Rendering OWL in LaTeX for Improved Readability: Extensions to the OWLAPI

A Thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science

by

Cogan M. Shimizu
B.S.C.S., Wright State University, 2016

2017
Wright State University

Wright State University
GRADUATE SCHOOL

May 10, 2017

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY Cogan M. Shimizu ENTITLED Rendering OWL in LaTeX for Improved Readability: Extensions to the OWLAPI BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF Master of Science.

Pascal Hitzler, Ph.D.
Thesis Director

Mateen Rizki, Ph.D.
Chair, Department of
Computer Science and Engineering

Committee on
Final Examination

Pascal Hitzler, Ph.D.

Michelle Cheatham, Ph.D.

Tanvi Banerjee, Ph.D.

Robert E. Fyffe, Ph.D.
Vice President for Research and
Dean of the Graduate School

ABSTRACT

Shimizu, Cogan M.. M.S., Department of Computer Science and Engineering, Wright State University, 2017. *Rendering OWL in LaTeX for Improved Readability: Extensions to the OWLAPI.*

As ontology engineering is inherently a multidisciplinary process, it is necessary to utilize multiple vehicles to present an ontology to a user. In order to examine the content of an ontology, formal logic renderings of the axioms appear to be a very helpful approach for some. This thesis introduces a number of incremental improvements to the OWLAPI's LaTeX rendering framework in order to improve the readability, concision, and correctness of OWL files translated into Description Logic and First Order Logic. In addition, we examine the efficacy of these renderings as vehicles for understanding an ontology.

Contents

List of Figures

List of Tables

Acknowledgment

To Pascal, for his encouragement down every wandering path.

This work was, in parts, supported by the National Science Foundation award 1440202 EarthCube Building Blocks: Collaborative Proposal: GeoLink – Leveraging Semantics and Linked Data for Data Sharing and Discovery in the Geosciences and Wright State University via the Graduate Council Scholar Fellowship. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation.

In Loving Memory of

William H. Shimizu, an inspiration.

Always.

1

Introduction

The Semantic Web is a dichotomy. It is both an active, growing area of research, as well a as an expansive ecosystem for the delivery and linkage of machine-readable knowledge.

As a field of research, there is a breadth and depth of activity that is simply astounding. This seems fitting for a field focused on the efficient representation of knowledge from any and all domains, even introspectively. It drives results behind the scenes in many applications, from Google's Knowledge Vault [3, 2] to major NSF initiatives [1] to "improve access, sharing, visualization, and analysis of all forms..." Ultimately, the Semantic Web, as a field, drives how knowledge is linked, published, reused, and analyzed across all fields of knowledge.

The Semantic Web, as an artifact, is closely related to the World Wide Web (WWW). This is ultimately unsurprising as they share the same goal: to proliferate knowledge in a widely accessible manner. They simply differ for whom they emphasize accessibility. Just as the WWW is an ecosystem of technologies and standards for sharing data amongst its human users, the Semantic Web is an analogous ecosystem for machines. Fundamentally, the Semantic Web is a way to ascribe to web content *meaning*. That is, to carefully describe the *semantics* of the content in a machine-readable way. Consequently, this enables

[1] https://www.earthcube.org/group/geolink

programmatic access, interpretation, and evaluation of knowledge previously encoded in an only *human*-readable format. Such an encoding, or model, is called an *ontology*.

In order to model a complex concept, it is expressed in terms of simpler concepts. These simpler concepts are used as building blocks and are heavily enriched with metadata that relate them to each other. In this way, we can represent a highly abstract and complex concept in a way that a machine can easily interpret. To this point, the Semantic Web thus enables computing with knowledge; a software system can leverage the relationships between concepts in order to extract latent information and make inferences about the content. Further, by reusing the same conceptual building blocks across multiple applications or knowledge bases, we can greatly increase our ability to link and share data over the web. These three main topics, model building, computation with knowledge, and information exchange and reuse, underpin the entire purpose of the Semantic Web.

The machine-readable formalisms that ontologies are based on are called *ontology languages* [4]. An ontology language can be nearly any knowledge representation language (e.g. taxonomies, modal logic, OWL, or first order logic). In some form or another, however, each one is based on some logic. Methods for representing knowledge logically have been incredibly rich and varied over the millenia, reaching all the way into antiquity with Aristotle [**?**] and continue as a modern, academic mainstay in Computer Science. Specific to the Semantic Web, there exist many standards for logic-based knowledge representation languages put forth by the World Wide Web Consortium (W3C), such as the Resource Description Framework (RDF) [11], the Rule Interchange Format (RIF) [7], and the Web Ontology Language (OWL) [15].

Ontology engineering is the process of encoding domain knowledge into a machine-readable format with respect to some (ideally) standardized, formalized ontology language. For this thesis, we are particularly interested in the OWL family of knowledge representation languages, as well as the family of description logics that support them. We provide a brief introduction in the next section. OWL is a popular, expressive ontology language that

benefits from a healthy community and active tool development.

Perhaps the most prominent of tools for the programmatic construction, manipulation, and rendering of an ontology is Stanford's Protégé [2] which is a sophisticated GUI tool for designing ontologies that is powered by the OWLAPI [5]. We also provide a brief introduction to the OWLAPI in the next section.

In order to promulgate the end goal of the Semantic Web, it is increasingly necessary to make the ontology engineering process more accessible to domain experts without necessitating that they also be experts in the Semantic Web.

There is a need to provide methods for visualizing the logical structure and formal content of an ontology under construction. For example, graphical representations are highly ambiguous and are therefore easy to misunderstand. In addition, some axiomatic structures have no intuitive visual analog. One tool, Visual Notation for OWL (VOWL) [12] attempts to render axioms graphically. However, there is some debate on the efficacy of this approach, especially for complex axioms as it is difficult to adequately visually represent some operations. We examine VOWL and the debate more closely in Section 3.1.

With no clear path forward in improving the graphical representation, this thesis instead examines the efficacy of rendering the logical structures natively. That is, translating the OWL syntax into DL or FOPL and rendering it in LaTeX. There is already some preliminary work that shows that using rules in the ontology engineering process leads to increased accuracy of the model [13].

For ontology developers and consumers intimately familiar with the logical and formal semantic underpinnings of OWL, the presentation of OWL files in a description logic syntax appears to be a very useful one for a quick assessment of expressivity and formal content. For those developers and consumers *not* as familiar with the formalizations of OWL, we intend to show that rules, written in FOPL, are even more useful over renderings

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

in other logical syntaxes. In summary, this thesis aims to prove the following.

Hypotheses:

- When presented in a Description Logic Syntax, users will more quickly and more correctly understand an ontology compared to renderings in Manchester syntax.
- When presented in a FOPL-Rule-like (FOLR) Syntax, users will more quickly and more correctly understand an ontology compared to renderings in Description Logic syntax.

As such, we have made extensive changes to the LaTeX rendering framework for the OWLAPI in order to generate syntactically correct and human readable renderings of an ontology in DL and FOPL.

Chapter Overview

The rest of the thesis is organized as follows.

Chapter 2: Preliminaries includes essential preliminary information. It briefly reviews description logics, first order predicate logic, and OWL.

Chapter 3: Related Work introduces other attempts to streamline the ontology engineering process. Specifically, we briefly examine alternative ways of specifying and interpreting the structure of an ontology (e.g. VOWL, ROWL, and OWLax [14]).

Chapter 4: Research Contributions gives an explanation into the changes made to the OWLAPI in order to facilitate the human readable LaTeX renderings in DL and FOL. We also introduce two GUI tools developed to leverage the OWLAPI for rendering.

Chapter 5: Evaluation covers the design of our evaluation and its results.

2

Preliminaries

As previously stated in the introduction, we intend to show that examining an ontology's axiomatization and formal content via description logic or rules syntax improves the ontology engineering process by facilitating ontology content understanding, for example, reducing the time spent developing an iteration of ontology design. We utilize this chapter to introduce (or provide an opportunity to re-acquaint with) some basic concepts of description logics (DL) in Section 2.1, first order predicate logic (FOPL) in Section 2.2, and the Web Ontology Language (OWL) in Section 2.3.

2.1 Description Logics

In this section, we briefly introduce the syntax and semantics used for DLs (with emphasis on $\mathcal{SHOIN}$), outline the motivation for using DLs for knowledge representation purposes, and show how OWL is directly related to DLs. For a closer examination of Description Logics and their history and applications, see [1].

2.1.1 Syntax & Semantics

Ontologies are constructed from two different semantic entities, *atomic concepts* and *atomic roles*. Together they are referred to as *atomic symbols*. Atomic concepts and roles can be considered to be the fundamental building blocks of an ontology, from which we can build arbitrarily complex descriptions via concept and role constructors. The sorts of constructors included in a language dictate its expressivity (and its decidability). Further, the allowed constructors are denoted in the name. For example, the language $\mathcal{ALC}$ is $\mathcal{AL}$ extended with $\mathcal{C}$ (for "complement"), thus allowing the negation of arbitrary concepts.

In this case, we are particularly interested in the DL $\mathcal{SHOIN}$ extended with datatypes. $\mathcal{S}$ is used denote the language $\mathcal{ALC}$ extended with transitive roles. Then, the following extensions $\mathcal{HOIN}$ allow role hierarchy (e.g. for two concepts R, S, we may express $R \sqsubseteq S$), nominals or individuals, inverse roles, and number restrictions, respectively. See Table 2.1 for a comprehensive list of possible extensions to DLs. To form $\mathcal{SHOIN}$-concepts, letting A be an atomic concept and R be an atomic role, we use the following syntax rule:

$$
\begin{array}{rcll}
C, D & \rightarrow & A \mid & \text{(atomic concept)} \\
 & & \top \mid & \text{(top concept)} \\
 & & \bot \mid & \text{(bottom concept)} \\
 & & \neg A \mid & \text{(atomic negation)} \\
 & & \neg C \mid & \text{(concept negation)} \\
 & & C \sqcap D \mid & \text{(intersection)} \\
 & & \forall R.C \mid & \text{(value restriction)} \\
 & & \exists R.C \mid & \text{(full existential quantification)} \\
 & & \leq nR \mid & \text{(at-least restriction)} \\
 & & \geq nR \mid & \text{(at-most restriction)} \\
 & & \{a\} & \text{(nominal)}
\end{array}
$$

To define a formal semantics for $\mathcal{SHOIN}$-concepts, we consider an interpretation $\mathcal{I}$ of some domain of interest $\Delta^{\mathcal{I}}$. Then for some interpretation function each atomic concept A is mapped to $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ and define a mapping for every atomic role R to a binary relation such that $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. We then inductively define the semantics of the other allowed concepts in Table 2.1.

$\{a\} = a \in \Delta^{\mathcal{I}}$	Nominal/Individual
$\top^{\mathcal{I}} = \Delta^{\mathcal{I}}$	Universal Concept
$\bot^{\mathcal{I}} = \emptyset$	Bottom Concept
$(\neg A)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus A^{\mathcal{I}}$	Atomic Negation
$(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$	Intersection
$(C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}$	Union
$(\forall R.C)^{\mathcal{I}} = \left\{a \in \Delta^{\mathcal{I}} \mid \forall b.(a,b) \in R^{\mathcal{I}} \rightarrow b \in C^{\mathcal{I}}\right\}$	Value Restriction
$(\exists R.C)^{\mathcal{I}} = \left\{a \in \Delta^{\mathcal{I}} \mid \exists b.(a,b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\right\}$	Full Existential Quantification
$(\geq nR)^{\mathcal{I}} = \left\{a \in \Delta^{\mathcal{I}} \mid \mid \left\{b \mid (a,b) \in R^{\mathcal{I}}\right\} \mid \geq n\right\}$	At-least Restriction
$(\leq nR)^{\mathcal{I}} = \left\{a \in \Delta^{\mathcal{I}} \mid \mid \left\{b \mid (a,b) \in R^{\mathcal{I}}\right\} \mid \leq n\right\}$	At-most Restriction

Figure 2.1: The semantics for the concept constructors included in $\mathcal{SHOIN}$.

2.1.2 Knowledge Representation

DLs are decidable fragments of FOL enhanced with a formal semantics, thus allowing both humans and machines to precisely and unambiguously interpret intended meanings within the ontology. Additionally, the formal semantics enables inference of latent or implicit knowledge from the explicitly stated facts in the ontology. The computation of these new inferences is known as reasoning. How the ontology is logically structured dictates exactly what can be inferred from its contents.

A DL ontology is split into two different components. Terminologies (also known as the *TBox*) are collections of statements on how concepts and roles relate to each other. The world description is eponymous: a description of the world, but in terms of the relations specified in the terminology. The world description is also known as the *ABox*, as it contains *assertional* axioms.

Terminologies

Terminologies consist of a set of terminological axioms. Generally, these axioms have the form

$$C \sqsubseteq D \ (R \sqsubseteq S) \qquad \text{or} \qquad C \equiv D \ (R \equiv S)$$

$\mathcal{AL}$	Attributive Language
$\mathcal{C}$	Concept Negation
$\mathcal{S}$	$\mathcal{AL}$ with role transitivity
$\mathcal{H}$	Role hierarchy
$\mathcal{O}$	Nominals
$\mathcal{I}$	Inverse Roles
$\mathcal{N}$	Number Restriction
$\mathcal{D}$	Datatypes
$\mathcal{F}$	Role Functionality
$\mathcal{Q}$	Qualified Cardinality Restriction
$\mathcal{R}$	Generalized Role Inclusion
$\mathcal{E}$	Existential Role Restriction

Table 2.1: Above is a list of possible extensions in the $\mathcal{AL}$-family of languages.

where C, D are concepts and R, S are roles. Respectively, these are known as *inclusions* and *equalities*. We provide an example Terminology in Figure 2.2.

