The University of Maine
DigitalCommons@UMaine

Electronic Theses and Dissertations | Fogler Library

Fall 12-11-2020

Approximating FOL Ontologies using OWL2

Robert W. Powell
robert.powell@maine.edu

Follow this and additional works at: https://digitalcommons.library.umaine.edu/etd

Part of the Theory and Algorithms Commons

Recommended Citation
Powell, Robert W., "Approximating FOL Ontologies using OWL2" (2020). *Electronic Theses and Dissertations*. 3475.
https://digitalcommons.library.umaine.edu/etd/3475

This Open-Access Thesis is brought to you for free and open access by DigitalCommons@UMaine. It has been accepted for inclusion in Electronic Theses and Dissertations by an authorized administrator of DigitalCommons@UMaine. For more information, please contact um.library.technical.services@maine.edu.

APPROXIMATING FOL ONTOLOGIES USING OWL2

By

Robert Wesley Powell II

B.S. University of Maine, 2016

A THESIS

Submitted in Partial Fulfillment of the

Requirements for the Degree of

Master of Science

(in Computer Science)

The Graduate School

The University of Maine

December 2020

Advisory Committee:

Torsten Hahmann, Assistant Professor of Spatial Informatics, Advisor

Roy Turner, Associate Professor of Computer Science

Sepideh Ghanavati, Assistant Professor of Computer Science

Copyright 2020 Robert Powell

APPROXIMATING FOL ONTOLOGIES USING OWL2

By Robert Wesley Powell II

Thesis Advisor: Dr. Torsten Hahmann

An Abstract of the Thesis Presented
in Partial Fulfilment of the Requirements for the
Degree of Master of Science
(in Computer Science)
December 2020

With the amount of data collected everyday ever expanding, techniques which allow computers to semantically understand data are growing in importance. Ontologies are a tool to describe the relationships connecting data so that computers can correctly interpret and combine data from many sources. An ontology about water needs to describe what the term "river" may refer to: An arbitrary river or one usable for navigation; a single tributary or an entire river network; the riverbed or the water itself? Well-designed ontologies can be shared, reused, and extended across multiple applications and facilitate betters integration of different data collections.

Common Logic (CL) and the Web Ontology Language (OWL) are two logic based languages of popular interest. However, ontologies developed in either of these languages are not easily consumed by users of the other language. By utilizing the first order properties of Common Logic, an automated approximation routine between CL and OWL is provided. OWL, being less expressive than CL, is capable of being totally represented by logically equivalent CL axioms. Leveraging the logical equivalence, we provide a method of axiom normalization and extraction in order to construct robust OWL ontologies from existing CL sources. This increases CL ontology intelligibility, and allows the automatic construction of OWL versions of existing reference ontologies. Further, the benefits of such a translation are demonstrated by applying previously exclusive OWL tooling and analysis techniques to evaluate the translated ontologies.

DEDICATION

To my family whose continued encouragement in my pursuit of higher education has been a constant driver.

ACKNOWLEDGEMENTS

I am thankful to all the individuals over the years who offered their encouragement and guidance to my pursuit of higher education.

The completion of this thesis is owed in no small part to the continued patience and efforts of my thesis advisor Torsten Hahmann. Without his ongoing support, even when I transitioned to being a part-time student while I pursued a full-time career, was fundamental to the completion of this thesis. I offer my sincere thanks for all the time spent over the years reviewing, discussing, and sometimes even debugging the contents of this thesis.

I owe my wife, Bai Li, a hearty thanks for her efforts to cajole me into writing the thesis itself. Even if it meant a holiday weekend at home while I tried to catch up on writing after a particularly busy week. Her support and willingness to listen to the more abstract details of the thesis and be a sounding board for practice runs of the defense was much needed and very appreciated.

I would also like to thank my committee members Roy Turner and Sepideh Ghanavati for their time spent reviewing the thesis itself. And for the valuable questions gathered during the defense which helped refine the thesis for those who may use it in the future.

TABLE OF CONTENTS

DEDICATION . . . iii

ACKNOWLEDGEMENTS . . . iv

LIST OF TABLES . . . vii

LIST OF FIGURES . . . viii

1 INTRODUCTION . . . 1

2 BACKGROUND AND RELATED WORK . . . 4
- 2.1 Kinds of Ontologies . . . 5
- 2.2 Application Ontologies . . . 5
- 2.3 Domain Ontologies . . . 6
- 2.4 Reference Ontologies . . . 7
- 2.5 Ontology Languages . . . 8
  - 2.5.1 First-order Logic . . . 8
  - 2.5.2 Web Ontology Language . . . 11
- 2.6 Related Works . . . 14

3 APPROACH . . . 16
- 3.1 OWL constructs in FOL . . . 18
- 3.2 Function-Free Prenex Normal Form . . . 20
- 3.3 Approximating FOL in OWL . . . 24
- 3.4 Pattern application . . . 28

4 IMPLEMENTATION . . . 30
- 4.1 Macleod . . . 30
- 4.2 Object-Oriented Representation . . . 31
- 4.3 Parsing Common Logic . . . 33
- 4.4 Conversion to FF-PCNF . . . 37
- 4.5 Filtering FF-PCNF Sentences . . . 39
- 4.6 Pattern Extraction . . . 39

5 EVALUATION AND DISCUSSION . . . 41
- 5.1 Materials: Ontologies Used for Evaluation . . . 41
- 5.2 Approximation Example . . . 42

5.3 Performance Evaluation 45
5.4 Efficacy Evaluation 46
5.5 Additional Benefits of OWL Approximation 49

6 SUMMARY 51

APPENDIX 53

BIBLIOGRAPHY 57

BIOGRAPHY OF THE AUTHOR 61

LIST OF TABLES

Table 3.1. OWL constructs expressed in FOL. 19
Table 3.2. FF-PCNF sentences expressing OWL constructs. 25
Table 3.3. Grouped filter metrics for OWL FF-PCNF constructs from Table 3.2 27

Table 5.1. Ontologies selected for analysis after approximation 42
Table 5.2. Basic ontology metrics collected 42
Table 5.3. Results of approximation for Dolce, OwlTime, Sumo Temporal, and Sumo Time ontologies. 47
Table 5.4. Results of approximation for PED, Voids Def, FullPhysCont, Inch Calculus, and MapSee ontologies. 48

LIST OF FIGURES

Figure 2.1. A simple example of PNF formatted FOL sentence. 10
Figure 2.2. An example of a CNF formatted FOL sentence. 11
Figure 2.3. Excerpt of the OWL2 Manchester syntax 12
Figure 2.4. Ontology snippet from Figure 1 specified in the OWL/XML syntax. 13

Figure 3.1. Overview of the approach to approximate a FOL sentence using OWL 17
Figure 3.2. Conversion of a FOL sentence to CNF 21
Figure 3.3. An example of function substitution in FF-PCNF 22
Figure 3.4. Conversion of the FOL sentence from Figure 3.2 into FF-PCNF 23
Figure 3.5. Logical equivalences used to coalesce like quantifiers 23
Figure 3.6. Ambiguous cases where the greedy heuristic can be used to decide which quantifier to promote when forming the FF-PCNF prenex 24
Figure 3.7. A decision tree showing how filtering occurs on a FF-PCNF sentence 26
Figure 3.8. Example of how the FOL sentence $\forall x, y[\neg R(x, y) \vee C_1(x) \vee C_2(x) \vee C_3(x)]$ can be expressed using the OWL `ObjectionUnionOf` construct. 29
Figure 3.9. Example showing the approximation of the FF-PCNF sentence $\forall x, y[\neg R(x, y) \vee S(y, x)]$ using the OWL `InverseOf` construct. 29

Figure 4.1. Lightweight UML diagram of the `macleod.logical` Python module. 31
Figure 4.2. Construction of FOL sentence $\exists x, y[A(f(x), x, y \vee B(x, y))]$ in Python using the OO API. 32
Figure 4.3. Python snippet which shows the construction of a FOL sentence. 32
Figure 4.4. List of tokenization rules applied to CLIF character streams 33
Figure 4.5. PLY implementation of the abbreviated CLIF BNF grammer 34
Figure 4.6. BNF grammar describing a subset of the CLIF syntax 36
Figure 4.7. Code snippet showing the FF-PCNF conversion in the Python 38
Figure 4.8. Code snippet of the Macleod extension 40

Figure 5.1. Snippet of the multidim_space_voids ontology 43
Figure 5.2. Excerpt of output produced from our automated approximation 44

CHAPTER 1

INTRODUCTION

The goals of the Semantic Web [BLHL01] and related projects, such as SAPPHIRE [Fei+07], outline the growing need to enable semantic understanding of information for software agents. Coupled with the growing amount of data generated and cataloged each day, efficient methods to enable software systems to leverage that data are growing in importance. Ontologies and the related field of ontology engineering are an established and growing area that can offer solutions to these demands. Through a standardized vocabulary and axioms, ontologies describe relationships betweens objects within a domain of interest. When these relationships are coupled with data and automated inference systems additional conclusions can be derived and a level of semantic understanding of data can be achieved. As the need for ontologies has grown in recognition, languages such as the web ontology language (OWL) and first order logic (FOL) have been utilized, each bringing its own level of expressivity and other benefits to the ontology engineering community.

This thesis is motivated by the observation that there already exists a large number of FOL ontologies available that are otherwise inaccessible to the broader community. This disuse of FOL ontologies is caused in part by the formalized nature of FOL which requires a understaning in logics to robustly utilize, and in part to the undeciablility of FOL as a whole which curbs its adoption in many application settings. Rather than let these rigorous FOL ontologies go unused, a mechanism is needed to allow the general community of ontology creators and consumers to benefit from previous efforts spent on developing FOL ontologies.

Our objective is to advance the work that has been put into the development of densely axiomatized, ontologically rigorous, FOL ontologies by devising a mechanism to automatically create lightweight approximations of FOL ontologies using OWL. Such approximations would allow for efficient reasoning and be more accessible to the larger community of ontology engineers, in particular those who would otherwise avoid FOL for the reasons given above. By creating the lightweight approximations we enable the reuse of existing FOL ontologies with the idea that the approximations could serve as a foundation to build additional ontologies using OWL. Existing OWL software would also add to the scarce set of development tooling currently available to FOL ontology engineers by allowing them to interface with their ontology using OWL specific tools.

Our approach is based on the observation that most natively developed OWL ontologies use only a restricted subset of the language. In particular, the number of synatctic constructs used in OWL ontologies is relatively small, thus the number of constructs needed to automatically produce "native looking" approximations of FOL ontologies is also small. In our approach, we specifically target 20 OWL constructs to use in the approximation of FOL ontologies.

For those 20 OWL constructs, we created a new normal form to enable filtering and comparison of FOL ontology sentences to efficiently search for patterns that indicate the presence of OWL constructs. Our normal form was created by altering the existing prenex and conjunctive normal forms to maintain properties useful for comparing FOL sentences. We identify patterns by manually translating 20 common OWL constructs to FOL which we then normalize using our new normal form. Once a FOL sentence is matched against a pattern, it is inspected further to see if we can automatically produce an approximation using an axiom schema of an OWL construct. If an axiom schema can be applied, then the axiom schema is used to create a corresponding OWL axiom.

This thesis contributes to the field of ontology development by way of an approach and implementation for the automated approximation of existing FOL ontologies. We implement the approach with an extension to the FOL development framework *Macleod* [1] to provide the following new features: 1) a new object-oriented representation of FOL sentences, 2) a syntactically aware common logic (CL) parser, 3) a method to convert FOL sentences to FF-PCNF, and 4) a method to generate OWL approximations of existing FOL ontologies. The underlying approach with the new normal form and implementation provides FOL developers another tool for the development and reuse of FOL ontologies.

Related works have targeted the generation of OWL axioms using specialized inputs, such as from rule-based languages, and other subsets of the FOL language [PB10; SK] or have examined the relationships between different axiomatizations within the same language [DMQ05]. The presented thesis work not only implements the approximation approach to generate OWL ontologies from FOL, but also tests it on a number of FOL ontologies from the COLORE ontology repository [Grü+10].

