the element after the subject (respectively, previous and next). The previous and next presidents should have a similar Wikidata page with a positionHeld property. To make that possible, special rules inspired by the behavior of the sequence qualifier will be presented in the next chapter. Table 6.2 shows the sequence qualifiers used in Wikidata and for each qualifier, his ID, his Wikidata classification, data type, and his prominence.

It is observed that the follows/followed by qualifiers are used more than replaces/replaced by. In our Wikidata search, it was identified that the sequence qualifier is frequently used

Qualifier name	ID	Wikidata classification	Data type	Prominence
followed by	P156	non-restrictive qualifier	item	728 439
follows	P155	non-restrictive qualifier	item	728 943
replaced by	P1366	other	item	157 195
replaces	P1365	other	item	175 818

Table 6.2 The sequence qualifiers

Qualifier	Start time	End time	Number of statements
replaced by	✓	✓	121 590
replaced by	✓		150 514
replaced by		✓	124 092
replaces	✓	✓	112 252
replaces	✓		171 534
replaces		✓	112 182
followed by	✓	✓	2 817
followed by	✓		3 113
followed by		✓	2 964
follows	✓	✓	2 274
follows	✓		3 176
follows		✓	2 296

Table 6.3 The coupling of sequence/validity time qualifiers

with validity time. Since we are interested to see the qualifiers coupling in Wikidata (i.e., the different types of qualifiers that come together), we present in Table 6.3 the coupling of the sequence qualifiers with start time and end time.

It can be seen that the replaces/replaced by qualifiers are coupled with time more than the follows/followed by qualifiers. It is also observed that most statements with a sequence qualifier are qualified with a start time more than an end time.

6.2.4 Causality

Wikidata contains causality qualifiers used to assert the causality of the beginning/end of some statements such as a marriage ending or a political or social event. In example 2, it can be seen that the causality was used to explain the reason of the end time of Carter Harrison being the head of the government of Chicago.

Qualifier name	**ID**	**Wikidata classification**	**Data type**	**Prominence**
has cause	P828	not classified	item	12 238
end cause	P1534	other [4]	item	124 911

Table 6.4 The causality qualifiers

Qualifier	**Start time**	**End time**	**Number of statements**
has cause	✓	✓	2 263
has cause	✓		4 186
has cause		✓	2 366
end cause	✓	✓	73 966
end cause	✓		74 146
end cause		✓	102 001

Table 6.5 The coupling of causality/validity time qualifiers

Example 2.

(Chicago, head government, Carter Harisson)[
start time : *1893*,
end time : *28 october 1893*,
end cause : *death in the office*,
]

Table 6.4 shows that the end cause qualifier is used more frequently than the has cause qualifier. The causality qualifiers are usually used in the presence of other qualifiers such as validity time. Table 6.5 shows the coupling of causality qualifiers with validity time qualifiers. The has cause qualifier is less coupled with validity time than the end cause. A strong coupling is observed between the end cause qualifier and the end time qualifier. The has cause is used with start time more frequently compared to has cause with end time. Furthermore, the coupling between the causality qualifiers and sequence qualifiers is less pertinent, as seen in Table 6.6. It can also be noted that the end cause qualifier is coupled with sequence qualifiers more than the has cause.

6.2.5 Provenance

We use the term "provenance" to refer to references used to state the sources of a statement. References consist of at least one property-value pair. They are usually used to support the claim. A provenance can consist of one or more references, although statements without references are found in Wikidata. The qualifiers used to assert the provenance provide information about the place the data was retrieved from, a URL, a date, etc.

Qualifier	**Replaces**	**Replaced by**	**Number of statements**
has cause	✓	✓	27
has cause	✓		49
has cause		✓	232
end cause	✓	✓	1300
end cause	✓		1974
end cause		✓	3286

Table 6.6 The coupling of causality/sequence qualifiers

Note The prominence numbers were retrieved using Wikidata query service. Hence, verifying the numbers can be difficult because they change over time. For instance, Table 6.3 shows that the prominence of the replaced by qualifier with start time is 150 514, but if the query is rerun now (a different time), another value (e.g., 150 517 for example) might be obtained. Therefore, a better solution perhaps is downloading a dump and having more accurate values. However, if the values in the tables are regularly updated, a more concrete idea could be obtained about which qualifiers are mostly used, which we plan to do.

6.3 Many-sorted logics for knowledge graphs

6.3.1 Many-sorted logics: preliminaries

A many-sorted logic is a logic, in a first-order language, in which the universe is divided into subsets called sorts. For example, one might divide the universe of discourse into different kinds (called sorts) of animals and plants (Walther, 1985), (Cohn, 1985). AI's main interest in sorted logics has been their computational efficiency; the search space may be much reduced compared with a corresponding formulation in unsorted logic because it is easy to detect that certain inferences cannot lead to a successful proof. Another advantage for sorted logics is that the ontology of the domain is made explicit and the sorted behavior of the non-logical symbols is explicitly declared. This can be valuable not only for documentation purposes but also for knowledge base verification and integrity checking through well-sortedness checking (Cohn, 1992).

Syntax. A many-sorted vocabulary is made of the usual logical symbols and sets of constant symbols, variable symbols, function symbols, and predicate symbols. In addition, it has a set S of sorts and:

- each constant symbol is associated to a sort
- each variable symbol is associated to a sort

- each function symbol has an arity (or profile) of the form $\sigma_1 \times \cdots \times \sigma_n \rightarrow \sigma$, where $n \geq 1$ and $\sigma_1, \ldots, \sigma_n, \sigma$ are sorts
- each predicate symbol has an arity of the form $\sigma_1 \times \cdots \times \sigma_n$, where $n \geq 1$ and $\sigma_1, \ldots, \sigma_n, \sigma$ are sorts

Sorted terms are defined as follows:

- Each variable x of sort σ is a term of sort σ.
- Each constant symbol c of sort σ is a term of sort σ.
- If f is a function symbol of arity $\sigma_1 \times \cdots \times \sigma_n \rightarrow \sigma$ and each t_i is a term of sort σ_i, $i = 1, ..., n$ then $f(t_1, ..., t_n)$ is a term of sort σ.

Atoms are expressions of the form $p(t_1, ..., t_n)$, where p is a predicate symbol of arity $\sigma_1 \times \cdots \times \sigma_n$, and each t_i is a term of sort σ_i, $i = 1, ..., n$.

Formulas are defined as in FOL:

- Each atom is a formula.
- If ϕ is a formula, then $\neg\phi$ formula.
- If ϕ and ψ are formulae, then $\phi \wedge \psi$, $\phi \vee \psi$, $\phi \rightarrow \psi$, $\phi \leftrightarrow \psi$ are formulae.
- If ϕ is a formula, $\sigma \in S$, and x is a variable of sort σ, then $\forall_\sigma x\ \phi$ and $\exists_\sigma x\ \phi$ are formulas.

Semantics. An interpretation I is a map with the following properties:

- Each sort $\sigma \in S$ is mapped to a non-empty set I_σ.
- Each variable x of sort σ is mapped to an element $x^I \in I_\sigma$.
- Each constant symbol c of sort σ is mapped to an element $c^I \in I_\sigma$.
- Each function symbol f of arity $\sigma_1 \times \cdots \times \sigma_n \rightarrow \sigma$ is mapped to a function $f^I : I_{\sigma_1} \times \cdots \times I_{\sigma_n} \rightarrow I_\sigma$.
- Each predicate symbol p of arity $\sigma_1 \times \cdots \times \sigma_n$ is mapped to a set $p^I \subseteq I_{\sigma_1} \times \cdots \times I_{\sigma_n}$. Formulae are interpreted in the same way as in FOL. For quantified formulae, $\forall_\sigma x\ \phi$ and $\exists_\sigma x\ \phi$, the variable assignments range over I_σ.

The above general logic is specialized, for our purpose, into Horn clauses (formulae with only one implication and condition with only conjunctions) and is using only one predicate, equality (with its basic axioms such as reflexivity, symmetry, and transitivity). This will add several interesting properties for reasoning.

To manage the definition domains, axioms can be defined for making precise when a term is defined; this will be done with the "Dom" predicate.

6.3.2 Knowledge graphs' representation in many-sorted logics

Previous research (Marx et al., 2017a), (Patel-Schneider and Martin, 2020) directly examines the attributes-value pairs level of the qualifiers. They handle qualifier values as datatypes such as strings, integer, etc., and propose common reasoning based on the type. In this thesis, we explore another line of research inspired by the common behaviour of each type of qualifier observed in the previous sections. We believe that the problem can be addressed using a many-sorted theory backed by an algebraic specification. The purpose of using the different sorts is to illustrate the different categories of qualifiers that were presented in the previous section. Later on, in chapter 7, using an algebraic specification, the different operations that could be applied on each category of qualifier or sort are shown. And it is subsequently used to build a rule-based system for reasoning on property graphs.

Definition 1. *Let G be a knowledge graph made of a set of statements with attributes-value pairs. A* many-sorted representation *of G is made of*

- *A many-sorted vocabulary Σ with a set of sorts S that contains the sort* resource*, a set F of functions symbols, a set of constants C that comprises all the resource denotations that appear as subject, object, or qualifier value in a statement, and a set of predicate symbols P that is made of all the resource denotations that appear in predicate position in some statement of G. Each $p \in P$ has the same arity* $\mathsf{resource} \times \mathsf{resource} \times s_1 \times \cdots \times s_k$
- *For each statement (s, p, o, A) of G*
  - *an atom*
    $$St(s, p, o, t_1(A), \ldots, t_k(A))$$
    where each $t_i(A)$ is a term of sort s_i, $i = 1, k$.
- *A theory* Spec *over Σ that describes the properties that the (interpretations of the) functions in F are required to satisfy.*

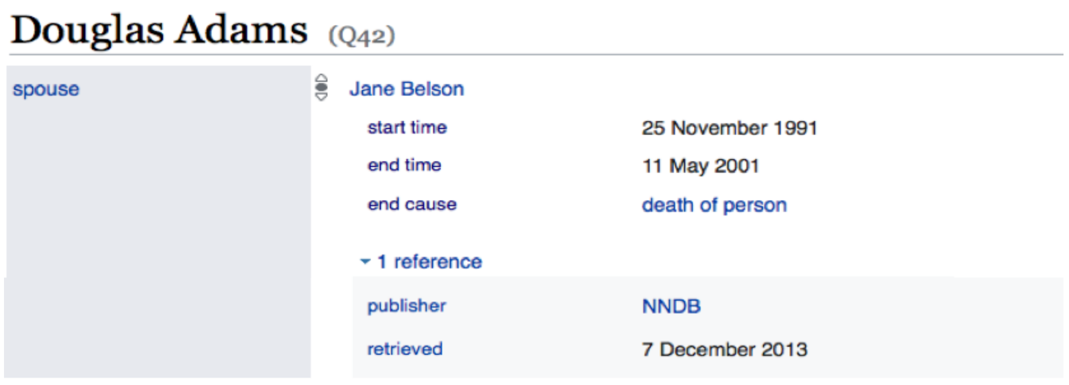
As opposed to the attribute-value pairs in related works, the MSL approach uses high-level objects (the $t_i(A)$'s) derived from the attribute values that correspond to each qualifier category. (Later on, we specify the behavior of each category in a module in an algebraic specification.)

6.3.3 Application on Wikidata

Following our analysis of the Wikdata qualifiers, we propose to represent Wikidata statements using the MSL approach as:

$$St(s, p, o, V, C, S, A, P)$$

, where


Douglas Adams (Q42)

spouse	Jane Belson	
	start time	25 November 1991
	end time	11 May 2001
	end cause	death of person
	1 reference	
	publisher	NNDB
	retrieved	7 December 2013

Fig. 6.4 Validity space (This figure is constructed using screenshots taken from the Wikidata page of Douglas Adam (Wik, c))

- s is the statement's subject.
- p is the statement's predicate.
- o is the statement's value.
- V is an object constructed from the statement's validity context qualifiers.
- C is an object constructed from the statement's causality qualifiers.
- S is an object constructed from the statement's sequence qualifiers.
- A is an object constructed from the statement's annotations qualifiers.
- P is an object constructed from the statement's provenance qualifiers.

Example 1. *From a Wikidata example to MSL:*

Let us take the Wikidata example given in Fig 6.4 where Douglas Adam is the subject of the statement, spouse is the predicate, and Jane Belson the value, and we have a set of qualifiers-values

$$[(\textsf{start time: } \textit{25/11/1991}), (\textsf{end time: } \textit{11/5/2001}), (\textsf{end cause: } \textit{death of person}),$$
$$(\textsf{publisher: } \textit{NNDB}), (\textsf{retreived: } \textit{7/12/2013})]$$

from different categories: time, causality and provenance. A possible MSL representation could be:

$$\begin{aligned}
&\mathsf{St}(\mathsf{Douglas\ Adams}, \mathsf{spouse}, \mathsf{Jane\ Belson},\\
&\quad \mathsf{addTime}(\mathsf{timeInterval}(25/11/1991, 11/5/2001), \mathsf{empty}),\\
&\quad \mathsf{addEndCause}(\text{death of person}, \mathsf{empty}),\\
&\quad \mathsf{empty}, \mathsf{empty},\\
&\quad \mathsf{addPublisher}(\mathit{NNDB}, \mathsf{addRetreived}(7/12/2013), \mathsf{empty}))\\
&)
\end{aligned}$$

To make the object generation of each sort possible, functions are required. For instance, using the validity time qualifiers of the validity context, a function can construct a timeInterval object. In another example, a time interval might have to be created using an instant time and duration. Hence, there should be a function that can do that. The same concept applies to the other sorts. The causality object here will be constructed from an end cause qualifier. A special operation should be available to do that. In another case, a causality object might have to be created from a has cause qualifier. In this example, the functions addTime and addEndCause construct the values of the sorts validity and causality using the qualifier values of the statement. To formalize all these detailed aspects, we designed an algebraic specification that contains a module for each category of qualifier. Each module describes the different functions used to create the sort objects and other operations that will be useful in reasoning.

Chapter 7

The algebraic specification and reasoning rules

Chapter Contents

7.1 Introduction

This chapter presents the algebraic specification that specifies the qualifiers' behaviors when reasoning on Wikidata. We have designed the specification to correspond to the different qualifier categories described in the previous chapter. The user can add other modules, if interested, to a new category of qualifiers that is not already defined.

7.2 Algebraic specification for specifying qualifiers' behaviors

7.2.1 Preliminaries: algebraic specification

Algebraic specifications are usually used to describe abstract data types (ADTs). They have found a straightforward application in object-oriented design to define classes and interfaces. A typical algebraic specification usually consists of two parts: syntax and semantics. The syntactical part specifies the signatures of the operations (or functions), including their names, inputs, outputs, domains, and ranges. The semantic part contains a set of axioms that describe the equivalence relations among the composition of operations.

7.2.2 Algebraic specification for Wikidata

We use an algebraic specification to encode the behavior of each category of qualifiers in reasoning. The specification contains seven main modules that correspond to the specification of the sort operations for different categories of qualifiers:

- a parametrized module for validity contexts specifying the general behavior of a validity context. This module take as parameters two modules: the TimeParam and SpaceParam.
- a module for temporal validity
- a module for spatial validity
- a module for causality
- a module for sequence (sequenceNode module)
- a module for provenance
- a module for annotations

In this chapter, the operations, sorts, and some representative axioms of each module are presented. We use CASL (Astesiano et al., 2002). The complete specification is given in appendix B.

7.2.2.1 The resource module

The resource module contains one sort: the resource sort. It also uses the IRI sort from the IRI module. The generator of this module creates a resource from an IRI. The equal predicate is used to test the equality between two resources. In some cases, resources are not defined, so we use the operation undefined.

```
spec Resource =
sort resource
sort IRI

%% generators
op rsrc : IRI −> resource

pred equal: resource * resource

op undefined : resource

forall r1, r2, r3: resource, i:IRI
. equal(r1, r1)
. equal(r1, r2) /\ equal(r2, r3) => equal(r1, r3)
. equal(r1, r2) => equal(r2, r1)

. not equal(rsrc(i), undefined)
. not equal(undefined , rsrc(i))

end
```

7.2.2.2 The validity context module

This module is a characterization of all of the validity contexts. In the upcoming modules, we show the specific behavior of each validity context (e.g., temporal validity and spatial validity) on its own.

```
spec ValidityContext[TimeParam][SpaceParam] =

sort validityContext

%% Generators
op emptyValidity : validityContext %% empty means valid in every context
op timeValidity : time −> validityContext
op spaceValidity : space -> validityContext
op timespaceValidity : time * space -> validityContext

pred testIntersectValidity : validityContext * validityContext
pred incl : validityContext * validityContext
pred equal : validityContext * validityContext

op union : validityContext * validityContext -> validityContext
op interValidity : validityContext * validityContext -> validityContext
op extractTime : validityContext -> time
op extractSpace : validityContext -> space
op setTime : validityContext * time -> validityContext
op setSpace : validityContext * space -> validityContext


forall c, c1, c2, c3 : validityContext, t, t1 , t2 : time , s, s1, s2 : space
. setTime(emptyValidity, t) = timeValidity(t)
. setTime(timeValidity(t1), t2) = timeValidity(t2)
. setTime(spaceValidity(s), t) = timespaceValidity(t, s)
. setTime(timespaceValidity(t, s), t2) = timespaceValidity(t2, s)

....
```

The parametrized module of validity context has two parameters: the TimeParam and SpaceParam. In the case of this specification, we limit validity contexts to validity time and space. If a user would like to work on more validity contexts, they should specify these contexts in the parameters. The timeParam and spaceParam also have specific modules that define what the concrete modules must guarantee in terms of properties. This allows us to take into account different notions of time and space. A time can be a time interval or time instant, for example.

```
spec TimeParam =

sort time
op undefined : time
op union : time * time -> time
op inter : time * time -> time

pred testIntersect : time * time
pred incl : time * time
pred equal : time * time

forall t1, t2, t3 : time
. equal(t1, t1)
. equal(t1, t2) /\ equal(t2, t3) => equal(t1, t3)
. equal(t1 , t2) => equal(t2, t1)

end
```

```
spec SpaceParam =

sort space

pred equal : space * space
pred incl : space * space

op union : space * space -> space
op inter : space * space -> space



forall s1, s2, s3 : space
. equal(s1, s1)
. equal(s1, s2) /\ equal(s2, s3) => equal(s1, s3)
. equal(s1, s2) => equal(s2, s1)
end
```

The validity context module uses the validity context sort. We find the following four generators:

- a generator that creates an empty validity context
- a generator that creates a validity context from time
- a generator that creates a validity context from space
- a generator that creates a validity context from a validity time and space

We also find general operations such as union, interValidity, etc., as well as operations for specific type of contexts such as the extractTime operation that extracts the time from a validity context and the extractSpace operation that extracts the geographicalRegion, for example, from validity context.