World Description

In the world description or ABox, the domain of interest is stated in terms of the concepts and roles defined in the TBox. The statements included in the ABox are called *concept assertions* and *role assertions*. Respectively, they have the form

$$C(a), \qquad\qquad R(b, c).$$

That is, for $C(a)$, we say that a belongs to the interpretation of C. For $R(b, c)$, we say that c is a filler of the role R for b. In Figure 2.3, we provide an example ABox to accompany

$$\text{fatherOf} \sqsubseteq \text{parentOf} \tag{2.1}$$
$$\text{motherOf} \sqsubseteq \text{parentOf} \tag{2.2}$$
$$\text{Woman} \equiv \text{Person} \sqcap \text{Female} \tag{2.3}$$
$$\text{Man} \equiv \text{Person} \sqcap \text{Male} \tag{2.4}$$
$$\text{Mother} \equiv \text{Woman} \sqcap \exists \text{hasChild.Person} \tag{2.5}$$
$$\text{Father} \equiv \text{Man} \sqcap \exists \text{hasChild.Person} \tag{2.6}$$
$$\text{Parent} \equiv \text{Mother} \sqcup \text{Father} \tag{2.7}$$

Figure 2.2: An Example TBox (Terminology).

$$\text{Father}(Peter) \tag{2.8}$$
$$\text{hasChild}(Mary, Paul) \tag{2.9}$$
$$\text{motherOf}(Michelle, Peter) \tag{2.10}$$
$$\text{Man}(Paul) \tag{2.11}$$

Figure 2.3: An Example ABox (World Description).

the TBox in Figure 2.2.

2.2 First Order Predicate Logic

First order predicate logic is an extension of propositional logic. That is, FOPL allows one to express certain notions that cannot be expressed in propositional logic.

Consider a set of arbitrary concepts of particular interest; we call this a *universe* which is analogous to the *domain of interest* from the previous section. Now, consider two disjoint subsets of the universe. In propositional logic, it is not possible to express a relation between these two subsets such that the relation is not surjective, that is to say, the relation only holds *sometimes*. FOPL allows us to express these sorts of relations in terms of existential or universal quantifiers, denoted via the symbols $\exists$ and $\forall$, respectively. The quantifiers, and thus FOPL, allow us a level of expressivity that more closely matches reality. In the next section we see how these relations are constructed.

2.2.1 Syntax & Semantics

The following definitions are taken from [16].

First, we define *terms* inductively.

1. Each variable is a term.
2. If f is a function symbol with arity k, and if $t_1, \ldots, t_k$ are terms, then $f(t_1, \ldots, t_k)$ is a term.

Then, *formulas* are defined inductively as follows.

1. If P is a predicate symbol with arity k, and if $t_1, \ldots, t_k$ are terms, then $P(t_1, \ldots, t_k)$ is a formula.
2. For each formula F, $\neg F$ is a formula.
3. For all formulas F and G, $(F \wedge G)$ and $(F \vee G)$ are formulas.
4. If x is a variable and F is a formula, then $\exists x F$ and $\forall x F$ are formulas

A formula is considered to be atomic if it is constructed using only (1). Finally, we describe the semantics of predicate logic. We note, too, the strong connections to the semantics of Description Logic. We define a *structure*, to be a pair $\mathcal{A} = (U_\mathcal{A}, I_\mathcal{A})$, where $U_\mathcal{A}$ is an arbitrary, non-empty set called the domain of interest and $I_\mathcal{A}$ is a mapping such that it maps

- each k-ary predicate symbol P to a k-ary predicate on $U_\mathcal{A}$ (if $I_\mathcal{A}$ is defined on P).
- each k-ary function symbol f to a k-ary function on $U_\mathcal{A}$ (if $I_\mathcal{A}$ is defined on f).
- each variable x to an element of $U_\mathcal{A}$ (if $I_\mathcal{A}$ is defined on P).

For further examination, please see [4].

$$\text{fatherOf}(x, y) \rightarrow \text{parentOf}(x, y) \tag{2.12}$$
$$\text{motherOf}(x, y) \rightarrow \text{parentOf}(x, y) \tag{2.13}$$
$$\text{Person}(x) \wedge \text{Female}(x) \rightarrow \text{Woman}(x) \tag{2.14}$$
$$\text{Person}(x) \wedge \text{Male}(x) \rightarrow \text{Man}(x) \tag{2.15}$$
$$\text{Woman}(x) \wedge \text{Person}(y) \wedge \text{hasChild}(x, y) \rightarrow \text{Mother}(x) \tag{2.16}$$
$$\text{Man}(x) \wedge \text{Person}(y) \wedge \text{hasChild}(x, y) \rightarrow \text{Father}(x) \tag{2.17}$$
$$\text{Parent}(x) \wedge \neg\text{Father} \rightarrow \text{Mother}(x) \tag{2.18}$$
$$\text{Parent}(x) \wedge \neg\text{Mother} \rightarrow \text{Father}(x) \tag{2.19}$$

Figure 2.4: An example FOPL program showing the same information as Figure 2.2.

2.2.2 Knowledge Representation

First Order Predicate Logic is highly expressive. Unfortunately, its expressivity is a double edged sword as, in general, FOPL is undecidable. However, that does not detract from its usefulness as a vehicle for representation, in particular the use of FOPL rules.

A FOPL rule is a formula that contains a single implication. The left hand side (LHS) of the implication is called the antecedent, or body, of the rule. The right hand side (RHS) of the implication is called the consequence, or head, of the rule. The antecedent must be in negation normal form and the consequence must be atomic.

It is a standing debate that rules are a more intuitive method for conveying logical statements than description logic axioms [13, 17]. The strong restrictions on the rule's structure are partially what makes a rule so readable. In Section 4.4 we cover translations from Description Logics to FOL rules.

2.2.3 Rule Conversion

From the previous section, we know that a *rule*, has the form $F \rightarrow G$, where G is atomic. Now, we briefly cover the conversion of a FOL formula, where G is non-atomic, to a rule. There are two initial requirements that must be met in order for such a conversion to be successful.

1. F must not contain a universal quantifier.

2. G must not contain an existential quantifier.

If these two requirements are met, then the following steps are taken for the formula-to-rule conversion.

1. Disambiguate all implications (i.e. $F \rightarrow G \equiv \neg F \vee G$).

2. Convert the resulting formula into negation normal form (NNF).

3. Move quantifiers to outermost scope.

4. Perform operations to move desired atomic concept to the end of the formula.

5. Convert resulting formula back into an implication.

The result has the desired rule form.

2.3 The Web Ontology Language

The Web Ontology Language is a family of knowledge representation languages specifically for modelling ontologies. The current specification, as endorsed by W3C, includes three different variants- also called sublanguages or *species*- of OWL: OWL-Lite, OWL-DL, and OWL-Full. In addition, there are two versions, OWL and OWL2. Each of the OWL species has a different level of expressivity, and thus a different associated description logic. The expressivity of the sublanguage also determines its scalability, i.e. the complexity of reasoning in the sublanguage.

This allows an ontology engineer or publisher to choose an expressivity (and thus scalability) that more appropriately fits their use-case. Table 2.2 shows the different sublanguages and to which DL they are paired. We note that each of the sublanguages is also

OWL Variant	DL	Worst Case Complexity
OWL-Lite	$\mathcal{SHOIF}^{(\mathcal{D})}$	ExpTime-Complete
OWL-DL	$\mathcal{SHOIN}^{(\mathcal{D})}$	NExpTime-Complete
OWL2-DL	$\mathcal{SROIQ}^{(\mathcal{D})}$	NExpTime-Hard
OWL/OWL2-Full	Not DLs	Undecidable

Table 2.2: The OWL Family and their associated description logics. See Table 2.1 for definitions of these DL extensions and the complexities[2] of their concept satisfiability.

```
Ontology(<http://www.example.com/family.owl>

Declaration(Class(:Female))
Declaration(Class(:Person))
Declaration(Class(:Woman))
SubClassOf(:Woman ObjectIntersectionOf(:Female :Person))
        )
```

Figure 2.5: OWL Functional Syntax

hierarchical. That is, every OWL-Lite ontology is a valid OWL-DL ontology and every OWL-DL ontology is a valid OWL-Full Ontology.

The OWL family also supports many different syntaxes: OWL2 functional syntax, OWL RDF/XML, OWL2 XML, and Manchester Syntax.

We provide a small example that is demonstrative of the *flavor* of each syntax in Figures 2.5-2.8. The examples portray exactly the same information; we choose an excerpt from the logic program in Figure 2.2. Each of these examples was generated using Protégé, a popular ontology creation tool. We describe some of Protégé's capabilities in Section 3.2.1. For a more in-depth examination of the syntaxes, semantics, and specification of the OWL family, see [5, 15]. Additionally, we provide an example of an entire ontology in OWL/XML in Appendix A.

```
<rdf:RDF xmlns=...>
    <owl:Ontology rdf:about="family.owl"/>

    <owl:Class rdf:about="\#Female"/>

    <owl:Class rdf:about="\#Person"/>

    <owl:Class rdf:about="\#Woman">
        <rdfs:subClassOf>
            <owl:Class>
                <owl:intersectionOf rdf:parseType="Collection">
                    <rdf:Description rdf:about="\#Female"/>
                    <rdf:Description rdf:about="\#Person"/>
                </owl:intersectionOf>
            </owl:Class>
        </rdfs:subClassOf>
    </owl:Class>
</rdf:RDF>
```

Figure 2.6: RDF/XML

```
<Ontology ontologyIRI="http://www.example.com/family.owl">
    <Declaration>
        <Class IRI="\#Person"/>
    </Declaration>
    <Declaration>
        <Class IRI="\#Woman"/>
    </Declaration>
    <Declaration>
        <Class IRI="\#Female"/>
    </Declaration>
    <SubClassOf>
        <Class IRI="\#Woman"/>
        <ObjectIntersectionOf>
            <Class IRI="\#Female"/>
            <Class IRI="\#Person"/>
        </ObjectIntersectionOf>
    </SubClassOf>
</Ontology>
```

Figure 2.7: OWL/XML Syntax

```
Ontology: <http://www.example.com/family.owl>
Class: <http://www.example.com/family.owl\#Female>
Class: <http://www.example.com/family.owl\#Person>
Class: <http://www.example.com/family.owl\#Woman>
    SubClassOf:
        <http://www.example.com/family.owl\#Female>
         and <http://www.example.com/family.owl\#Person>
```

Figure 2.8: Manchester Syntax

```
@prefix : <http://www.example.com/family.owl\#> .

<http://www.example.com/family.owl> rdf:type owl:Ontology .

:Female rdf:type owl:Class .
:Person rdf:type owl:Class .
:Woman rdf:type owl:Class ;
       rdfs:subClassOf [ owl:intersectionOf ( :Female
                                              :Person
                                            ) ;
                         rdf:type owl:Class
                       ] .
```

Figure 2.9: RDF/Turtle Syntax

3

Related Work

As previously mentioned, the ontology engineering process can be highly iterative. In this chapter, we discuss a number of recently developed tools for streamlining this process. Specifically, we chose technologies that provide alternative methods for visualizing an ontology during the design process. However, of these tools, none touch on translating OWL to FOPL. In fact, to the author's knowledge, and although the *mechanisms* for such translations are very well known [1, 9, 8], there is no existing tool that programmatically translates OWL or DL to FOPL.

In Section 3.1, we introduce the Visual Notation for OWL (VOWL). Section 3.2 describes Protégé and two plugins providing alternative means for programmatic ontology development, ROWL (FOL Rules to OWL) and OWLax (axioms from graphical relationships). Finally, Section 3.3 provides a brief description of the OWLAPI and its rendering framework.

3.1 Visual Notation for OWL: VOWL

The Visual Notation for OWL Ontologies (VOWL) is a specification for a visual language that represents ontologies to a user [12]. The specification defines a number of *graphical*

primitives that are used to build the alphabet of the visual notation. The alphabet is then used to generate a force-directed graph that visualizes the ontology. VOWL is implemented as a Protégé plugin and as a web service.

There has been some debate on the efficacy of representing complex axioms. Figure 3.2 is an exact copy of the rendering of an ontology benchmark utilized in [12]. Of particular interest are the shaded areas labeled 2 and 10. In Area 2, the *visual semantics* of a circle labeled "disjoint class" are not clear. Furthermore, axioms involving the traditional set operations (i.e. conjunction, disjunction, and complement) are unclear. Area 10, and those edges emitted from the area, are good examples of this. The specification is not forthcoming on the semantics of the dashed line. It is used to indicate set operators or class disjointness, but does not impart directionality (i.e. does not exactly and unambiguously show how the class is related to the set operator). For example, with respect to Figure 3.2 how is Class 1 related to the $\neg$ and the subsequent $\cap$? We argue that these inexact, graphical primitives serve to obfuscate the relationships between concepts in the presence of complex axioms. However, [12] does clearly and intuitively communicate which entities are Classes, Properties, or Datatypes. An additional example, of a non-synthetic variety, of a rendering in VOWL is provided in Figure 3.1. The graph itself is aesthetically pleasing, but the overall structure is obscured by the level of detail. Additionally, subproperty relations seem especially confusing amidst all the other visualized relations.

As a final remark, the force-directed graph visualization does not take into account any semantics in the final visualization of the ontology. That is, the visualization itself is dependent upon graph metrics such as node degree and centrality, rather than emphasizing semantically important relationships. That is, classes (or concepts) that are tightly semantically coupled have no guarantee that this coupling is emphasized, or even clear, in the final visualization. Moreover, as the ontology engineering process is frequently iterative, there is no guarantee that semantically similar iterations of the ontology have similar visualizations. This can make visualizing the contents and semantics of an ontology under

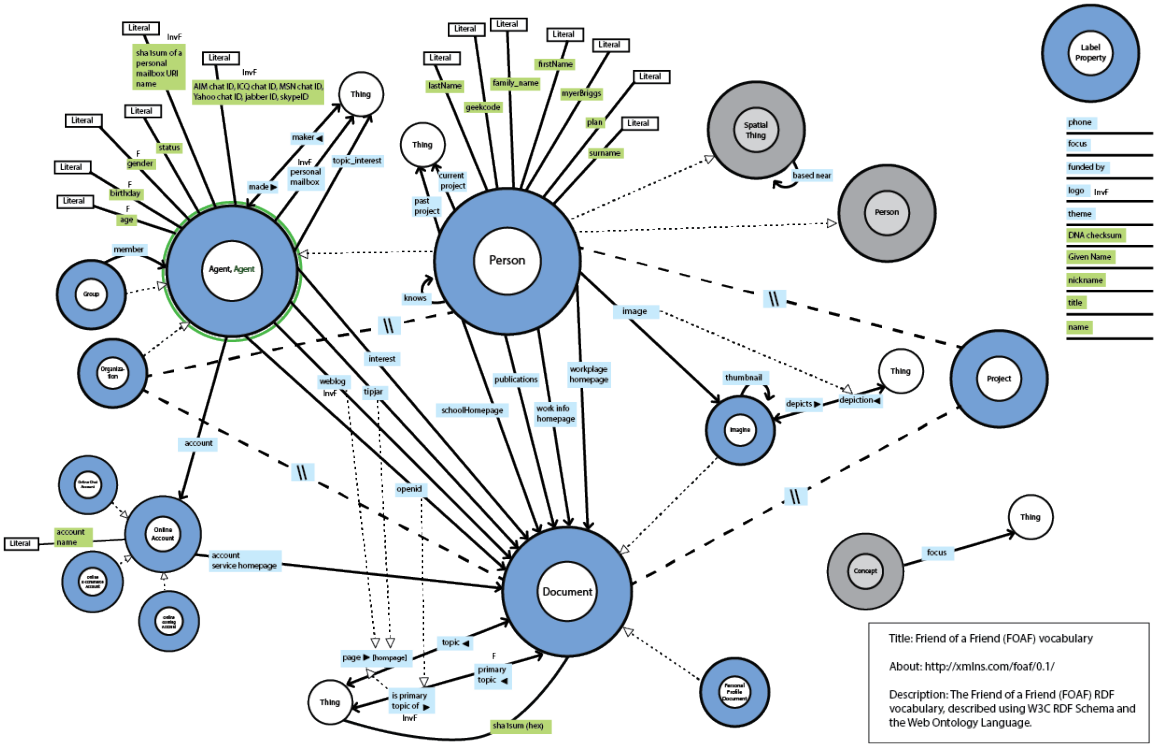

Figure 3.1: An example of rendering of an ontology using the Visual Notation for OWL Ontologies.

development difficult during the iterative design process.