The approximation method was applied to FOL ontologies from COLORE and nine different ontologies were inspected as the basis for evaluating efficacy. All generated approximations were

[1] Source code is maintained and freely available at https://github.com/thahmann/macleod

logically consistent with no unintended effects observed when inspected using the OWL Protege editor. The approximations captured class and property hierarchies well with OWL `SubclassOf` and `SubObjectPropertyOf` constructs. These kinds of constructs appeared most often in the approximations. OWL axioms `ObjectPropertyDomain` and `ObjectPropertyRange` were often produced in ontologies that also produced approximations with many `SubObjectProperty` axioms. FOL ontologies that use more complex axioms, especially those with multiple nested quantifiers, yielded fewer axioms in the approximated ontology. The inability to produce many axioms from complex FOL ontologies was expected due to OWL's lack of expressivity when compared to FOL. With our approach, axioms containing predicates with an arity greater than 3 do not yield approximations outside of `Class` and `ObjectProperty` definitions. While it may be possible to work around this limitation by aggressively altering the FOL ontology with operations to remove higher arity predicates, the resulting approximation would likely be unrecognizable against the original ontology. In addition, of the many ontologies available in the COLORE repository, only a few make use of ternary predicates which further limits the impact of this restriction.

The remainder of this thesis is organized as follows. Background and related works provides necessary context into the kinds of ontologies and the languages used to specify them. Afterwards in Section 3, we detail the approach taken to satisfy the objective and provide reasoning for the implementation. The implementation is then explained in Section 4, highlighting interesting details for the specific ontologies, languages, and libraries presented and how they were used. In section 5 we detail our findings on the selected ontologies and the efficacy of the approach. Finally, we discuss the key findings and present areas of potential future work in Section 6.

CHAPTER 2

BACKGROUND AND RELATED WORK

To date, the term "ontology" does not have a single shared meaning, rather it encompasses a broad array of differing artifacts. The philosophical study of Ontologies (with a capital "O"), has played a role in this confusion [Hoe15]. Throughout this thesis ontologies will be presented pragmatically, focusing on intention and implementation rather than idealization. For this purpose we view ontologies as artifacts that structure knowledge and that have been captured and formatted in such a way that makes them accessible to computer systems.

By far the most commonly referenced definition of an ontology belongs to Gruber [Gru93] who defined an ontology as, "a specification of a shared conceptualization'. A specification requires a language in order to capture the conceptualization. In practice, these ontology languages are typically formal languages which are mathematically grounded. Just like programming languages, formal ontology languages are geared towards a specific type of problem which determines the kind of constructs they provide [Hen11]. But unlike most programming languages, ontology languages differ in their expressivity.

A key observation is that all ontology languages struggle with a fundamental trade-off between expressivity and tractability. The seminal literature by Levesque and Brachman [LB85] demonstrates that minor changes between languages can have far-reaching consequences on the practical utilization of a particular language. For example, first-order logic is a very expressive language with the caveat that it is undecidable [Bor96; Hod01]. Meanwhile, the relational algebra used by database systems is much less expressive, but possesses the benefits of being impressively efficient [Cod82].

Conceptualization is the harder term to unambiguously define in regards to Gruber's definition. A conceptualization can be described as a specific view of some observation. It could be the observation of a particular business or the whole domain of some scientific field of study. From an implementation perspective, this conceptualization is the collection of concepts of the domain in that observation and how they relate to one another from within that view. The concept of a "seat" might be observed as being, "something to sit on." Then depending on your view, a large rock or stump may qualify as a "seat" whereas another view may strictly limit "seats" as being parts of

chairs. Yet another conceptualization might view “seats” more abstractly as places in a venue that can be sold for event tickets. The important part is that the conceptualization of a “seat” is largely defined via its relationships (“sits”, “has”, and “is made out of”, “available”) with other concepts such as (“people”, “chairs”, “wood”, and etc). If two systems share a conceptualization of a seat they can exchange information about that conceptualization [CJB; GS98]. The point of ontologies is to enable such information exchange.

2.1 Kinds of Ontologies

Through their development and growth, different kinds of ontologies have emerged. A key difference amongst these ontologies is their expressivity. Certain kinds of ontologies utilize very expressive languages while others use more restrictive languages. The two ontology languages of interest to this thesis, first-order logic and OWL2, are covered in Section 2.5. Differences in expressivity are often rooted in and tied to the purpose of an ontology. By purpose, the literature distinguishes three types of ontologies: application, domain, and reference.

2.2 Application Ontologies

As their name suggests, application ontologies (AO) are developed for use within a specific application or application area. Within the literature, AOs have sometimes gone by the name knowledge representation (KR) ontologies [LB85]. By developing AOs, an ontologist aims at utilizing narrower semantics for use on a particular problem. Additionally, because AOs are used by some application they must be formalized, that is, specified in a machine-readable language. AOs are typically formalized in rather inexpressive ontology languages, such as RDF-S, and seldom use the whole expressivity of OWL or first-order logic. This restriction allows AOs to be computationally efficient and more responsive to different reasoning and inferencing programs [LB85; Men03]. In many cases, applications contain implicit ontologies that have been “baked” into the “business logic” rather than being a standalone artifact.

An AO is the least reusable type of ontology in terms of its scope. Unlike other kinds of ontologies, AOs are not developed to be reused multiple times across or even within some domain. A good example of an implicit AO can be found in most application board games, such as chess,

that offer an AI opponent to play against. Obviously the game is programmed with the rules of chess otherwise it wouldn't be a very good game. However, while the application can judge what's a valid move or not it's probably not within its scope to be able to relate what a pawn chess piece is versus a rook. Further, the business logic of knowing how to play the game is probably so tightly coupled with the application itself that it can't be easily extracted and expanded. From a different perspective, even if you were to refactor the application into some form of query interface the queries you could make would be limited. Queries such as, "Is white 1.e4 a valid move?" or "What is white's current set of valid moves?" might be expected. In more advanced versions of the game one could make queries such as, "What is black's next best move?" or "What next sequence of moves would end the game quickest?" However for all variants of the game the queries, "How many players does chess require?" or, "How does chess relate to the other game called Go?" would very likely fall out of scope. The AO is focused on the "how" of playing chess and not necessarily the remaining, "who, what, when, where, and why?"

2.3 Domain Ontologies

In most respects, domain ontologies (DO) are similar to AOs except that they are more general, though in practice rarely cover an entire domain. Instead, they are mostly used to integrate two or more specific applications. Building on the chess example from Section 2.2 a DO *would* be able to encapsulate generalized information about chess the board game rather than chess the application. Imagine now that instead of just having a single chess application there were multiple game applications available and a catalog was needed that could be queried by players. Queries such as, "Number of players required", "Duration of game play", and possibly even "What are the rules to play…" all could fit within the scope of a domain ontology. A goal here would be a certain level of interoperability between the DO and the AO where generalized terms like "Player" or "Game play rules" could be shared to enable information exchange and reuse of the DO across multiple AOs.

2.4 Reference Ontologies

Reference ontologies (RO) sit at the opposite end of AOs in terms of expressivity, scope, and sometimes even formalization [CJB; Men03]. A RO, as the name suggests, is developed to be used as a kind of reference; thus ROs have a broader scope than AOs and DOs. Unlike the chess game example, an RO would ideally be applicable across many different domains. To be useful, ROs require a greater degree of expressivity so that abstract concepts can be captured in a generic enough form for reuse.

Unlike AOs, the term RO has often been used interchangeably to denote several kinds of ontologies within the literature [Hah14; Men03]. Using scope as a metric it is possible to further break down the term RO into sub-categories: Upper and domain reference ontologies. An upper ontology (UO), also called a foundational ontology, is how the term reference ontology has most often been used in the existing literature [Fal+13; GS98]. Upper ontologies focus on capturing truly domain independent knowledge. Thus, they focus on generic terms that span many or virtually all domains and include concepts such as: space, time, mathematics, social constructs, and etc. Examples of upper ontologies are the Descriptive Ontology for Linguistic and Cognitive Engineering (DOLCE) [Mas+03], the Basic Formal Ontology (BFO) [GS04], the General Formal Ontology (GFO) [Her+06], and the Unified Foundational Ontology (UFO) [Gui05].

Domain reference ontologies (DRO) have a more restricted scope than upper ontologies, but still encompass broader concepts than that of AOs or even DOs [Hah14; HSB16]. DROs solve the problem of providing concepts that are not pervasive across domains but are central to a particular domain and all its subdomains and that help connect knowledge about that domain to other, closely related domains. An example is *HyFO* from [HSB16] which encompasses hydrological knowledge. The concepts within *HyFO* are central to hydrology-related domains, but may still be of use in marine biology, oceanography, or even fisheries stock assessment. In terms of expressivity, DROs tend to utilize the same highly expressive languages used for ROs [Fal+13; Grü+10].

Two kinds of reference ontologies can be distinguished based on how they are specified: formalized ROs and informal ROs. A formalized RO is one that has been expressed using a formalized logic such as first-order logic or other higher-order logic. *DOLCE* is an example of a formalized upper ontology. Conceptual reference ontologies are those that have not been specified using a

formal language. An example of a conceptual upper ontology would be the *UFO* project which provides a conceptual model that could be used to create a formalized ontology but does not go so far as to be formalized itself. A conceptual DRO is possible, however at the time of writing no relevant examples exist to the best of our knowledge.

2.5 Ontology Languages

As we have seen in the previous section, differences in the purpose and scope of an ontology are also reflected in the level of formalization needed and thus the choice of specification language used. Informal ROs, sometimes referred to as conceptual ontologies, often use a graphical notation geared towards human consumption such as mind maps or UML diagrams over machine readable specifications. For formalized ontologies, the two logic-based languages first-order logic and description logic have emerged as the most common ontology specification languages. First-order logic is more widely used for ROs whereas OWL and other description logic based languages are the standard for AOs and DOs. Both FOL and OWL are covered in detail in sections 2.5.1 and 2.5.2.

2.5.1 First-order Logic

First-order logic (FOL) is a highly expressive language that is widely used for the specification of most formalized ROs [Men03]. The significant expressive capabilities of FOL are often required in RO ontologies, but come at the cost of increased computational complexity. Unrestricted usage of FOL results in an undecidable ontology [LB85]. Undecidability within an ontology effectively curbs its utilization in any system where tractable reasoning is required. Though work has been done to address this limitation with extensive examination of different syntactic forms of FOL, it remains as a defining limitation [GKV97]. Despite this, the expressive capabilities of FOL, and its established formal underpinnings, solidify its position as the language of choice for the formalization of ROs.

FOL as a language is comprised of an infinite set of variables and a fixed set of logical symbols which include quantifiers ($\forall, \exists$) and a set of logical connectives including: and ($\wedge$), or ($\vee$), negation ($\neg$), and implications ($\rightarrow, \leftrightarrow$). A set of non-logical symbols ($x..y$), predicates (P_n), functions (f_x), and constants (C) construct terms and formulae (e.g. $\forall x[P_n(x) \wedge (P_n(f_x(x)) \vee P_n(C))]$). Syntactically, FOL defines a set of well-formed formulas (WF) recursively as follows:

1. Any single predicate over arbitrary variables, constants, or terms (which can involve function symbols as well) is WF
2. If φ is WF then $\neg\varphi$ is also WF
3. If φ and ϵ are WF then $\varphi \wedge \epsilon$, $\varphi \vee \epsilon$, $\varphi \rightarrow \epsilon$, and $\varphi \leftrightarrow \epsilon$ are also WF
4. If φ is WF and x is a variable then $\exists x\varphi$ and $\forall x\varphi$ are also WF

A formula that contains no free variables, that is a variable not bounded by any quantifier, is said to be a first order sentence. The semantics of FOL are defined using an interpretation I which is a structure consisting of:

1. Domain of discourse D
2. Set of functions $F : D^n \rightarrow D$ mapping each function symbol $f_n \in \Sigma$ such that $f_n(a_1, \ldots, a_k) = b$ iff $I(F_n(I(a_1), \ldots, I(a_k))) = I(b)$ where $b \in D$
3. Set of relations p which map each predicate $p_n(a_n, \ldots, a_k)$ to a truth value $\{true, false\}$

Truth value of sentences are then evaluated given the following rules:

1. An atom $p_n(a_1, \ldots, a_k)$ is true if $(v_1, \ldots, v_n) \in I(P_n)$ and false if $(v_1, \ldots, v_n) \notin I(P_n)$ where $(v_1, \ldots v_n)$ are the mapped terms $(a_1, \ldots, a_k)$ under the interpretation
2. Logical connectives $(\rightarrow, \leftrightarrow, \vee, \wedge)$ and negation $(\neg)$ over terms are evaluated according to their associated truth tables

    a. If φ is true then $\neg\varphi$ is false and vice-versa

    b. If $\varphi = true$ and $\epsilon = false$ then only $\varphi \wedge \varphi$ is $true$ whilst $\varphi \wedge \epsilon$ and $\epsilon \wedge \epsilon$ are $false$

    c. Similarly if $\varphi = true$ and $\epsilon = false$ then $\varphi \vee \epsilon$ is $true$ whereas only $\epsilon \vee \epsilon$ is $false$

    d. Implication is evaluated following the above rules where $\varphi \rightarrow \epsilon$ is equivalent to $\neg\varphi \vee \epsilon$

    e. Biconditionals are evaluated following the rules above where $\varphi \leftrightarrow \epsilon$ is equivalent to $(\varphi \rightarrow \epsilon) \wedge (\epsilon \rightarrow \varphi)$

$$\underbrace{\forall x, y, z}_{\text{Prenex}}[\underbrace{R(x,y) \rightarrow [R(x,z) \rightarrow y = z]}_{\text{Matrix}}] \tag{2.1}$$

Figure 2.1: A simple example of PNF formatted FOL sentence.