7.2.2.3 The validity time module

We observed in chapter 6 the variety of validity time qualifiers often used. In the following, the operations that can be applied on the validity time qualifiers are discussed.

```
spec validityInstantTime =

NAT

then

sort instantTime
sort duration

%% generators
op undefined : instantTime
op undefinedDuration: duration
op instant : nat -> instantTime

op min : instantTime * instantTime -> instantTime
op max : instantTime * instantTime -> instantTime

pred equal : instantTime * instantTime

op union : instantTime * instantTime -> instantTime
op inter : instantTime * instantTime -> instantTime

pred testIntersect : instantTime * instantTime

pred ___<___ : instantTime * instantTime
pred ___<=___ : instantTime * instantTime

op ___-___ : instantTime * instantTime -> duration
op ___+___ : instantTime * duration -> instantTime

forall x, y : nat , i, i1, i2 : instantTime
. x < y => min(instant(x), instant(y)) = instant(x)
. x < y => max(instant(x), instant(y)) = instant(y)
. y < x => min(instant(x), instant(y)) = instant(y)
. y < x => max(instant(x), instant(y)) = instant(x)
. x = y => max(instant(x), instant(y)) = instant(x)
. x = y => min(instant(x), instant(y)) = instant(x)

...
```

```
spec ValidityTimeInterval =

validityInstantTime

then

sort timeInterval

%% Generators
op undefined : timeInterval
op interval : instantTime * instantTime -> timeInterval
op interval : instantTime * duration -> timeInterval

pred equal : timeInterval * timeInterval
pred disjoint : timeInterval * timeInterval
pred inside : instantTime * timeInterval
pred testIntersectInterval : timeInterval * timeInterval

op union : timeInterval * timeInterval -> timeInterval
op interInterval : timeInterval * timeInterval -> timeInterval

op startTime : timeInterval -> instantTime
op endTime : timeInterval -> instantTime
op duration : timeInterval -> duration

forall t1, t2: timeInterval, x, x1, x2, y1, y2: instantTime, d: duration

. equal(t1, t2) <=> equal(startTime(t1), startTime(t2)) /\ equal(endTime(t1),
endTime(t2))

...
}
```

The validity time is represented through two modules that have to follow the "TimeParam" parameter constraint: the validityTimeInterval and validityInstantTime.

The validityInstantTime has two sorts: instantTime and duration. They are used to represent a particular time instant or a temporal entity with an extent or duration. The module contains generators, operations such as union and predicates such as equal to test the equality of two instantTime, etc.

The ValidityTimeInterval has the timeInterval sort used to represent a temporal entity with an extent or duration. It also imports the sorts of validityInstantTime. Generators (i.e., the operations building the elements of the sorts) are also three in cases:

- undefined
- interval: creates a time interval using a time instant as a beginning and a time instant as an end
- interval: creates a time interval using a time instant as a beginning and a duration to set the end

The module also contains predicates and operations to:

- test if two time intervals are equal (e.g., pred equal)
- test if two time intervals are disjoint (e.g., pred disjoint)
- test if a time instant falls within a time interval (e.g., pred inside)
- test if a time interval falls within another (e.g., op incl)
- test if the intersection of two time intervals is empty or not (e.g., pred testIntersectInterval)
- find the union between two time intervals (e.g., op union)
- find the intersection between two time intervals (e.g., op interInterval)
- obtain the start time of a time interval (e.g., op startTime)
- obtain the end time of a time interval (e.g., op endTime)
- obtain the duration of a time interval (e.g., op duration)

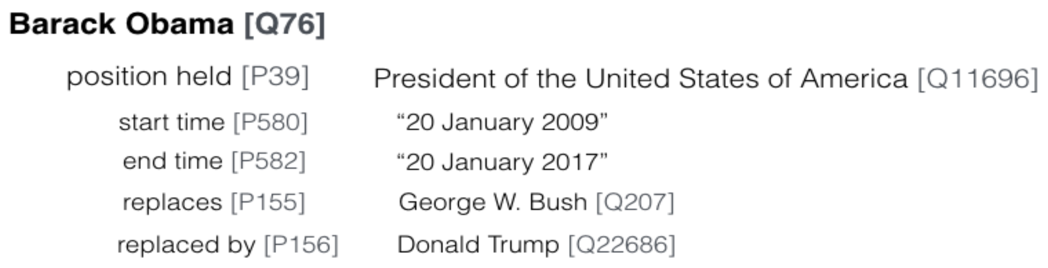
7.2.2.4 The validity space module

The validity space module is used to specify the behavior of the validity space qualifiers in the reasoning. Compared with the validity time qualifiers, validity space qualifiers are less pertinent in Wikidata. We can find the country qualifier used to state that a person was convicted of a crime in a certain country. In this module, city and state are also added as sorts. They can also be further refined depending on the application.

```
spec ValiditySpace =

sort geographicalRegion
sort country
sort city
sort state

%% generators
op geographicalRegion : country -> geographicalRegion
op geographicalRegion : city -> geographicalRegion
op geographicalRegion : state -> geographicalRegion

pred inside : geographicalRegion * geographicalRegion
pred testIntersectSpace : geographicalRegion * geographicalRegion

op interSpace : geographicalRegion * geographicalRegion -> geographicalRegion
op union : geographicalRegion * geographicalRegion -> geographicalRegion



forall s1, s2, s3 : geographicalRegion
. inside(s1, s2) /\ inside(s2, s3) => inside(s1, s3)
. inside(s1, s1)
...
}
```

7.2.2.5 The sequence module

In Wikidata, the frequent use of the sequence qualifiers is observed. A typical example of the behavior of the sequence qualifiers in reasoning is illustrated in the following example in Figure 7.1.



Barack Obama [Q76]

position held [P39]	President of the United States of America [Q11696]
start time [P580]	“20 January 2009”
end time [P582]	“20 January 2017”
replaces [P155]	George W. Bush [Q207]
replaced by [P156]	Donald Trump [Q22686]

Fig. 7.1 Presidency of Barack Obama

Using the replaces qualifier, we should be able to infer (Georges Bush, position held, president of the US) with the qualifiers (end time: 20/1/2009), (replaced by: Barack Obama), and using the replaced by qualifier, we should be able to infer that (Donald Trump, position held, president of the US) with the qualifiers (start time: 20/1/2017), (replaces: Barack Obama).

To make this inference possible, the behavior of the sequence qualifiers should be well formalized. Part of this formalization lies in the sequence module and is later completed with adequate inference rules (section 7.4). In fact, after observing the two previous inferences, it can be seen that the sequence qualifiers are of two types: previous and next. The previous type – illustrated using qualifiers such as replaces, follows – point to the item that was replaced by the subject of the current statement. The next type – illustrated using qualifiers such as replaced by, followed by – point to the item that is replacing the subject. To make it formal, we present the sequenceNode module that contains the sequenceNode sort. It is a pair of two resources. In the following, the operations are discussed. The axioms are given in the appendix.

```
spec SequenceNode =
 Resource
 then

sort sequenceNode

%% generators
op emptySequence : sequenceNode
op seq : resource * resource -> sequenceNode
op seqWithNext : resource -> sequenceNode
op seqWithPrev : resource -> sequenceNode

%% operations
op next : sequenceNode -> resource
op previous : sequenceNode -> resource

pred hasNext : sequenceNode
pred hasPrevious : sequenceNode

forall x, y : resource, n : nat
. next(seq(x, y)) = y
. previous(seq(x, y)) = x


...
```

The module contains four generators. We will use the sequence qualifiers in Figure 7.1 for more clarity :

- the emptySequence that generates an empty sequence
- the seq generator that takes two resources as input (e.g., (George W. Bush and Donald Trump))
- the seqWithNext generator that takes one resource of (implicit) type next (e.g., (, Donald Trump))
- the seqWithPrevious generator that takes one resource of (implicit) type previous (e.g., (George W. Bush,)).

In addition, the module contains operations to output resources of type next or previous, plus predicates that checks if they exist.

7.2.2.6 The causality module

Causalities in Wikidata are used to explain the reason an event exists or ends using has cause and end cause qualifier. Causalities in Wikidata are represented as items. We consider that there can be many causes for an event, so we use a set to represent causality qualifiers' value. Other structures can certainly be used depending on the application.

```
spec Causality =

Resource

then
SET[sort resource]
then
sort causality

%% generators
op emptyCause : causality
op addEndCause : set[resource] * causality -> causality
op addHasCause : set[resource] * causality -> causality

pred equal : causality * causality

op getEndCause : causality -> set[resource]
op getHasCause : causality -> set[resource]
op unionCause : causality * causality -> causality

forall r1 : resource, c1, c2, c3 : causality
. equal(c1, c1)
. equal(c1, c2) /\ equal(c2, c3) => equal(c1, c3)
. equal(c1, c2) => equal(c2, c1)
...


}
```

The causality module uses the causality sort and imports the resource sort. It also uses the set sort of the set module. This module contains three generators. The main idea behind the three generators is related to the nature of causality qualifiers we have in Wikidata. As a reminder, we have the end cause qualifier and has cause qualifier. The end cause qualifier is used to state the cause of an ending event, and the has cause is used to state the cause of an event that is starting or happening. For this reason, we have:

- an emptyCause generator that creates an empty causality object,
- addEndCause generators that add one or more ressources – representing an end cause – to the causality object, and
- addHasCause generators that add one or more ressources – representing a has cause – to the causality object.

Among the operations, we have getters, namely getEndCause and getHasCause, which return the set of causes for each type of qualifiers. Other operations and predicates such as unionCause and equal respectively are also available. Their utilities in the rules are shown later.

7.2.2.7 The provenance module

We use the term "provenance," as explained previously, to refer to the references in Wikidata. Each statement is usually supported by one or more references. And each reference contains one or more attribute-value pairs. For this reason, we create two modules: one that represents a reference and one for provenance.

```
spec R1 =
 sort s1
end

spec Rn =
 sort sn
end

spec Reference = R1 and Rn
then
 SET[sort s1] and SET[sort sn]
then
%% we consider that each property is multi—valued

sort reference

%% generators
op emptyReference : reference

op addP1 : reference * s1 —> reference
%%...
op addPn : reference * sn —> reference

op getP1 : reference —> set[s1]
%%...
op getPn : reference —> set[sn]

pred equal : reference * reference

forall r, r1, r2, a : reference, v1 : s1, vn : sn

%% for each Pi
. getP1(addP1(r, v1)) = add(v1, emptyset) union getP1(r)
...
```

Many properties are used to define references in Wikidata: archive URL, reference URL, etc. They are represented in these modules by properties P1, ..., Pn. To represent all these properties compactly, we use a sort for each reference value (Let s1, ..., sn be the sorts of these properties [it can be a datatype like string, int, real, etc., or the sort resource]). Afterward, we create a generator (AddPi) and a get operation for each sort.

```
spec Provenance = Reference then

sort provenance
sort reference
```

```
%% generators
op emptyProvenance : provenance
op addReference : provenance * reference -> provenance

pred equal : provenance * provenance
pred containedIn : provenance * provenance
pred containsReference : provenance * reference

op union : provenance * provenance -> provenance

forall p1, p2, p3 : provenance, r1, r2, r3 : reference
. containsReference(addReference(p1, r1), r2) <=> (equal(r1, r2) \/
containsReference(p1, r2))

...
```

The module provenance uses two sorts: provenance and reference. It contains two generators: one that creates an empty provenance (emptyProvenance) and the addReference. Other predicates and operations are also available to:

- test the equality of two provenances (e.g., equal).
- test if a provenance is contained in another one (e.g., containedIn)
- test if a reference is contained in a provenance (e.g., containsReference)
- define the union of two provenances (e.g., union)

7.2.2.8 The annotations module

We use the annotations modules to group all the qualifiers that do not belong to any of the previous categories.

```
spec A1 =
 sort s1
end

spec An =
 sort sn
end

spec Annotations = A1 and An
```

```
then
  SET[sort s1] and SET[sort sn]
then
sort annotations

%% Let A1, ..., An be the qualifiers that we consider as annotations
%% Let s1, ..., sn be the sorts of these qualifiers
%% (it can be a datatype like string, int, real, ... or the sort resource)
%% we consider that each attribute may be multi—valued

%%generators
op emptyAnnotations : annotations
op addA1 : annotations * s1 —> annotations
%% . . .
op addAn : annotations * sn —> annotations

op getA1 : annotations —> set[s1]
%% . . .
op getAn : annotations —> set[sn]

forall a : annotations, v1 : s1 %{... }% , vn : sn

. getA1(addA1(a, v1)) = {v1} union getA1(a)
. getA1(emptyAnnotations) = emptyset

...
```

The problem with this module's design is the variety of the annotation types. Their value can range from string, int, real, etc. To have a compact and adequate representation of the annotations (because otherwise, we would have to add an operation to thousands of Wikidata annotations), we consider that we have a sort for each qualifier value. For example, if A_1 (resp. A_n) is the annotation qualifier, let s_1 (resp. s_n) be the qualifier sort. For each sort, there is a generator that adds the annotation of sort s_i to annotations ($1 \leq \text{i} \leq \text{n}$). Getters operations are also available to obtain the annotations values. Since an annotation can have multiple values, we use a set in the operation output signature.

7.3 Formalizing the ontological axioms in Wikidata

As mentioned in the previous chapter, Wikidata contains some "ontological" properties and property constraints. Therefore, an essential part of providing a formal logic for Wikidata is formalizing the ontological concepts in Wikidata with formal rules. In this section, we analyze these properties and propose adequate inference rules. To axiomatize the "ontological" properties, we want to see if they are qualified with qualifiers. Table 7.1 shows how many times the same qualifier appears with the properties instance of, subclass of , and subproperty of. The commonly used validity contexts qualifiers are grouped in blue, the sequence qualifiers in gray, and the causality qualifiers in white. We omitted the provenance and annotations because all the three properties are qualified with annotations qualifiers and generally have a provenance. It can be seen that the instance of property is qualified more than the other properties with the three sorts: validity contexts, causality, and sequence.

Qualifier name	instance of	subclass of	subproperty of
start time	64089	497	0
end time	31646	0	0
valid in period	23853	17	0
point in time	2803	9	0
valid in place	3423	148	0
applies to juridiction	213	0	0
followed by	326096	1522	0
follows	326083	1549	0
replaced by	3189	0	0
replaces	134	1	0
has cause	37	64	0
end cause	54	1	0

Table 7.1 Number of a qualifier appearing with an ontological property (taken from our 114GB [almost complete] downloaded version of the Wikidata 2020-11-09 dump on November 22, 2020)

After providing an MSL view of Wikidata in chapter 6, we show in this section how we transform the ontological concept of Wikidata into inference rules. The rules are given as universally quantified (many-sorted) first-order implications over a n+3-ary (5+3 for Wikidata) predicate St, the $=$ predicate, and the functions defined in the algebraic specification. As formalized in chapter 6, an atom $St(s, p, o, V, C, S, A, P)$ represents a Wikidata statement with the subject s representing the item, p representing the property, the object o representing the property value, V representing the validity context object, C denoting the causality object, S signifying the sequence object, A representing annotations object, and P denoting the provenance object. On the left-hand side (body) of the rule, we find the statements the rule is applied on and the condition on the sort objects. On the right-hand side (head), we have the result of the inference.

Note In the following, we use the term null to refer to the absence of qualifiers for a sort. It is a substitution to the operations `emptyValidity`, `emptyCause`, `emptySequence`, `emptyProvenance`, `emptyAnnnotations` for brevity reasons. For instance, if we have a statement $St(s, p, o, null, C, S, A, P)$. Then, the null on the validity context position means that the statement has no validity context qualifiers.