3.2 Protégé and Plugins

3.2.1 Protégé

Protégé is an open-source, free-ware ontology engineering platform. It is developed and maintained by Stanford University. Protégé is powered by the OWLAPI, an incredibly powerful API for programmatically creating, manipulating, and rendering ontologies. We briefly cover its rendering framework in Section 3.3. Additionally, in order to address the needs of a constantly growing audience, it supports a number of plugins, two of which are

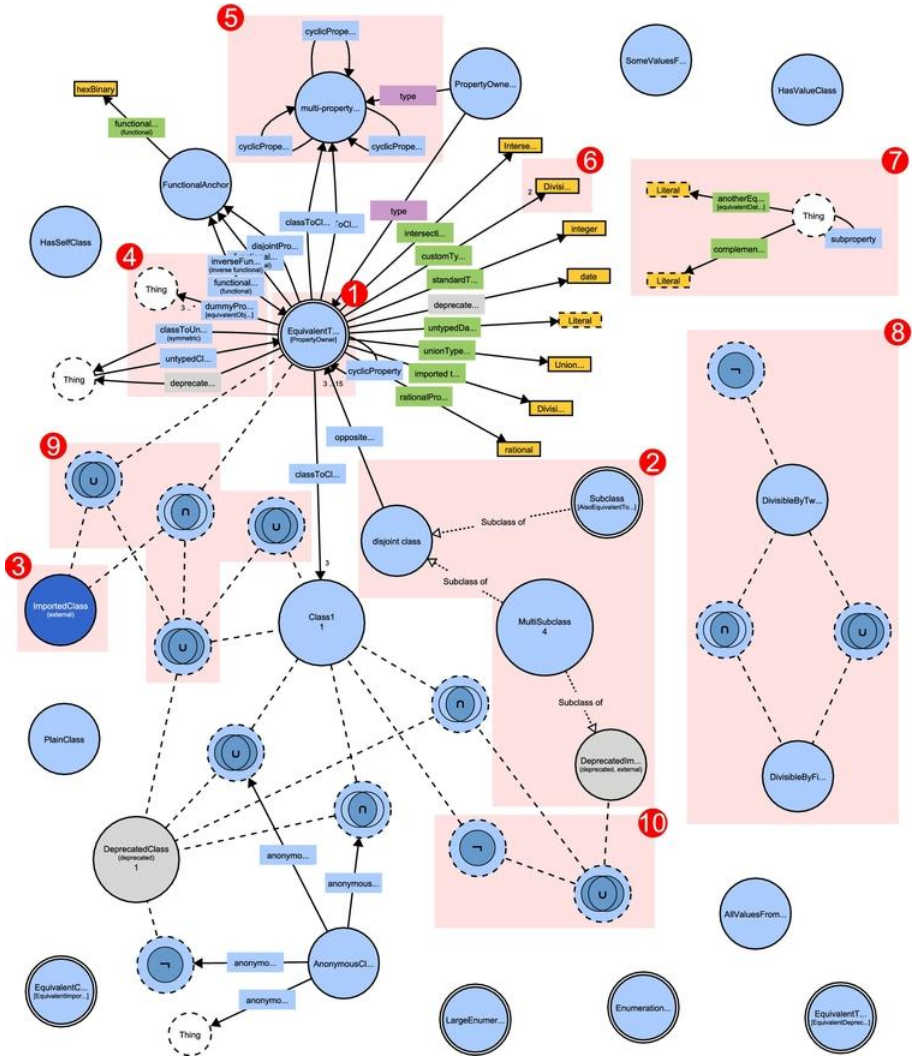

Figure 3.2: The benchmark ontology for VOWL. This figure is exactly reproduced from [12].

detailed below in Sections 3.2.2 and 3.2.3.

3.2.2 ROWLTab: Rule-based OWL Modeling

The ROWLTab is a plugin for Protégé developed by the Data Semantics Laboratory at Wright State University. It provides users with an alternate means to generate OWL axioms by providing *rules* to the system. The plugin automatically attempts to convert these rules into appropriate OWL-DL axioms, if possible.

Preliminary usage statistics show that creating ontologies in this manner leads to a number of improvements in the ontology engineering process [13]. For example, in the author's experience, domain experts frequently have a difficult time examining the formal axioms in an ontology. Having this tool early in the design phase is extremely helpful as it can save and reload axioms during the iterative process. However, it is not possible, at this point in time, to load in an existing ontology and generate the rules from the ontology. In Chapter 6, we discuss the possible integration of contributions described in Chapter 4 as future work.

3.2.3 OWLAx: A Protégé Plugin

OWLAx is another plugin for Protégé developed by the Data Semantics Lab at Wright State University. It allows us to *begin* with a graphical representation. The plugin will then attempt to translate the graphical representation into the appropriate OWL axioms.

In general, an ontology modeling session, such as a GeoVocamp [1], begins graphically. That is, it is easy to describe the overall structure, relations, classes, and properties on a whiteboard. The OWLax plugin facilitates this strategy by providing that whiteboard virtually. The plugin will attempt to create axioms based on the graphical representation. This approach also allows users to quickly specify disjointness of classes and domains and

[1] http://ontolog.cim3.net/wiki/GeoVoCamp.html

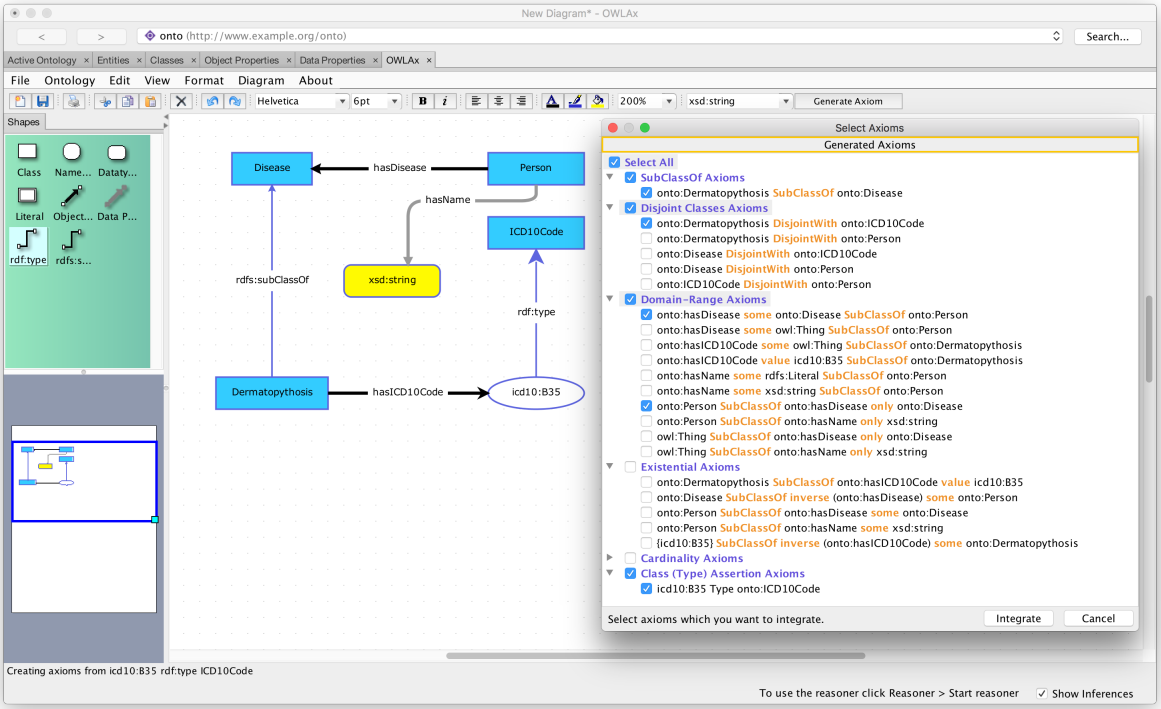

Figure 3.3: An example of generating an ontology using the OWLax Protégé plugin.

ranges of properties. While this plugin is a great addition to the ontology development ecosystem it does not contribute to the formal axiom development. [14]

3.3 The OWLAPI

The OWLAPI [5], which is a powerful tool for the programmatic construction, manipulation, and rendering of ontologies, has for considerable time had limited support for the rendering of OWL ontologies in description logic syntax via LaTeX. Unfortunately, this LaTeX rendering framework, which outputs description logic in a LaTeX source file, was never developed beyond an early experimental stage. As a consequence, translations suffered from a number of syntax errors and poor readability of the output. In practice, translations were further impacted by the presence of illegal characters in the LaTeX source, thus preventing

nearly all renderings from typesetting. In Chapter 4, we cover a number of changes made to the OWLAPI as partial fulfillment of this thesis.

4

Research Contributions

This chapter describes the entirety of concrete changes made to the OWLAPI for advancing human-readable renderings of OWL files, as well as two tools for making these renderings accessible. Tutorials for using these tools can be found online.[1] The changes to the OWLAPI herein described, at the time of this writing, have been submitted to the source developers and maintainers. Some changes will be present in version 5.0.6.

The remainder of this chapter can be partitioned into two parts. Section 4.1 describes characteristics of the LaTeX rendering framework that span the development of both tools. Sections 4.3 and 4.4 describe the rendering tools: OWL to Description Logic and OWL to First Order Logic, respectively. Chapter 5 will cover the evaluation of both the described changes as well as the general efficacy of rendering an ontology in a logic.

4.1 LaTeX Rendering in the OWLAPI

4.1.1 Overview

For immediate context, we provide a very brief overview of how the OWLAPI renders an ontology in LaTeX. First, the renderer examines a ontology that has been loaded into

[1] http://dase.cs.wright.edu/content/owl2dl-rendering

memory. Then, for each entity, (i.e Class, Object Property, Data Property, Individual, and Datatype) in the ontology, it prints associated axioms and facts. An axiom is considered to be *associated* to an entity if the entity appears somewhere in the axiom. For example, the axiom

$$\textit{DisjointClasses}(\text{A, B, C})$$

is associated with classes A, B, and C. While this does result in redundantly rendered axioms (i.e. the same axiom may occur in multiple entity subsections), we stress that the renderer is meant to summarize the entities in an ontology, rather than exhaustively enumerate all axioms in the ontology.

4.2 Reduction of Duplicate Axioms

Several OWL concepts provide a way for succinctly expressing pairwise relations (e.g. equivalence and disjointness). However, the translations of these concepts into description logic can potentially generate a huge number of axioms. For example, in order to express that n classes are mutually disjoint requires $2 \cdot \binom{n}{2}$ axioms. Furthermore, under the current framework all these axioms are related and will thus be printed in each class's section, for a total of $2n \cdot \binom{n}{2}$ axioms. This can quickly obscure the actual relationship between all the Classes. As such, we adopt the functional syntax as defined in the specification as follows (using an example from the Semantic Trajectory ODP)

$$\textit{disjoint}(\text{Attribute}, \text{Fix}, \text{MovingObject}, \text{Place}, \text{Segment}, \text{TimeEntity}, \text{Trajectory})$$

The equivalent axioms for expressing this single line is represented in Table 4.1.

$Attribute \not\sqsubseteq Fix$	$Fix \not\sqsubseteq Attribute$
$Attribute \not\sqsubseteq MovingObject$	$Fix \not\sqsubseteq MovingObject$
$Attribute \not\sqsubseteq Place$	$Fix \not\sqsubseteq Place$
$Attribute \not\sqsubseteq Segment$	$Fix \not\sqsubseteq Segment$
$Attribute \not\sqsubseteq TimeEntity$	$Fix \not\sqsubseteq TimeEntity$
$Attribute \not\sqsubseteq Trajectory$	$Fix \not\sqsubseteq Trajectory$
$MovingObject \not\sqsubseteq Attribute$	$Segment \not\sqsubseteq Attribute$
$MovingObject \not\sqsubseteq Fix$	$Segment \not\sqsubseteq Fix$
$MovingObject \not\sqsubseteq Place$	$Segment \not\sqsubseteq MovingObject$
$MovingObject \not\sqsubseteq Segment$	$Segment \not\sqsubseteq Place$
$MovingObject \not\sqsubseteq TimeEntity$	$Segment \not\sqsubseteq TimeEntity$
$MovingObject \not\sqsubseteq Trajectory$	$Segment \not\sqsubseteq Trajectory$
$TimeEntity \not\sqsubseteq Attribute$	$Place \not\sqsubseteq Attribute$
$TimeEntity \not\sqsubseteq Fix$	$Place \not\sqsubseteq Fix$
$TimeEntity \not\sqsubseteq MovingObject$	$Place \not\sqsubseteq MovingObject$
$TimeEntity \not\sqsubseteq Place$	$Place \not\sqsubseteq Segment$
$TimeEntity \not\sqsubseteq Segment$	$Place \not\sqsubseteq TimeEntity$
$TimeEntity \not\sqsubseteq Trajectory$	$Place \not\sqsubseteq Trajectory$
$Trajectory \not\sqsubseteq Attribute$	
$Trajectory \not\sqsubseteq Fix$	
$Trajectory \not\sqsubseteq MovingObject$	
$Trajectory \not\sqsubseteq Place$	
$Trajectory \not\sqsubseteq Segment$	
$Trajectory \not\sqsubseteq TimeEntity$	

Table 4.1: In order to rigorously define that seven classes are mutually disjoint, it is necessary to express it in 42 axioms. This very quickly obscures the fact that they are expressing pairwise disjoint relationships.