3. Existential quantifications $(\exists x \varphi(x)))$ is true if and only if there is some element $d \in D$ such that $I(\varphi(d))$ is true
4. Universal quantifications $(\forall \varphi(x))$ is true if and only if for all individuals $y \in D$ it holds that $I(\varphi(y))$ is true

A sentence ϕ is said to be satisfiable if there exists some interpretation M (often called the model) under which it evaluates to true. In this instance is it said that M satisfies ϕ denoted as $M \models \phi$. A set of sentences (e.g. an ontology) for which there exists a model is said to be consistent. For sentences that contain free variables (e.g. variables not bound by a quantifier), an interpretation I is not sufficient to assign a truth value. Free variables in a sentence must be mapped by a variable assignment μ to obtain some individual within the domain of discourse. After a mapping has been made, sentences with free values are evaluated to true under M and μ given the rules above [Men97].

Normal Forms Normals forms within FOL are well documented and studied structures of sentences. Representation of a sentence in a particular form is usually done for some utility. The utilities of normal forms range from simplification in certain reasoning tasks to easier examination of certain properties. Prenex normal forms are sentences of the form $(Q_1 y_1, ...Q_n y_n)\varphi$ where each $Q_i y_i$ is either $\forall y_i$ or $\exists y_i$ and $y_i \neq y_j$ when $i \neq j$ and φ contains no quantifiers. The chain of $Q_1 y_1, ...Q_n y_n$ is called the prenex while φ is called the matrix [Hin05]. Note that the notational form $\forall x, y, z \exists a, b$ is a syntactic shortcut and represents the prenex $\forall x \forall z \forall y \exists a \exists b$. Any sentence can be expressed in an equivalent prenex normal form as shown in [Men97]. A benefit of working with prenex normal form in this thesis comes from the relocation and clear ordering of the quantifiers in the prenex of each sentence. In the more general sense, prenex normalized sentences are often simpler to work with and analyze since the quantifiers are separated from the matrix as show in Figure 2.1.

$$\underbrace{\forall x, y, z}_{\text{Prenex}}[\underbrace{A(x) \wedge B(y) \wedge \overbrace{(C(z)}^{\text{Term}} \vee D(z))}_{\text{Conjunction}}] \tag{2.2}$$

Figure 2.2: An example of a CNF formatted FOL sentence.

Conjunctive normal form (CNF) is another well-studied normal form which is a specialized prenex normal form (see Figure 2.2). A CNF sentence is denoted by a prenex comprised of only universal quantifiers and whose matrix is a strict conjunction over disjunctive terms such as $(A \wedge (B \vee C) \wedge D)$. Unlike generic prenex normal form, CNF does not maintain logical equivalence because of skolemization that occurs to remove existential quantifiers. Sentences that have been converted to CNF do maintain satisfiability [Rus+]. While prenex normal form gives structure to a sentence in the form of a prenex, CNF gives structure to the remaining matrix. CNF is popular for a number of applications including automated theorem proving where first-order resolution techniques are needed and depend on the matrix having a certain structure.

2.5.2 Web Ontology Language

The web ontology language (OWL) is a standard for ontologies on the web [Hit+09]. Created as part of the semantic web project, it has become the preferred language for explicit AOs and DOs. OWL is based on description logics (DL), which generally are restricted versions of first-order logic [BHS04; Hit+09]. By restricting what can be expressed, DLs tackle the undecidability problem of FOL by providing different complexity guarantees depending on how, and how much of, the language is used. In the latest version of the OWL specification, OWL2, the creators provide different profiles of the language. Each profile combines the different semantic constructs of the OWL2 specification in such a way that controls the tractability of reasoning within the language. The result is a flexible specification that can be adapted according to developers' computational requirements. OWL is specified using well documented constructs which address specific modeling needs and are more readily accessible to developers who may be unfamiliar with formal logics. OWL comes in a variety of syntaxes including extensible markup language (XML), functional, Manchester, and Turtle as given in the OWL specification [Hit+09].

```
Class: Father
  SubClassOf: Man and Parent

Class: Parent
  EquivalentTo: hasChild some Person

DisjointClasses: Mother, Father

ObjectProperty: hasUncle
  SubPropertyChain: hasFather o hasBrother

Individual: Bill
  Facts: not hasDaughter Susan
```

Figure 2.3: Excerpt of the OWL2 Manchester syntax from the OWL2 Language Primer [Hit+09]. Entities `Father` and `Parent` are denoted by class expression axioms. The individual `Bill` is described through usage of an assertion.

Ontologies specified in OWL are built from entities, expressions, and axioms. Entities denote classes, properties, and individuals and make up the non-logical part of the language much like non-logicals symbols in FOL. Individuals represent literal objects within the domain where classes are sets of individuals and properties connect pairs of individuals. Expressions are a inductively defined as part of OWL2's syntax used to describe classes, properties, and individuals. OWL expressions are broadly categorized into object property expressions, data property expressions, and class expressions. Object property expressions are the simpler category and consist of either object properties or inverse object properties. Data properties are a bit of a syntactic shortcut to relate individuals to well defined datatypes like integers, strings, and etc. Class expressions are more varied and offer a larger set of primitives to construct complex expressions. Finally, axioms are those expressions which are interpreted as statements of truth about the domain.

Axioms are typically either assertions about individuals (e.g. axioms that are only about specific individuals and not about classes of individuals), object property axioms (e.g. axioms that constrain the interpretation of one or a set of object properties), or class expression axioms (e.g. axioms that constrain the interpretation of one or multiple classes). Class expression axioms establish relationships between classes within the domain and include expressions for declaring subclasses, equivalency, or disjointness between classes.

```
<SubClassOf>
  <Class IRI="Father"/>
  <ObjectIntersectionOf>
    <Class IRI="Man"/>
    <Class IRI="Parent"/>
  </ObjectIntersectionOf>
</SubClassOf>

<EquivalentClasses>
  <Class IRI="Parent"/>
  <ObjectSomeValuesFrom>
    <ObjectProperty IRI="hasChild"/>
    <Class IRI="Person"/>
  </ObjectSomeValuesFrom>
</EquivalentClasses>

<DisjointClasses>
    <Class IRI="Father"/>
    <Class IRI="Mother"/>
</DisjointClasses>

<SubObjectPropertyOf>
  <ObjectPropertyChain>
    <ObjectProperty IRI="hasFather"/>
    <ObjectProperty IRI="hasBrother"/>
  </ObjectPropertyChain>
  <ObjectProperty IRI="hasUncle"/>
</SubObjectPropertyOf>

<NegativeObjectPropertyAssertion>
  <ObjectProperty IRI="hasDaughter"/>
  <NamedIndividual IRI="Bill"/>
  <NamedIndividual IRI="Susan"/>
</NegativeObjectPropertyAssertion>
```

Figure 2.4: Ontology snippet from Figure 1 specified in the OWL/XML syntax.

Object property axioms establish and describe relationships between object property expressions. Object property axioms include sub-properties, equivalency, disjointness, and inversions much like the available class expression axioms. Object property axioms also include more complex characterizations of the interaction of object properties and classes, such as restrictions on the domain or range of a property as well as declaring property traits such as being functional, reflexive, symmetric, and transitive.

Unlike axioms, assertions make truth statements about individuals within the domain much like a database of facts. The available assertions in OWL include assertions for individual equivalency, distinctness, class membership, and relational assertions via an object property. It is important to note that OWL does not make the unique name assumption so explicitly declaring individuals to be different is regularly needed.

2.6 Related Works

Within the field of ontology engineering much work has been done examining the relationships between different ontology languages and the underlying logics they are based upon. Largely the work can be partitioned into two areas: (1) works concerned with examining the relation between ontologies specified in the same language [DMQ05; Grü+12], but with different semantics, and (2) works concerned with relating or converting between ontologies specified in different languages [Bor96; Mos17; MK11]. This thesis focuses on the latter and specifically looks at a more pragmatic approach to produce a working approximation of a FOL ontology in OWL2.

The collective works of Hendler [Hen11], Borgida [Bor96], and Baader [BHS04] form the basis of much of the current literature on works bridging ontologies specified in different languages. Respectively each piece of literature provides the formal comparison of the underlying formal logics (propositional logic, description logic, and first-order logic) and their variants and provides the foundations on which further analysis is based. In particular, [Bor96] provides formal translations to FOL for the syntactic constructs found in DL which is the formal underpinning of OWL. Many of the translations are still applicable to OWL and were leveraged in this thesis to create FOL *patterns* for approximation which will be discussed in Chapter 3.

The ongoing work in the definition of the distributed ontology language (DOL) from [Mos17; MK11] aims to provide a meta-language solution for specifying relationships between ontologies in different logical languages. Similar to this thesis, Mossakowski [MK11] recognizes the widening gap between the available ontology languages and the benefit that could be acquired by providing a mechanism that could bridge the languages to aid ontology engineers and avoid redundancies in developing ontologies. But unlike this thesis, which provides a direct approximation method between FOL and OWL2, the DOL provides a formal method to connect ontologies specified in different ontology languages. However, this comes at the price of expensive reasoning (which involves meta-reasoning over multiple logics) and not being able to reuse the many available reasoners for FOL and OWL. At the time of writing, the heterogeneous toolset (HETS) [MML07] is the only tool that supports a set of the DOL language and is leveraged by the Ontohub (see http://ontohub.org) project.

Other works are more focused on supporting the development of OWL ontologies for domain experts not formally trained in logic. Sarker [SK] provides a Protege [Mus15] plugin to enable entry of axioms into an OWL ontology using the FOL-alike rule based language SWRL. A difference between Sarker's work and this thesis is that their approach in [SK] is focused on helping experts formulate new axioms. This thesis supports reusing existing axiomatizations, currently in the form of FOL ontologies, to create entire OWL2 ontologies on the fly without much user input.

CHAPTER 3

APPROACH

Approximating first order logic (FOL) sentences into web ontology language (OWL) expressions presents multiple issues to be solved. The overarching challenge is that some highly expressive FOL statements are inexpressible using OWL. A second but closely related challenge is the ability to encode the same logical theory using multiple different FOL axiomatizations. A final challenge is how to identify FOL axioms that correspond to available OWL expressions. In this chapter, we present our developed approach (see Figure 3.1) for addressing these challenges in order to automatically produce an OWL ontology that approximates a given FOL ontology.