7.3.1 The *instance of* rule

Taking inspiration from the RDFS9 entailment pattern, we designed the inference rule of the Wikidata instance of (P31) predicate as follows:

$$
\begin{aligned}
&\mathsf{St}(x, \mathsf{instance\ of}, y, V_1, C_1, S_1, A_1, P_1)\\
&\wedge\quad \mathsf{St}(y, \mathsf{subclass\ of}, z, V_2, C_2, S_2, A_2, P_2)\\
&\wedge\quad \mathsf{testIntersectValidity}(V_1, V_2)\\
&\longrightarrow \mathsf{St}(x, \mathsf{instance\ of}, z, \mathsf{interValidity}(V_1, V_2), \mathsf{unionCause}(C_1, C_2), null, null, \mathsf{union}(P_1, P_2))
\end{aligned}
$$

It represents the fact that if x is an instance of y with some validity context V_1, causality C_1, sequence S_1, annotations A_1, and provenance P_1; y is a subclass of z with some validity context V_2, causality C_2, sequence S_2, and provenance P_2; and there is an intersection between V_1 and V_2, then x is the instance of z with:

- a validity context that is the intersection of the two validity contexts statements (i.e., V_1 and V_2). The interValidity used is taken from the module validity context.
- a causality that is the union of the two causalities (i.e., C_1 and C_2). We use the unionCause function of the causality module.
- no particular sequence can be inferred.
- no particular annotations can be inferred.
- a provenance that is the union of the two provenances (i.e., P_1 and P_2). We use the union function of the provenance module.

7.3.2 The *subclass of* rule

Inspired by the RDFS11 entailment pattern, we designed the inference rule of the subclass of (P279) predicate as follows:

$$
\begin{aligned}
&\mathsf{St}(x, \mathsf{subclass\ of}, y, V_1, C_1, S_1, A_1, P_1) \\
&\land \quad \mathsf{St}(y, \mathsf{subclass\ of}, z, V_2, C_2, S_2, A_2, P_2) \\
&\land \quad \mathsf{testIntersectValidity}(V_1, V_2) \\
&\longrightarrow \mathsf{St}(x, \mathsf{subclass\ of}, z, \mathsf{interValidity}(V_1, V_2), \mathsf{unionCause}(C_1, C_2), null, null, \mathsf{union}(P_1, P_2))
\end{aligned}
$$

It represents the fact that if x is a subclass of y with some validity context V_1, causality C_1, sequence S_1, annotations A_1; and provenance P_1; y is a subclass of z with some validity context V_2, causality C_2, sequence S_2, and provenance P_2; and there is an intersection between V_1 and V_2, then, x is a subclass of z with:

- a validity context that is the intersection of the two validity contexts statements (i.e., V_1 and V_2. The interValidity function used is taken from the module validity context.
- a causality that is the union of the two causalities (i.e., C_1 and C_2). We use the unionCause function of the causality module.
- no particular sequence is inferred.
- no particular annotations is inferred.
- a provenance that is the union of the two provenances (i.e., P_1 and P_2). We use the union function of the provenance module.

7.3.3 The *subproperty of* rule

Inspired by the *subproperty of (P1647)* predicate in Wikidata.

$$
\begin{aligned}
&\mathsf{St}(p, \mathsf{subproperty\ of}, q, null, null, null, A_1, P_1) \\
&\land \quad \mathsf{St}(q, \mathsf{subproperty\ of}, r, null, null, null, A_2, P_2) \\
&\longrightarrow \mathsf{St}(p, \mathsf{subproperty\ of}, r, null, null, null, null, \mathsf{union}(P_1, P_2))
\end{aligned}
$$

As seen in Table 7.1, the subproperty is not usually qualified with a validity context, causality, or sequence, so we use *null* in the rule. The rule represents the fact that if p is a subproperty of q with annotations A_1 and provenance P_1 and q is a subclass of r with annotations A_2 and provenance P_2, then p is a subproperty of r with:

- null for the validity context
- null for the causality
- null for the sequence
- no particular annotation

- a provenance that is the union of the two provenances (i.e., P_1 and P_2). We use the union function of the provenance module.

7.4 Formalizing the inference rules of the *sequence* sort

The sequence qualifiers have a particular behavior compared with other qualifiers (in terms of reasoning). Once they are present, they force the generation of new statements. Hence, it is not enough to formalize their behavior in a module. We use inference rules as well.

7.4.1 The *sequence previous* rule

Inspired from the replaces(P1365)/ follows(P155) predicate in Wikidata.

$$\begin{aligned}
&\mathsf{St}(x, p, y, V_1, C_1, S_1, A_1, P_1)\\
&\wedge \quad \mathsf{hasPrevious}(S_1)\\
&\longrightarrow \mathsf{St}(\mathsf{previous}(S_1), p, y,\\
&\mathsf{addTime}(\mathsf{interval}(undefined, \mathsf{startTime}(\mathsf{extractTime}(V_1)))),\\
&null,\\
&\mathsf{seqWithNext(x)},\\
&null,\\
&, P_1\\
&)
\end{aligned}$$

This rule checks if there is a sequence qualifier of type "previous" in the statement. If the statement has a sequence qualifier of type previous, which will be detected using the $\mathsf{hasPrevious}(S_1)$, then in the conclusion of the rule, we generate a new statement where:

- we construct the validity context of this statement using the addTime function of the validity context module. This function takes as a parameter an interval. For that, we use the interval function. The interval function has undefined as the start time because nothing can be inferred for the start time. The end time is equal to the validity context's start time of the original statement V_1; this is achieved by using the $\mathsf{startTime}(\mathsf{extractTime}(V_1))$ function.

- the causality, in this case, is null. We cannot copy the original statement causality nor infer a new one.

- the sequence of the inferred statement is constructed using the subject of the original statement x. Since the qualifier that generated this inference is of type "previous," the sequence of the generated inference will contain a qualifier of type "next," and this is represented using the function seqWithNext(x) of the sequence module.

- we keep the provenance of the original statement P_1.

Many statements in Wikidata do not have the inferences this rule can generate.

7.4.2 The *sequence next* rule

Inspired by the replaced by (P1366)/ followed by(P156) predicate in Wikidata.

$$
\begin{aligned}
&\mathsf{St}(x, p, y, V_1, C_1, S_1, A_1, P_1)\\
&\wedge \quad \mathsf{hasNext}(S_1)\\
&\longrightarrow \mathsf{St}(\mathsf{next}(S_1), p, y,\\
&\mathsf{addTime}(\mathsf{interval}(\mathsf{endTime}(\mathsf{extractTime}(V_1)), \mathsf{undefined}),\\
&null,\\
&\mathsf{seqWithPrevious}(x),\\
&null,\\
&, P_1\\
&)
\end{aligned}
$$

This rule checks if there is a sequence qualifier of type "next" in the statement. If the statement has a sequence qualifier of type "next," which will be detected using the $\mathsf{hasNext}(S_1)$, then the conclusion of the rule is a new statement where:

- the validity context of this statement is constructed using the addTime function of the validity context module. This function takes as parameter an interval. For that, we use the interval function. The interval function has a start time that is equal to the validity context end time of the original statement; this is achieved by using the endTime(extractTime(V_1)) function. The second input of the interval function is undefined because nothing can be inferred for the end date from the original statement.

- the causality, in this case, is null. We cannot copy the original statement causality nor infer a new one.

- the sequence of the inference is constructed using the subject of the original statement. Since the qualifier that generated this inference is of type "next," the sequence of the generated inference will contain a qualifier of type "previous," and this is represented using the function $\mathsf{seqWithPrevious}(x)$ of the sequence module.

- we keep the provenance of the original statement P_1.

This rule applies to properties for which we are sure that the next subject replaces the previous one exactly when the previous' validity time ends. This is the case for s being a president of the US. It is not the case in some other countries[1].

[1]In this case, we can infer that the next president's start time is at least equal to the actual president's end time

7.5 Formalizing the constraints in Wikidata

Constraints are instrumental in Wikidata as they express regularities (patterns of data) that should hold. In practice, they identify potential problems (constraint violations) for interested contributors who can either fix the problem or determine that the particular anomaly is acceptable (Martin and Patel-Schneider, 2020). In this section, we show how to formalize these constraints as MSL formulas and that some of these constraints can give rise to simple (logic programming) inference rules. For example, a constraint of the form $\forall x_1...x_m(B_1 \wedge ... \wedge B_k \rightarrow H)$, where each variable appears in at least one B_i, will correspond to an inference rule $B_1...B_k \longrightarrow H$. This rule can be seen as a correction rule (by completion) that ensures that the constraint is satisfied in the graph. Otherwise, the constraint can be associated to a query that indicates for which values of the variables there is a problem. For example, if there is an invariant $\forall x \exists y(B(x) \rightarrow H(x, y))$, the corresponding query is the open formula $\neg(\exists y(B(x) \rightarrow H(x, y))$. The values of x that make the formula true on the graph are exactly those for which the constraint is not satisfied.

7.5.1 Existing constraints

In the following, we discuss the modeling of the popular Wikidata constraints [2].

Symmetric property (Q18647518) *This constraint[3] specifies that a property is symmetric, and values for that property should have a statement with the same property pointing back to the original item.*

A property p has a symmetric property constraint – (p, property constraint, symmetric constraint) – is qualified usually with this list of five qualifiers: {exception to constraint (P2303), constraint status (P2316), wikibase:rank, wikibase:hasViolationForConstraint, and rdf:type }. The qualifiers in this list belongs to the annotations sort. The MSL formalization of this constraint is as follows:

$$\begin{aligned}
&\forall x, p, y, V_1, C_1, S_1, A_0, A_1, P_1,\\
&\mathsf{St}(p, \text{property constraint}, \text{symmetric property}, \text{null}, \text{null}, \text{null}, A_0, \text{null})\\
&\wedge \quad \mathsf{St}(x, p, y, V_1, C_1, S_1, A_1, P_1)\\
&\longrightarrow \mathsf{St}(y, p, x, V_1, C_1, null, null, P_1)
\end{aligned}$$

The corresponding inference rule is:

$$\begin{aligned}
&\mathsf{St}(p, \text{property constraint}, \text{symmetric property}, \text{null}, \text{null}, \text{null}, A_0, \text{null})\\
&\wedge \quad \mathsf{St}(x, p, y, V_1, C_1, S_1, A_1, P_1)\\
&\longrightarrow \mathsf{St}(y, p, x, V_1, C_1, null, null, P_1)
\end{aligned}$$

[2] https://www.wikidata.org/wiki/Wikidata:WikiProject_property_constraints
[3] https://www.wikidata.org/wiki/Help:Property_constraints_portal/Symmetric

The inference of a statement $\mathsf{St}(x, p, y, V_1, C_1, S_1, A_1, P_1)$ involving a symmetric property is a statement $\mathsf{St}(y, p, x, V_1, C_1, null, null, P_1)$ that preserves the same sorts of the original statement except for the sequence and annotations sort that will be a *null*.

Single-value constraint (Q19474404) *This constraint* [4] *is usually used to specify that a property generally has only a single value for an item.*

As of June 1, 2021, there were 5826 properties using this property constraint. A property p that has a single value constraint – (p, property constraint, single value constraint) – is qualified usually with one out of this list of ten qualifiers: {exception to constraint (P2303), constraint status (P2316), separator (P4155), constraint scope (P4680), constraint clarification (P6607), syntax clarification (P2916), wikibase:rank, wikibase:hasViolationForConstraint, prov:wasDerivedFrom, and rdf:type }. They belong to the annotations and provenance sorts. This constraint gives rise to an inference rule: if two values y1 and y2 are associated to the same subject, then these values are equal.

$$\begin{aligned}&\mathsf{St}(p,\ \text{property constraint, single value constraint, null, null, null},\ A_0,\ P_0)\\ \wedge\quad &\mathsf{St}(x, p, y_1, V_1, C_1, S_1, A_1, P_1) \wedge \mathsf{St}(x, p, y_2, V_2, C_2, S_2, A_2, P_2)\\ \longrightarrow\ &\mathsf{S}(y_1, \mathsf{equal}, y_2, null, null, null, null, \mathsf{union}(P_1, P_2))\end{aligned}$$

There is currently no predicate in Wikidata that represents the equality, so we propose to create a Wikidata predicate called equal.

In some cases, the separator qualifier is used. Multiple values for the two statements (having x as a subject) are allowed under the single-value constraint if they have different values for the separator property. This is typically used to indicate that the constraint must hold in each (temporal, spatial, etc.) context but not globally. For example, suppose the separator is a temporal qualifier such point in time or time period. In that case, the value must remain the same within a period or at a given point in time, i.e., within a temporal validity context. In this case, the corresponding rule is as follows:

$$\begin{aligned}&\mathsf{St}(p,\ \text{property constraint, single value constraint, null, null, null},\ A_0,\ P_0)\\ \wedge\quad &\mathsf{St}(x, p, y_1, V_1, C_1, S_1, A_1, P_1) \wedge \mathsf{St}(x, p, y_2, V_2, C_2, S_2, A_2, P_2)\\ \wedge\quad &\mathsf{testIntersectValidity}(V_1, V_2)\\ \longrightarrow\ &\mathsf{St}(y_1, \mathsf{equal}, y_2, \mathsf{interValidity}(V_1, V_2), null, null, null, \mathsf{union}(P_1, P_2))\end{aligned}$$

Similar rules can be written for other contextual dimensions if the seperator property represents a validity space, a causality, etc.

Distinct values constraint (Q21502410) *This constraint*[5] *is used to specify that the value for this property is likely to be different from all other items.*

[4] https://www.wikidata.org/wiki/Help:Property_constraints_portal/Single_value
[5] https://www.wikidata.org/wiki/Q21502410

A property p that has a single value constraint – (p, property constraint, distinct value constraint) – is qualified usually with one out of this list of 11 qualifiers: {exception to constraint (P2303), constraint status (P2316), separator (P4155), constraint clarification (P6607), syntax clarification (P2916), group by (P2304), item of property constraint (P2305), wikibase:rank, wikibase:hasViolationForConstraint, prov:wasDerivedFrom, and rdf:type }. They belong to the annotations and provenance sorts. This constraint gives rise to an inference rule as follows:

$$\begin{aligned}
&\mathsf{St}(p,\ \text{property constraint, distinct value constraint, null, null, null},\ A_0,\ P_0)\\
&\wedge\quad \mathsf{St}(x_1, p, y, V_1, C_1, S_1, A_1, P_1) \wedge \mathsf{St}(x_2, p, y, V_2, C_2, S_2, A_2, P_2)\\
&\longrightarrow \mathsf{St}(x_1, \mathsf{equal}, x_2, null, null, null, \mathsf{union}(P_1, P_2))
\end{aligned}$$

Format constraint (Q21502404) *This constraint* [6] *is used to specify that values for this property should conform to a certain regular expression pattern.*

A property p that has a format constraint – (p, property constraint, format constraint) – is usually qualified with one out of this list of 11 qualifiers: {exception to constraint (P2303), constraint status (P2316), constraint clarification (P6607), syntax clarification (P2916), format as a regular expression (P1793), constraint scope (P4680), reason for deprecation (P2241), wikibase:rank, wikibase:hasViolationForConstraint, prov:wasDerivedFrom, and rdf:type }. They belong to the annotations and provenance sorts. The corresponding rule is the following:

$$\begin{aligned}
&\forall x, p, y, V1, C1, S1, A0, A1, P1.\\
&\mathsf{St}(p,\ \text{property constraint, formal constraint, null, null, null},\ A_0,\ P_0)\\
&\wedge\quad \mathsf{getFormatAsRegularExpression}(A_0) = regex\\
&\wedge\quad \mathsf{St}(x, p, y, V_1, C_1, S_1, A_1, P_0, P_1)\\
&\longrightarrow \mathsf{Regex.matches}(y, regex)
\end{aligned}$$

In this case, we cannot create an inference rule (the rule head is not a statement). If y does not match *regex*, we can just report the problem.

Type constraint (Q21503250) *This constraint* [7] *is used to specify that items with the specified property should have the given type.*

This property is similar to the rdfs:domain property in RDFS. The qualifiers that can qualify this constraint are 13: {exception to constraint (P2303), constraint clarification (P6607), group by (P2304), class (P2308), relation (P2309), constraint status (P2316), constraint scope (P4680), item of property constraint (P2305), property scope (P5314), wikibase:rank, wikibase:hasViolationForConstraint, prov:wasDerivedFrom, and rdf:type)}. They belong to the annotations and provenance sorts. The qualifiers class (P2308) and relation (P2309) play an important role in the formalization of this constraint. They indicate that relation must hold

[6] https://www.wikidata.org/wiki/Q21502404

[7] https://www.wikidata.org/wiki/Help:Property_constraints_portal/Subject_class

between the subject of a statement with this property and one of the given classes. The relation (P2309) is either instance of, subclass of, or instance or sub-class of. Note that the identifier used for instance of is wd:Q21503252, which is surprisingly different from (P31), the identifier of the conceptual instance of property. Similarly, for subclass of, the identifier used is "wd:Q21514624," which is completely different from the identifier of the conceptual Wikidata subclass of (P279). In the following, we formalize this Wikidata constraint only when the relation (P2309) is equal to instance of (wd:Q21503252) or subclass of(wd:Q2151462). We do not formalize the instance or sub-class of because it is unclear what to do with it.