4.2.1 Limitations

In Section 4.4, we see that the OWL2FOL tool provides only a direct translation to first order logic and *not* to rules, in most cases. This is due to limitations inherent to adapting the OWLAPI's rendering framework. The OWLAPI disconnects the underlying data structure representing an axiom from its traversal via implementation of the visitor design pattern. For the OWL2DL tool (Section 4.3), this provides no problem.

The rendering framework considers an ontology to be a *forest*, where each axiom is a tree. The axiom is traversed in a stateless manner. That is, actions performed at each node of the tree are independent of actions occurring in parent nodes. A node in this "axiom tree" is either an operator, concept constructor or role constructor. At each of these nodes, the DL rendering is written to a LaTeX source file. However, rendering FOL *rules* from the OWL source cannot be done in such a traversal, as is done with the DL rendering. As is elaborated in Section 4.4, most OWL axioms have mappings into first order logic. Inconveniently, the one-to-one mapping from OWL to FOL is, in general, not immediately in rule format. Non-trivial manipulation of the FOL formula would be necessary to translate it to the rule format. As such, it would be necessary to develop an intermediate data structure that captures the behavior and structure of FOL. This is problematic for two reasons. First, developing and maintaining such a data structure is definitively outside the scope of the OWLAPI. Secondly, the structure of the OWLAPI interferes. That is, arbitrarily nested complex OWL classes prevent an intuitive way forward in utilizing the visitor design pattern to properly bind variables in the FOL formula.

4.2.2 Spacing & Alignment

At the top level, we have also made several quality of life improvements irrespective of the rendered language. Previously, axioms were rendered such that mathematical operators were embedded in ensuremath LaTeX commands. While this is convenient for having

plain text renderings of concept and property names, the subsections devoted to each OWL entity were disorganized and could be difficult to read.

To rectify this, an entity's associated axioms are now embedded in the align environment included as part of the amsmath LaTeX package. This allows us to align related axioms over their principal relation (i.e. $\equiv$, $\not\equiv$, $\sqsubseteq$, $\rightarrow$) or after a function name or argument.

4.2.3 Line-Breaking Heuristic

In some cases, axioms would result in an excessively long rendering (i.e. result in hbox overflow, placing text in or even beyond the page margin). For the most part, LaTeX handles itself in knowing when to break a line. However, this behavior does not occur in the math environments. As such, it was necessary to look into methods for preventing unacceptable overflow.

The first examined option was the LaTeX package breqn. This package is an experimental package that employs its own heuristics for breaking excessively long equations. Unfortunately, breqn's heuristics take into account only a select number of operators as potential breaking points. Due to the uncommon operators that description logic employs, breqn was unable to find appropriate breaking points.

The next option was the split environment from the LaTeX package amsmath. However, split does not dynamically split an equation; it is an entirely manual process. At this point, we developed our own heuristic to determine when the split environment would be necessary.

In the rendering tools, we examine the raw LaTeX source. First, we control for the LaTeX commands that are employed by the rendering framework and then count an empirically determined number of characters; we found 125 characters to be a reasonable equation

length before a newline would be required.

$$\text{DataGranule}(x_1) \rightarrow \geq 1 x_2 \text{ hasDataSet}(x_1, x_2) \wedge \text{DataSet}(x_2) \\ \wedge \leq 1 x_3 \text{ hasDataSet}(x_1, x_3) \wedge \text{DataSet}(x_3)$$

There are some limitations to this approach, as each entity's subsection is a single align environment. The split environment is thus, in turn, embedded in it. As such, if the antecedent of the principal operator of any axiom per subsection is sufficiently long, the line breaks may occur significantly into the margin. However, in an evaluation of 117 ontologies[2] rendered to Description Logic, the line-breaking heuristic did not display this anomalous behavior.

4.2.4 Namespaces & URIs

In natural language, especially in situations where context is unclear, it can be difficult to parse the exact semantics of a word. In OWL, every entity has its own "Uniform Resource Identifier." As such, this allows a reader to know exactly which semantics are used for the entity. It is generally customary to use a URI for this purpose. The URI is considered to be a *namespace* for the ontology. It is for this reason an OWL ontology may contain an external class with the same name as a class already existing in the ontology; that class would have a different URI.

When considering the entities in an axiom, it is important to use the entire URI. However, when rendering an entity (with the intent to be human readable), the entire URI can obfuscate the meaning. Consider the following DL rendering with full namespaces. It clearly would not even render completely on the page. In addition, it has characters that do not render properly and the # in particular would normally prevent the LaTeX source from

[2] pulled from ontologydesignpatterns.org

typesetting in the first place.

¡http://www.example.com/family.owl#Woman¿ ⊑
¡http://www.example.com/family.owl#Female¿⊓
¡http://www.example.com/family.owl#Person¿

Now, consider a "shortform" rendering. For entities that are defined in the *current* namespace, the namespace is omitted.

Woman ⊑ Female ⊓ Person

We contend that this is significantly more convenient, and thus readable. We examine this claim formally in the next chapter. Now, externally defined entity namespaces are included using the shortform notation. For example, datatypes specified as XML Schema Datatypes or in RDFS are prepended with the popular, shortened namespaces of xsd and rdfs, respectively.

xsd:string or xsd:int

4.3 OWL to Description Logic

This section describes changes made to the OWLAPI and its original LaTeX rendering framework. Sections 4.3.1-4.3.5 cover these changes. Section 4.3.6 introduces the GUI tool that implements these changes.

4.3.1 Datatypes

With respect to the syntax of datatypes, there were a number of small changes necessary to align the LaTeX renderer with the OWL standard [15] (e.g. using the bracket and double carat notation).

$$\{\text{“five”}\hat{}\hat{}\text{xsd:string}\} \qquad \text{or} \qquad \{5\hat{}\hat{}\text{xsd:int}\}$$

4.3.2 Nominals

Literals, when used as nominals, are now properly rendered using set notation. In accordance with the above, the example below includes a shortform namespace for its datatype.

$$\exists \text{hasSigrid3IceFormCode}.\{\text{“05”}\hat{}\hat{}\text{xsd:string}\}$$

4.3.3 DatatypeRestriction Axiom

Previously, DatatypeRestriction axioms were not rendered in an intuitive manner. We have made changes in order to make it more similar to the functional syntax specified in [15]. However, we diverge slightly from the specification in the interest of readability. The constrained datatype is followed by a colon to differentiate it from its facets. Further, the constraining facets are rendered using their respective relational operators instead of keywords. In general, DatatypeRestriction axioms are now rendered using the following form, where the ‘+’ indicates one or more of the preceding tokens.

$$\textit{DatatypeRestriction}(\text{datatype: } (\text{constrainingFacet restrictionValue})+)$$

4.3.4 HasKey Axiom

In OWL2 Functional Syntax, a *key* axiom has the form

$$\textit{HasKey}(\text{CE } (\text{OPE}_1 \ldots \text{OPE}_m) \ (\text{DPE}_1 \ldots \text{DPE}_n))$$

where CE is a Class Expression, OPE is an Object Property Expression, and DPE is a Data Property Expression. It must be the case that $n + m > 0$, i.e. the lists OPE and DPE must both not be empty. From [15], the following is a valid *key* axiom.

$$\textit{HasKey}(\text{owl:Thing } () \ (\text{hasSSN}))$$

This axiom states that all owl:Things are uniquely identified by their SSN. This axiom has no analog in description logic [10]. We contend that this functional syntax form is unwieldy and that distinguishing between Object Properties and Data Properties is unnecessary for rendering in DL. As such, we have adopted the following infix notation for a HasKey axiom, where the '+' means one or more of the preceding token. The parentheses are omitted for $n + m = 1$.

$$\text{ClassExpression } \textit{hasKey } (\text{Property}+)$$

$$\text{owl:Thing } \textit{hasKey } \text{hasSSN}$$

This infix notation is much more concise and clearly mirrors the same format of other (non-functional syntactic) DL axioms.

4.3.5 Miscellaneous Corrections to the Existing LaTeX Renderer

Below is a quick summary of syntactical errors previously present in the OWLAPI LaTeX rendering framework. These fixes are unique to the DL rendering.

- The Subproperty axiom now completely renders subproperties.
- Extraneous spacing after logical symbols (e.g. ¬) has been fixed.
- Number Restriction Axioms now have correctly rendered cardinality.
- Role Restriction axioms now have correct "." syntax.

4.3.6 GUI Converter Tool

As mentioned at the beginning of this chapter, some of the changes described here will be present in version 5.0.6 of the OWLAPI. However, some changes require some careful consideration as to their overall impact in the design of the rendering framework. In the interest of making all of these changes accessible to the ontology developer or consumer, we have developed an open-source tool.

The tool can be used to launch a GUI, as shown in Figure 4.3.6, or can be used in the command line. This tool requires Java on the host machine. Additionally, we provide an online portal containing supplemental information regarding this tool: usage tutorial, benchmark patterns and output, and the source code repository.

4.4 OWL to First Order Logic

4.4.1 Direct Translations

As description logics are (decidable) fragments of first order logic, the syntax for DL maps into the syntax for FOL. These translations are very well known [9, 1, 8]. For convenience, we provide these translations and comment on the potential for conversion to FOL rules in the next section. Section 4.4.2 covers the treatment of Datatypes, Section 4.4.3 covers

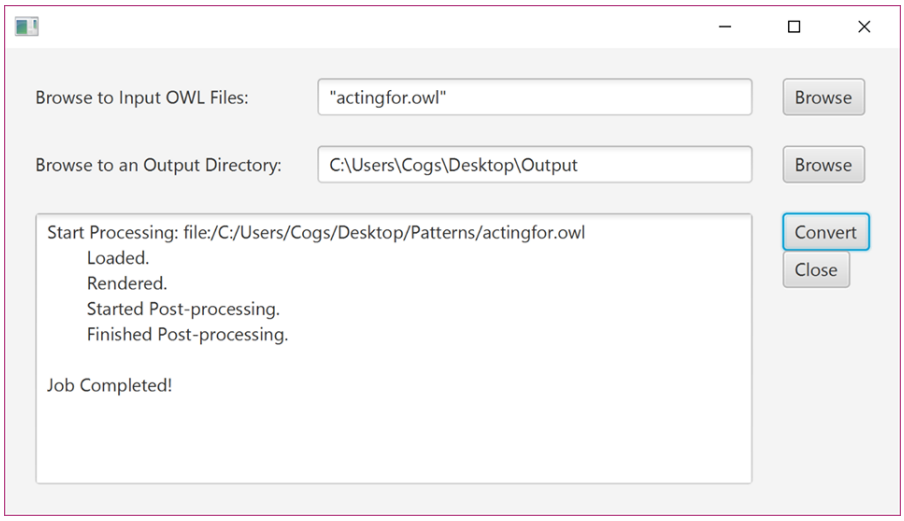

Figure 4.1: A snapshot of the GUI tool used for converting OWL to Description Logic Syntax.

nominals, Section 4.4.4 covers those axioms that have no FOL analog, and finally, Section 4.4.5 covers the developed translation tool.

Concept Inclusions TBox axioms that have the form $C \sqsubseteq D$, called *concept inclusions* or a subconcept relationship, correspond to FOL rules of the form $C(x) \rightarrow D(x)$. For example, the DL axiom Mother $\sqsubseteq$ Parent is equivalent to the FOL formula Mother$(x) \rightarrow$ Parent(x).

Existential Quantification The concept constructor for existential quantification in $\mathcal{SROIQ}$ has the form $\exists R.C$ for atomic R and arbitrary concept C. Consider the axiom

$$\text{Parent} \sqsubseteq \exists \text{hasChild.Person}$$

In natural language, this corresponds to "A Parent is one such that there exists a Person to whom the Parent is related via *hasParent*." The translation to a FOL formula results in

$$\text{Parent}(x) \rightarrow \exists y \ (\text{hasChild}(x, y) \wedge \text{Person}(y))$$

Note that this particular formula can not be converted to a FOL rule due to the presence of an existential quantifier in the consequent.

Value Restriction The concept constructor for value restriction in $\mathcal{SROIQ}$ has the form $\forall R.C$ for atomic R and arbitrary C. This constructor corresponds to the FOL formula $\forall y R(x, y) \rightarrow C(y)$. Consider the axiom

$$\text{Parent} \sqsubseteq \forall \text{hasChild.Child}$$

This axiom is used to express that: for all things related to a Parent through *hasChild*, those things are Children. In FOL, this results in the formula

$$\text{Parent}(x) \rightarrow \forall y \ (\text{hasChild}(x, y) \rightarrow \text{Child}(y))$$

Note that as the LHS, Person, is atomic and that the filler for hasChild is also atomic, the value restriction axiom can be successfully converted into a rule and has the following form

$$\forall y \ \text{Parent}(x) \wedge \text{hasChild}(x, y) \rightarrow \text{Child}(y)$$

Local Reflexivity The DL concept $\exists R.\mathsf{Self}$ are those things that are related to themselves through R. To borrow from [9] an example and its translation to FOL:

$$\exists \text{loves.}\mathsf{Self} \sqsubseteq \text{Narcissist}$$

$$\text{loves}(x, x) \rightarrow \text{Narcissist}(x)$$

Top & Bottom Concept The top and bottom concepts are represented using $\top$ and $\bot$, respectively. In FOL, we choose to use these symbols as predicate names for clarity. That

is, for all things in the domain of interest $\top(x)$ is true. In contrast, $\bot$ may be used to denote the empty set or succinctly express disjointness. For example, if two concepts C, D are disjoint, we may use the functional syntax (which scales well when expressing mutual disjointness of $n > 2$ concepts) or we may say

$$C(x) \wedge D(x) \rightarrow \bot(x) \tag{4.1}$$

4.4.2 Datatypes

For the purposes of FOPL rendering, datatypes are considered to be a predicate. That is, an individual or nominal that has a datatype is expressed as

$$\text{xsd:string}(\text{“}Cogan\text{”})$$

4.4.3 Nominals

Literals are assertions of a datatype predicate. For example, the following axiom contains a nominal with a datatype.

$$\text{BigFloe} \sqsubseteq \exists \text{hasSigrid3IceFormCode.}\{\text{“05”\^\^xsd:string}\}$$

In order to align with the treatment of datatypes in this system, we construct the following formula.

$$\text{BigFloe}(x) \rightarrow \text{hasSigrid3IceFormCode}(x, \text{“05”}) \wedge \text{xsd:string}(\text{“05”})$$

4.4.4 Axioms with No FOPL Analog

For those axioms that do not have analogs in FOPL, we borrow the notation from the OWL Functional Syntax. For example, the HasKey and DataTypeRestriction axiom renderings from Section 4.3 are utilized.