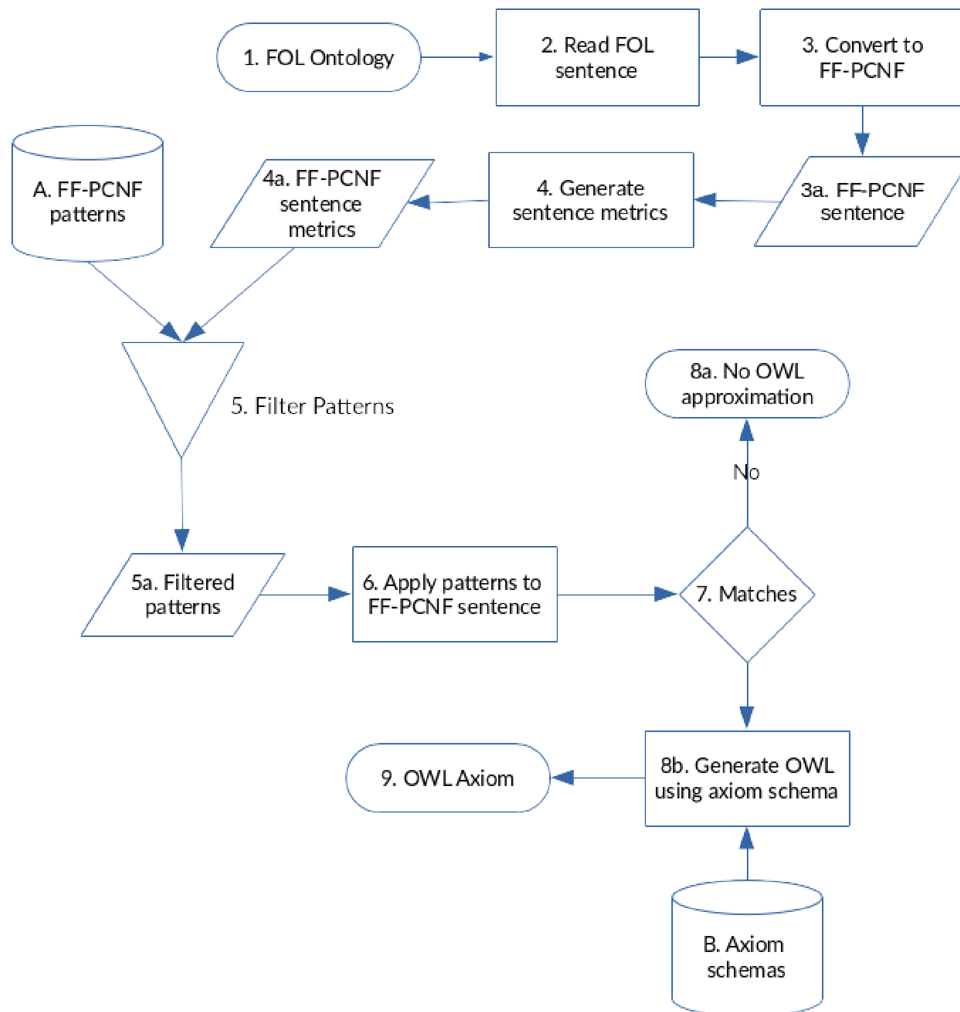

Figure 3.1: Overview of the approach to approximate a FOL sentence using OWL. Sentences are read from a FOL ontology (1) and then converted to yield function-free prenex normal form (FF-PCNF) sentences (3a). Once converted the sentences are examined using generalized metrics and compared (5) against a set of already generated patterns (A). Patterns that match (7) are then applied to a sentence using an axiom schema (B) to produce an OWL axiom (9).

3.1 OWL constructs in FOL

FOL provides a small and generic set of logical connectives to capture the semantics of a domain but it does not constrain or guide how to express it logically [Hod01]. OWL axioms are built using a larger fixed set of constructs that are much closer to the kind of knowledge that people want to capture. These OWL constructs capture specific semantic intuitions of a domain and thus limit the kind of axioms people can write. Because of this difference in syntax, it is more appropriate to create templates of common OWL axioms and express them as FOL statements rather than build an inventory of the possible ways people could write FOL axioms.

Using the OWL ontology in Figure 2.3 as an example we can see the difference in difficulty working backwards from OWL versus trying to enumerate all the possibilities. The class Father can be defined succinctly as $\forall x[Father(x) \leftrightarrow Man(x) \wedge Parent(x)]$. Parent can be defined as $\forall x[Parent(x) \leftrightarrow \exists y[hasChild(x, y) \wedge Person(y)]]$. And the final assertion, "Susan is not Bill's daughter" is also straightforward as $\neg hasDaughter(Bill, Susan)$. However, if we tried to enumerate all the possible ways we could define Father in FOL it is clear that FOL expressiveness starts to get in the way: $\forall x[\neg Father(x) \vee Man(x) \wedge Parent(x)]$, $\forall x[\neg Father(x) \vee Man(x) \wedge \neg Father(x) \vee Parent(x)]$, $\neg\exists x[Father(x) \wedge \neg Man(x) \vee \neg Parent(x)]$, are all correct FOL sentences with even more complex FOL manipulations becoming possible. By starting with a fixed set of OWL axioms, and clear FOL approximations of those axioms, we can identify templates of what is needed for a good approximation.

By leveraging these templates we can work backwards to identify common expressions in OWL that we want to be able to extract from FOL sentences. By doing so we ensure that the OWL approximation of the FOL ontology is comprised of the kinds of axioms that developers would expect from a "native OWL ontology" (e.g. one that was originally written in OWL). While greater attention can be paid to capturing as much of the domain as possible in the approximation, a goal of our work is to produce OWL ontologies that are readily accessible to developers to be utilized further. To this end, we captured the "developer expected" OWL constructs (e.g. axioms like those found in Figures 2.3 and 2.4) and left the more complex or expert oriented expressions (such as the hasUncle assertion in Figure 2.3) for future extensions. Table 3.1 lists the OWL constructs that we cover in our study with an accompanying FOL representation for reference.

	OWL	Equivalent FOL sentence
1	`SubClassOf(C D)`	$\forall x[C(x) \rightarrow D(x)]$
2	`DisjointClass(C D)`	$\forall x[C(x) \rightarrow \neg D(x)]$
3	`SubObjectPropertyOf(R S)`	$\forall x, y[R(x,y) \rightarrow S(x,y)]$
4	`DisjointObjectPropertyOf(R S)`	$\forall x, y[R(x,y) \rightarrow S(x,y)]$
5	`ObjectPropertyDomain(R C)`	$\forall x, y[R(x,y) \rightarrow C(x)]$
6	`ObjectPropertyRange(R C)`	$\forall x, y[R(x,y) \rightarrow C(y)]$
7	`InverseObjectProperty(R S)`	$\forall x, y[R(x,y) \leftrightarrow S(y,x)]$
8	`ReflexiveProperty(R)`	$\forall x[R(x,x)]$
9	`IrreflexiveProperty(R)`	$\forall x[\neg R(x,x)]$
10	`SymmetricProperty(R)`	$\forall x, y[R(x,y) \rightarrow R(y,x)]$
11	`AsymmetricProperty(R)`	$\forall x, y[R(x,y) \rightarrow \neg R(y,x)]$
12	`TransitiveProperty(R)`	$\forall x, y, z[R(x,y) \rightarrow [R(y,z) \rightarrow R(x,z)]]$
13	`FunctionalProperty(R)`	$\forall x, y, z[R(x,y) \rightarrow [R(x,z) \rightarrow y = z]]$
14	`InverseFunctionalProperty(R S)`	$\forall x, y, z[R(x,y) \rightarrow [R(z,y) \rightarrow x = z]]$
15	`SubClassOf(C ObjectSomeValuesFrom(R D))`	$\forall x \exists y[C(x) \rightarrow R(x,y) \wedge D(y)]$
16	`SubClassOf(ObjectSomeValuesFrom(R D) C)`	$\forall x \exists y[R(x,y) \wedge D(y) \rightarrow C(x)]$
17	`SubClassOf(C ObjectAllValuesFrom(R D))`	$\forall x, y[C(x) \wedge R(x,y) \rightarrow D(y)]$
18	`SubClassOf(ObjectAllValuesFrom(R D) C)`	$\forall x, y[R(x,y) \rightarrow D(y) \rightarrow C(x)]$
19	`SubClassOf(C ObjectHasSelf(R))`	$\forall x[C(x) \rightarrow R(x,x)]$
20	`SubClassOf(ObjectHasSelf(R) C)`	$\forall x[R(x,x) \rightarrow C(x)]$

Table 3.1: OWL constructs expressed in FOL.

As illustrated by the Father concept it is important to note that the FOL sentences shown in Table 3.1 are not the only ways to express the OWL constructs. Because of the small and generic set of logical connectives in FOL, it is possible to express the same OWL construct in multiple ways with no loss of equivalency. As another example, the `DisjointClass(C D)` construct can also be axiomatized as $\forall x[\neg C(x) \vee \neg D(x)]$ or $\neg\exists x[C(x) \wedge D(x)]$. A good algorithm for approximation should be able to cope with such syntactic variations and deal more directly with the intended semantic meaning. To achieve this a suitable normalization needs to be identified and adapted to catch a wider range of semantically equivalent but syntactically different FOL axiomatizations.

3.2 Function-Free Prenex Normal Form

Normal forms exist to constrain the structure of an expression for better fitness in a particular application. In our work, we adapted conjunctive normal form (CNF) to make it easier to compare FOL sentences with the FOL form of the OWL constructs we wanted to appear in the approximations. Specifically, we needed the normal form to fulfill three key requirements: 1) to minimize syntactic variability of expressions, 2) to maintain satisfiability, and 3) to maintain both universal and existential quantifiers.

The requirements were chosen to enable efficient comparison between the FOL specified OWL constructs and FOL ontology sentences. By minimizing syntactic variation we reduce the likelihood that equivalent sentences appear with different structures which may make them difficult to compare. For the second requirement, the normal form needs to maintain satisfiability so that any OWL constructs that are found equivalent to the FOL sentence would, by the equivalency, also maintain satisfiability. Finally, by preserving both the universal and existential quantifiers it becomes easier to compare sentences with the OWL constructs which may contain either type of quantifier.

CNF is a well studied normal form that structures a FOL expression as a universally quantified sentence comprised of a single conjunction over several disjunctive terms. CNF does not maintain logical equivalence but does maintain the satisfiability of the pre-CNF FOL sentence. A key component of CNF that was not well suited to our goals was the removal of existential quantifiers. A common OWL construct, `SomeValuesFrom`, utilizes existential quantification so it was desirable to

$$\forall x, y[A(x,y) \rightarrow \exists z[B(f(y),z) \wedge D(z)]] \tag{3.1a}$$
$$\equiv \forall x, y[\neg A(x,y) \vee \exists z[B(f(y),z) \wedge D(z)]] \tag{3.1b}$$
$$\equiv \forall x, y \exists z[\neg A(x,y) \vee (B(f(y),z) \wedge D(z))] \tag{3.1c}$$
$$\equiv \forall x, y[\neg A(x,y) \vee (B(f(y),f_z(x,y)) \wedge D(f_z(x,y))] \tag{3.1d}$$
$$\equiv \neg A(x,y) \vee (B(f(y),f_z(x,y)) \wedge D(f_z(x,y)) \tag{3.1e}$$
$$\equiv (\neg A(x,y) \vee B(f(y),f_z(x,y))) \wedge (\neg A(x,y) \vee D(f_z(x,y))) \tag{3.1f}$$

Figure 3.2: Conversion of the FOL sentence $\forall x, y[A(x,y) \rightarrow \exists z[B(f(y),z) \wedge D(z)]]$ to CNF. Note that the intermediate sentence 3.1c has already been put into PNF where the existential is still present and the prenex has been formed. Sentence 3.1d is where the existentially quantified variable z is skolemized and replaced by the skolem function f_z, it is at this step where only satisfiability to the original FOL sentence is maintained. Sentences 3.1e and 3.1f drop the existential quantifier and distribute terms until the sentence has been put into CNF.

keep it in the normalized form for easier identification (see removal of the existential quantifier in Figure 3.2).

CNF is a specialization of another normal form called prenex normal form (PNF). PNF is a well studied normal form that, unlike CNF, maintains logical equivalence with the original FOL sentence. For a sentence to be in PNF, all quantifiers must exist in the very front of a sentence (maximally scoped), in a portion called the prenex, whereas the rest of the terms preside in a quantifier free portion called the matrix (see sentence 3.1c of Figure 3.2). Unlike CNF, PNF doesn't impose any additional restrictions on the structure of the matrix so it by itself didn't meet our first requirement to try and minimize variance in the structure of the sentence.

Having reviewed the existing PNF and CNF, we present a new modified normal form, function-free prenex conjunctive normal form (FF-PCNF), to enable the task of efficient filtering and comparison of sentences for approximating FOL ontologies using OWL. FF-PCNF is an altered CNF where existential quantifiers are retained and function symbols are replaced with functional predicates. FF-PCNF has the same matrix structure of regular CNF sentence where the sentence matrix consists of a single conjunction over several disjunctive terms. However, unlike regular CNF, the prenex may contain existential quantifiers. FF-PCNF also differs from CNF in that function symbols are replaced with a new predicate within a universally quantified FOL formula which maintains satisfiability with the original FOL sentence. In addition, in cases where the new FOL formula in-

$$\forall x, y[A(y,x) \rightarrow B(x, d(x))] \tag{3.2a}$$
$$\equiv \forall x, y[A(y,x) \rightarrow \forall a[B(x,a) \wedge d'(x,a)] \wedge \forall x, y, z[d'(x,y) \wedge d'(x,z) \rightarrow y = z] \tag{3.2b}$$
$$\equiv \forall x, y, a[A(y,x) \rightarrow B(x,a) \wedge d'(x,a)] \wedge \forall x, y, z[d'(x,y) \wedge d'(x,z) \rightarrow y = z] \tag{3.2c}$$

Figure 3.3: An example of function substitution in FF-PCNF. In 3.2a d is the unary function which will be replaced in 3.2b with a new FOL binary predicate d' in $\forall x[eq(x,z) \wedge d'(x,a)]$. Sentence 3.2b also shows the addition of a new FOL sentence to define d' as being functional which is critical to maintaining satisfiability. Sentence 3.2c shows quantifier promotion to simplify the sentence. $=$ is used to represent the equality operator.

troduces a binary predicate, another FOL sentence is added which defines the new predicate as being functional as shown in Figure 3.3. The addition of a new axiom where applicable ensures the introduced predicate better approximates the replaced function symbol. Like CNF, FF-PCNF is a specialization of PNF.