$\forall x, p, y, V_1, C_1, S_1, A_0, A_1, P_0, P_1, \exists T$
$\mathsf{St}(p, \text{ property constraint, type constraint, null, null, null, } A_0, P_0)$
$\wedge \quad \mathsf{St}(x, p, y, V_1, C_1, S_1, A_1, P_1)$
$\wedge \quad \mathsf{contains}(\mathsf{getRelation}(A_0), \text{"instance of"})$
$\longrightarrow \mathsf{St}(x, \text{instance of}, T, null, null, null, null, \mathsf{union}(P_0, P_1)) \wedge \mathsf{contains}(\mathsf{getClass}(A_0), T)$

Another form of the MSL formalization is possible when the getRelation predicate is subclass of.

$\forall x, p, y, V_1, C_1, S_1, A_0, A_1, P_0, P_1, \exists T$
$\mathsf{St}(p, \text{ property constraint, type constraint, null, null, null, } A_0, P_0)$
$\wedge \quad \mathsf{St}(x, p, y, V_1, C_1, S_1, A_1, P_1)$
$\wedge \quad \mathsf{contains}(\mathsf{getRelation}(A_0), \text{"subclass of"})$
$\longrightarrow \mathsf{St}(x, \text{subclass of}, T, null, null, null, null, \mathsf{union}(P_0, P_1)) \wedge \mathsf{contains}(\mathsf{getClass}(A_0), T)$

Value type constraint (Q21510865) *The referenced item should be a subclass or instance of the given type.*

This property is similar to the rdfs:range property in RDFS. The qualifiers that can qualify this constraint are 10: {exception to constraint (P2303), constraint clarification (P6607), group by (P2304), class (P2308), relation (P2309), constraint status (P2316), wikibase:rank, wikibase:hasViolationForConstraint, prov:wasDerivedFrom, and rdf:type)}. They belong to the annotations and provenance sorts. The MSL formalization of this constraints takes two forms:

$\forall x, p, y, V_1, C_1, S_1, A_0, A_1, P_0, P_1, \exists T$
$\mathsf{St}(p, \text{ property constraint, value type constraint, null, null, null, } A_0, P_0)$
$\wedge \quad \mathsf{St}(x, p, y, V_1, C_1, S_1, A_1, P_1)$
$\wedge \quad \mathsf{contains}(\mathsf{getRelation}(\mathsf{A}_0), \text{"instance of"})$
$\longrightarrow \mathsf{St}(y, \text{instance of}, T, null, null, null, null, \mathsf{union}(P_0, P_1)) \wedge \mathsf{contains}(\mathsf{getClass}(\mathsf{A}_0), \mathsf{T})$

$\forall x, p, y, V_1, C_1, S_1, A_0, A_1, P_0, P_1, \exists T$
$\mathsf{St}(p,$ property constraint, value type constraint, null, null, null, A_0, $P_0)$
$\wedge\ \ \mathsf{St}(x, p, y, V_1, C_1, S_1, A_1, P_1)$
$\wedge\ \ \mathsf{contains}(\mathsf{getRelation}(\mathsf{A_0}), \text{"subclass of"}) = \mathsf{true}$
$\longrightarrow \mathsf{St}(y, \text{subclass of}, T, null, null, null, null, \mathsf{union}(P_0, P_1)) \wedge \mathsf{contains}(\mathsf{getClass}(\mathsf{A_0}), \mathsf{T})$

7.5.2 Proposed Constraints

We propose to add the following rule to the Wikidata entailment regime.

Transitive property *This constraint is to be used to declare properties to be transitive.* The MSL formalization is as follows:

$\forall x, p, y, V_1, V_2, C_1, C_2, S_1, S_2, A_0, A_1, A_2, P_0, P_1, P_1$
$\mathsf{St}(p,$ property constraint, transitive constraint, null, null, null, A_0, $P_0)$
$\wedge\ \ \mathsf{St}(x, p, y, V_1, C_1, S_1, A_1, P_1)$
$\wedge\ \ \mathsf{St}(y, p, z, V_2, C_2, S_2, A_2, P_2)$
$\wedge\ \ \mathsf{testIntersectValidity}(V_1, V_2)$
$\longrightarrow \mathsf{St}(x, p, z, \mathsf{interValidity}(V_1, V_2), \mathsf{unionCause}(C_1, C_2), \mathsf{null}, \mathsf{null}, \mathsf{union}(P_1, P_2))$

The corresponding inference rule is the following:

$\mathsf{St}(p,$ property constraint, transitive constraint, null, null, null, A_0, $P_0)$
$\wedge\ \ \mathsf{St}(x, p, y, V_1, C_1, S_1, A_1, P_1)$
$\wedge\ \ \mathsf{St}(y, p, z, V_2, C_2, S_2, A_2, P_2)$
$\wedge\ \ \mathsf{testIntersectValidity}(V_1, V_2)$
$\longrightarrow \mathsf{St}(x, p, z, \mathsf{interValidity}(V_1, V_2), \mathsf{unionCause}(C_1, C_2), \mathsf{null}, \mathsf{null}, \mathsf{union}(P_1, P_2))$

If two statements $\mathsf{St}(x, p, y, V_1, C_1, S_1, A_1, P_1)$ and $\mathsf{St}(y, p, z, V_2, C_2, S_2, A_2, P_2)$ have p as a transitive property and there is an intersection between the two validity contexts, then the inferred statement $\mathsf{St}(x, p, z, \mathsf{interValidity}(V_1, V_2)$, $\mathsf{unionCause}(C_1, C_2)$, null, null, $\mathsf{union}(P_1, P_2))$ holds within the intersection of the two validity contexts (V_1, V_2), the union of the causality (C_1, C_2), no particular sequence and annotations, the union of the provenances (P_1, P_2).

7.6 Formalizing the domain rules in Wikidata

Domain-specific rules can also be written in the same way we wrote the previous rules. In this section, we show two rules inspired by two Wikidata events:

7.6.1 Nationalities

A famous problem about challenging reasoning on Wikidata qualifers – proposed by the Wikidata reasoning group[8] – is about the nationalities. More concretely, in Wikidata, we find a property called country of citizenship (P27) used for persons. The value of this property is a country, and the property is usually qualified with qualifiers including validity time and causality. The problem is that sometimes, in history, countries change; they have new names, are divided into two or more countries, or are perhaps reunited with others. We propose the following rule that enriches the knowledge graph with new statements while providing a solution to the problem of qualifiers:

$$\begin{aligned}
&\mathsf{St}(x_1, \mathsf{countryOfCitizenship}, y_1, V_1, C_1, S_1, A_1, P_1)\\
&\wedge \quad \mathsf{St}(y_1, \mathsf{replaced\ by}, y_2, V_2, C_2, S_2, A_2, P_2)\\
&\wedge \quad \mathsf{endTime}(\mathsf{extractTime}(V_1)) = \mathsf{getTime}(A_2)\\
&\longrightarrow \mathsf{St}(x1, \mathsf{country\ of\ Citizenship}, y2,\\
&\qquad \mathsf{setTime}(\mathsf{empty}, \mathsf{interval}(\mathsf{getTime}(A_2), \mathsf{null})),\\
&\qquad \mathsf{addHasCause}(\mathsf{getEndCause}(C_1), \mathsf{empty}),\\
&\qquad \mathsf{null},\\
&\qquad \mathsf{null},\\
&\qquad \mathsf{union}(P_1, P_2)\\
&)
\end{aligned}$$

This rule is applied if the knowledge graph contains a statement $\mathsf{St}(x_1$, countryOfCitizenship, y_1, V_1, C_1, S_1, A_1, $P_1)$, which means a person x who has a country of citizenship y for some contexts. The status of the country y_1 changed during time to reach y_2. In this case, if the end time of the validity time of the citizenship of x_1 in y_1 (e.g., $\mathsf{endTime}(\mathsf{extractTime}(V_1))$) is equal to the time the change happened to the country (e.g., $\mathsf{getTime}(\mathsf{A}_2)$), then we can infer the citizenship of x to y_2 with:

- a validity context containing only a validity time. The latter has a starting time equal to the time when the country's change happened $\mathsf{setTime}(\mathit{empty}, \mathsf{interval}(\mathsf{getTime}(A_2), \mathit{null}))$
- the causality of the statement is the same as the original statement. It is added to the causality set using the addHascause operation.
- no particular inference for sequence object
- no particular inference for annotations
- the union of the provenances

[8] https://www.wikidata.org/wiki/Wikidata:WikiProject_Reasoning/Use_cases#Nationalities

Applying this rule to countries that change names with time is best. However, the inferences may not be valid if the country splits with time. For example, if West Germany and East Germany replace Germany, the new nationality of someone depends on where they live (geographically). Thus, the rule applies only when there is no territorial change, which is more complex to check.

7.6.2 Enrich fluent properties with causality

We find many fluent properties in Wikidata. These are properties that are valid for a limited time. Usually, they are used to define a certain event like marriage. For instance, if two persons are married and one of them dies, then the marriage statement should be enriched with an end cause property, which is *death*. More generally, some properties hold only between two living entities. If one of them dies, the property ceases to hold precisely at that moment, and the end cause is *death*.

$$\begin{aligned}
&\mathsf{St}(x_1, p, y_1, V_1, C_1, S_1, A_1, P_1)\\
&\wedge \quad \mathsf{St}(y_1, \mathsf{dateofdeath}, date, V_2, C_2, S_2, A_2, P_2)\\
&\longrightarrow \mathsf{St}(x_1, p, y_1, \mathsf{setTime}(V_1, \mathsf{interval}(\mathsf{startTime}(\mathsf{extractTime}(V_1)), date)),\\
&\quad\ \mathsf{addEndCause}(\mathsf{death}, C_1), S_1, A_1, \mathsf{union}(P_1, P_2))
\end{aligned}$$

7.7 Discussion

The MSL, algebraic specification, and the inference rules we presented in this thesis answer the Wikidata reasoning group questions [9]. Writing rules in MSL might require an effort to define the sorts and formally specify their operations. However, we gain a more abstract and therefore simpler vision of the knowledge graph. We no longer need to go down to the level of qualifiers; we work on validity time and not on start/end time qualifiers. The introduction of operations on the sorts also makes it possible to derive new values through complex calculations that would be very difficult to express in a pure logical formalism. All this requires some implementation work, for instance, to map the values of qualifiers to the values of sorts, as we will see in the next chapter.

7.8 Generality of the proposed approach

We base our approach on a study/classification of the qualifiers, which will lead to the definition of the sorts. The sorts identified for Wikidata are very general, and there is a good chance that they are found (in part) in most graphs; but it is always possible to define new ones. An analysis of the qualifiers of the graph is a prerequisite for determining the relevant categories. Once the categories are determined, so are the sorts. Similarly, we can find the

[9] https://www.wikidata.org/wiki/Wikidata:WikiProject_Reasoning/Use_cases

ontological rules and other constraints presented for Wikidata in other graphs, and it is easy to adapt them. In the case of Wikidata, the subproperty was not qualified with any of the formalized sorts. If in another graph, it is, this should be well represented in the rule. We can also create rules to define an "entailment regime" specific to the domain of the graph in question. In all cases, observing the behaviors of qualifiers from different categories together is necessary because it highly influences the rules.

Chapter 8

Implementation

Chapter Contents

8.1 Implementation goal

This chapter aims to show the usability of the MSL approach. We describe the steps to follow in order to implement the MSL reasoning on a KG. We illustrate the description with the Wikidata use case.

8.2 MSL in practice: methodology

Practical reasoning with MSL is a two-level approach that starts with the setup of the sorts and their algebraic specification and ends with implementation choices (related to existing tools). Each step can be decomposed into the following tasks.

Sorts and specification

1. Identify the relevant qualifier categories. They are application-dependant.
2. Specify the sorts corresponding to the categories, i.e., define the operations on the sorts (by their signatures and logical axioms). Define auxiliary sorts if necessary, for instance, to represent categories that have a complex structure.
3. Specify the inference rules in MSL using the sorts and sort operations.

Implementation

1. For each sort
   - find all the qualifiers that are used in the knowledge graph. For instance, for the causality, we find “has cause” and “end cause” in Wikidata.
   - define a representation for the sorts
   - define a mapping from sets of (qualifier, value) pairs to sort values. Note that in practice, this mapping may be very complex because 1) a dimension, such as validity, may have a complex structure (time + space) and 2) the same context may be expressed in different ways (time interval = start + end or start + duration).
2. Implement the mappings from qualifier values to sort values.
3. Implement the sort operations (typically in Javascript, SPARQL, ...).
4. Express the MSL inference rules into executable expressions in an SW language such as SPARQL, SHACL, or SPARQL + Javascript functions. These expressions must use the sort operations.

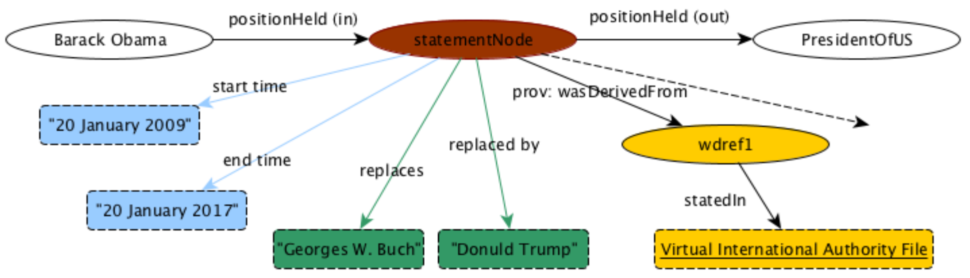


Fig. 8.1 The positionHeld statement of Barack Obama in RDF

8.3 Implementation choices

The implementation of contextual reasoning reveals the lack of tools for context-based graph management. Therefore, we conducted the work in two ways.

1. We did a first partial implementation [1] using SHACL + Javascript in TopBraidComposer[2]. We started by modifying the Wikidata data model to enable the inclusion of the sorts. Fig 8.1 shows the representation of some qualifiers from the Barack Obama positionHeld's statement. It can be seen that all the qualifiers are attached directly to the statement node. However, a difference can be noted with the modeling of the references. There is always a middle node connecting the statement node to the reference qualifiers. This is because each statement can have multiple references. To make this data model adequate with the MSL, a representation of sorts must be introduced. In the MSL representation of Wikidata, we find sorts for validity contexts that refer to validity time, validity space, etc. There are also sorts for annotations and sequences. We added a layer of nodes representing these sorts, as illustrated in Fig 8.2. These auxiliary nodes are located between the statement node and the qualifier. They represent the sorts of each qualifier. Examples of the rules and the datasets can be found in Gitlab. The problem with this approach is that the functions of the specification, which we wrote in Javascript, deal with graphs and return nodes/graphs. The communication between SHACL and JS is then based on graphs, which are not easily handled in Topbraid.

[1]the first implementation is a collaboration work with my colleague MS. Ashley Caselli
[2]https://www.topquadrant.com/products/topbraid-composer

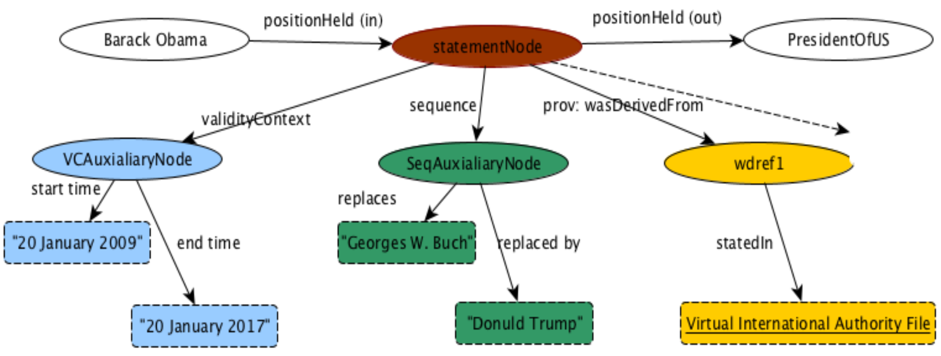


Fig. 8.2 Adding a middle layer for qualifiers representation

2. Therefore, we decided to implement the work using SPARQL + Javascript in GraphDb [3]. In this case, we represented the sorts as literals using JSON because it is easier to work with in Graphdb [4]. In the following, this implementation is explained in detail.

The implementation of all the rules was done using the second approach. However, the translation from the JSON approach into the model proposed in 8.2 is straightforward. The code can be found in http://ke.unige.ch/wikidata/.