4.4.5 FOL Translator Tool

The FOL Translator Tool has the same GUI and CLI functionality as in Section 4.3.6.

5

Evaluation

For convenience we restate the research hypothesis.

Hypotheses:

- When presented in a Description Logic Syntax, users will more quickly and more correctly understand an ontology compared to renderings in Manchester syntax.
- When presented in a FOPL-Rule-like (FOL[1]) Syntax, users will more quickly and more correctly understand an ontology compared to renderings in Description Logic syntax.

In this chapter, we cover our method for measuring the impact of the different logical renderings of OWL files have on understanding the content of an ontology. In Section 5.1, we cover the overall design of the evaluation, our evaluation method and criteria, and our test populations. In Section 5.2, we cover the results of our evaluation.

[1] For brevity, we will use FOL in place of FOPL-Rule-like (FOPLRL)

5.1 Evaluation Design

5.1.1 Design Overview

In order to evaluate this claim, we have taken the following steps.

- Choose an Ontology
- Render the Ontology
- Choose Test Subjects
- Ask Questions for Understanding
- Evaluate Results

These points are explained in more detail in the following sections.

5.1.2 Choose an Ontology

In this case, we opted to utilize the Semantic Trajectory [6] Ontology Design Pattern. We chose this ODP as it is sufficiently abstract that the semantic relations between its concepts can not be immediately assumed via common sense. That is, in order to truly understand the content of the ontology, the source material must be consulted, thus ensuring that each of the syntaxes must be parsed in order to answer the questions. The entire contents of this ontology are provided in OWL format in Appendix A.

5.1.3 Render the Ontology

As stated in the hypothesis, we aim to show that renderings in FOL result in better and quicker understanding of the ontology. To that end, we will provide the test subjects with renderings in each of the syntaxes. We generate the Manchester Syntax rendering using

an online tool[2] powered by the OWLAPI's Manchester Syntax Renderer. The resultant rendering was only modified to remove annotations and import data. Otherwise, the rendering is provided to the subject *exactly* as the tool outputs. For reference, we include this rendering in Appendix B. The FOPLR-like and the DL Syntax renderings were generated using the tools developed during the course of this thesis and described in Chapter 4. No changes were made to these renderings. These renderings are included in Appendices C and D, respectively.

5.1.4 Choose Test Subjects

The test subjects are unpaid volunteers from among the computer science and engineering graduate students at Wright State University. These students are not expected to be experts in any logic, but should have passing familiarity with modeling data in an abstract manner. In general, we believe that the volunteers are representative of domain experts interested in ontology modeling. For this evaluation we did not assess prior knowledge. We note this lack of assessment in an opportunity for improvement in future work.

5.1.5 Ask Questions of Understanding

This step of the evaluation process includes the actual assessment of the test subject's understanding. This section is split into two parts: development of the evaluation and the assessment.

First, for development, we created four questions designed to assess understanding of the ontology. These questions come in two flavors: those questions that concern exactly one axiom and questions that require the understanding of multiple axioms to answer. We call these four questions, collectively, a test set. In the same manner, we generate two more test sets. Each test set is similar to the others in intent and content. For example, the first

[2] http://www.ldf.fi/service/owl-converter/

1. T [F] All Fixes are StartingFixes.
2. T [F] A Segment can never start from an EndingFix.
3. Which of the following statements is **TRUE**?
   (a) A Segment may start at multiple Fixes.
   (b) [A Segment must have a StartingFix.]
   (c) A Segment consists of subsegments.
4. Which of the following statements is **TRUE**?
   (a) A Segment must be traversed by a MovingObject.
   (b) A Segment must start at an EndingFix of another Segment.
   (c) [Only MovingObjects may traverse a Segment.]

Table 5.1: Questions Provided with Manchester Syntax. The correct answers are outlined in rectangles.

question in each test set relates to the disjointness (or not) of StartingFixes, EndingFixes, and Fixes. Now, we assign each of the test sets to a syntax. The test-set syntax pair, a*section*, is constant (i.e. each test subject answers the same questions for each syntax). The test sets are tests, and their answers, are provided in Tables 5.1 - 5.3.

For assessment, each subject is instructed that they are timed and that they cannot return to a section after they have completed it. We time the test subject as they complete each section. The test subjects are presented with the sections in different orders in order to combat an effect of "increasing familiarity." We do this in case there is a significant "subsidizing effect" of learning the ontology during the first test section. That way our results are agnostic to section ordering. Finally, after completing all the sections, the subjects are asked to rank the sections by readability. We examine the results of this assessment in Section 5.2.

1. T [F] Only some EndingFixes are Fixes.
2. T [F] A Trajectory has an EndingFix.
3. Which of the following statements is **TRUE**?
   (a) [A Segment is not a Trajectory.]
   (b) A Segment is always connected to another Segment.
   (c) A Segment is a Fix.
4. Which of the following statements is **TRUE**?
   (a) [A Trajectory must have a Segment.]
   (b) Segments may not be part of a Trajectory.
   (c) The domain of 'atPlace' consists only of Fixes.

Table 5.2: Questions Provided with Description Logic Syntax. The correct answers are outlined in rectangles.

1. T [F] StartingFixes can be EndingFixes.
2. [T] F An EndingFix is always at a Place.
3. Which of the following statements is **TRUE**?
   (a) A Trajectory may act as a StartingFix.
   (b) A Trajectory does not have subtrajectories.
   (c) [A Trajectory consists of Segments.]
4. Which of the following statements is **TRUE**?
   (a) A Place cannot have Attributes.
   (b) [A Fix that does not begin a Segment is an EndingFix.]
   (c) Any Place is also a TimeEntity.

Table 5.3: Questions Provided with FOLR-like Syntax. The correct answers are outlined in rectangles.

	MS	DL	FOL
Readability	24	14	16
Correctness	2.89	3.22	2.2

Table 5.4: The Readability metric is the summed user ranking. A lower score indicates greater readability. The Correctness metric is the average number of questions answered correctly per test section (4 questions in each section).

5.2 Results

The evaluation was taken by 10 participants. As each test section was taken by every participant, we use the two-tailed Student's *t*-test to measure the significance of our results (we believe assuming an underlying normal distribution is reasonable for this evaluation). We conduct the *t*-test pairwise comparing the aggregate results of each test section.

5.2.1 Readability

In this section, we discuss the results of the *Readability* evaluation; these results are shown in Table 5.4. As stated above, each test subject was asked to rank the different renderings 1, 2, and 3, where each rank may only be used once and a rank of 1 is considered to be 'most readable.' These rankings were then summed; a low score is desirable for this metric.

The Description Logic Rendering was ranked most readable, followed by the FOL Rule-like (FOLRL) Syntax and Manchester Syntax. The significance of these results are shown in Table 5.6. We see that when comparing the Manchester Syntax (MS) results to DL results, we reject the null hypothesis with $p \approx 0.030 < 0.05$, validating the first hypothesis. For MS to FOL, the results are not significant with $p \approx 0.052 \not< 0.05$. When comparing DL to FOL, we fail to reject the null hypothesis.

First, the Manchester Syntax rendering is exactly the output from Protégé or online converter tools. As such, there is little to no organization of the contents of the rendering and full URIs are used; no class is in the shortform notation. The lower ranking for the

DL is likely a result of the number of test subjects already familiar with Description Logic, although we cannot be sure as we did not assess prior knowledge.

5.2.2 Correctness

The results for the *Correctness* evaluation are shown in Table 5.4. As is shown, correctness was highest on the DL test section, followed by the Manchester Syntax and FOL sections. While many people found the FOL syntax more readable, the performance on the test sections does not substantiate the hypothesis that their understanding of the ontology would increase. However, when using comparing the correctness results between MS and DL and FOLRL via Student's *t*-test, we see that the correctness scores are not significant $p \approx 0.40$ and $p \approx 0.71$, respectively. As such, we fail to substantiate our first hypothesis. With respect to FOL to DL, we see that we reject the null hypothesis with $p \approx 0.0400 < 0.05$, substantiating our second hypothesis.

With respect to these results, we offer some comments on the failure to achieve significant results. The higher mean correctness on the MS test section may indicate that it is indeed easier to understand; it is the lack of viable human readable tooling. Alternatively, the high mean correctness may be conflated with increased attention due to the difficulty presented in parsing the dense URI markup. Finally, we again note that some of the test subjects were already intimately familiar with DL, thus increasing their performance on those sections. Finally, we note that performance on the DL section by those familiar with DL was perfect, thus dramatically increasing the performance score.

5.2.3 Timing Statistics

We tracked the amount of time it took for each participant to complete each test section. In the hypothesis, we contend that an ontology rendered in a FOL syntax allows a user to more quickly understand the hypothesis. We have provided these timing statistics in

	MS	DL	FOL
Mean	9:10	6:42	9:51
Median	9:31	5:27	9:22

Table 5.5: Time Taken was measured per test-section.

	MS-DL	MS-FOL	DL-FOL
Readability	0.0304	0.0516	0.5943
	significant	not significant	not significant
Correctness	0.3972	0.3734	0.0400
	not significant	not significant	significant
Timing	0.0476	0.7074	0.1978
	significant	not significant	not significant

Table 5.6: We use Student's *t*-Distribution to measure the pair-wise significance of our results.

Table 5.5. From these data, we conclude via Student's *t*-distribution that only the relative improvement between MS and DL renderings is significant with $p \approx 0.048 < 0.05$ which substantiates our first hypothesis.

These results fail to substantiate the second hypothesis. To this point, we note that timing *understanding* is a nebulous in concept in general, and thus difficult pinpoint a single point of failure. For example, our test set had varied levels of understanding of each of the syntaxes- does this variety have a disproportionate affect over different syntaxes? In future studies, it will be necessary to have a more controlled population of participants.

6

Conclusion

The Semantic Web is a field of research dedicated to modeling complex concepts in a machine readable way, as well as an eponymous extension of the World Wide Web. One of the most prominent tools available to a Semantic Web research is the *ontology*, a codified way of expressing complex concepts in terms of simpler concepts.

As one of the goals of the field is to be able to model *any* arbitrary concept in a machine readable way, ontology engineering is an inherently multidisciplinary process. Thus, ontologies are usually developed in tandem with Semantic Web researchers and domain experts. This collaborative approach necessitates that there be multiple vehicles for visualizing an ontology under development allowing the team to more fully understand the nascent structure.

Furthermore, we wish to lower the barrier for entry to domain experts in order to increase adoption, thus growing the Semantic Web as an artifact. To do this, there are already many tools that are attempts to improve the ontology engineering process (OEP). For example, a visual notation for OWL and new ways to input axioms into Protégé.

In this thesis, we have described the development of two tools to add to the OEP tool set. These tools are powered by a number of changes made to the OWLAPI in order to provide human readable renderings of ontologies in Description Logic and a First Or-

der Predicate Logic Rule-like syntaxes. While the Description Logic syntax was already present as the OWLAPI's LaTeX rendering framework, it suffered from a number of significant problems in correctness and readability.

Hypotheses:

- When presented in a Description Logic Syntax, users will more quickly and more correctly understand an ontology compared to renderings in Manchester syntax.
- When presented in a FOPL-Rule-like (FOL[1]) Syntax, users will more quickly and more correctly understand an ontology compared to renderings in Description Logic syntax.

The results from the evaluation, with respect to above hypotheses, were, at best, inconclusive. However, we contend that the key take away is that there is need for a tool set optimized for rendering an ontology in a human readable manner, whether the target is Manchester Syntax, Description Logic, or a FOL-rule-like syntax. This take away is substantiated by the fact that, in general, DL or FOL renderings resulted in statistically significant improvements over MS. As noted, though, this may be due to the usage of *exactly* the output of the converter tool, which did not utilize shortform notation for any class names.

Future Work

There are three major points discussed in this thesis that can be targeted for improvement in future work.

First and foremost, we would like to create a tool for creating First Order Logic rules. As previously mentioned, simply extending the OWLAPI is infeasible as it is out of scope. Thus, it will be necessary to create a standalone tool with the OWLAPI as a dependency.

We imagine such a tool translating the ontology's underlying data structure into an analogous first order logic data structure. At this point, the FOL can be manipulated in such a way that we can generate rules for those axioms that are translatable. In addition, we may borrow the hybridized syntax from [9] in order to maintain the intuitiveness of rules. It would then be our position that FOL-*rules* are the most readable of the available syntaxes.

Secondly, any subsequent evaluation would require a more carefully chosen and prepared set of test subjects. While each participant was a graduate student and familiar with knowledge representation in general, we noted that varying levels of familiarity may have had a significant impact on the results.

Finally, we will investigate making further changes to the OWLAPI to support a "human readable" rendering of Manchester Syntax

Bibliography

[1] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[2] Xin Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Ni Lao, Kevin Murphy, Thomas Strohmann, Shaohua Sun, and Wei Zhang. Knowledge vault: a web-scale approach to probabilistic knowledge fusion. In Sofus A. Macskassy, Claudia Perlich, Jure Leskovec, Wei Wang, and Rayid Ghani, editors, *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, New York, NY, USA - August 24 - 27, 2014*, pages 601–610. ACM, 2014.

[3] Xin Luna Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Kevin Murphy, Shaohua Sun, and Wei Zhang. From data fusion to knowledge fusion. *CoRR*, abs/1503.00302, 2015.

[4] Pascal Hitzler, Markus Krötzsch, and Sebastian Rudolph. *Foundations of Semantic Web Technologies*. Chapman and Hall/CRC Press, 2010.

[5] Matthew Horridge and Sean Bechhofer. The OWL API: A Java API for OWL Ontologies. *Semantic Web*, 2(1):11–21, 2011.