Converting into FF-PCNF closely resembles the procedure to convert to CNF [Men97]:

1. Convert all implications into disjunctions within the sentence
2. Any instance of negation needs to be distributed such that it applies to predicates only
3. All functions must be replaced with a new predicate and an additional sentence must be introduced to maintain equivalence for each replaced function
4. All variables must be standardized to be uniquely identified.
5. The prenex must be created by promoting quantifiers to maximize their scope
6. Disjunctions must be distributed inwards over conjunctions until the sentence is in FF-PCNF

Note, like with conversion to CNF, the final distribution step (6) in the conversion to FF-PCNF has the potential to exponentially increase in the number of terms in the sentence.

Quantifier Coalescing Unlike with CNF, FF-PCNF retains existential quantifiers and requires additional care when crafting the sentence prenex in step 5 of the conversion. In particular it's possible to create a sentence prenex with a lesser number of quantified variables than in the original sentence through the application of quantifier coalescence. Quantifier coalescing leverages

$$
\begin{aligned}
&\forall x, y[A(x,y) \rightarrow \exists z[B(f(y),z) \wedge D(z)]] && \text{(3.3a)}\\
&\quad = \forall x, y[\neg A(x,y) \vee \exists z[B(f(y),z) \wedge D(z)]] && \text{(3.3b)}\\
&\quad = \forall x, y[\neg A(x,y) \vee \exists z[\forall a[B(a,z) \wedge f_y(y,a)] \wedge D(z)]] && \text{(3.3c)}\\
&\quad = \forall x, y[\neg A(x,y) \vee \exists z \forall a[B(a,z) \wedge f_y(y,a)] \wedge D(z)] && \text{(3.3d)}\\
&\quad = \forall a, b[\neg A(a,b) \vee \exists c \forall d[B(d,c) \wedge f_y(b,d)] \wedge D(c)] && \text{(3.3e)}\\
&\quad = \forall a, b \exists c \forall d[\neg A(a,b) \vee (B(d,c) \wedge f_y(b,d) \wedge D(c))] && \text{(3.3f)}\\
&\quad = \forall a, b \exists c \forall d[(\neg A(a,b) \vee B(d,c)) \wedge (\neg A(a,b) \vee f_y(b,d)) \wedge (\neg A(a,b) \vee D(c))] && \text{(3.3g)}
\end{aligned}
$$

Figure 3.4: Conversion of the FOL sentence from Figure 3.2 into FF-PCNF. Note that final FOL sentence is very different from the result of both the CNF and PNF sentences from Figure 3.2. Sentence 3.3c is where function substitution occurs where the term $B(f(y), z)$ is replaced by the new clause $B(a, z) \wedge f_y(y, a)$. Note that the additional FOL sentence which restricts f_y to being functional is omitted for brevity (see Figure 3.3 for example). Sentence 3.3f is where the PNF quantifier promotion occurs and finally 3.3g is the result of distribution of disjunctions over conjunctive terms.

$$
\begin{aligned}
\forall x[A(x)] \wedge \forall y[B(y)] &\iff \forall z[A(z) \wedge B(z)] && \text{(3.4)}\\
\exists x[A(x)] \vee \exists y[B(y)] &\iff \exists z[A(z) \vee B(z)] && \text{(3.5)}
\end{aligned}
$$

Figure 3.5: Logical equivalences used to coalesce like quantifiers. Universal quantifiers coalesce over conjunctions (3.4) whereas existential quantifiers coalesce over disjunction (3.5).

the logical equivalences shown in Figure 3.5 to combine variables of like-quantifiers under certain conditions.

As is covered in more detail in Section 3.3, sentences with a lesser number of quantified variables have a better chance of producing an approximation. While not a specific step in the general conversion to FF-PCNF, we leverage this observation to implement a greedy heuristic while creating the sentence prenex to minimize the number of quantified variables.

The greedy heuristic uses a single lookahead to decide what quantifier to promote in cases where there are more than a single option available. As shown in sentences 3.6a and 3.6d there is a choice available as to which quantifier to place first in the prenex. In this case, we would look to see if sentence 3.6a is actually a term in another connective and then promote the quantifier which has the potential to be coalesced under that connective. In the case where there are no parent connectives we promote the $\forall$ quantifier as that matches more patterns as shown in Section 3.3.

$$\forall x[\varphi] \wedge \exists y[\epsilon] \tag{3.6a}$$
$$\equiv \forall x \exists y[\varphi \wedge \epsilon] \tag{3.6b}$$
$$\equiv \exists y \forall x[\varphi \wedge \epsilon] \tag{3.6c}$$

$$\forall x[\varphi] \vee \exists y[\epsilon] \tag{3.6d}$$
$$\equiv \forall x \exists y[\varphi \vee \epsilon] \tag{3.6e}$$
$$\equiv \exists y \forall x[\varphi \vee \epsilon] \tag{3.6f}$$

Figure 3.6: Ambiguous cases where the greedy heuristic can be used to decide which quantifier to promote when forming the FF-PCNF prenex. Note that this is only valid when x does not appear free in ϵ and y does not appear free in φ. $\forall x \exists y[A(x,y)]$ is *not* equivalent to $\exists y \forall x[A(x,y)]$. In cases where differently quantified variables appear in a function or predicate then the ordering of the quantifiers is fixed and cannot be altered as shown in the example above.

3.3 Approximating FOL in OWL

FF-PCNF provides the foundation to represent FOL statements in a standardized format to identify the presence of OWL constructs. Using FF-PCNF we manually converted each OWL construct from Table 3.1 into a FF-PCNF sentence, which are shown in Table 3.2. One might suspect this is sufficient to enable the approximation of FOL ontologies using OWL. However, given that a FOL ontology may contain many sentences, and the conversion to FF-PCNF can exponentially grow the number of terms in the sentence, an efficient mechanism is needed to filter the number of OWL constructs we attempt to approximate from each sentence.

In general, a FOL ontology may contain only a few sentences or several hundred. During conversion to FF-PCNF those FOL sentences have the potential to exponentially grow in the number of terms present. This growth in sentence size, along with the chance of a more expressive FOL sentence not yielding a single OWL approximation, leaves open the possibility for many wasted comparisons and attempts at approximation. To optimize the approximation of FOL ontologies using OWL we introduce a method of filtering FOL sentences to minimize the number of attempts at approximation.

By inspecting the sentences in Table 3.2 it is possible to create a filtering method based on simple metrics found in each FF-PCNF sentence. In order, these metrics are: type and order of quantifiers, the number of quantified variables, the presence and sign of unary predicates, and

#	OWL	FOL FF-PCNF
	Class Expressions	
1	`SubClassOf(C D)`	$\forall x[\neg C(x) \vee D(x)]$
2	`DisjointClass(C D)`	$\forall x[\neg C(x) \vee \neg D(x)]$
	Property Expression	
3	`SubObjectPropertyOf(R S)`	$\forall x, y[\neg R(x, y) \vee S(x, y)]$
4	`DisjointObjectPropertyOf(R S)`	$\forall x, y[\neg R(x, y) \vee \neg S(x, y)]$
5	`ObjectPropertyDomain(R C)`	$\forall x, y[\neg R(x, y) \vee C(x)]$
6	`ObjectPropertyRange(R C)`	$\forall x, y[\neg R(x, y) \vee C(y)]$
7	`InverseObjectProperty(R S)`	$\forall x, y[\neg R(x, y) \vee S(y, x)], \forall xy[S(y, x) \vee R(x, y)]$
8	`ReflexiveProperty(R)`	$\forall x[R(x, x)]$
9	`IrreflexiveProperty(R)`	$\forall x[\neg R(x,]$
10	`SymmetricProperty(R)`	$\forall x, y[\neg R(x, y) \vee R(y, x)]$
11	`AsymmetricProperty(R)`	$\forall x, y[\neg R(x, y) \vee \neg R(y, x)]$
12	`TransitiveProperty(R)`	$\forall x, y, z[\neg R(x, y) \vee \neg R(y, z) \vee R(x, z)]$
13	`FunctionalProperty(R)`	$\forall x, y, z[\neg R(x, y) \vee \neg R(x, z) \vee = (y, z)]$
14	`InverseFunctionalProperty(R S)`	$\forall x, y, z[\neg R(x, y) \vee \neg R(z, y) \vee = (x, z)]$
	Quantification	
15	`SubClassOf(C ObjectSomeValuesFrom(R D))`	$\forall x \exists y[[\neg C(x) \vee R(x, y)] \wedge [\neg C(x) \vee D(y))]]$
16	`SubClassOf(ObjectSomeValuesFrom(R D) C)`	$\forall x \exists y[\neg R(x, y) \vee \neg D(x) \vee C(y)]$
17	`SubClassOf(C ObjectAllValuesFrom(R D))`	$\forall x, y[\neg C(x) \vee \neg R(x, y) \vee D(y)]$
18	`SubClassOf(ObjectAllValuesFrom(R D) C)`	$\forall x, y[[R(x, y) \vee C(x)] \wedge [\neg D(y) \vee C(x)]]$
19	`SubClassOf(C ObjectHasSelf(R))`	$\forall x[\neg C(x) \vee R(x, x)]$
20	`SubClassOf(ObjectHasSelf(R) C)`	$\forall x[\neg R(x, x) \vee C(x)]$

Table 3.2: FF-PCNF sentences expressing OWL constructs.

$$\forall x, y[\neg R(x, y) \vee C(x)] \tag{3.7}$$

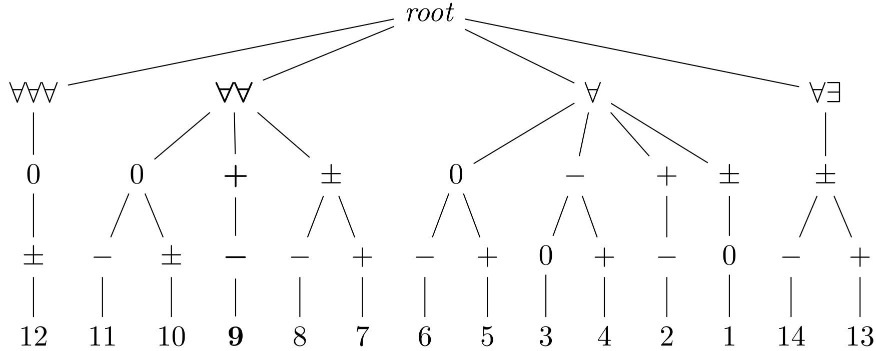

Figure 3.7: A decision tree showing how filtering occurs on the OWL `ObjectPropertyDomain` FF-PCNF construct. By following the metrics we see that it matches: Two univesally quantified variables, presence of a positive unary predicate, and finally presence of a negated binary predicate. Filtering of this FF-PCNF sentence places it in filter group 9 (see Table 3.3).

finally the presence and sign of binary predicates. Using these simple metrics we create fourteen "filter" groups as shown in Table 3.3. Better still, nine of the fourteen groups consist of a single OWL construct. By filtering FF-PCNF sentences from a FOL ontology using the metrics from Table 3.3 to determine what group they correspond with we reduce the number of OWL constructs we try to approximate from a single sentence from twenty down to at most three.

The four identified metrics partition the OWL constructs into the fourteen groups as shown in Table 3.3. Each group has no more than three possible patterns, and more than half contain only a single pattern. Further, it becomes possible to filter FOL sentences in a tree like manner as shown in Figure 3.7. By using the filter tree we short-circuit and stop comparison for FF-PCNF sentences that are no longer able to be filtered into a group. In cases of complex FOL ontologies this has the potential to save many comparisons on sentences that will not yield any OWL approximation. Once a FF-PCNF sentence has been filtered it is then possible to inspect the sentence more closely to determine if it matches an OWL construct.