8.4 Sort representation and mapping

The first implementation step involves constructing the sort values for the different qualifier categories. We chose the literal approach to represent the sorts' value because literal values are much easier to handle in the auxiliary language (JS, Java, ...). This is because implementing the sort operations does not involve the creation of new nodes and links in the datagraph. A JSON string represents the value of each sort. For a statement S and each qualifier category, it consists of mapping the Wikidata subgraph that represents a value of this category for s to a JSON value (object, array, number, string, or one of false null true). This JSON value denotes a sort value. For example, the causality of a statement S is represented by the subgraph made of all the triples (S, p, o), where p is either pq:P828 (has cause) or pq:P1534 (end cause). A possible mapping can be: If

$$(S, \mathsf{pq{:}P828}, h_1), \ldots, (S, \mathsf{pq{:}P828}, h_k)$$

[3]https://graphdb.ontotext.com/

[4]wikidata is also internally stored in a document-centric form using JSON.

are all the triples with subject S and predicate pq:P828 and a non-blank object, and

$$(S, \text{pq:P1534}, e_1), ..., (S, \text{pq:P1534}, e_n)$$

are all the triples with subject S and predicate pq:P1534 and a non-blank object, then the corresponding JSON object j is:

$$\{\text{"hasCause"}: [h'_1 , h'_2 , ..., h'_k], \text{"endCause"}: [e'_1 , e'_2 , ..., e'_n]\}$$

where h'_i (resp. e'_i) is a string representation of the node h_i (resp. e_i). This value is finally added to the graph we are working on as a triple (S, pq:causalityJ , j). This mapping can be implemented in different ways:

1. with a (python) script that reads all the statements of interest in the graph, computes the representation, and stores it in the 'context' graph.
2. with a SPARQL INSERT statement, if the value computation is algorithmically simple.
3. with a SPARQL INSERT + calls to SPARQL/SHACL functions that correspond to the constructor functions of the target sort.

We implemented this mapping with a Python script (sort-maker.py).

8.5 Sort operations

The sort operations were implemented in Javascript. This is because Graphdb, the triple store we chose, allows users to define and execute JavaScript code, further enhancing data manipulation with SPARQL. Since the sort values are JSON values stored in strings (literals), the functions typically follow the pattern:

```
    function op(string_p1, string_p2, ...){
      var p1 = JSON.parse(string_p1)
      var p2 = JSON.parse(string_p2)
      ...
      // do the op computation
      result = ...
      string_result = JSON.stringify(result)
      return string_result
}
```

This is an extract of the causality module:

```
PREFIX extfn: <http://www.ontotext.com/js#>
```

```
    INSERT DATA {
        [] <http://www.ontotext.com/js#register>
'''

///// module Causality

function emptyCause(){
 var ec = {hasCause: [], endCause: []}
 var jec = JSON.stringify(ec)
 return jec

}

function getEndCause(c){
 var ec = (JSON.parse(c)).endCause
 var jec = JSON.stringify(ec)
 return org.eclipse.rdf4j.model.impl.SimpleValueFactory.getInstance().
 createLiteral(jec);
}

function getHasCause(c){
    var hc = (JSON.parse(c)).hasCause
    var jhc = JSON.stringify(hc)
    return org.eclipse.rdf4j.model.impl.SimpleValueFactory.getInstance().
    createLiteral(jhc);
}

function unionCause(c1, c2){
 var ec1 = (JSON.parse(c1)).endCause
 var ec2 = (JSON.parse(c2)).endCause
 var hc1 = (JSON.parse(c1)).hasCause
 var hc2 = (JSON.parse(c2)).hasCause
 var jc1uc2 = JSON.stringify({endCause: set_union(ec1, ec2),
 hasCause: set_union(hc1, hc2)})
 return org.eclipse.rdf4j.model.impl.SimpleValueFactory.getInstance()
 .createLiteral(jc1uc2)
}

...
'''
}
```

The module develops different types of operations to create, retreive or modify a cause .

8.6 MSL rules

8.6.1 SPARQL Construct for the rule implementation

Consider the rule

```
St(p, <<property_constraint>>, <<transitive_constraint>>, null, null, null,
A0, P0)
^ St(x, p, y, V1, C1, S1, A1, P1)
^ St(y, p, z, V2, C2, S2, A2, P2)
^ testIntersect(V1, V2)
->
St(x, p, z, interValidity(V1, V2), unionCause(C1, C2), null, null,
union(P1, P2))
```

where `<<property_constraint>>` is the Wikidata property `p:P2302` and `<<transitive_constraint>>` is the item `wd:Q18647515`.

It can be implemented on the contextualized Wikidata RDF dump graph as a CONSTRUCT statement:

```
CONSTRUCT {?x ?p_p _:newstatement. _:newstatement ?ps_p ?z;
  pq:validityJ ?hvc; pq:causalityJ ?hcc; pq:provenanceJ ?hpc] }

WHERE { ?p p:P2302 ?s0 . ?s0 ps:P2302 wd:Q18647515 .
  ?x ?p_p ?s1. ?s1 ?ps_p ?y;
     pq:validityJ ?V1; pq:causalityJ ?C1; pq:provenanceJ ?P1 .
  ?y ?p_p ?s2. ?s2 ?ps_p ?z;
    pq:validityJ ?V2; pq:causalityJ ?C2; pq:provenanceJ ?P2 .

        FILTER(strafter(str(?p),"P")=strafter(str(?p_p),"P") &&
               strafter(str(?p),"P")=strafter(str(?ps_p),"P")
        ).
        FILTER(ext:testIntersectValidity(?V1, ?V2))

        BIND(ext:interValidity(?V1, ?V2) as ?hvc)
        BIND(ext:unionCause(?C1, ?C2) as ?hcc)
        BIND(ext:union(?P1, ?P2) as ?hpc)
        }
```

This example provides an intuition on how a rule compiler could transform MSL rules into executable code, in SPARQL, in a very straightforward manner. To generalize, a rule of the form:

$$\begin{aligned}
&\mathsf{St}(s_1, p_1, o_1, V_1, C_1, S_1, A_1, P_1)\\
&\wedge \quad \mathsf{St}(s_2, p_2, o_2, V_2, C_2, S_2, A_2, P_2)\\
&\wedge \quad ...\\
&\wedge \quad pred_1(params_1) \wedge \quad pred_2(params_2)\\
&\wedge \quad ...\\
&\longrightarrow \mathsf{St}(s_h, p_h, o_h, V_h, C_h, S_h, A_h, P_h)
\end{aligned}$$

translates to a SPARQL statement of the following form:

```
CONSTRUCT {
 $s_h$ ?pp_h _:newstatement . _:newstatement ?psp_h $o_h$ ;
   pq:validity ?hvc ; pq:causality ?hcc ; ... }

WHERE {
  $s_1$ $p(p_1)$ ?statement_1 . ?statement_1 $ps(p_1)$ o_1 ;
    pq:validity $V_1$ ... .
  $s_2$ $p(p_2)$ ?statement_2 . ?statement_2 $ps(p_2)$ $o_2$ ;
    pq:validity $V_2$ ... .
  ...
  FILTER ($pred_1(params_1)$)
  FILTER ($pred_2(params_2)$)
  ...
  FILTER ($removePrefix(p(p_1)) = removePrefix(ps(p_1))$)
  FILTER ($removePrefix(p(p_2)) = removePrefix(ps(p_2))$)
  ...
  BIND (expression for $V_h$ as ?hvc)
  BIND (expression for $C_h$ as ?hcc)
  ...
  BIND ($p(p_h)$ as ?pp_h)
  BIND ($ps(p_h)$ as ?psp_h)
}
```

In the RDF representation of Wikidata, the same entity may have different IRIs, depending on its role in a statement representation. For instance, the property P19 (place of birth) is represented by p:P19[5] when it connects the subject of a statement to the statement node and by ps:P19[6] when it connects a statement node to its object. It can also be represented as wd:P19[7] when it is the subject of a statement. Therefore, if the same variable appears in different positions within a rule, it must be represented by several variables in the SPARQL expression (`?p`, `?p_p`, `?ps_p` in the above example), and a filter must ensure that they have the same value after their prefixes. In the translation process, the $p()$ and $ps()$ compile-time functions replace the prefix of their argument by `p:` and `ps:` respectively.

8.6.2 Termination for the rules

Some of the preceding rules can create new facts that can be reused by the same rule to generate additional facts. Hence, a complete implementation must store the inferred triples in the graph or in a dataset graph and iterate up to saturation. Moreover, since we represent the St(...) predicate in the rule's head by a subgraph having a new blank node, the same St(...) statement (with the same s, p, o, and sort values) could be inferred multiple times. This could lead to infinite loops in the inference process. Therefore, before creating a new St(...), the inference system must check that an equivalent statement representation does not already exist in the graph. We can do this by incorporating an additional filter in the SPARQL CONSTRUCT for the rule.

```
CONSTRUCT {?x ?p_p _:newstatement. _newstatement ?ps_p ?z;
  pq:validityJ ?hvc;  pq:causalityJ ?hcc;  pq:provenanceJ ?hpc]}

WHERE { ...
        # avoid multiple inferences of equivalent statements
        FILTER NOT EXISTS
          {?x ?p_p [ ?ps_p ?z;
             pq:validityJ ?hvc_ ; pq:causalityJ ?hvc_ ; pq:provenanceJ ?hpc_]
           FILTER( ext:validity_equal(?hvc, ?hvc_) &&
                   ext:causality_equal(?hvc, ?hvc_) &&
                   ext:provenance_equal(?hpc, ?hpc_))}
}
```

The MSL rules generate new facts of the form $\mathsf{St}(s, p, o, V, C, S, P, A)$, where s, p, o are individuals that exist in the original KG and V, C, S, P, A are values of the sorts validity,

[5]where p: <http://www.wikidata.org/prop/>
[6]where ps: <http://www.wikidata.org/prop/statement/>
[7]where wd: <http://www.wikidata.org/entity/>

causality, etc., that can be generated by functions of these sorts. Therefore, the rules can potentially generate an infinite number of new St(...) facts. We hypothesize that the structure of the rules and operations is such that this should not happen. For instance, for the validity sort, the newly generated values are (roughly speaking) intersections, and we cannot generate an infinite number of intersections starting from a finite set of constants (the start-time, end-time, etc., that exist in the KG). A similar argument should work for C, S, and P. The problem could arise with A. On the other hand, if we allow arbitrary functions to combine annotation values (multiplications, divisions, etc.), the process could undoubtedly generate an infinite number of new annotations.

8.7 Use cases description and observations

In this section, we show the benefits of applying the inferences rules and constraints on Wikidata. We chose a use case per example.

8.7.1 Inference rules

Instance of The use of instance of in Wikidata is not always appropriate. In chapter 7, we saw that from two statements $St(x, instanceof, z, V_1, C_1, S_1, A_1, P_1)$ and $St(y, subclassof, z, V_2, C_2, S_2, A_2, P_2)$, we must infer:

$$St(x, instanceof, z, interValidity(V_1, V_2), unionCausality(C_1, C_2), null, null, union(P_1, P_2))$$

Some examples in Wikidata do not follow this theory. For instance, we find the following two statements related to the Dioceze of Fez(Q27779890):

(Dioceze of Fez, instance of, titular see)
[start time : *1496*, end time : *1730*].

(titular see, subclass of, dioceze of the catholic church)

And the following statement, which looks like an inference result:

(Dioceze of Fez, instance of, dioceze of the catholic church)
[start time : *1225*, end time : *1237*].

This statement may be true (considering open-world semantics), but it is not a consequence of the former two statements. This is because its validity interval does not contain nor intersect the validity interval of the statement one can infer with the rule (i.e., no intersection or inclusion between [1496, 1730] and [1225, 1237]). Therefore, it must be complemented with the inferred statement:

(Dioceze of Fez, instance of, dioceze of the catholic church)
[start time : *1496*, end time : *1730*].

We have found 227 similar cases in which the instance of statement must be complemented with the intersection of the two original statements validity tine.

Subproperty of This property is not often used in Wikidata compared with the instance of and subclass of properties. It is also qualified only with the provenance. The Wikidata dump we are using contains 783 statements with subproperty of with 282 statement couples (?p subproperty of ?q)(?q subproperty ?K) not having their inferences.

Sequence previous Applying the sequence previous inference rule on the position held(P39) property, we can infer 186 869 statements from 195 367 statements annotated with a replaces qualifier. However, on top of generating inferences, we can use this inference to check the validation of the data. For instance, the position held (P39) property is qualified with sequence qualifiers (replaces and replaced by) and validity time (start time and end time). For example, we find the following statement:

(Jacques Chirac, position held, member of the French National Assembly)
[start time : *11 July 1968*, replaces : *henri Belcour*]

Using this statement, we can infer a new statement:

(Henri Belcour, position held, member of the French National Assembly)
[end time : *11 July 1968*, replaced by : *Jacques Chirac*]

which is actually present in Wikidata but in a slightly different form, where the time is *May 30, 1968*.

(henri Belcour, position held, member of the French National Assembly)
[end time : *30 May 1968*, replaced by : *Jacques Chirac*]

The sequence inferences can help validate the data in Wikidata if they are already present such as in this example, or complete them if they are not.

In addition, as mentioned in the previous chapter, the sequence rules generate rigid inferences in terms of time. A better way to deal with that is to use earliest date (P1319) and latest date (P1326) instead of start time and end time respectively. To make this possible, we should update the algebraic specification to deal with uncertain time, which we will do in future works.

Sequence next Running the rule on the position held(P39) property, we can infer 162 124 statements from 169 781 statements. However, similar to sequence previous, we can use this inference to check the validation of the data. For instance, Wikidata uses a position held (P39) property qualified with sequence qualifiers (replaces and replaced by) and validity time (start time and end time). For example, from the following statement:

$$\begin{gathered}(\text{Walerly Slawek, position held, Prime Minister of Poland})\\ [\text{end time}: \textit{23 August 1930}, \quad \text{replaced by}: \textit{Jozef Pilsudski}]\end{gathered}$$

we can infer:

$$\begin{gathered}(\text{Jozef Pilsudski, position held, Prime Minister of Poland})\\ [\text{start time}: \textit{23 August 1930}, \quad \text{replaces}: \textit{Walerly Slawek}]\end{gathered}$$

while Wikidata contains:

$$\begin{gathered}(\text{Jozef Pilsudski, position held, Prime Minister of Poland})\\ [\text{start time}: \textit{15 August 1930}, \quad \text{replaces}: \textit{Walerly Slawek}]\end{gathered}$$

We can report this mismatch to potential contributors to check if it is alright.

8.7.2 Constraints

As mentioned previously, constraints are instrumental in Wikidata as they express regularities (patterns of data) that should hold. In practice, they identify potential problems (constraint violations) for interested contributors who can either fix the problem or determine that the particular anomaly is acceptable (Martin and Patel-Schneider, 2020). We use SPARQL queries to find constraint violations.

Symmetry constraint: *This constraint specifies that a property is symmetric, and values for that property should have a statement with the same property pointing back to the original item.*

We run the symmetry constraint on the spouse property (P26). From 721 491 statement annotated with time and causality, we find 13 972 statements that violate the constraint. For instance, we find (Danilo Medina (Q57429), spouse (P26), Candida Montilla de Medina (Q24302385)), but we do not find (Candida Montilla de Medina (Q24302385), spouse (P26)). This can be corrected using the corresponding inference rule proposed in chapter 7.

Type constraint: *used to specify that items with the specified property should have the given type*[8]

[8]Probably, the formulation is misleading because, as we will see, most of the times, "the specified property have many types".

We run the type constraint on the head of government property (P6) where the relation (P2309) attribute is equal to "instance of" and the class (P2308) attribute can be one of the following elements (government (Q7188), administrative territorial entity (Q56061), government agency (Q327333), term of office (Q524572), cabinet (Q640506), political territorial entity (Q1048835), fictional country (Q1145276), ministry (Q6866562), and fictional administrative territorial entity (Q64034456)).

It reveals 284 violations from 36 354 (P6) statements. Some of the reasons behind this are the following:

1. the property P6 is used in the wrong direction, such as in this statement: (Gian Galeazzo Visconti (Q357539), head of government(P6), Milan(Q490)) instead of (Milan(Q490), head of government(P6), Gian Galeazzo Visconti (Q357539))

2. the property P6 is used with the wrong semantics (Anastas Ishirkov (Q4751826), head of government(P6), Bulgarian geographical society(Q12274348)). In addition, the Bulgarian geographical society is not in the allowed classes.

Value type constraint: *the referenced item should be a subclass or instance of the given type.*[9]

We run the value type constraint query on the country of citizenship property (P27) where the relation (P2309) attribute is equal to "instance of", and the class (P2308) attribute can be one of the following elements (country (Q6256), nation(Q6266), empire(Q48349), colony(Q133156), city-state(Q133442), polis(Q148837), dependant territory(Q161243), confederation(Q170156), nationality(Q231002), political territorial entity(Q1048835), fictional country(Q1145276), fictional planet(Q2775969), historical country(Q3024240), sovereign state(Q3624078), disputed territory(Q15239622), country in a fiction work(Q57662985), and state(former or current) (Q96196009)). The query reveals 5170 violations. Some examples are as follows:

- (Emperor Xizong of Jin(Q5071), country of citizinship, Jin dynasty)
- (Julie Ringuette (Q103818698), country of citizinship, Canadian French(Q1450506)

Distinct value constraint: *This constraint* [10] *is usually used to specify that a property generally only has a single value for an item.* Wikidata contains many properties that have a single-value constraint including the property RealGM basketball player ID(P3957). There are 44 statements that violate this constraint. For example, *Jefferson Andrale da Silva Antônio* has two statements with different values.

- (Jefferson Andrale da Silva Antônio), RealGM basketball player ID, 58 460)
- (Jefferson Andrale da Silva Antônio), RealGM basketball player ID, 40 989)

[9]The formulation of the type constraint should be changed to "the referenced ...instance of the given types"
[10]https://www.wikidata.org/wiki/Help:Property_constraints_portal/Single_value

8.8 Discussion

This experiment demonstrates that the MSL approach is a solution to the questions raised by the Wikidata reasoning group and others. It also shows that with the MSL approach, we can deal with massive numbers of qualifiers and generate many others by following a simple procedure.