[6] Yingjie Hu, Krzysztof Janowicz, David Carral, Simon Scheider, Werner Kuhn, Gary Berg-Cross, Pascal Hitzler, Mike Dean, and Dave Kolas. A geo-ontology design pattern for semantic trajectories. In Thora Tenbrink, John G. Stell, Antony Galton, and Zena Wood, editors, *Spatial Information Theory - 11th International Conference, COSIT 2013, Scarborough, UK, September 2-6, 2013. Proceedings*, volume 8116 of *Lecture Notes in Computer Science*, pages 438–456. Springer, 2013.

[7] Michael Kifer. *Rule Interchange Format: The Framework*, pages 1–11. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[8] Markus Krötzsch, Sebastian Rudolph, and Pascal Hitzler. Description logic rules. In Malik Ghallab, Constantine D. Spyropoulos, Nikos Fakotakis, and Nikolaos M. Avouris, editors, *ECAI 2008 - 18th European Conference on Artificial Intelligence, Patras, Greece, July 21-25, 2008, Proceedings*, volume 178 of *Frontiers in Artificial Intelligence and Applications*, pages 80–84. IOS Press, 2008.

[9] Markus Krötzsch, Sebastian Rudolph, and Pascal Hitzler. ELP: tractable rules for OWL 2. In Amit P. Sheth, Steffen Staab, Mike Dean, Massimo Paolucci, Diana Maynard, Timothy W. Finin, and Krishnaprasad Thirunarayan, editors, *The Semantic Web - ISWC 2008, 7th International Semantic Web Conference, ISWC 2008, Karlsruhe, Germany, October 26-30, 2008. Proceedings*, volume 5318 of *Lecture Notes in Computer Science*, pages 649–664. Springer, 2008.

[10] Markus Krötzsch, František Simančík, and Ian Horrocks. A description logic primer. In Jens Lehmann and Johanna Völker, editors, *Perspectives on Ontology Learning*, chapter 1. IOS Press, 2014.

[11] Ora Lassila, Ralph R. Swick, World Wide, and Web Consortium. Resource description framework (rdf) model and syntax specification, 1998.

[12] Steffen Lohmann, Florian Haag, and Stefan Negru. Towards a visual notation for owl: A brief summary of vowl. In *Revised Selected Papers of the 12th International Experiences and Directions Workshop on Ontology Engineering - Volume 9557*, pages 143–153, New York, NY, USA, 2016. Springer-Verlag New York, Inc.

[13] Sarker Md Kamruzzaman, Adila Krisnadhi, David Carral, and Pascal Hitzler. Rule-based owl modeling with rowltab protege plugin, 2017.

[14] Sarker Md Kamruzzaman, Adila Krisnadhi, and Pascal Hitzler. Owlax: A protege plugin to support ontology axiomatization through diagramming. Kobe, Japan, 2016. 15th International Semantic Web Conference, ISWC2016, Kobe, Japan, October 2016, 15th International Semantic Web Conference, ISWC2016, Kobe, Japan, October 2016.

[15] Boris Motik, Peter Patel-Schneider, and Bijan Parsia, editors. *OWL 2 Web Ontology Language: Structural Specification and Functional-Style Syntax (Second Edition)*. W3C Recommendation 11 December 2012, 2012.

[16] Uwe Schöning. *Logic for Computer Scientists*. Birkhuser, 1989.

[17] Cogan Shimizu, Pascal Hitzler, and Matthew Horridge. Rendering owl in description logic syntax.

Appendices

Appendix A: Semantic Trajectory (OWL)

```
<?xml version="1.0"?>
<rdf:RDF xmlns="http://w3id.org/daselab/onto/trajectory#"
    xml:base="http://w3id.org/daselab/onto/trajectory"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:xml="http://www.w3.org/XML/1998/namespace"
    xmlns:cpannotationschema="http://www.
        ontologydesignpatterns.org/schemas/
        cpannotationschema.owl#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:trj="http://w3id.org/daselab/onto/trajectory#"
    xmlns:dc="http://purl.org/dc/elements/1.1/">
  <owl:Ontology rdf:about="http://w3id.org/daselab/onto/
      trajectory">
    <owl:imports rdf:resource="http://www.
        ontologydesignpatterns.org/schemas/
```

cpannotationschema.owl"/>
<dc:creator>Adila Krisnadhi, Pascal Hitzler</dc:creator>
<cpannotationschema:coversRequirements>Show the birds which stop at x and y,
Show the birds which move at a ground speed of 0.4 m/s,
Show the trajectories of rivers which cross national parks,
Where are the ports at which the oceanographic cruise A3221 stopped after leaving Woods Hole?,
List the places and times that represent the spatiotemporal extent of the 1990 World Chess Championship event,</cpannotationschema:coversRequirements>
<cpannotationschema:reengineeredFrom>Yingjie Hu; Krzysztof Janowicz; David Carral; Simon Scheider; Werner Kuhn; Gary Berg-Cross; Pascal Hitzler; Mike Dean; Dave Kolas: A Geo-ontology Design Pattern for Semantic Trajectories. In International Conference on Spatial Information Theory (COSIT) 2013) 438-456</cpannotationschema:reengineeredFrom>
<cpannotationschema:scenarios>Mike's trip to the GeoVoCamp 2012 from his home integrating data from GPS device, vehicle information, and personal information.

A toucan flies through the air as recorded by researchers in the MoveBank.