	Construct	Form of the Prenex	Presence of Unary Predicates	Presence of Binary Predicates
1	`SubClassOf(C D)`	$\forall$	$\pm$	0
2	`SubClassOf(ObjectHasSelf(R) C)`	$\forall$	$+$	$-$
3	`DisjointClass(C D)`	$\forall$	$-$	0
4	`SubClass(C ObjectHasSelf(R))`	$\forall$	$-$	$+$
5	`ReflexiveProperty(R)`	$\forall$	0	$+$
6	`IrreflexiveProperty(R)`	$\forall$	0	$-$
7	`SubClassOf(ObjectAllValuesFrom(R D) C)`	$\forall\forall$	$\pm$	$+$
8	`SubClassOf(C ObjectAllValuesFrom(R D))`	$\forall\forall$	$\pm$	$-$
9	`ObjectPropertyDomain(R C)`	$\forall\forall$	$+$	$-$
	`ObjectPropertyDomain(R C)`	$\forall\forall$	$+$	$-$
10	`SubObjectProperty(R S)`	$\forall\forall$	0	$\pm$
	`InverseObjectProperty(R S)`	$\forall\forall$	0	$\pm$
	`SymmetricProperty(R)`	$\forall\forall$	0	$\pm$
11	`DisjointObjectPropertyOf(R S)`	$\forall\forall$	0	$-$
	`AsymmetricProperty(R)`	$\forall\forall$	0	$-$
12	`TransitiveProperty(R)`	$\forall\forall\forall$	0	$\pm$
	`FunctionalProperty(R)`	$\forall\forall\forall$	0	$\pm$
	`InverseFunctionalProperty(R S)`	$\forall\forall\forall$	0	$\pm$
13	`SubClassOf(C ObjectSomeValuesFrom(R D))`	$\forall\exists$	$\pm$	$+$
14	`SubClassOf(ObjectSomeValuesFrom(R D) C)`	$\forall\exists$	$\pm$	$-$

Table 3.3: Grouped filter metrics for OWL FF-PCNF constructs from Table 3.2. In total there are 14 groups expressing 20 different OWL constructs. In the prenex column, the quantifiers $\forall$ and $\exists$ denote the structure of the prenex. The number of quantifiers indicates the number of quantified variables. In the unary and binary columns, 0 indicates the absence of that predicate whereas $+$ indicates the presence of a positive, $-$ presence of a negated predicate, and $\pm$ the presence of both.

3.4 Pattern application

Once a FF-PCNF sentence has been filtered and its filter group identified it's time to test if it's possible to generate OWL approximations. Testing whether or not a sentence will yield an OWL approximation most often involves counting predicates, locating variables within those predicates, and then correlating those locations across predicates. In sentences with a large number of predicates this can be computationally expensive, which is why filtering out the number of applicable OWL constructs is useful. Not every OWL pattern within a filter group will yield a successful approximation, in some cases it is possible for none of the OWL patterns to produce an approximation.

Continuing the example of the OWL construct `ObjectPropertyDomain` from Figure 3.7, we see that there are two applicable OWL constructs in group 9. From Table 3.3 we see that `ObjectPropertyDomain` ($\forall x, y[\neg R(x, y) \vee C(x)]$) and `ObjectPropertyRange` ($\forall x, y[\neg R(x, y) \vee C(y)]$) have very similiar FF-PCNF sentences. Testing for a match to either of those constructs involves checking whether the variable in the domain or range of the binary predicate, R, is being used in the unary predicate C. In our case, we can see that the variable used in C is in the domain of R, thus this FF-PCNF sentence yields an OWL `ObjectPropertyDomain`.

Another complication that arises while testing if a sentence can yield an approximation is the variance in the number of predicates within a FF-PCNF sentence. The filter metrics used in Table 3.3 only account for the presence of unary and binary predicates, not the total count of each. If we are trying to generate OWL approximations for a FF-PCNF sentence such as $\forall x, y[\neg R(x, y) \vee C_1(x) \vee C_2(x) \vee C_3(x)]$ and strictly interpret the OWL `ObjectPropertyDomain` construct ($\forall x, y[\neg R(x, y) \vee C(x)]$) then we will not yield an OWL approximation. This is inefficient because within the original FF-PCNF there does exist an approximable construct, though it requires the introduction of the OWL class expression operator `"ObjectUnionOf"` to be appropriately expressed. In this case the correct approximation can be seen in Figure 3.8.

Binary predicate inverses are another complication that require special attention when trying to match OWL constructs to FF-PCNF sentences. Comparing the FF-PCNF sentence $\forall x, y[\neg R(x, y) \vee S(y, x)]$ against Table 3.2 shows it is similar to the OWL constructs `SubObjectProperty` and `InverseObjectProperty`. In this case, it is approximated by `SubObjectProperty(R InverseOf(S))`

```xml
<ObjectPropertyDomain>
    <ObjectProperty IRI="R"/>
    <ObjectUnionOf>
        <Class IRI="C1"/>
        <Class IRI="C2"/>
        <Class IRI="C3"/>
    </ObjectUnionOf>
</ObjectPropertyDomain>
```

Figure 3.8: Example of how the FOL sentence $\forall x, y[\neg R(x, y) \vee C_1(x) \vee C_2(x) \vee C_3(x)]$ can be expressed using the OWL `ObjectionUnionOf` construct.

```xml
<SubObjectProperty>
    <ObjectProperty IRI="R"/>
    <ObjectInverseOf>
        <ObjectProperty IRI="S"/>
    </ObjectInverseOf>
</SubObjectProperty>
```

Figure 3.9: Example showing the approximation of the FF-PCNF sentence $\forall x, y[\neg R(x, y) \vee S(y, x)]$ using the OWL `InverseOf` construct.

as shown in Figure 3.9. Unlike the `ObjectPropertyDomain` and `ObjectPropertyRange` constructs, where the placement of variables determines which construct will be approximated; other constructs handle the different placement of variables across predicates by introducing the `ObjectInverseOf` construct. Overall this adds to the complexity needed to correctly approximate a FOL sentence and reinforces the need to minimize the number of attempted approximations.

Matching the OWL constructs from the filter groups to sentences requires a more general approach than strict matching against the FF-PCNF sentences given in Table 3.2. The approach taken in this thesis is to generalize the structure of the FF-PCNF sentences given in Table 3.2 based on their structure. After inspecting each OWL FF-PCNF sentence we define a pattern such that each OWL construct has a more flexible definition to implement against. A full listing of the Pattern descriptions can be found in the Appendix.

CHAPTER 4

IMPLEMENTATION

Our approach was implemented as a refactor and extension to the Macleod ontology framework.[1] It features an object oriented design (OOD) that provides a semantically intuitive application programming interface (API) to manipulate FOL sentences. Python was chosen as the implementation language because of its community support, rich third-party libraries, and because Macleod is written in Python already. All implementation details can be found online in the GitHub repository [HS18] under the GPL v3+ license.

4.1 Macleod

Macleod consists of a set of scripts designed to support key reasoning tasks frequently encountered during ontology design and verification. Currently it focuses on automating tasks that can be accomplished independent of the semantics of concepts and relations. These tasks are consistency checking of ontologies and their modules as well as checking whether competency questions, provided as "lemmas", are entailed through the use of various theorem provers and model finders. Our extension to the Macleod framework brings a new parsing methodology and an object-oriented (OO) representation of FOL ontologies [HS18].

Originally Macleod leveraged an internal string representations of the Common Logic Interchange Format (CLIF) [ISO18] FOL syntax. The string representations were primarily used to convert CLIF ontologies to different formats accepted by automated theorem provers and model finders like Vampire and Paradox [CS03; RV02]. In addition, the string representations were extremely slow to manipulate once ontologies got larger or data was added. For Macleod's original use-case, the string representations were sufficient for the limited programmatic interaction that was originally needed. However, our approach requires the conversion to FF-PCNF and the inspection of FOL sentences to test for matches against OWL constructs and thus needed a more robust API to manipulate and inspect FOL sentences. To achieve this we designed, implemented, and integrated an object oriented (OO) representation of FOL ontologies into Macleod.

[1] https://github.com/thahmann/macleod

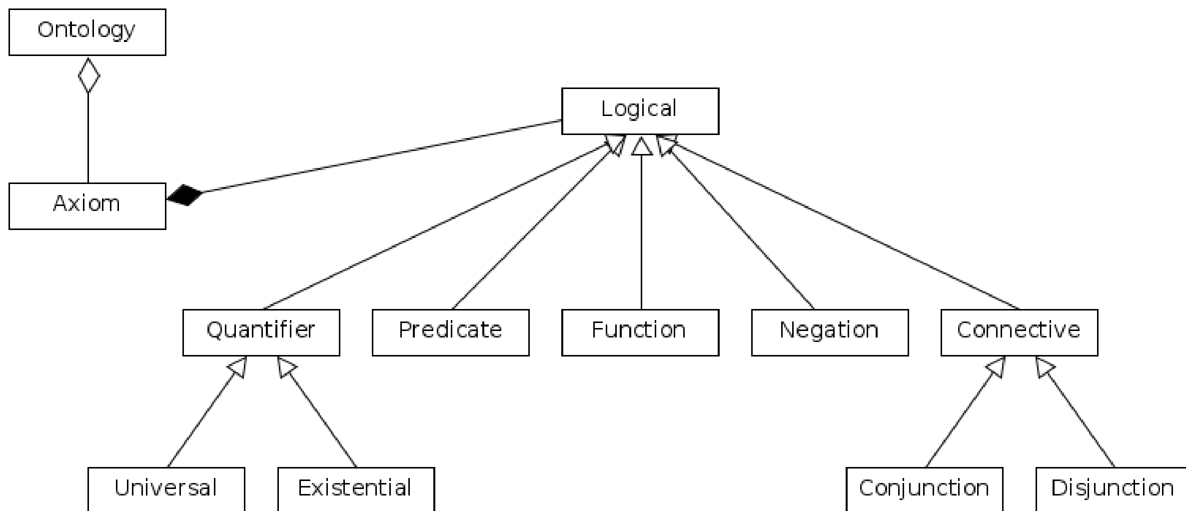

Figure 4.1: Lightweight UML diagram of the `macleod.logical` Python module.

4.2 Object-Oriented Representation

The OO implementation provides a semantically intuitive interface to programmatically operate on parsed FOL ontologies. In our design an `Ontology`, which is composed of many `Axioms`, represents the top-most object in our hierarchy. An `Axiom` possesses a single `Logical` which represents a basic FOL expression and itself contains a list of terms that are of type `Logical`. A `Logical` is a abstract base class which is extended by the following derived classes: `Connective`, `Quantifier`, `Predicate`, `Function`, and `Negation`. `Connectives` are further extended by `Disjunctions` and `Conjunctions`. `Quantifiers` are extended by `Universals` and `Existentials`. A complete class hierarchy can be found in Figure 4.1. By structuring the FOL representation in this way we provide a rich API to interact with parsed FOL ontologies.

Our OO structure was based on the natural tree-like structure that FOL sentences exhibit when specified in prefix form (e.g. `(and P1 P2 P3)` rather than `(P1 and P2 and P3)`). Using a tree-like structure made implementing several of the algorithms straight-forward as they could use the existing structure of well known tree traverals like depth-first and breadth-first search. This also made maintaining one of Macleod's original functions, conversion to automated theorem prover specific formats, easier because it could be succinctly expressed in a single recursive traversal [2].

[2]https://github.com/thahmann/macleod/blob/DL/src/macleod/logical/axiom.py

```
f = Function('f', ['x'])
a = Predicate('A', [f, 'x', 'y'])
b = Predicate('B', ['x', 'y'])
disjunction = a | b
Existential(['x','y'], disjunction)
```

Figure 4.2: Construction of FOL sentence $\exists x, y[A(f(x), x, y \vee B(x, y))]$ in Python using the OO API.

```
Universal(['x','y'], (a | b) & (a | (b & ~b)))
```

Figure 4.3: Python snippet which shows the construction of a FOL sentence.

We implemented several logical manipulations required by our approach using the OO representation. With the OO API it is relatively simple, and more concise, to abstract away the formation and logical operations on FOL expressions. For example, the FOL sentence $\exists xy[A(f(x), x, y) \vee B(x)]$ could be written using the Python snippet shown in Figure 4.2. While the more complicated expression $\forall xy[(A(f(x), x, y) \vee B(y)) \wedge (A(f(x), x, y) \vee (B(x) \wedge \neg B(x))]$ would similarly be expressed as shown in Figure 4.3.