Chapter 9

Conclusion

Chapter Contents

9.1 Summary of scientific contributions

This thesis presents models that deal with the formal representation of context and the practical aspects of contextual reasoning. Along the way, we have offered several insights regarding i) the semantics of the term "context," ii) the usability of the OWL formalism within a context-based setting, and iii) the use of context semantics in reasoning. The focus was to provide a conceptual framework for reasoning on contexts in knowledge graphs. We used Wikidata, an open-free property graph extremely rich with contextual data. Having a semantic web background, we started by proposing a contextual extension of OWL. Then, we studied the degree of compatibility of this extension with modern knowledge graphs, especially Wikidata. This study resulted in a novel contribution: a logical framework that formalizes the impact of Wikidata qualifiers in reasoning. We see it as an abstraction of the massive number of Wikidata's qualifiers formalized within a many-sorted logic. Sorts represent a qualifier category and are considered a high-level abstraction of several Wikidata qualifiers of a category. Finally, we coupled the model with an algebraic specification to formalize the behavior of qualifiers in reasoning.

Let us summarize the contributions by restating the answers to the research questions formulated at the beginning of the manuscript.

RQ1 *What are the obstacles encountered when one wants to implement practical contextual reasoning for ontologies on the SW? How to add validity contexts in an OWL2 profile?*
The topic of context representation and reasoning has been around for a long time. Theoretically, the subject has been discussed in artificial intelligence and logic, mainly, including a context term in a logical formula and the complexity of contextual reasoning. On the practical level, there has been plenty of work on including contexts in RDF, a language with binary predicates. However, to date, encoding contexts remain adhoc, and there are no standardized context-based reasoners. On top of that, there are no clear guidelines for creating a contextual knowledge base, storing it, querying it, or reasoning upon it.

Extending OWL with the two-dimensional approach can mitigate this problem. Therefore, we proposed OWL^C, which is an extension of OWL with two dimensions: one dimension for the core language and one for the context language. It is based on existing theoretical works because much has been done along this direction. We chose the two-dimensional approach of Klarman (Klarman and Gutiérrez-Basulto, 2011a) for its simplicity and clarity. Furthermore, the extension is equipped with a set of rules that allow efficient reasoning (scalable) and is therefore simple to implement.

RQ2 *To what extent can the semantic web languages serve the purpose of context representation and reasoning on property graphs?*
Property graphs like Wikidata can benefit from knowledge representation technologies that

help with knowledge organization and quality control. Although Wikidata is a data management platform, contributors use properties such as instance of and subclass of to represent schematic information and others like symmetric property to capture ontological-like information. However, these properties do not have a particular treatment; ultimately, they remain data like any other in Wikidata! It would thus be desirable to axiomatize their semantics. Unfortunately, most of the logic used by most KR formalisms, notably semantic web languages, are incompatible with property graphs because they cannot directly capture n-ary relations of any fixed arity n>2. Statements in these graphs are usually attributed with many attributes that explain the n-ary structure.

RQ3 *What types of context exist in property graphs such as Wikidata? And how to use them in contextual reasoning on property graphs?*
Knowledge graphs, especially property graphs like Wikidata, are rich with attribute-value pairs used to express contextual information. Such attributes in Wikidata are called qualifiers, and they express different types of contexts. The two major categories of contexts are validity contexts and non-validity contexts. Validity contexts interfere with the semantics of the original statement, whereas non-validity contexts provide only additional information that adds to the original statement without interfering with its semantics. These two categories are again divided into subcategories. Validity contexts in Wikidata include validity time, validity space, and "applies to" attributes. Among non-validity contexts, we can mention causality, sequence, and annotations. Causality qualifiers express the end/has cause of a statement. Sequence qualifiers behave like pointers referring to the elements that come before or after the subject of the statement. Annotations are different types of qualifiers expressing property constraints or descriptive properties. To reason on these categories, we propose representing them as sorts in a first-order many-sorted logic language. Each sort in this logic represents an abstraction of a category of qualifiers.

RQ4 *How to use this logical formalism to extract/infer implicit contextual knowledge from these graphs?*
The Wikidata community uses properties such as subclass of and instance of to capture schematic information. Furthermore, property constraints are declared and processed in an ad-hoc fashion. Without being part of an axiomatized theory, these properties remain data (they are not used to infer new facts/knowledge)! In a many-sorted logic, the meaning of these properties can be formalized with respect to sorts and then represented in a rule format. Based on that, we analyzed the types of qualifiers that accompany each conceptual property or constraint and proposed corresponding inference rules. What is particular about these rules is that they axiomatize the properties and constraints and provide a way to deal with

the qualifiers, which is one of the most critical problems of the Wikidata reasoning group[1]. We also showed that this formalism is applicable to more specific domain rules.

9.2 Impact statement

This thesis discusses a crucial topic in knowledge representation and reasoning: the representation and reasoning with contexts. The lack of standards and practical tools for reasoning on contexts is a long-standing issue. The root cause behind this problem is the term "context" itself. Researchers use it to refer to different things underlying different semantics, thus creating confusion when comparing existing works.

The first contribution of this thesis is that it formalizes the semantics of contexts. We did it transversally, starting from the semantic web settings and moving to the modern knowledge graphs. Clarifying context semantics and categorizing contexts will facilitate the work for future context researchers and implementers. This clarification is not only theoretical, but it has also been tested on Wikidata; it effectively allows the classification of the types of contexts present in Wikidata (and in other KGs). Additionally, the work answers the Wikidata reasoning group question by presenting solutions for the state of the qualifiers in reasoning. Furthermore, the work can be adapted to different applications. For example, historians interested in reasoning on time and causality can use the time and causalities modules in their reasoning rules.

From a semantic web perspective, we presented a contextual extension of OWL known as OWL^C. It will be a reference for rule-based reasoning when implementing context-dependent rules. Finally and most importantly, the work can be seen as a guideline for the design and implementation of reasoners on KGs.

9.3 Outlook and future works

This thesis strengthens the link between the theoretical and practical communities of contextual reasoning. We must say that the focus was more on formal approaches. Why? Because as observed in Martin and Patel-Schneider (2020), "the value of formal characterizations is foundational in Computer Science. We rely on them for clarity in specification in most of our activities." We used logic to formalize the different aspects of context representation in knowledge graphs. However, the practical aspect needs to be further advanced. In the case of OWL^C, we should deploy usability test cases in different applications, in particular, to evaluate to what extent this language helps ontology designers to incorporate validity contexts in their conceptualizations. This also raises the question of creating new ontology design methods that account for validity contexts. In the case of the many-sorted logic, we will work on the automated translation of logical rules and algebraic specifications into

[1]https://www.wikidata.org/wiki/Wikidata:WikiProject$_{R}easoning/Use_{c}ases$

operational languages (SPARQL, SHACL, Javascript, etc). This translator would be helpful for applications considering using Wikidata subgraphs in their knowledge base. Another aspect is to study the performance of reasoning. Finally, an essential part is to create evaluation frameworks for contextual languages, reasoning systems, and design methodologies for knowledge graphs.

References

Wikidata page of barack obama. https://www.wikidata.org/wiki/Q76, a.

Wikidata page of beatrice saubin. https://www.wikidata.org/wiki/Q42304190, b.

Wikidata page of douglas adam. https://www.wikidata.org/wiki/Q42, c.

Sahar Aljalbout and Gilles Falquet. Un modèle pour la représentation des connaissances temporelles dans les documents historiques. In *{IC} 2017 :Proceedings of the 28th French Knowledge Engineering Conference, Caen, France, July 3-7, 2017.*, pages 145–150, 2017.

Sahar Aljalbout, Didier Buchs, and Gilles Falquet. Introducing contextual reasoning to the semantic web with owl. In *International Conference on Conceptual Structures*, pages 13–26. Springer, 2019a.

Sahar Aljalbout, Didier Buchs, and Gilles Falquet. OWLˆ C: A Contextual Two-Dimensional Web Ontology Language. In *2nd Conference on Language, Data and Knowledge (LDK 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019b.

Renzo Angles and Claudio Gutierrez. Survey of graph database models. *ACM Computing Surveys (CSUR)*, 40(1):1–39, 2008.

Egidio Astesiano, Michel Bidoit, Hélene Kirchner, Bernd Krieg-Brückner, Peter D Mosses, Donald Sannella, and Andrzej Tarlecki. Casl: the common algebraic specification language. *Theoretical Computer Science*, 286(2):153–196, 2002.

Franz Baader and Armin Laux. Terminological logics with modal operators. 1994.

Colin Batchelor, Christian YA Brenninkmeijer, Christine Chichester, Mark Davies, Daniela Digles, Ian Dunlop, Chris T Evelo, Anna Gaulton, Carole Goble, Alasdair JG Gray, et al. Scientific lenses to support multiple views over linked chemistry data. In *International Semantic Web Conference*, pages 98–113. Springer, 2014.

Wouter Beek, Stefan Schlobach, and Frank van Harmelen. A contextualised semantics for owl: sameas. In *European Semantic Web Conference*, pages 405–419. Springer, 2016.

François Belleau, Marc-Alexandre Nolin, Nicole Tourigny, Philippe Rigault, and Jean Morissette. Bio2RDF: towards a mashup to build bioinformatics knowledge systems. *Journal of biomedical informatics*, 41(5):706–716, 2008.

Massimo Benerecetti, Paolo Bouquet, and Chiara Ghidini. Contextual reasoning distilled. *Journal of Experimental & Theoretical Artificial Intelligence*, 12(3):279–305, 2000.

Massimo Benerecetti, Paolo Bouquet, and Chiara Ghidini. On the dimensions of context dependence. Technical report, University of Trento, 2002.

Tim Berners-Lee, James Hendler, Ora Lassila, and Others. The semantic web. *Scientific american*, 284(5):28–37, 2001.

Christian Bizer, Jens Lehmann, Georgi Kobilarov, Sören Auer, Christian Becker, Richard Cyganiak, and Sebastian Hellmann. Dbpedia-a crystallization point for the web of data. *Journal of web semantics*, 7(3):154–165, 2009.

Paolo Bouquet, Fausto Giunchiglia, Frank Van Harmelen, Luciano Serafini, and Heiner Stuckenschmidt. C-owl: Contextualizing ontologies. In *International Semantic Web Conference*, pages 164–179. Springer, 2003a.

Paolo Bouquet, Bernardo Magnini, Luciano Serafini, and Stefano Zanobini. A sat-based algorithm for context matching. In *International and Interdisciplinary Conference on Modeling and Using Context*, pages 66–79. Springer, 2003b.

Loris Bozzato and Luciano Serafini. Materialization calculus for contexts in the Semantic Web. *CEUR Workshop Proceedings*, 1014:552–572, 2013a. ISSN 16130073.

Loris Bozzato and Luciano Serafini. Materialization Calculus for Contexts in the Semantic Web. In *Description Logics*, pages 552–572, 2013b.

Loris Bozzato, Martin Homola, and Luciano Serafini. Context on the semantic web: Why and how. *ARCOE-12*, page 11, 2012.

Loris Bozzato, Chiara Ghidini, and Luciano Serafini. Comparing contextual and flat representations ofknowledge: a concrete case about football data. In *Proceedings of the seventh international conference on Knowledge capture*, pages 9–16. ACM, 2013.

Dan Brickley and R.V. Guha. Rdf schema 1.1. https://www.w3.org/TR/rdf-schema/.

Jeremy J Carroll, Christian Bizer, Pat Hayes, and Patrick Stickler. Named graphs, provenance and trust. In *Proceedings of the 14th international conference on World Wide Web*, pages 613–622, 2005.

Anthony G Cohn. On the solution of schubert's steamroller in many-sorted logic. In *IJCAI*, pages 1169–1174, 1985.

Anthony G Cohn. A many sorted logic with possibly empty sorts. In *International Conference on Automated Deduction*, pages 633–647. Springer, 1992.

Renata Dividino, Sergej Sizov, Steffen Staab, and Bernhard Schueler. Querying for provenance, trust, uncertainty and other meta knowledge in RDF. *Web Semantics: Science, Services and Agents on the World Wide Web*, 7(3):204–219, 2009.

David Ferrucci, Eric Brown, Jennifer Chu-Carroll, James Fan, David Gondek, Aditya A Kalyanpur, Adam Lally, J William Murdock, Eric Nyberg, John Prager, and Others. Building Watson: An overview of the DeepQA project. *AI magazine*, 31(3):59–79, 2010.

Aldo Gangemi and Peter Mika. Understanding the semantic web through descriptions and situations. In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*, pages 689–706. Springer, 2003.

Chiara Ghidini and Fausto Giunchiglia. Local models semantics, or contextual reasoning= locality+ compatibility. *Artificial intelligence*, 127(2):221–259, 2001.

José M Giménez-García, Antoine Zimmermann, and Pierre Maret. Ndfluents: An ontology for annotated statements with inference preservation. In *European Semantic Web Conference*, pages 638–654. Springer, 2017.

Fausto Giunchiglia. Contextual reasoning. *Epistemologia, special issue on I Linguaggi e le Macchine*, 16:345–364, 1993.

Fausto Giunchiglia and Luciano Serafini. Multilanguage hierarchical logics, or: how we can do without modal logics. *Artificial intelligence*, 65(1):29–70, 1994.

Benjamin Grosof, Ian Horrocks, Raphael Volz, and Stefan Decker. Description Logic Programs: Combining Logic Programs with Description Logic. In *Proceedings of the World-Wide Web Conferance: WWW2003*. ACM, 2003.

Paul Groth, Andrew Gibson, and Jan Velterop. The anatomy of a nanopublication. *Information Services & Use*, 30(1-2):51–56, 2010.

Nicola Guarino, Daniel Oberle, and Steffen Staab. What is an ontology? In *Handbook on ontologies*, pages 1–17. Springer, 2009.

Ramanathan V Guha. *Contexts: a formalization and some applications*, volume 101. Stanford University Stanford, CA, 1991.

Claudio Gutierrez, Carlos A Hurtado, and Alejandro Vaisman. Introducing time into RDF. *IEEE Transactions on Knowledge and Data Engineering*, 19(2):207–218, 2006.

Peter Haase, Andriy Nikolov, Johannes Trame, Artem Kozlov, and Daniel M Herzig. Alexa, ask wikidata! voice interaction with knowledge graphs using amazon alexa. In *International Semantic Web Conference (Posters, Demos & Industry Tracks)*, 2017.

Olaf Hartig and Bryan Thompson. Foundations of an alternative approach to reification in RDF. *arXiv preprint arXiv:1406.3399*, 2014.

Daniel Hernández, Aidan Hogan, and Markus Krötzsch. Reifying RDF: What works well with wikidata? *SSWS@ ISWC*, 1457:32–47, 2015.

Johannes Hoffart, Fabian M Suchanek, Klaus Berberich, and Gerhard Weikum. YAGO2: A spatially and temporally enhanced knowledge base from Wikipedia. *Artificial Intelligence*, 194:28–61, 2013.

A Hogan. Context in graphs. In *Proceedings of the 1st International Workshop on Conceptualized Knowledge Graphs. RWTH Aachen University, Aachen*, 2018.

Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia D'Amato, Gerard de Melo, Claudio Gutierrez, José Emilio Labra Gayo, Sabrina Kirrane, Sebastian Neumaier, Axel Polleres, Roberto Navigli, Axel-Cyrille Ngonga Ngomo, Sabbir M. Rashid, Anisa Rula, Lukas Schmelzeisen, Juan Sequeda, Steffen Staab, and Antoine Zimmermann. Knowledge Graphs. 2020. URL http://arxiv.org/abs/2003.02320.

Martin Homola, Andrei Tamilin, and Luciano Serafini. Modeling contextualized knowledge. In *CIAO@ EKAW*, 2010.

Ian Horrocks and Peter F Patel-Schneider. Reducing owl entailment to description logic satisfiability. In *International semantic web conference*, pages 17–29. Springer, 2003.

Ian Horrocks, Peter F Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof, Mike Dean, et al. Swrl: A semantic web rule language combining owl and ruleml. *W3C Member submission*, 21(79):1–31, 2004.

Mathew Joseph and Luciano Serafini. Simple Reasoning for Contextualized RDF Knowledge. In *WoMO*, pages 79–93, 2011.

Michael Kifer and V S Subrahmanian. Theory of generalized annotated logic programming and its applications. *The Journal of Logic Programming*, 12(4):335–367, 1992.

Szymon Klarman. *Reasoning with contexts in description logics.* PhD thesis, Citeseer, 2013.

Szymon Klarman, 2017. URL https://www.slideshare.net/SzymonKlarman/description-logics-of-context.

Szymon Klarman and V\'\ictor Gutiérrez-Basulto. Two-Dimensional Description Logics for Context-Based Semantic Interoperability. In *AAAI*, 2011a.

Szymon Klarman and Víctor Gutiérrez-Basulto. Two-dimensional description logics for context-based semantic interoperability. *Proceedings of the National Conference on Artificial Intelligence*, 1:215–220, 2011b.

Markus Krötzsch. Ontologies for Knowledge Graphs? In *Description Logics*, 2017.