The 1990 World Chess Championship event that was held in two locations at two different times.</cpannotationschema:scenarios>
<rdfs:label>Semantic Trajectory Pattern</rdfs:label>
<cpannotationschema:hasIntent>The pattern provides a model of trajectory, which is understood as a sequence of spatiotemporal points. The model generalizing the Semantic Trajectory pattern from [Hu, et al., COSIT 2013] by employing the notion of place, instead of location/geo-coordinate, to represent the spatial extent of the trajectory. This pattern is suitable for a variety of trajectory datasets and easily extendible by by aligning to or matching with existing trajectory ontologies, foundational ontologies, or other domain specific vocabularies.</cpannotationschema:hasIntent>
<cpannotationschema:hasConsequences>Unlike the original version of Semantic Trajectory, this pattern omits the hook to the data source for fixes (which was a subclass of ssn:Device) because instead of location/geo-coordinate, the notion of place is employed to capture the spatial extent. Nevertheless, it should be relatively straightforward to extend this version if the user wises to attach data source

```
        information to the fixes.</cpannotationschema:
        hasConsequences>
    <cpannotationschema:relatedCPs>Place, Time,
        MovingObject</cpannotationschema:relatedCPs>
</owl:Ontology>



<!--
///////////////////////////////////////////////////////////////////

//
// Annotation properties
//
///////////////////////////////////////////////////////////////////

 -->



<!-- http://purl.org/dc/elements/1.1/creator -->

<owl:AnnotationProperty rdf:about="http://purl.org/dc/
    elements/1.1/creator"/>
```

```
<!--
////////////////////////////////////////////////////////////////////////

//
// Object Properties
//
////////////////////////////////////////////////////////////////////////

 -->



<!-- http://w3id.org/daselab/onto/trajectory#atPlace -->

<owl:ObjectProperty rdf:about="http://w3id.org/daselab/
    onto/trajectory#atPlace">
     <rdfs:range rdf:resource="http://w3id.org/daselab/
         onto/trajectory#Place"/>
     <rdfs:comment>Connects anything (including fixes in
         this pattern) to Place.</rdfs:comment>
     <rdfs:label>atPlace</rdfs:label>
</owl:ObjectProperty>
```

```
<!-- http://w3id.org/daselab/onto/trajectory#atTime -->

<owl:ObjectProperty rdf:about="http://w3id.org/daselab/
    onto/trajectory#atTime">
     <rdfs:range rdf:resource="http://w3id.org/daselab/
         onto/trajectory#TimeEntity"/>
     <rdfs:comment>Connects anything (including fixes in
         this pattern) to TimeEntity</rdfs:comment>
     <rdfs:label>atTime</rdfs:label>
</owl:ObjectProperty>



<!-- http://w3id.org/daselab/onto/trajectory#endsAt -->

<owl:ObjectProperty rdf:about="http://w3id.org/daselab/
    onto/trajectory#endsAt">
     <rdf:type rdf:resource="http://www.w3.org/2002/07/
         owl#FunctionalProperty"/>
     <rdfs:comment>Connects a segment to the fix it ends
         at.</rdfs:comment>
     <rdfs:label>endsAt</rdfs:label>
</owl:ObjectProperty>



<!-- http://w3id.org/daselab/onto/trajectory#
```

```
    hasAttribute -->

<owl:ObjectProperty rdf:about="http://w3id.org/daselab/
    onto/trajectory#hasAttribute">
     <rdfs:comment>Connects a fix or a segment to an
         additional information as represented by an
         instance of Attribute.</rdfs:comment>
     <rdfs:label>hasAttribute</rdfs:label>
</owl:ObjectProperty>



<!-- http://w3id.org/daselab/onto/trajectory#hasFix -->

<owl:ObjectProperty rdf:about="http://w3id.org/daselab/
    onto/trajectory#hasFix">
     <owl:propertyChainAxiom rdf:parseType="Collection">
          <rdf:Description rdf:about="http://w3id.org/
              daselab/onto/trajectory#hasSegment"/>
          <rdf:Description rdf:about="http://w3id.org/
              daselab/onto/trajectory#endsAt"/>
      </owl:propertyChainAxiom>
     <owl:propertyChainAxiom rdf:parseType="Collection">
          <rdf:Description rdf:about="http://w3id.org/
              daselab/onto/trajectory#hasSegment"/>
          <rdf:Description rdf:about="http://w3id.org/
              daselab/onto/trajectory#startsFrom"/>
```

```
    </owl:propertyChainAxiom>
    <rdfs:comment>Relating the trajectory to each of its
        fixes.</rdfs:comment>
    <rdfs:label>hasFix</rdfs:label>
</owl:ObjectProperty>


<!-- http://w3id.org/daselab/onto/trajectory#hasSegment
   -->

<owl:ObjectProperty rdf:about="http://w3id.org/daselab/
   onto/trajectory#hasSegment">
    <rdfs:comment>Relating the trajectory to each of its
        segments.</rdfs:comment>
    <rdfs:label>hasSegment</rdfs:label>
</owl:ObjectProperty>


<!-- http://w3id.org/daselab/onto/trajectory#
   hasTrajectory -->

<owl:ObjectProperty rdf:about="http://w3id.org/daselab/
   onto/trajectory#hasTrajectory">
    <rdfs:range rdf:resource="http://w3id.org/daselab/
       onto/trajectory#Trajectory"/>
```

```
    <rdfs:comment>Anything that has a trajectory can use
        this property to connect it to the trajectory
       instance.</rdfs:comment>
    <rdfs:label>hasTrajectory</rdfs:label>
</owl:ObjectProperty>



<!-- http://w3id.org/daselab/onto/trajectory#nextFix -->

<owl:ObjectProperty rdf:about="http://w3id.org/daselab/
   onto/trajectory#nextFix">
    <rdfs:comment>Relates one fix to the immediately
       following fix in the sequence.</rdfs:comment>
    <rdfs:label>nextFix</rdfs:label>
</owl:ObjectProperty>



<!-- http://w3id.org/daselab/onto/trajectory#startsFrom
   -->

<owl:ObjectProperty rdf:about="http://w3id.org/daselab/
   onto/trajectory#startsFrom">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/
       owl#FunctionalProperty"/>
    <rdfs:comment>Connects a segment to the fix it
```

```
        starts from.</rdfs:comment>
    <rdfs:label>startsFrom</rdfs:label>
</owl:ObjectProperty>



<!-- http://w3id.org/daselab/onto/trajectory#traversedBy
    -->

<owl:ObjectProperty rdf:about="http://w3id.org/daselab/
    onto/trajectory#traversedBy">
    <rdfs:comment>Connect a segment to the moving object
        that traverses it.</rdfs:comment>
    <rdfs:label>traversedBy</rdfs:label>
</owl:ObjectProperty>



<!--
////////////////////////////////////////////////////////////////////////

//
// Classes
//
////////////////////////////////////////////////////////////////////////

 -->
```

```
<!-- http://w3id.org/daselab/onto/trajectory#Attribute
    -->

<owl:Class rdf:about="http://w3id.org/daselab/onto/
    trajectory#Attribute">
     <rdfs:comment>Captures additional information that
         enriches some fix or segment.</rdfs:comment>
     <rdfs:label>Attribute</rdfs:label>
</owl:Class>



<!-- http://w3id.org/daselab/onto/trajectory#EndingFix
    -->

<owl:Class rdf:about="http://w3id.org/daselab/onto/
    trajectory#EndingFix">
     <owl:equivalentClass>
          <owl:Class>
               <owl:intersectionOf rdf:parseType="
                   Collection">
                    <rdf:Description rdf:about="http://w3id.
                        org/daselab/onto/trajectory#Fix"/>
```

```
                <owl:Class>
                    <owl:complementOf>
                        <owl:Restriction>
                            <owl:onProperty>
                                <rdf:Description>
                                    <owl:inverseOf rdf:
                                        resource="http://
                                        w3id.org/daselab/
                                        onto/trajectory#
                                        startsFrom"/>
                                </rdf:Description>
                            </owl:onProperty>
                            <owl:someValuesFrom rdf:
                                resource="http://w3id.org
                                /daselab/onto/trajectory#
                                Segment"/>
                        </owl:Restriction>
                    </owl:complementOf>
                </owl:Class>
            </owl:intersectionOf>
        </owl:Class>
    </owl:equivalentClass>
    <rdfs:subClassOf rdf:resource="http://w3id.org/
        daselab/onto/trajectory#Fix"/>
    <rdfs:comment>The last fix in a particular sequence
        of fixes.</rdfs:comment>
    <rdfs:label>EndingFix</rdfs:label>
```

```
</owl:Class>


<!-- http://w3id.org/daselab/onto/trajectory#Fix -->

<owl:Class rdf:about="http://w3id.org/daselab/onto/
    trajectory#Fix">
     <rdfs:subClassOf>
          <owl:Restriction>
               <owl:onProperty rdf:resource="http://w3id.
                    org/daselab/onto/trajectory#atPlace"/>
               <owl:someValuesFrom rdf:resource="http://
                    w3id.org/daselab/onto/trajectory#Place"/>
          </owl:Restriction>
     </rdfs:subClassOf>
     <rdfs:subClassOf>
          <owl:Restriction>
               <owl:onProperty rdf:resource="http://w3id.
                    org/daselab/onto/trajectory#atTime"/>
               <owl:someValuesFrom rdf:resource="http://
                    w3id.org/daselab/onto/trajectory#
                    TimeEntity"/>
          </owl:Restriction>
     </rdfs:subClassOf>
     <rdfs:subClassOf>
          <owl:Restriction>
```

```
            <owl:onProperty>
                <rdf:Description>
                    <owl:inverseOf rdf:resource="http://
                        w3id.org/daselab/onto/trajectory#
                        hasFix"/>
                </rdf:Description>
            </owl:onProperty>
            <owl:someValuesFrom rdf:resource="http://
                w3id.org/daselab/onto/trajectory#
                Trajectory"/>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="http://w3id.
                org/daselab/onto/trajectory#hasAttribute
                "/>
            <owl:allValuesFrom rdf:resource="http://w3id
                .org/daselab/onto/trajectory#Attribute"/>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="http://w3id.
                org/daselab/onto/trajectory#nextFix"/>
            <owl:allValuesFrom rdf:resource="http://w3id
                .org/daselab/onto/trajectory#Fix"/>
```

```
            </owl:Restriction>
        </rdfs:subClassOf>
        <rdfs:comment>Describes a fix, which is an adorned
            spatiotemporal point. A sequence of fixes form
            the trajectory.</rdfs:comment>
        <rdfs:label>Fix</rdfs:label>
    </owl:Class>



    <!-- http://w3id.org/daselab/onto/trajectory#
        MovingObject -->

    <owl:Class rdf:about="http://w3id.org/daselab/onto/
        trajectory#MovingObject">
        <rdfs:comment>This is the hook to an ontology/
            pattern that describes the moving object, if any,
             which moves along the trajectory.</rdfs:comment>
        <rdfs:label>MovingObject</rdfs:label>
    </owl:Class>



    <!-- http://w3id.org/daselab/onto/trajectory#Place -->

    <owl:Class rdf:about="http://w3id.org/daselab/onto/
        trajectory#Place">
```

```
    <rdfs:comment>This is the hook to other pattern/
        ontology that describes the notion of place,
        which is more general than just a location/geo-
        coordinate.</rdfs:comment>
    <rdfs:label>Place</rdfs:label>
</owl:Class>



<!-- http://w3id.org/daselab/onto/trajectory#Segment -->

<owl:Class rdf:about="http://w3id.org/daselab/onto/
    trajectory#Segment">
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="http://w3id.
                org/daselab/onto/trajectory#endsAt"/>
            <owl:someValuesFrom rdf:resource="http://
                w3id.org/daselab/onto/trajectory#Fix"/>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="http://w3id.
                org/daselab/onto/trajectory#startsFrom"/>
            <owl:someValuesFrom rdf:resource="http://
                w3id.org/daselab/onto/trajectory#Fix"/>
```

```
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty>
                <rdf:Description>
                    <owl:inverseOf rdf:resource="http://
                        w3id.org/daselab/onto/trajectory#
                        hasSegment"/>
                </rdf:Description>
            </owl:onProperty>
            <owl:someValuesFrom rdf:resource="http://
                w3id.org/daselab/onto/trajectory#
                Trajectory"/>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="http://w3id.
                org/daselab/onto/trajectory#endsAt"/>
            <owl:allValuesFrom rdf:resource="http://w3id
                .org/daselab/onto/trajectory#Fix"/>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="http://w3id.
```

```
                org/daselab/onto/trajectory#hasAttribute
                "/>
            <owl:allValuesFrom rdf:resource="http://w3id
                .org/daselab/onto/trajectory#Attribute"/>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="http://w3id.
                org/daselab/onto/trajectory#startsFrom"/>
            <owl:allValuesFrom rdf:resource="http://w3id
                .org/daselab/onto/trajectory#Fix"/>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="http://w3id.
                org/daselab/onto/trajectory#traversedBy
                "/>
            <owl:allValuesFrom rdf:resource="http://w3id
                .org/daselab/onto/trajectory#MovingObject
                "/>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:comment>The Segment class captures the &quot;
        connection&quot; between two consecutive fixes.
        That is, a segment starts from a fix and ends at
```

```
        another fix. If the pattern is used to model the
        trajectory of some moving object, each segment is
         traversed by that moving object. Additional
        information about a segment can be attached as
        attributes.</rdfs:comment>
    <rdfs:label>Segment</rdfs:label>
</owl:Class>



<!-- http://w3id.org/daselab/onto/trajectory#StartingFix
    -->

<owl:Class rdf:about="http://w3id.org/daselab/onto/
    trajectory#StartingFix">
    <owl:equivalentClass>
        <owl:Class>
            <owl:intersectionOf rdf:parseType="
                Collection">
                <rdf:Description rdf:about="http://w3id.
                    org/daselab/onto/trajectory#Fix"/>
                <owl:Class>
                    <owl:complementOf>
                        <owl:Restriction>
                            <owl:onProperty>
                                <rdf:Description>
                                    <owl:inverseOf rdf:
```

```
                                    resource="http://
                                    w3id.org/daselab/
                                    onto/trajectory#
                                    endsAt"/>
                                </rdf:Description>
                            </owl:onProperty>
                            <owl:someValuesFrom rdf:
                                resource="http://w3id.org
                                /daselab/onto/trajectory#
                                Segment"/>
                        </owl:Restriction>
                    </owl:complementOf>
                </owl:Class>
            </owl:intersectionOf>
        </owl:Class>
    </owl:equivalentClass>
    <rdfs:subClassOf rdf:resource="http://w3id.org/
        daselab/onto/trajectory#Fix"/>
    <rdfs:comment>The first fix in a particular sequence
        of fixes.</rdfs:comment>
    <rdfs:label>StartingFix</rdfs:label>
</owl:Class>



<!-- http://w3id.org/daselab/onto/trajectory#TimeEntity
    -->
```

```
<owl:Class rdf:about="http://w3id.org/daselab/onto/
    trajectory#TimeEntity">
     <rdfs:comment>The hook to class/pattern/ontology
         that models time, this class provides the
         temporal extent of the trajectory. One example of
          time model is the W3C Time Ontology.</rdfs:
         comment>
     <rdfs:label>TimeEntity</rdfs:label>
     <rdfs:seeAlso rdf:resource="http://www.w3.org/2006/
         time"/>
</owl:Class>



<!-- http://w3id.org/daselab/onto/trajectory#Trajectory
    -->

<owl:Class rdf:about="http://w3id.org/daselab/onto/
    trajectory#Trajectory">
     <rdfs:subClassOf>
          <owl:Restriction>
               <owl:onProperty rdf:resource="http://w3id.
                   org/daselab/onto/trajectory#hasSegment"/>
               <owl:someValuesFrom rdf:resource="http://
                   w3id.org/daselab/onto/trajectory#Segment
                   "/>
```

```
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="http://w3id.
                org/daselab/onto/trajectory#hasFix"/>
            <owl:allValuesFrom rdf:resource="http://w3id
                .org/daselab/onto/trajectory#Fix"/>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="http://w3id.
                org/daselab/onto/trajectory#hasSegment"/>
            <owl:allValuesFrom rdf:resource="http://w3id
                .org/daselab/onto/trajectory#Segment"/>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:comment>Represents the notion of trajectory,
        this is the main class that can be hooked with
        other patterns that use the Trajectory pattern.
        Trajectory in this model is understood as a
        sequence of fixes connected by segments. There is
         exactly one starting fix and exactly one ending
        fix. Each fix has a temporal extent and a place (
        which is more general than just a location).</
        rdfs:comment>
```

```
    <rdfs:label>Trajectory</rdfs:label>
</owl:Class>



<!--
///////////////////////////////////////////////////////////////////////

//
// General axioms
//
///////////////////////////////////////////////////////////////////////

 -->

<owl:Restriction>
    <owl:onProperty rdf:resource="http://w3id.org/
        daselab/onto/trajectory#endsAt"/>
    <owl:someValuesFrom rdf:resource="http://w3id.org/
        daselab/onto/trajectory#Fix"/>
    <rdfs:subClassOf rdf:resource="http://w3id.org/
        daselab/onto/trajectory#Segment"/>
</owl:Restriction>
<owl:Restriction>
    <owl:onProperty rdf:resource="http://w3id.org/
        daselab/onto/trajectory#hasAttribute"/>
    <owl:someValuesFrom rdf:resource="http://w3id.org/
```

```
        daselab/onto/trajectory#Attribute"/>
    <rdfs:subClassOf>
        <owl:Class>
            <owl:unionOf rdf:parseType="Collection">
                <rdf:Description rdf:about="http://w3id.
                    org/daselab/onto/trajectory#Fix"/>
                <rdf:Description rdf:about="http://w3id.
                    org/daselab/onto/trajectory#Segment
                    "/>
            </owl:unionOf>
        </owl:Class>
    </rdfs:subClassOf>
</owl:Restriction>
<owl:Restriction>
    <owl:onProperty rdf:resource="http://w3id.org/
        daselab/onto/trajectory#hasFix"/>
    <owl:someValuesFrom rdf:resource="http://w3id.org/
        daselab/onto/trajectory#Fix"/>
    <rdfs:subClassOf rdf:resource="http://w3id.org/
        daselab/onto/trajectory#Trajectory"/>
</owl:Restriction>
<owl:Restriction>
    <owl:onProperty rdf:resource="http://w3id.org/
        daselab/onto/trajectory#hasSegment"/>
    <owl:someValuesFrom rdf:resource="http://w3id.org/
        daselab/onto/trajectory#Segment"/>
    <rdfs:subClassOf rdf:resource="http://w3id.org/
```

```
        daselab/onto/trajectory#Trajectory"/>
</owl:Restriction>
<owl:Restriction>
    <owl:onProperty rdf:resource="http://w3id.org/
        daselab/onto/trajectory#nextFix"/>
    <owl:someValuesFrom rdf:resource="http://w3id.org/
        daselab/onto/trajectory#Fix"/>
    <rdfs:subClassOf rdf:resource="http://w3id.org/
        daselab/onto/trajectory#Fix"/>
</owl:Restriction>
<owl:Restriction>
    <owl:onProperty rdf:resource="http://w3id.org/
        daselab/onto/trajectory#startsFrom"/>
    <owl:someValuesFrom rdf:resource="http://w3id.org/
        daselab/onto/trajectory#Fix"/>
    <rdfs:subClassOf rdf:resource="http://w3id.org/
        daselab/onto/trajectory#Segment"/>
</owl:Restriction>
<owl:Restriction>
    <owl:onProperty rdf:resource="http://w3id.org/
        daselab/onto/trajectory#traversedBy"/>
    <owl:someValuesFrom rdf:resource="http://w3id.org/
        daselab/onto/trajectory#MovingObject"/>
    <rdfs:subClassOf rdf:resource="http://w3id.org/
        daselab/onto/trajectory#Segment"/>
</owl:Restriction>
<rdf:Description>
```

```
        <rdf:type rdf:resource="http://www.w3.org/2002/07/
            owl#AllDisjointClasses"/>
        <owl:members rdf:parseType="Collection">
            <rdf:Description rdf:about="http://w3id.org/
                daselab/onto/trajectory#Attribute"/>
            <rdf:Description rdf:about="http://w3id.org/
                daselab/onto/trajectory#Fix"/>
            <rdf:Description rdf:about="http://w3id.org/
                daselab/onto/trajectory#MovingObject"/>
            <rdf:Description rdf:about="http://w3id.org/
                daselab/onto/trajectory#Place"/>
            <rdf:Description rdf:about="http://w3id.org/
                daselab/onto/trajectory#Segment"/>
            <rdf:Description rdf:about="http://w3id.org/
                daselab/onto/trajectory#TimeEntity"/>
            <rdf:Description rdf:about="http://w3id.org/
                daselab/onto/trajectory#Trajectory"/>
        </owl:members>
    </rdf:Description>
</rdf:RDF>



<!-- Generated by the OWL API (version 4.2.5.20160517-0735)
    https://github.com/owlcs/owlapi -->
```

Appendix B: Semantic Trajectory (Manchester Syntax)

```
Ontology: <http://w3id.org/daselab/onto/trajectory>

Import: <http://www.ontologydesignpatterns.org/schemas/
    cpannotationschema.owl>

Datatype: rdf:PlainLiteral


ObjectProperty: <http://w3id.org/daselab/onto/trajectory#
    atPlace>

     Range:
          <http://w3id.org/daselab/onto/trajectory#Place>


ObjectProperty: <http://w3id.org/daselab/onto/trajectory#
    atTime>
```

```
    Range:
        <http://w3id.org/daselab/onto/trajectory#TimeEntity>


ObjectProperty: <http://w3id.org/daselab/onto/trajectory#
    endsAt>

    Characteristics:
        Functional


ObjectProperty: <http://w3id.org/daselab/onto/trajectory#
    hasAttribute>


ObjectProperty: <http://w3id.org/daselab/onto/trajectory#
    hasFix>


    SubPropertyChain:
        <http://w3id.org/daselab/onto/trajectory#hasSegment>
            o <http://w3id.org/daselab/onto/trajectory#
            endsAt>

    SubPropertyChain:
        <http://w3id.org/daselab/onto/trajectory#hasSegment>
```