Note that the OO implementation makes use of operator overloading to simplify the creation of FOL sentences. In Figure 4.2, `(a | b)` is shorthand for creating a disjunction by overloading the pipe (`|`) operator on the base `Logical` class. Operator overloading for `Negation` (~) and `Conjunction` (`&`) are also implemented to simplify the creation of more complex features such as the conversion to FF-PCNF, quantifier coalesence, and inspection of sentences to match an OWL pattern.

By providing an OO representation of common FOL structures and operations we were able to implement our approach in a way that would be accessible for future expansion. Other areas of Macleod, such as the conversion of CLIF ontologies to formats required by various other automated theorem provers and model finders also benefited from the new approach. With the OO format implemented, we then needed a method to parse existing CLIF ontologies to create the new OO structures.

4.3 Parsing Common Logic

The ontologies used in our research are written in the Common Logic Interchange Format (CLIF) defined by an ISO standard [ISO18]. The CLIF ontologies are publicly available as part of the COLORE project [Grü+10]. To leverage the COLORE repository an efficient method was needed to create the Python OO structure out of the CLIF ontologies. OO structures were created via a left-right (LR) parser written using the Python Lex-Yacc (`PLY`[3]) library against a subset of the CLIF standard. In particular, we focused on the portion of the CLIF grammar dealing with what the standard calls “sentences”. Our implemented parser does not deal with the more flexibile CLIF constructs such as restricted imports, axiom schemas, or domain restrictions.

Tokenization with the Python `PLY` library works based on regular expressions. As streams of text are read into the parser a series of regular expressions are applied, and if matched, a token is produced. The full list of tokenization rules is found in Figure 4.4.

⟨*not*⟩ ::= `not`

⟨*and*⟩ ::= `and`

⟨*or*⟩ ::= or

⟨*exists*⟩ ::= exists

⟨*forall*⟩ ::= forall

⟨*iff*⟩ ::= iff

⟨*if*⟩ ::= if

⟨*clcomment*⟩ ::= cl-comment

⟨*start*⟩ ::= cl-text

⟨*import*⟩ ::= cl-imports

⟨*lparen*⟩ ::= (

⟨*rparen*⟩ ::=)

⟨*newline*⟩ ::= `\n`

⟨*nonlogical*⟩ ::= `[<>=\w\-=]+`

⟨*string*⟩ ::= `['\"](.+)['\"]`

Figure 4.4: List of tokenization rules applied to character streams of CLIF ontologies. A tokenization is applied if the supplied regular expression matches a portion of the character stream.

[3] https://www.dabeaz.com/ply/ply.html

Likewise our BNF grammar is easily represented in Python via the PLY library. Given a stream of tokens obtained as the output of applying the above tokenization rules PLY will attempt to match rules from the BNF grammar and then execute custom code. In our implementation when parsing rules are matched we construct the relevant Python object from our OO design. A snippet of the PLY implementation is found in Figure 4.5. A complete list of grammar rules are found in Figure 4.6.

```
def p_conditional(p):
    """
    conditional : LPAREN IF axiom axiom RPAREN
    """

    p[0] = Disjunction([Negation(p[3]), p[4]])

def p_existential(p):
    """
    existential : LPAREN EXISTS LPAREN nonlogicals RPAREN axiom RPAREN
    """

    p[0] = Existential(p[4], p[6])

def p_universal(p):
    """
    universal : LPAREN FORALL LPAREN nonlogicals RPAREN axiom RPAREN
    """

    p[0] = Universal(p[4], p[6])

def p_predicate(p):
    """
    predicate : LPAREN NONLOGICAL parameter RPAREN
    """

    p[0] = Predicate(p[2], p[3])
```

Figure 4.5: PLY implementation of the abbreviated CLIF BNF grammar. When a parsing rule has been applied to executes the code contained in the Python function.

It is important to note that the application of the parsing rules in Figure 4.5 are recursive in their production. `Universal(p[4], p[6])` will supply in `p[6]` whatever Python object is the result of further application of parsing rules. In this way, the end result is a fully constructed OO

representation of a parsed CLIF file that is ready to be manipulated. With this object it becomes possible to automate analysis and manipulation of CLIF ontologies that were otherwise inaccessible.

⟨*clif*⟩ ::= ⟨*comment*⟩ ⟨*ontology*⟩ | ⟨*ontology*⟩

⟨*ontology*⟩ ::= `(` cl-text ⟨*uri*⟩ ⟨*statement*⟩ `)` | ⟨*statement*⟩

⟨*statement*⟩ ::= ⟨*axiom*⟩ ⟨*statement*⟩
| ⟨*import*⟩ ⟨*statement*⟩
| ⟨*comment*⟩ ⟨*statement*⟩
| ⟨*axiom*⟩
| ⟨*import*⟩
| ⟨*comment*⟩

⟨*comment*⟩ ::= `(` cl-comment ⟨*string*⟩ `)`

⟨*import*⟩ ::= `(` cl-imports ⟨*uri*⟩ `)`

⟨*axiom*⟩ ::= ⟨*negation*⟩
| ⟨*universal*⟩
| ⟨*existential*⟩
| ⟨*conjunction*⟩
| ⟨*disjunction*⟩
| ⟨*conditional*⟩
| ⟨*biconditional*⟩
| ⟨*predicate*⟩

⟨*negation*⟩ ::= `(` not ⟨*axiom*⟩ `)`

⟨*conjunction*⟩ ::= `(` and ⟨*axiomList*⟩ `)`

⟨*disjunction*⟩ ::= `(` or ⟨*axiomList*⟩ `)`

⟨*axiomList*⟩ ::= ⟨*axiom*⟩ ⟨*axiom*⟩ | ⟨*axiom*⟩ ⟨*axiom*⟩ ⟨*axioms*⟩

⟨*axioms*⟩ ::= ⟨*axiom*⟩ ⟨*axioms*⟩ | ⟨*axiom*⟩

⟨*conditional*⟩ ::= `(` if ⟨*axiom*⟩ ⟨*axiom*⟩ `)`

⟨*biconditional*⟩ ::= `(` iff ⟨*axiom*⟩ ⟨*axiom*⟩ `)`

⟨*existential*⟩ ::= `(` exists `(` ⟨*nonlogicals*⟩ `)` ⟨*axiom*⟩ `)`

⟨*universal*⟩ ::= `(` forall `(` ⟨*nonlogicals*⟩ `)` ⟨*axiom*⟩ `)`

⟨*predicate*⟩ ::= `(` ⟨*nonlogical*⟩ ⟨*parameter*⟩ `)`

⟨*parameter*⟩ ::= ⟨*function*⟩ ⟨*parameter*⟩
| ⟨*nonlogicals*⟩ parameter
| ⟨*function*⟩
| ⟨*nonlogicals*⟩

⟨*function*⟩ ::= `(` ⟨*nonlogical*⟩ ⟨*parameter*⟩ `)`

Figure 4.6: BNF grammar describing a subset of the CLIF syntax. The URI production rule is omitted for brevity. The NONLOGICAL and STRING production rules are tokenization regular expressions used to denote the respective tokens.

4.4 Conversion to FF-PCNF

Once a FOL ontology has been parsed and the OO representation has been created the next step is converting each sentence to FF-PCNF. Leveraging the tree-like structure of the OO representation the 6 steps in the conversion to FF-PCNF are implemented as methods on the different Python classes to simplify the code. The entrypoint to the FF-PCNF conversion on a FOL sentence is implemented on the Python `Axiom` class and is shown in Figure 4.7.

```
def ff_pcnf(self):
  """
  Apply logical operations to translate the axiom into a function free
  prenex conjunctive normal form.
  """

  # Don't modify structure in place!
  copied = copy.deepcopy(self)

  function_free, declaration = copied.substitute_functions()
  unique_variables = function_free.standardize_variables()
  distributed_negation = unique_variables.push_negation()
  prenex_form = distributed_negation.create_prenex()
  cnf = prenex_form.distribute_disjunctions()

  #Add additional function declarations if they exist
  if declaration is not None:
    cnf.extra_sentences = declaration

  return cnf
```

Figure 4.7: Code snippet showing the FF-PCNF conversion in the Python `Axiom` class. Note that the method starts out by creating a deep copy. Operations that change the structure of an `Axiom` prefer to return a new copy of the object rather than modify it in place.

4.5 Filtering FF-PCNF Sentences

Filtering of FF-PCNF sentences to place them into a pattern group is implemented using Python's efficient set intersection operation. At the root of the filter "tree", the set of applicable patterns includes each pattern listed in Table 3.3. From there, each comparison is associated with it's own set of patterns (e.g. the set of patterns which have a single quantifier) and if the comparison evaluates to true then an intersection is done between the currently applicable patterns and those from the comparison. With each comparison the set of currently applicable patterns is reduced. At the end of the four filtering comparisons the remainder are the applicable patterns ready for extraction as covered in Section 4.6.

4.6 Pattern Extraction

After a FOL sentence has been converted to FF-PCNF and filtered to produce a list of applicable patterns we now need to attempt to apply those patterns to produce approximations. By using the pattern definitions given in the Annex, we implemented a set of specialized functions which examine a FOL sentence to dicern if it contains an approximable axiom. Of the whole approximation process, this is the most tedious and expensive operation. Each applicable pattern requires additional checks to be conducted to see if a pattern can successfully be applied to extract an OWL approximation. In most cases this involves checking the placement of variables within binary predicates and ensuring their ordering matches a pattern. Figure 4.8 shows a snippet of the function which applies the transitive property pattern to a FOL sentence.

```
# Ensure at least two of them are negated
negated = [p for p in axiom.negated()]
if len(negated) != 2:
    return None

# Ensure the three binary predicates are all the same name
name = properties[0].name
if not all([x.name == name for x in properties]):
    return None

# Compare placement of the variables for transitive chain
x_one, y_one = negated[0].variables
x_two, y_two = negated[1].variables

if y_one == x_two:
    x = x_one
    z = y_two
elif x_two == y_one:
    x = x_two
    z = y_one
else:
    return None

# Ensure transitive predicate matches first and last vars
positive_x, positive_z = axiom.positive()[0].variables
if positive_x != x or positive_z != z:
    return None

return ('transitive', [axiom.positive()[0]])
```

Figure 4.8: Code snippet of the Macleod extension code that applies the transitive relation pattern to a parsed FOL sentence. If the pattern is successfully applied the function returns a tuple indicating that a particular binary predicate should be marked as transitive in OWL.

CHAPTER 5

EVALUATION AND DISCUSSION

We applied our approximation method to nine ontologies from the COLORE repository. The resulting approximated OWL ontologies were then inspected using the Protege ontology editor. Efficacy of the approximation was evaluated qualitatively by manual inspection of how well the approximated ontology matched the source ontology and included details such as ability to capture class or property hierarchies. Additional metrics for each ontology were gathered using Protege's inspection utilities to aid with the comparison. Metrics collected included the number of axioms and types of axioms that were present in the approximated ontology.

5.1 Materials: Ontologies Used for Evaluation

Early on it was recognized that specific ontologies would be better suited to approximation than others. In particular, FOL ontologies that model highly complex mathematical constructs which often leverage predicates with an arity greater than two were expected to approximate poorly. This was expected because of the limitation of OWL which can only directly express unary and binary predicates. In order to observe the overall efficacy of our approach a subset of the COLORE ontology repository was examined which tended towards using binary predicates and taxonomic hierarchies. The selected ontologies are listed in Table 5.1 with the ontology name and the name of the top-level file of the ontology. Basic metrics such as number of FOL sentences and total number of unary, binary, and n-ary predicates are listed in Table 5.2.

Ontology	Filename	Description
dolce_taxonomy	dolce_taxonomy	Upper ontology
inch mapsee multidim ped multidim physcont multidim voids	inch_calculus_extended_full mapsee multidim_space_ped fullphycont_full void_def	Various spatial ontologies, multidim specifically combines spatial representations to `DOLCE`'s taxonomy.
owltime sumo sumo	owltime sumo_temporalPart sumo_time	Temporal ontologies, where `SUMO` is another upper ontology like `DOLCE`.

Table 5.1: Ontologies selected for analysis after approximation. All ontologies can be found in the COLORE repository on Github.