Markus Krötzsch. Too much information: Can AI cope with modern knowledge graphs? *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 11511 LNAI:17–31, 2019. ISSN 16113349. doi:10.1007/978-3-030-21462-3_2.

Markus Krötzsch and Veronika Thost. Ontologies for knowledge graphs: Breaking the rules. In *International Semantic Web Conference*, pages 376–392. Springer, 2016.

Markus Krötzsch, František Simančík, and Ian Horrocks. A description logic primer. *Perspectives on Ontology Learning*, 18(June):3–20, 2014.

Markus Krötzsch, Maximilian Marx, Ana Ozaki, and Veronika Thost. Attributed Description Logics: Reasoning on Knowledge Graphs. In *IJCAI*, pages 5309–5313, 2018.

Doug Lenat. The dimensions of context-space, 1998.

David L Martin and Peter F Patel-Schneider. A logical approach to representing constraints in wikidata. *arXiv preprint arXiv:2008.03900*, 2020.

Maximilian Marx and Markus Krötzsch. Sqid: Towards ontological reasoning for wikidata. In *International Semantic Web Conference (Posters, Demos & Industry Tracks)*, 2017.

Maximilian Marx, Markus Krötzsch, and Veronika Thost. Logic on MARS: Ontologies for generalised Property graphs. *IJCAI International Joint Conference on Artificial Intelligence*, 155629:1188–1194, 2017a. ISSN 10450823. doi:10.24963/ijcai.2017/165.

Maximilian Marx, Markus Krötzsch, and Veronika Thost. Logic on MARS: Ontologies for Generalised Property Graphs. In *IJCAI*, pages 1188–1194, 2017b.

John McCarthy. Generality in artificial intelligence. *Communications of the ACM*, 30(12): 1030–1035, 1987.

John McCarthy. Notes on formalizing context. 1993.

Deborah L McGuinness, Richard Fikes, James Hendler, and Lynn Andrea Stein. Daml+ oil: an ontology language for the semantic web. *IEEE Intelligent Systems*, 17(5):72–80, 2002.

Eric Miller. An introduction to the resource description framework. *Bulletin of the American Society for Information Science and Technology*, 25(1):15–19, 1998.

Boris Motik. Representing and querying validity time in rdf and owl: A logic-based approach. In *International Semantic Web Conference*, pages 550–565. Springer, 2010.

Boris Motik, Bernardo Cuenca Grau, Ian Horrocks, Zhe Wu, Achille Fokoue, and Carsten Lutz. OWL 2 Web Ontology Language Profiles (Second Edition). W3c recommendation, W3C, 2012.

Vinh Nguyen, Olivier Bodenreider, and Amit Sheth. Don't like RDF reification?: making statements about statements using singleton property. In *Proceedings of the 23rd international conference on World wide web*, pages 759–770. ACM, 2014.

Natasha Noy, Alan Rector, Pat Hayes, and Chris Welty. Defining n-ary relations on the semantic web. *W3C working group note*, 12(4), 2006.

Peter F Patel-Schneider. Contextualization via qualifiers. In *Contextualized Knowledge Graphs @ ISWC 2018*, 2018.

Peter F Patel-Schneider and David Martin. Wikidata on mars. *arXiv preprint arXiv:2008.06599*, 2020.

Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. Semantics and complexity of sparql. *ACM Transactions on Database Systems (TODS)*, 34(3):1–45, 2009.

Edward W Schneider. Course Modularization Applied: The Interface System and Its Implications For Sequence Control and Data Analysis. 1973.

John F Sowa. *Principles of semantic networks: Explorations in the representation of knowledge.* Morgan Kaufmann, 2014.

Umberto Straccia, Nuno Lopes, Gergely Lukacsy, and Axel Polleres. A general framework for representing and reasoning with annotated semantic web data. In *Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.

VS Subrahmanian. On the semantics of quantitative logic programs. In *SLP*, pages 173–182, 1987.

Octavian Udrea, Diego Reforgiato Recupero, and V S Subrahmanian. Annotated rdf. *ACM Transactions on Computational Logic (TOCL)*, 11(2):1–41, 2010.

Michael Uschold, Michael Gruninger, et al. Ontologies: Principles, methods and applications. *TECHNICAL REPORT-UNIVERSITY OF EDINBURGH ARTIFICIAL INTELLIGENCE APPLICATIONS INSTITUTE AIAI TR*, 1996.

Denny Vrande. i{{\textcent}} and Markus Krötzsch. Wikidata: A free collaborative knowledgebase. *Commun. ACM*, 57(10), 2014.

Christoph Walther. A mechanical solution of schubert's steamroller by many-sorted resolution. *Artificial Intelligence*, 26(2):217–224, 1985.

Chris Welty. Context slices: representing contexts in OWL. In *Proceedings of the 2nd International Conference on Ontology Patterns-Volume 671*, pages 59–60. CEUR-WS. org, 2010.

Chris Welty, Richard Fikes, and Selene Makarios. A reusable ontology for fluents in OWL. In *FOIS*, volume 150, pages 226–236, 2006.

Frank Wolter and Michael Zakharyaschev. Multi-dimensional description logics. In *IJCAI*, volume 99, pages 104–109. Citeseer, 1999.

Antoine Zimmermann, Nuno Lopes, Axel Polleres, and Umberto Straccia. A general framework for representing, reasoning and querying with annotated Semantic Web data. *Journal of Web Semantics*, 11:72–95, 2012a. ISSN 15708268. doi:10.1016/j.websem.2011.08.006. URL http://dx.doi.org/10.1016/j.websem.2011.08.006.

Antoine Zimmermann, Nuno Lopes, Axel Polleres, and Umberto Straccia. A general framework for representing, reasoning and querying with annotated semantic web data. *Web Semantics: Science, Services and Agents on the World Wide Web*, 11:72–95, 2012b.

Appendices

Appendix A

Appendix A

A.1 Completeness of the rule set

The OWL^C inference rules are complete, in the same sense as the OWL-RL rules[1]. In particular, we can rephrase the theorem PR1 of Motik et al. (2012)

Theorem 1. *Let $\mathcal{R}$ be the OWL-C rule set as defined in section 4. Let O_1 and O_2 be OWL-C ontologies satisfying the following properties:*

- *neither O_1 nor O_2 contains a IRI that is used for more than one type of entity (i.e., no IRIs is used both as, say, a class and an individual);*
- *each axiom in O_2 is either a contextual class assertion $k : C(a)$ or a property assertion $k : P(a_1, a_2)$ on the core vocabulary or a class assertion $K(k)$ or a property assertion $Q(k_1, k_2)$ on the context vocabulary.*

Furthermore, let $TQ(O_1)$ and $TQ(O_2)$ be translations of O_1 and O_2, respectively, into FOL atoms on the predicates T (ternary) and Q (quaternary) (see 4.4.1) Then, O_1 entails O_2 under the OWL^C semantics if and only if $TQ(O_1) \cup \mathcal{R}$ entails $TQ(O_2)$ under the standard first-order semantics.

Proof. To prove this theorem we re-use the proof PR1[2]. This means that we must first define a mapping from OWL^C to logic programs (definite Horn clauses). Following the lines of Grosof et al. (2003) we first observe that OWL^C statements are equivalent to first order formulas on unary, binary, and ternary predicates (the predicates for the core contexts and roles have an additional parameter for the context). Table A.1 shows some of these equivalences

Then we define a mapping $\mathcal{T}^C$ from OWL^C statements to FOL that preserves the semantics of OWL^C. This mapping extends the $\mathcal{T}$ mapping of Grosof et al. (2003). The $\mathcal{T}$ mapping is

[1]https://www.w3.org/TR/owl2-profiles/

[2]https://www.w3.org/TR/owl2-profiles/#Theorem-PR1

OWL^C Expression	FOL Expression
$k : C(a)$	$C(a,k)$
$K : C(a)$	$\forall k\ K(k) \rightarrow C(a,k)$
$k : R(a,b)$	$R(a,b,k)$
$k : C \sqsubseteq D$	$\forall x\ C(x,k) \rightarrow D(x,k)$
$k : \mathrm{dom}(R,C)$	$\forall x \forall y\ R(x,y,k) \rightarrow C(y,k)$
$k : \mathrm{trans}(R)$	$\forall x \forall y \forall z\ R(x,y,k) \wedge R(y,z,k) \rightarrow R(x,z,k)$
A	$A(x,k)$
$C \sqcap D$	$C(x,k) \wedge D(x,k)$
$C \cup D$	$C(x,k) \vee D(x,k)$
$\exists R.C$	$\exists y\ R(x,y,k) \wedge C(y,k)$
$\forall R.C$	$\forall y\ R(x,y,k) \rightarrow C(y,k)$
$\langle K \rangle\, C$	$\exists j.K(j) \wedge C(x,j)$
$[K]C$	$\forall j.K(j) \rightarrow C(x,j)$
$\{i\}C$	$C(x,i)$
$\langle R.K \rangle C$	$\exists j\, R(k,j) \wedge K(j) \wedge C(x,j)$
$[R.K]C$	$\forall j\, (R(k,j) \wedge K(j)) \rightarrow C(x,j)$

Table A.1 Sample OWL^C-FOL equivalences (k represent a context instance, K represents a context class)

used for the context language (which is not contextual). This mapping is defined only for OWL-C statements that satisfy the following syntactic restrictions:

- $\sqcup$, $\exists$, $\langle . \rangle$ may only appear in the left hand side of a $\sqsubseteq$ axiom
- $\forall$, $[.]$ may only appear in the right hand side of a $\sqsubseteq$ axiom

$\mathcal{T}^C$ does not yield Horn rules. However, the produced formulas can be directly transformed to Horn rules of the form $\mathit{head} \leftarrow \mathit{body}_1 \wedge \cdots \mathit{body}_n$ (rules with an $\vee$ in the body yield two rules, rules with a $\leftarrow$ in the head yield a rule with an $\wedge$ in the body, i.e. $(A \leftarrow B) \leftarrow C \equiv A \leftarrow (B \wedge C)$), rules with a $\wedge$ in the head yield two rules)

$$\begin{array}{rcl}
\mathcal{T}^{\mathcal{C}}(k : C(a)) & = & \mathcal{T}^{C}_{head}(C, a, k) \\
\mathcal{T}^{\mathcal{C}}(K : C(a)) & = & \mathcal{T}^{C}_{head}(C, a, k) \leftarrow \mathcal{T}_{body}(K, k) \\
\mathcal{T}^{\mathcal{C}}(k : R(a, b)) & = & R(a, b, k) \\
\mathcal{T}^{\mathcal{C}}(k : C \sqsubseteq D) & = & \mathcal{T}^{C}_{head}(D, x, k) \leftarrow \mathcal{T}^{C}_{body}(C, x, k) \\
\mathcal{T}^{\mathcal{C}}(K : C \sqsubseteq D) & = & \mathcal{T}^{C}_{head}(D, x, k) \leftarrow \mathcal{T}_{body}(K, k) \wedge \mathcal{T}^{C}_{body}(C, x, k)
\end{array}$$

$$\begin{array}{rcl}
\mathcal{T}^{C}_{body}(A, x, k) & = & A(x, k) \\
\mathcal{T}^{C}_{head}(A, x, k) & = & A(x, k) \\
\mathcal{T}^{C}_{head}(C \sqcap D, x, k) & = & \mathcal{T}^{C}_{head}(C, x, k) \wedge \mathcal{T}^{C}_{head}(D, x, k) \\
\mathcal{T}^{C}_{body}(C \sqcup D, x, k) & = & \mathcal{T}^{C}_{head}(C, x, k) \vee \mathcal{T}^{C}_{head}(D, x, k) \\
\mathcal{T}^{C}_{body}(\exists R.C, x, k) & = & R(x, y, k) \wedge \mathcal{T}^{C}_{body}(C, y, k) \\
\mathcal{T}^{C}_{head}(\forall R.C, x, k) & = & \mathcal{T}^{C}_{head}(C, y, k) \leftarrow R(x, y, k) \\
\mathcal{T}^{C}_{head}([K]C, x, k) & = & \mathcal{T}^{C}_{head}(C, x, k') \leftarrow \mathcal{T}_{body}K(k') \\
\mathcal{T}^{C}_{head}([R.K]C, x, k) & = & \mathcal{T}^{C}_{head}(C, x, k') \leftarrow \mathcal{T}_{body}R(k, k') \wedge \mathcal{T}_{body}K(k') \\
\mathcal{T}^{C}_{body}(\langle K \rangle\, C, x, k) & = & \mathcal{T}_{body}K(k') \wedge \mathcal{T}^{C}_{body}(C, x, k') \\
\mathcal{T}^{C}_{body}(\langle R.K \rangle\, C, x, k) & = & \mathcal{T}_{body}R(k, k') \wedge \mathcal{T}_{body}K(k') \wedge \mathcal{T}^{C}_{body}(C, x, k') \\
\mathcal{T}^{C}_{head/body}(\{i\}C, x, k) & = & \mathcal{T}^{C}_{head/body}(C, x, i)
\end{array}$$

It is straightforward to see, by comparing these mappings with table **??**, that this mapping preserves the semantics of CDL statements, provided they obey the syntactic restrictions.

From this point it is possible to adapt the completeness proof of theorem PR1 in Motik et al. (2012).

Without loss of generality, it can be assumed that all axioms in O_1 are fully normalized — that is, that all class expressions in the axioms are of depth at most one.

For example

$$k : \exists R.[C1 \sqcup C2] \sqsubseteq [S.K](D1 \sqcap D2)$$

would be normalized as

$$k : \exists R.C_1 \sqsubseteq [S.K]D_1$$

$$k : \exists R.C_1 \sqsubseteq [S.K]D_2$$

$$k : \exists R.C_2 \sqsubseteq [S.K]D_1$$

$$k : \exists R.C_2 \sqsubseteq [S.K]D_2$$

Moreover, all the $\sqcup$ can be removed because they only appear in the left-hand side and $\{C_1 \sqcup C_2 \sqsubseteq D\}$ is equivalent to $\{C_1 \sqsubseteq D, C_2 \sqsubseteq D\}$.

Let $CDLP(O_1)$ be the set of rules obtained by applying $\mathcal{T}^C$ to O_1.

Consider now each assertion $A \in O_2$ that is entailed by $CDLP(O_1$ (or, equivalently, by O_1).

Let dt be a derivation tree for A from $CDLP(O_1\ )$.

Each node of the tree is a (fully instantiated) Horn rule $H \leftarrow B_1, ..., B_n$

By examining the set of OWL^C constructs, it is possible to see that each such tree can be transformed to a derivation tree dt' for $TQ(A)$ from $TQ(O_1) \cup \mathcal{R}$. Each assertion B occurring in dt is of the form as specified in the theorem. The tree dt' can, roughly speaking, be obtained from dt by replacing each assertion B with $TQ(B)$) and by replacing each rule from $CDLP(O_1)$ with a corresponding rule from $\mathcal{R}$ (in fact, each OWL^C rule corresponds to the mapping $\mathcal{T}^C$ of some OWL^C construct). Consequently, $TQ(O_1) \cup \mathcal{R}$ entails $TQ(A)$).

More precisely, consider a CDLP rule that is used (instantiated) in the derivation tree. This rule necessarily comes from an axiom $k : \phi \sqsubseteq \psi$ in O_1 where ϕ has the form A (a class name) or $\exists R.C$ or $\langle K\rangle C$ or $\langle R.K\rangle C$ and ψ has the form A or $\forall R.C$ or $[K]C$ or $[R.K]C$

One must check that for all possible category of rules (ϕ - ψ combination) there are OWL^C rules that generate an equivalent result.

As an illustration, consider the axiom

$$k : \exists R.C \sqsubseteq \forall R'.C'$$

It yields the CDLP rule

$$(C'(y', k) \leftarrow R'(x, y', k)) \leftarrow R(x, y, k) \wedge C(y, k)$$

which is equivalent to

$$C'(y', k) \leftarrow R'(x, y', k) \wedge R(x, y, k) \wedge C(y, k)$$

It's mapping to triples and quadruples yields

1. Q(`_:e1`, `rdfs:subClassOf`, `_:e2`, k)
2. Q(`_:e1`, `owl:someValueFrom`, C, `owlc:Context`)
3. Q(`_:e1`, `owl:onProperty`, R, `owlc:Context`)
4. Q(`_:e2`, `owl:allValueFrom`, C', `owlc:Context`)
5. Q(`_:e2`, `owl:onProperty`, R', `owlc:Context`)

If the rule belongs to a derivation tree it means that other rules have produced atoms that can be unified with the body, for instance $R'(a, b', k,\ R(a, b, x),\ C(b, k)$. Their representation as quadruples are If the quadruples

1. Q(a, R', b', k)
2. Q(a, R, b, k)

3. $\mathrm{Q}(b, \texttt{rdf:type}, C, k)$

The application of the rules cls-svf1-1, cax-sco, and cls-avf1-1 would successively produce

$\mathrm{Q}(a, \texttt{rdf:type}, \texttt{\_:e1}, k)$

$\mathrm{Q}(a, \texttt{rdf:type}, \texttt{\_:e2}, k)$

$\mathrm{Q}(b', \texttt{rdf:type}, C', k)$

Therefore this category of CDLP rule has equivalent OWL^C rules.
It is straightforward to apply the same reasoning for the other categories of rules.