o <http://w3id.org/daselab/onto/trajectory#
startsFrom>

ObjectProperty: <http://w3id.org/daselab/onto/trajectory#
hasSegment>

ObjectProperty: <http://w3id.org/daselab/onto/trajectory#
hasTrajectory>

Range:
<http://w3id.org/daselab/onto/trajectory#Trajectory>

ObjectProperty: <http://w3id.org/daselab/onto/trajectory#
nextFix>

ObjectProperty: <http://w3id.org/daselab/onto/trajectory#
startsFrom>

Characteristics:

```
        Functional


ObjectProperty: <http://w3id.org/daselab/onto/trajectory#
    traversedBy>



Class: <http://w3id.org/daselab/onto/trajectory#Attribute>



Class: <http://w3id.org/daselab/onto/trajectory#EndingFix>


    EquivalentTo:
        <http://w3id.org/daselab/onto/trajectory#Fix>
          and (not ( inverse (<http://w3id.org/daselab/onto/
              trajectory#startsFrom>) some <http://w3id.org/
              daselab/onto/trajectory#Segment>))


    SubClassOf:
        <http://w3id.org/daselab/onto/trajectory#Fix>


Class: <http://w3id.org/daselab/onto/trajectory#Fix>
```

```
    SubClassOf:
        <http://w3id.org/daselab/onto/trajectory#atPlace>
            some <http://w3id.org/daselab/onto/trajectory#
            Place>,
        <http://w3id.org/daselab/onto/trajectory#atTime>
            some <http://w3id.org/daselab/onto/trajectory#
            TimeEntity>,
          inverse (<http://w3id.org/daselab/onto/trajectory#
              hasFix>) some <http://w3id.org/daselab/onto/
              trajectory#Trajectory>,
        <http://w3id.org/daselab/onto/trajectory#
            hasAttribute> only <http://w3id.org/daselab/onto/
            trajectory#Attribute>,
        <http://w3id.org/daselab/onto/trajectory#nextFix>
            only <http://w3id.org/daselab/onto/trajectory#Fix
            >


Class: <http://w3id.org/daselab/onto/trajectory#MovingObject
    >



Class: <http://w3id.org/daselab/onto/trajectory#Place>

    el "Place"
```

```
Class: <http://w3id.org/daselab/onto/trajectory#Segment>

    al information about a segment can be attached as
       attributes.",
        rdfs:label "Segment"

    SubClassOf:
        <http://w3id.org/daselab/onto/trajectory#endsAt>
           some <http://w3id.org/daselab/onto/trajectory#Fix
           >,
        <http://w3id.org/daselab/onto/trajectory#startsFrom>
            some <http://w3id.org/daselab/onto/trajectory#
           Fix>,
          inverse (<http://w3id.org/daselab/onto/trajectory#
            hasSegment>) some <http://w3id.org/daselab/onto/
            trajectory#Trajectory>,
        <http://w3id.org/daselab/onto/trajectory#endsAt>
           only <http://w3id.org/daselab/onto/trajectory#Fix
           >,
        <http://w3id.org/daselab/onto/trajectory#
           hasAttribute> only <http://w3id.org/daselab/onto/
           trajectory#Attribute>,
        <http://w3id.org/daselab/onto/trajectory#startsFrom>
            only <http://w3id.org/daselab/onto/trajectory#
           Fix>,
```

```
        <http://w3id.org/daselab/onto/trajectory#traversedBy
            > only <http://w3id.org/daselab/onto/trajectory#
            MovingObject>


Class: <http://w3id.org/daselab/onto/trajectory#StartingFix>


    EquivalentTo:
        <http://w3id.org/daselab/onto/trajectory#Fix>
         and (not ( inverse (<http://w3id.org/daselab/onto/
            trajectory#endsAt>) some <http://w3id.org/
            daselab/onto/trajectory#Segment>))

    SubClassOf:
        <http://w3id.org/daselab/onto/trajectory#Fix>


Class: <http://w3id.org/daselab/onto/trajectory#TimeEntity>


Class: <http://w3id.org/daselab/onto/trajectory#Trajectory>

    SubClassOf:
        <http://w3id.org/daselab/onto/trajectory#hasSegment>
            some <http://w3id.org/daselab/onto/trajectory#
            Segment>,
```

```
        <http://w3id.org/daselab/onto/trajectory#hasFix>
            only <http://w3id.org/daselab/onto/trajectory#Fix
            >,
        <http://w3id.org/daselab/onto/trajectory#hasSegment>
             only <http://w3id.org/daselab/onto/trajectory#
            Segment>


DisjointClasses:
    <http://w3id.org/daselab/onto/trajectory#Attribute>,<
        http://w3id.org/daselab/onto/trajectory#Fix>,<http://
        w3id.org/daselab/onto/trajectory#MovingObject>,<http
        ://w3id.org/daselab/onto/trajectory#Place>,<http://
        w3id.org/daselab/onto/trajectory#Segment>,<http://
        w3id.org/daselab/onto/trajectory#TimeEntity>,<http://
        w3id.org/daselab/onto/trajectory#Trajectory>
```

Appendix C: Semantic Trajectory (First Order Logic)

Classes

Attribute

AllDifferent(Attribute, Fix, MovingObject, Place, Segment, TimeEntity, Trajectory)

EndingFix

$$\text{EndingFix}(x_1) \rightarrow \text{Fix}(x_1) \wedge \forall x_2(\text{startsFrom}(x_2, x_1) \rightarrow \neg\text{Segment}($$

$$\text{Fix}(x_1) \wedge \forall x_2(\text{startsFrom}(x_2, x_1) \rightarrow \neg\text{Segment}(x_2)) \rightarrow \text{EndingFix}(x_1)$$

$$\text{EndingFix}(x_1) \rightarrow \text{Fix}(x_1)$$

Fix

$$\text{Fix}(x_1) \rightarrow \exists x_2(\text{atPlace}(x_1, x_2) \wedge \text{Place}(x_2))$$

$$\text{Fix}(x_1) \rightarrow \forall x_2(\text{nextFix}(x_1, x_2) \rightarrow \text{Fix}(x_2))$$

$$\text{Fix}(x_1) \rightarrow \exists x_2(\text{atTime}(x_1, x_2) \wedge \text{TimeEntity}(x_2))$$

$$\text{Fix}(x_1) \rightarrow \forall x_2(\text{hasAttribute}(x_1, x_2) \rightarrow \text{Attribute}(x_2))$$

$$\text{Fix}(x_1) \rightarrow \exists x_2(\text{hasFix}(x_2, x_1) \wedge \text{Trajectory}(x_2))$$

$$\textit{AllDifferent}(\text{Attribute}, \text{Fix}, \text{MovingObject}, \text{Place}, \text{Segment}, \text{TimeEntity}, \text{Trajectory})$$

MovingObject

$$\textit{AllDifferent}(\text{Attribute}, \text{Fix}, \text{MovingObject}, \text{Place}, \text{Segment}, \text{TimeEntity}, \text{Trajectory})$$

Place

$$\textit{AllDifferent}(\text{Attribute}, \text{Fix}, \text{MovingObject}, \text{Place}, \text{Segment}, \text{TimeEntity}, \text{Trajectory})$$

Segment

$$\text{Segment}(x_1) \rightarrow \exists x_2(\text{startsFrom}(x_1, x_2) \wedge \text{Fix}(x_2))$$

$$\text{Segment}(x_1) \rightarrow \exists x_2(\text{endsAt}(x_1, x_2) \wedge \text{Fix}(x_2))$$

$$\text{Segment}(x_1) \rightarrow \forall x_2(\text{hasAttribute}(x_1, x_2) \rightarrow \text{Attribute}(x_2))$$

$$\text{Segment}(x_1) \rightarrow \forall x_2(\text{startsFrom}(x_1, x_2) \rightarrow \text{Fix}(x_2))$$

$$\text{Segment}(x_1) \rightarrow \exists x_2(\text{hasSegment}(x_2, x_1) \wedge \text{Trajectory}(x_2))$$

$$\text{Segment}(x_1) \rightarrow \forall x_2(\text{traversedBy}(x_1, x_2) \rightarrow \text{MovingObject}(x_2))$$

$$\text{Segment}(x_1) \rightarrow \forall x_2(\text{endsAt}(x_1, x_2) \rightarrow \text{Fix}(x_2))$$

$$\textit{AllDifferent}(\text{Attribute}, \text{Fix}, \text{MovingObject}, \text{Place}, \text{Segment}, \text{TimeEntity}, \text{Trajectory})$$

StartingFix

$$\text{StartingFix}(x_1) \rightarrow \text{Fix}(x_1) \wedge \forall x_2(\text{endsAt}(x_2, x_1) \rightarrow \neg\text{Segment}(x_2))$$

$$\text{Fix}(x_1) \wedge \forall x_2(\text{endsAt}(x_2, x_1) \rightarrow \neg\text{Segment}(x_2)) \rightarrow \text{StartingFix}(x_1)$$

$$\text{StartingFix}(x_1) \rightarrow \text{Fix}(x_1)$$

TimeEntity

$$\textit{AllDifferent}(\text{Attribute}, \text{Fix}, \text{MovingObject}, \text{Place}, \text{Segment}, \text{TimeEntity}, \text{Trajectory})$$

Trajectory

$\text{Trajectory}(x_1) \rightarrow \exists x_2(\text{hasSegment}(x_1, x_2) \wedge \text{Segment}(x_2))$

$\text{Trajectory}(x_1) \rightarrow \forall x_2(\text{hasFix}(x_1, x_2) \rightarrow \text{Fix}(x_2))$

$\text{Trajectory}(x_1) \rightarrow \forall x_2(\text{hasSegment}(x_1, x_2) \rightarrow \text{Segment}(x_2))$

$\mathit{AllDifferent}(\text{Attribute}, \text{Fix}, \text{MovingObject}, \text{Place}, \text{Segment}, \text{TimeEntity}, \text{Trajectory})$

Object properties

atPlace

$$\exists x_1\ \text{Thing}(x_1) \rightarrow \forall x_2(\text{atPlace}(x_1, x_2) \rightarrow \text{Place}(x_2))$$

atTime

$$\exists x_1\ \text{Thing}(x_1) \rightarrow \forall x_2(\text{atTime}(x_1, x_2) \rightarrow \text{TimeEntity}(x_2))$$

endsAt

$$\rightarrow\ \ \leq 1 x_2\ \text{endsAt}(x_1, x_2) \wedge \text{Thing}(x_2)$$

hasAttribute

hasFix

hasSegment

hasTrajectory

$$\exists x_1 \text{ Thing}(x_1) \rightarrow \forall x_2(\text{hasTrajectory}(x_1, x_2) \rightarrow \text{Trajectory}(x_2))$$

nextFix

startsFrom

$$\rightarrow \ \leq 1 x_2 \text{ startsFrom}(x_1, x_2) \wedge \text{Thing}(x_2)$$

traversedBy

Data properties

Individuals

Datatypes

string

Appendix D: Semantic Trajectory (Description Logic)

Classes

Attribute

$$AllDisjoint(\text{Attribute}, \text{Fix}, \text{MovingObject}, \text{Place}, \text{Segment}, \text{TimeEntity}, \text{Trajectory})$$

EndingFix

$$EquivalentClasses(\text{EndingFix}, \text{Fix} \sqcap \neg(\exists \text{startsFrom}^{-}.\text{Segment}))$$
$$\text{EndingFix} \sqsubseteq \text{Fix}$$

Fix

$$\text{Fix} \sqsubseteq \exists \text{atPlace}.\text{Place}$$
$$\text{Fix} \sqsubseteq \forall \text{nextFix}.\text{Fix}$$
$$\text{Fix} \sqsubseteq \exists \text{atTime}.\text{TimeEntity}$$
$$\text{Fix} \sqsubseteq \forall \text{hasAttribute}.\text{Attribute}$$
$$\text{Fix} \sqsubseteq \exists \text{hasFix}^{-}.\text{Trajectory}$$
$$AllDisjoint(\text{Attribute}, \text{Fix}, \text{MovingObject}, \text{Place}, \text{Segment}, \text{TimeEntity}, \text{Trajectory})$$

MovingObject

$AllDisjoint$(Attribute, Fix, MovingObject, Place, Segment, TimeEntity, Trajectory)

Place

$AllDisjoint$(Attribute, Fix, MovingObject, Place, Segment, TimeEntity, Trajectory)

Segment

Segment $\sqsubseteq \exists$startsFrom.Fix
Segment $\sqsubseteq \exists$endsAt.Fix
Segment $\sqsubseteq \forall$hasAttribute.Attribute
Segment $\sqsubseteq \forall$startsFrom.Fix
Segment $\sqsubseteq \exists$hasSegment^{-}.Trajectory
Segment $\sqsubseteq \forall$traversedBy.MovingObject
Segment $\sqsubseteq \forall$endsAt.Fix
$AllDisjoint$(Attribute, Fix, MovingObject, Place, Segment, TimeEntity, Trajectory)

StartingFix

$EquivalentClasses$(StartingFix, Fix $\sqcap \neg(\exists$endsAt^{-}.Segment$)$)
StartingFix $\sqsubseteq$ Fix

TimeEntity

$AllDisjoint$(Attribute, Fix, MovingObject, Place, Segment, TimeEntity, Trajectory)

Trajectory

$\text{Trajectory} \sqsubseteq \exists \text{hasSegment}.\text{Segment}$
$\text{Trajectory} \sqsubseteq \forall \text{hasFix}.\text{Fix}$
$\text{Trajectory} \sqsubseteq \forall \text{hasSegment}.\text{Segment}$
$AllDisjoint(\text{Attribute}, \text{Fix}, \text{MovingObject}, \text{Place}, \text{Segment}, \text{TimeEntity}, \text{Trajectory})$

Object properties

atPlace

$$\top \sqsubseteq \forall \text{atPlace}.\text{Place}$$

atTime

$$\top \sqsubseteq \forall \text{atTime}.\text{TimeEntity}$$

endsAt

$$\top \sqsubseteq \leq 1 \text{endsAt}.\top$$

hasAttribute

hasFix

hasSegment

hasTrajectory

$$\top \sqsubseteq \forall \text{hasTrajectory}.\text{Trajectory}$$

nextFix

startsFrom

$$\top \sqsubseteq \leq 1 \text{startsFrom}.\top$$

traversedBy

Data properties

Individuals

Datatypes

string

Appendix E: Question Pool

For each answer, write the axiom that justifies your answer.

1. T F Only some EndingFixes are Fixes
2. T F All Fixes are StartingFixes
3. T F Fixes and Segments are equivalent concepts
4. T F Trajectories, Segments, and Fixes are mutually disjoint concepts
5. T F Trajectories may have more than one possible StartingFix

6. A Segment
   A. Is not a trajectory
   B. Is always connected to another segment
   C. Is a Fix
   D. Is equivalent to a Place
   E. All of the above

7. A Trajectory
   A. Can act as a StartingFix
   B. Does not have subtrajectories
   C. Is a segment
   D. Consists of segments
   E. None of the above

8. Which of the following statements is **TRUE**
   A. A Trajectory does not have Fixes
   B. All Segments belong to a Trajectory.
   C. The domain of 'atPlace' is Place
   D. Both A & B
   E. Both A & C

9. Which of the following statements is **TRUE**
   A. Only MovingObjects traverse a Segment
   B. A Segment has a MovingObject
   C. A Segment must start at a Fix.
   D. A Segment must start at a Place.
   E. Both A & C

10. Which of the following statements is **TRUE**
    A. A Place may have Attributes
    B. If any Fix is the start of a Segment, then it is a StartingFix
    C. A Fix that does not begin a Segment is an EndingFix
    D. Any Place is also a TimeEntity
    E. Both B & C

11. Which of the following statements is **TRUE**
    A. A Fix may have Attributes
    B. A Segment may have Attributes
    C. A Trajectory may have Attributes
    D. Both A & B
    E. A, B, & C