$\square$

Appendix B

Appendix B

B.1 Algebraic specification

```
 spec NAT =

 sort nat

 op ___+___ : nat * nat -> nat
 op ___-___ : nat * nat -> nat
 pred ___<___ : nat * nat

end


spec Resource =
sort resource
sort IRI

%% generators
op rsrc : IRI -> resource

pred equal: resource * resource

op undefined : resource

forall r1, r2, r3: resource, i:IRI
. equal(r1, r1)
. equal(r1, r2) /\ equal(r2, r3) => equal(r1, r3)
. equal(r1, r2) => equal(r2, r1)

. not equal(rsrc(i), undefined)
. not equal(undefined , rsrc(i))

end

spec TimeParam =

sort time
op undefined : time
op union : time * time -> time
op inter : time * time -> time

pred testIntersect : time * time
pred incl : time * time
pred equal : time * time

forall t1, t2, t3 : time
. equal(t1, t1)
. equal(t1, t2) /\ equal(t2, t3) => equal(t1, t3)
. equal(t1 , t2) => equal(t2, t1)

end


spec SpaceParam =

sort space

pred equal : space * space
pred incl : space * space
op union : space * space -> space
op inter : space * space -> space


forall s1, s2, s3 : space
. equal(s1, s1)
. equal(s1, s2) /\ equal(s2, s3) => equal(s1, s3)
. equal(s1, s2) => equal(s2, s1)
end


spec ValidityContext[TimeParam][SpaceParam] =


sort validityContext


%% Generators
op emptyValidity : validityContext %% empty means valid in every context
op timeValidity : time -> validityContext
op spaceValidity : space -> validityContext
op timespaceValidity : time * space -> validityContext

pred testIntersectValidity : validityContext * validityContext
pred incl : validityContext * validityContext
pred equal : validityContext * validityContext

op union : validityContext * validityContext -> validityContext
op interValidity : validityContext * validityContext -> validityContext
op extractTime : validityContext -> time
op extractSpace : validityContext -> space
op setTime : validityContext * time -> validityContext
op setSpace : validityContext * space -> validityContext


forall c, c1, c2, c3 : validityContext, t, t1 , t2 : time , s, s1, s2 : space
. setTime(emptyValidity, t) = timeValidity(t)
. setTime(timeValidity(t1), t2) = timeValidity(t2)
. setTime(spaceValidity(s), t) = timespaceValidity(t, s)
. setTime(timespaceValidity(t, s), t2) = timespaceValidity(t2, s)

. setSpace(emptyValidity, s) = spaceValidity(s)
. setSpace(timeValidity(t), s) = timespaceValidity(t, s)
. setSpace(spaceValidity(s), s2) = spaceValidity(s2)
. setSpace(timespaceValidity(t, s), s2)= timespaceValidity(t, s2)
. extractTime(timeValidity(t1)) = t1
. extractTime(timespaceValidity(t, s))= t
. extractTime(spaceValidity(s)) = undefined

. extractSpace(spaceValidity(s)) = s
. extractSpace(timespaceValidity(t, s)) = s

. union(emptyValidity, c) = emptyValidity
. union(c, emptyValidity) = emptyValidity

. union(timeValidity(t1), spaceValidity(s2)) = emptyValidity
. union(timeValidity(t1), timeValidity(t2)) = timeValidity(union(t1, t2))
. union(timeValidity(t1), timespaceValidity(t2, s2)) = timespaceValidity(union(t1, t2),
s2)

. union(spaceValidity(s1), spaceValidity(s2)) = spaceValidity(union(s1, s2))
. union(spaceValidity(s1), timeValidity(t2)) = emptyValidity
. union(spaceValidity(s1), timespaceValidity(t2, s2)) = timespaceValidity(t2 , union(s1,
s2))

. union(timespaceValidity(t1, s1), timespaceValidity(t2, s2)) =
 timespaceValidity(union(t1, t2) , union(s1, s2))
. union(timespaceValidity(t1, s1), spaceValidity(s2)) = timespaceValidity(t1 , union(s1,
s2))
. union(timespaceValidity(t1, s1), timeValidity(t2)) = timespaceValidity(union(t1, t2),
s1)


. interValidity(emptyValidity, c) = c
. interValidity(c, emptyValidity) = c

. interValidity(timeValidity(t1), spaceValidity(s2)) = timespaceValidity(t1, s2)
. interValidity(timeValidity(t1), timeValidity(t2)) = timeValidity(inter(t1, t2))

. interValidity(timeValidity(t1), timespaceValidity(t2, s2)) = timespaceValidity(inter(t1,
t2), s1)

. interValidity(spaceValidity(s1), spaceValidity(s2)) = spaceValidity(inter(s1, s2))
. interValidity(spaceValidity(s1), timeValidity(t2)) = timespaceValidity(t2, s1)

. interValidity(spaceValidity(s1), timespaceValidity(t2, s2)) = timespaceValidity(t2,
inter(s1, s2))

. interValidity(timespaceValidity(t1, s1), spaceValidity(s2)) = timespaceValidity(t1,
inter(s1, s2))

. interValidity(timespaceValidity(t1, s1), timeValidity(t2)) = timespaceValidity(inter(t1,
t2), s1)

. interValidity(timespaceValidity(t1, s1), timespaceValidity(t2, s2)) =
 timespaceValidity(inter(t1, t2) , inter(s1, s2))

. testIntersectValidity(c1, c2) <=> not equal(interValidity(c1, c2), emptyValidity)
%% equality axioms

. incl(c1, c2) <=> incl(extractTime(c1), extractTime(c2))
 /\ incl(extractSpace(c1), extractSpace(c2))

%% theorems for setTime, setSpace, extractTime and extractSpace

. extractTime(setTime(c, t1)) = t1
. extractSpace(setSpace(c, s)) = s

. extractTime(interValidity(c1, c2)) = inter(extractTime(c1), extractTime(c2))
. extractSpace(interValidity(c1, c2))= inter(extractSpace(c1), extractSpace(c2))

. extractTime(union(c1, c2)) = union(extractTime(c1), extractTime(c2))
. extractSpace(union(c1, c2)) = union(extractSpace(c1), extractSpace(c2))

%% inclusion theorems
. incl(c1, c2) /\ incl(c2, c1)=> equal(c1, c2)
. incl(c1, c2) /\ incl(c2, c3) => incl(c1, c3)

%% equality theorems
. equal(c1, c1)
. equal(c1, c2) /\ equal(c2, c3) => equal(c1, c3)
. equal(c1, c2) => equal(c2, c1)
. equal(c1, c2) <=> equal(extractTime(c1), extractTime(c2))
 /\ equal(extractSpace(c1), extractSpace(c2))

end




spec validityInstantTime =

NAT

then

sort instantTime
sort duration

%% generators
op undefined : instantTime
op undefinedDuration: duration
op instant : nat -> instantTime

op min : instantTime * instantTime -> instantTime
op max : instantTime * instantTime -> instantTime

pred equal : instantTime * instantTime

op union : instantTime * instantTime -> instantTime
op inter : instantTime * instantTime -> instantTime

pred testIntersect : instantTime * instantTime

pred ___<___ : instantTime * instantTime
pred ___<=___ : instantTime * instantTime

op ___-___ : instantTime * instantTime -> duration
op ___+___ : instantTime * duration -> instantTime

forall x, y : nat , i, i1, i2 : instantTime
```

```
. x < y => min(instant(x), instant(y)) = instant(x)
. x < y => max(instant(x), instant(y)) = instant(y)
. y < x => min(instant(x), instant(y)) = instant(y)
. y < x => max(instant(x), instant(y)) = instant(x)
. x = y => max(instant(x), instant(y)) = instant(x)
. x = y => min(instant(x), instant(y)) = instant(x)

.min(i, i)=i
.max(i, i)=i
.min(instant(x), undefined) = undefined
.max(instant(x), undefined) = undefined
.min(undefined, instant(y)) = undefined
.max(undefined, instant(y)) = undefined

. instant(x) < instant(y) <=> x < y
. instant(x) = instant(y) <=> x = y
. instant(x) <= instant(y) <=> x < y \/ x = y

. i1 = i2 => union(i1, i2) = i1
. not(i1 = i2) => union(i1, i2) = undefined
. i1 = i2 => inter(i1, i2) = i1
. not(i1 = i2) => inter(i1, i2) = undefined
. i1 = i2 <=> testIntersect(i1, i2)

forall n, m : nat
.equal(instant(n), instant(m)) <=> (n=m)
.equal(undefined, undefined)

.instant(n) < instant(m) <=> n<m

end


spec ValidityTimeInterval =

validityInstantTime

then
```

```
sort timeInterval

%% Generators
op undefined : timeInterval
op interval : instantTime * instantTime -> timeInterval
op interval : instantTime * duration -> timeInterval

pred equal : timeInterval * timeInterval
pred disjoint : timeInterval * timeInterval
pred inside : instantTime * timeInterval
pred testIntersectInterval : timeInterval * timeInterval

op union : timeInterval * timeInterval -> timeInterval
op interInterval : timeInterval * timeInterval -> timeInterval

op startTime : timeInterval -> instantTime
op endTime : timeInterval -> instantTime
op duration : timeInterval -> duration

forall t1, t2: timeInterval, x, x1, x2, y1, y2: instantTime, d: duration

. equal(t1, t2) <=> equal(startTime(t1), startTime(t2)) /\ equal(endTime(t1),
endTime(t2))

%% an undefined start time means -infinity, an undefined end time means +infinity
. inside(x, interval(x1, x2)) <=> not(x < x1) /\ not(x2 < x)
. inside(x, interval(undefined, x2)) <=> not(x2 < x)
. inside(x, interval(x1, undefined)) <=> not(x < x1)
. inside(x, interval(undefined, undefined))

. inside(x, interval(x1, d)) <=> not(x < x1) /\ not(x1+d < x)

. disjoint(t1 , t2) <=> not((startTime(t1) <= endTime(t2) /\ startTime(t2) <=
endTime(t1)) \/ (startTime(t2) <= endTime(t1) /\ startTime(t1) <= endTime(t2)))

. union(t1, undefined)=undefined
. union(undefined, t2)=undefined
. (not disjoint(interval(x1, x2), interval(y1, y2))) =>
 union(interval(x1, x2), interval(y1, y2)) = interval(min(x1, y1), max(x2, y2))
```

```
. startTime(interval(x1, x2)) = x1
. startTime(interval(x1, d)) = x1
. startTime(undefined) = undefined

. endTime(interval(x1, x2)) = x2
. endTime(undefined) = undefined

. endTime(interval(x1, d)) = x1 + d
. endTime(interval(undefined, x2)) = undefined
. endTime(interval(x1, undefined)) = undefined

. duration(interval(x1, x2)) = x2 − x1
. duration(interval(x1, d)) = d
. duration(undefined) = undefinedDuration



. disjoint(t1, t2) => union(t1, t2) = undefined

end


spec ValiditySpace =

sort geographicalRegion
sort country
sort city
sort state

%% generators
op geographicalRegion : country −> geographicalRegion
op geographicalRegion : city −> geographicalRegion
op geographicalRegion : state −> geographicalRegion

pred inside : geographicalRegion * geographicalRegion
pred testIntersectSpace : geographicalRegion * geographicalRegion

op interSpace : geographicalRegion * geographicalRegion −> geographicalRegion
op union : geographicalRegion * geographicalRegion -> geographicalRegion


forall s1, s2, s3 : geographicalRegion
. inside(s1, s2) /\ inside(s2, s3) => inside(s1, s3)
. inside(s1, s1)

. inside(s1, interSpace(s2, s3)) <=> (inside(s1, s2) /\ inside(s1, s3))
. inside(s1, union(s2, s3)) <=> (inside(s1, s2) \/ inside(s1, s3))

end

spec SequenceNode =
 Resource
 then

sort sequenceNode

%% generators
op emptySequence : sequenceNode
op seq : resource * resource -> sequenceNode
op seq : resource * resource * nat -> sequenceNode
op seqWithNext : resource -> sequenceNode
op seqWithPrev : resource -> sequenceNode
op seqWithOrdinal : nat -> sequenceNode

%% operations
op next : sequenceNode -> resource
op previous : sequenceNode -> resource
op ordinal : sequenceNode -> nat

pred hasNext : sequenceNode
pred hasPrevious : sequenceNode
pred hasOrdinal : sequenceNode

forall x, y : resource, n : nat
. next(seq(x, y)) = y
. previous(seq(x, y)) = x
. next(seqWithNext(x)) = x
. next(seqWithPrev(x)) = undefined
. previous(seqWithNext(x)) = undefined
. previous(seqWithPrev(x)) = x
. ordinal(seq(x, y, n)) = n
. ordinal(seqWithOrdinal(n)) = n
. hasPrevious(s) /\ ordinal(s) = n => ordinal(previous(s)) = n-1
. hasNext(s) /\ ordinal(s) = n => ordinal(next(s)) = n+1

. hasNext(seqWithNext(x))
. hasNext(seq(x, y))
. hasNext(seq(x, y, n))
. hasPrevious(seqWithPrev(x))
. hasPrevious(seq(x, y))
. hasPrevious(seq(x, y, n))
. hasOrdinal(seq(x, y, n))
. hasOrdinal(seqWithOrdinal(n))


end



spec SET[sort Elem] =
  sort set[Elem]
  %%generator
  op emptyset : set[Elem]

  pred element : Elem * set[Elem]
  pred equal : set[Elem] * set[Elem]

  op ___ union ___ : set[Elem] * set[Elem] -> set[Elem]
  op add : Elem * set[Elem] -> set[Elem]
  op{___} : Elem -> set[Elem];

  forall x: Elem . {x} union {x} = {x}

end
```

```
spec Causality =

Resource

then
SET[sort resource]
then
sort causality

%% generators
op emptyCause : causality
op addEndCause : set[resource] * causality -> causality
op addHasCause : set[resource] * causality -> causality

pred equal : causality * causality

op getEndCause : causality -> set[resource]
op getHasCause : causality -> set[resource]

op unionCause : causality * causality -> causality

forall r1 : resource, c1, c2, c3 : causality
. equal(c1, c1)
. equal(c1, c2) /\ equal(c2, c3) => equal(c1, c3)
. equal(c1, c2) => equal(c2, c1)

. getEndCause(emptyCause) = emptyset
. getEndCause(addEndCause({r1}, c1))= add(r1, emptyset) union getEndCause(c1)
. getEndCause(addHasCause({r1}, c1))= getEndCause(c1)

. getHasCause(emptyCause) = emptyset
. getHasCause(addEndCause({r1}, c1))= getHasCause(c1)
. getHasCause(addHasCause({r1}, c1))= add(r1, emptyset) union getHasCause(c1)



spec R1 =
 sort s1
end
```

```
spec Rn =
 sort sn
end

spec Reference = R1 and Rn
then
 SET[sort s1] and SET[sort sn]
then
%% we consider that each property is multi—valued

sort reference

%% generators
op emptyReference : reference

op addP1 : reference * s1 —> reference
%%...
op addPn : reference * sn —> reference

op getP1 : reference —> set[s1]
%%...
op getPn : reference —> set[sn]

pred equal : reference * reference

forall r, r1, r2, a : reference, v1 : s1, vn : sn

%% for each Pi
. getP1(addP1(r, v1)) = add(v1, emptyset) union getP1(r)
. getP1(emptyReference) = emptyset
. getP1(addPn(r, vn)) = getP1(r)
. equal(r1, r2) <=> equal(getP1(r1), getP1(r2)) /\ equal(getPn(r1), getPn(r2))

end



spec Provenance = Reference then

sort provenance
sort reference

%% generators
op emptyProvenance : provenance
op addReference : provenance * reference —> provenance

pred equal : provenance * provenance
pred containedIn : provenance * provenance
pred containsReference : provenance * reference

op union : provenance * provenance —> provenance

forall p1, p2, p3 : provenance, r1, r2, r3 : reference
. containsReference(addReference(p1, r1), r2) <=> (equal(r1, r2) \/
containsReference(p1, r2))
. not containsReference(emptyProvenance, r1)

. containedIn(addReference(p1, r1), p2) <=> containsReference(p2, r1) /\
containedIn(p1, p2)
. containedIn(emptyProvenance, p2)
. not containedIn(addReference(p1, r1), emptyProvenance)

. equal(p1, p2) <=> containedIn(p1, p2) /\ containedIn(p2, p1)

end


spec A1 =
 sort s1
end

spec An =
 sort sn
end

spec Annotations = A1 and An

then
  SET[sort s1] and SET[sort sn]
```

```
then
sort annotations



%% Let A1, ..., An be the qualifiers that we consider as annotations
%% Let s1, ..., sn be the sorts of these qualifiers
%% (it can be a datatype like string, int, real, ... or the sort resource)
%% we consider that each attribute may be multi—valued

%%generators
op emptyAnnotations : annotations
op addA1 : annotations * s1 —> annotations
%% . . .
op addAn : annotations * sn —> annotations

op getA1 : annotations —> set[s1]
%% . . .
op getAn : annotations —> set[sn]

forall a : annotations, v1 : s1 %{... }% , vn : sn

. getA1(addA1(a, v1)) = {v1} union getA1(a)
. getA1(emptyAnnotations) = emptyset

%% . . .

. getAn(addAn(a, vn)) = {vn} union getAn(a)
. getAn(emptyAnnotations) = emptyset



end
```