Ontology	FOL sentences	# Unary predicates	# Binary predicates	# N-Ary predicates	Constants
dolce_taxonomy	48	37	0	0	0
inch	25	1	3	0	0
mapsee	39	13	12	0	0
multidim ped	14	7	1	0	0
multidim physcont	382	25	68	5	1
multidim voids	318	25	37	5	1
owltime	37	5	10	1	0
sumo_temporalPart	43	3	11	2	1
sumo_time	72	4	13	4	3

Table 5.2: Basic ontology metrics including the number of FOL sentences and number of unary, binary, and n-ary (3+) predicates found. Metrics for each ontology were obtained using the Macleod framework.

5.2 Approximation Example

The selected ontologies were approximated using our approach implemented as an extension to Macleod. Figure 5.1 shows an excerpt from the approximated multidim_space_voids ontology. The excerpt shows two example axioms in CLIF syntax comprised of only unary and binary predicates. Sentence V-A1, as called out by the comment in the snippet, features a function `r` of arity 1. In Figure 5.2 the logging output of the Macleod extension is shown.[1] From the two sentences a total of

[1]For full approximations see colore/link/here

```
(cl-comment 'V-D: void')
(forall (x)
    (iff
        (V x)
        (or
            (SimpleV x)
            (ComplexV x)
        )
    )
)
(cl-comment 'V-A1:')
(forall (x y)
    (if
        (hostsv x y)
        (and
            (hosts x y)
            (VS x (r y))
            (StrongC (r x) (r y))
        )
    )
)
```

Figure 5.1: Snippet of the multidim_space_voids ontology from the COLORE project. Note that sentence V-A1 contains a function “r” of arity 1 that needs to be substituted during the approximation. Both sentences V-D and V-A1 use only unary or binary predicates and are relatively short as written.

11 OWL axioms are successfully extracted and include the following types: `Class`, `ObjectProperty`, `SubClassOf`, `SubObjectPropertyOf`, and `FunctionObjectProperty`.

In the second sentence (`V-A1`) of figure 5.2 a clause $\forall a, b, c[(\neg r(a, b) \vee \neg r(a, c) \vee = (c, b))]$ appears that is absent from the original sentence. This additional clause is injected as part of the transformation to FF-PCNF where the function `r` is substituted with a binary predicate `r` and a new sentence declaring `r` to be functional is added. This transformation is done for all present functions however, only functions of arity 1 result in a usable approximation because of OWL's restriction to unary and binary relations.

```
1  Axiom: (x)[((~V(x) | SimpleV(x) | ComplexV(x))
2   & (~(SimpleV(x) | ComplexV(x)) | V(x)))]
3
4  FF-PCNF: (z)[((V(z) | ~SimpleV(z))
5   & (V(z) | ~ComplexV(z))
6   & (~V(z) | SimpleV(z) | ComplexV(z)))]
7
8   + yielded: (z)[(V(z) | ~SimpleV(z))]
9       - pattern subclass
10  + yielded: (z)[(V(z) | ~ComplexV(z))]
11      - pattern subclass
12  + yielded: (z)[(~V(z) | SimpleV(z) | ComplexV(z))]
13      - pattern subclass
14
15 Axiom: (x,y)[(~hostsv(x,y) | (hosts(x,y)
16  & VS(x,r(y)) & StrongC(r(x),r(y))))]
17
18 FF-PCNF: (z,y,w,v)[((~hostsv(z,y) | VS(z,w) | ~r(y,w))
19  & (~hostsv(z,y) | StrongC(w,v) | ~r(z,w) | ~r(y,v))
20  & (~hostsv(z,y) | hosts(z,y)))]
21
22  + yielded: (z,w,y)[(~hostsv(z,y) | VS(z,w) | ~r(y,w))]
23  + yielded: (z,v,w,y)[(~hostsv(z,y) | StrongC(w,v)
24                        | ~r(z,w) | ~r(y,v))]
25  + yielded: (z,y)[(~hostsv(z,y) | hosts(z,y))]
26      - pattern subproperty
27  + yielded: (a,b,c)[(~r(a,b) | ~r(a,c) | =(c,b))]
28      - (extra) pattern functional
```

Figure 5.2: Excerpt of output produced by running our automated approximation approach on the multidim_space_voids ontology from COLORE. The sentences are first printed in infix notation after being parsed then the FF-PCNF is shown. Applicable patterns are displayed along which top-level clause they are found in.

5.3 Performance Evaluation

We attempted to approximate 1984 ontologies from the COLORE project to benchmark our implementation. Of the 1984 ontologies, 986 were converted to FF-PCNF and approximated in OWL. Runtime metrics were collected on a Fedora 32 workstation equipped with an AMD Ryzen 7 3700x processor (3.6 GHz) and 32 gigabytes of RAM. Although the workstation had multiple cores available, metrics were collected using a single core for consistency. Memory usage of the implementation was negligible beyond what is normally required by the Python interpretor and was not recorded.

For each FOL ontology the average time to convert to FF-PCNF was 0.117 seconds. The standard deviation after removal of a single outlier was 0.07 seconds. The single outlier was periods_over_rationals ontology, where the conversion to FF-PCNF took 109 seconds. This appears to be caused by the deep nesting and structure of the sentences in the ontology which impressively demonstrates the $O(N^2)$ complexity of the FF-PCNF distribution step.

Conversion from a FOL ontology to OWL (including the conversion to FF-PCNF) took on average 0.21 seconds with a standard deviation of 0.3 seconds. The outlier periods_over_rationals took a total time of 114 seconds which showed the additional OWL approximation had a negligible effect on the runtime and was not affected by the $O(N^2)$ complexity of the FF-PCNF conversion. With the exception of the periods_over_rationals ontology the approach scaled well to handle the range of different ontologies within COLORE. In ontology engineering as a whole, that the worst case conversion took 114 seconds is not much of an issue. Approximation does not happen frequently, and it is not accompanied by any type of time constraint.

The 998 ontologies which did not approximate successfully were caused by a myriad of reasons. The most prominent reason was that the underlying ontology used an unsupported syntax of our OWL parser. Following behind unsupported syntax it was found that the ontologies had unsupported character sets (e.g. Unicode characters) embedded in the ontology. Finally, there were several cases where ontologies were found to have simple syntax errors such as missing parenthesis or malformed sentences.

5.4 Efficacy Evaluation

Metrics to evaluate approximated FOL ontologies included the number and kind of OWL constructs, DL complexity rating, and logical correctness. After approximation and correction of errors found in the FOL ontologies (see Section 5.5 on syntax checking), every approximation was found consistent after a logical evaluation using Protege. Unsurprisingly, `Class` and `ObjectProperty` declarations were the most commonly extracted OWL construct followed by `SubClassOf` and `SubObjectPropertyOf`. The number and kind of extracted OWL axioms are summarized in Tables 5.3 and 5.4.

All of the ontologies were able to produce at least an approximation consisting of classes, properties, and either class hierarchies or object property hierarchies. The most commonly recognized patterns were `SubClassOf`, `SubObjectPropertyOf`, `ObjectPropertyDomain`, and `ObjectPropertyRange`. Ontologies containing more properties, such as `FullPhysCont`, approximated more specialized object properties such as `DisjointObjectProperties` and `FunctionalObjectProperties`. Only the `SUMO Time` approximation successfully captured `ClassAssertion` and `Individuals`. This is expected because of the ontologies selected only the `SUMO` Time ontology actually had instance data which is uncommon for FOL ontologies.

Ontology	**Dolce Taxonomy**	**OwlTime**	**SUMO TemporalPart**	**SUMO Time**
Axioms	110	36	35	53
Classes	37	5	3	4
ClassAssertion	0	0	0	2
Individuals	0	0	0	2
SubClassOf	48	4	0	3
EquivalentClasses	0	0	0	0
DisjointClasses	25	1	0	1
Properties	0	10	11	13
ObjectPropertyDomain	0	6	7	10
ObjectPropertyRange	0	6	7	10
SubObjectPropertyOf	0	2	3	4
EquivalentObjectProperties	0	0	0	0
InverseObjectProperties	0	0	0	0
DisjointObjectProperties	0	1	0	0
FunctionalObjectProperty	0	0	2	2
InverseFunctionalObjectProperty	0	0	0	0
TransitiveObjectProperty	0	0	0	0
SymmetricObjectProperty	0	0	1	1
AsymmetricObjectProperty	0	1	1	1
ReflexiveObjectProperty	0	0	0	0
IrreflexiveObjectProperty	0	0	0	0

Table 5.3: Results of approximation for Dolce, OwlTime, Sumo Temporal, and Sumo Time ontologies.

Ontology	PED	Voids Def	FullPhysCont	Inch Calculus	MapSee
Axioms	23	163	249	10	77
Classes	7	25	25	1	14
ClassAssertion	0	0	0	0	0
Individuals	0	0	0	0	0
SubClassOf	8	23	23	0	23
EquivalentClasses	0	0	0	0	0
DisjointClasses	4	9	9	0	6
Properties	1	37	68	5	12
ObjectPropertyDomain	1	9	19	0	12
ObjectPropertyRange	1	17	25	1	11
SubObjectPropertyOf	0	27	57	2	0
EquivalentObjectProperties	0	0	0	0	0
InverseObjectProperties	0	0	0	0	0
DisjointObjectProperties	0	5	10	0	0
FunctionalObjectProperty	0	6	7	0	0
InverseFunctionalObjectProperty	0	1	1	0	0
TransitiveObjectProperty	0	0	0	0	0
SymmetricObjectProperty	0	3	4	1	0
AsymmetricObjectProperty	1	1	1	0	0
ReflexiveObjectProperty	0	0	0	0	0
IrreflexiveObjectProperty	0	0	0	0	0

Table 5.4: Results of approximation for PED, Voids Def, FullPhysCont, Inch Calculus, and MapSee ontologies.

5.5 Additional Benefits of OWL Approximation

Syntax Checking While implementing our approach we noted several latent problems in various FOL ontologies in the COLORE repository. In some cases the problems found were caused by incorrect CLIF syntax in the original ontologies. In part, these errors may stem from a lack of available tooling when developing using CLIF. With our contribution of a CLIF syntactically aware parser, we were able to identify and fix these problems when they occurred.

Where previous iterations of Macleod utilized raw string manipulation to translate to model and prover friendly formats, our implemented approach was CLIF syntax aware. Effectively when attempting to parse a CLIF specified ontology we were able to produce useful error output when an ontology failed to parse with a syntax error. In most cases it was trivial to find and fix the errors because the parser reports the line on which the error was found and what about the line it thinks is causing the error (e.g. unmatched parenthesis while parsing). This was quite useful in increasing the general usability of the ontologies in the COLORE repository.

Availability of OWL Tools Protege is a development environment for working with OWL ontologies. It features a streamlined interface and rich functionality to identify and expand various parts of an ontology. FOL ontology development does not have an equivalent tool and misses out on many of the benefits offered by Protege. One such benefit, which we overlooked, was the simple visualization that Protege offers when inspecting classes and their relations.

After approximating a FOL ontology in OWL it was trivial to inspect the class and property hierarchies using Protege. It turned out that some existing FOL ontologies had unintended inferences that resulted in incorrect axiomatizations. With the robustness of the OWL provers Fact++ and Pellet, coupled with Proteges' GUI, these types of errors were easily identified. Identifying these issues directly from the CLIF source in some cases was non-trivial because they were the result of axioms being combined across multiple CLIF sources.

Further still, the inference engines of the Fact++ and Pellet OWL provers provide a breakdown for each inferred axiom they create. This can, and was, used in several instances to trace back an implicit assertion made in the approximated OWL ontology to the originating explicit axiomatiza-

tions in FOL. This feedback loop for ontology verification was a much welcomed addition to the scope of benefits introduced by this research.

CHAPTER 6

SUMMARY

There exists a set of densely axiomatized FOL ontologies that are the result of countless hours of ontology development and verification, such as those in the COLORE project, which are inaccessible to knowledge engineers unfamiliar with FOL. Further, there is currently a dearth of stable free and open source or commercial off the shelf products which enable development, extension, or adoption of FOL ontologies. We widen the accessibility of those FOL ontologies and make them more usable with existing tools by a new approach to approximate FOL ontologies using OWL. The lightweight OWL approximations of the FOL ontologies can be inspected, extended, and used as the foundation for future work with the available OWL tooling which includes provers, model finders, and development environments. This helps to verify, evolve, and reuse the FOL ontology and avoid redundant ontology engineering efforts or maintaining copies of the ontologies in two languages with different expressivity (FOL and OWL).

By using FF-PCNF to constrain the free-form structure of FOL sentences, recognizable patterns were identified to be used in the approximation. With an efficient search structure the patterns were applied to FOL sentences which yielded OWL approximations. The approximations were then aggregated into a functional OWL approximation of the original FOL ontology. When tested our approach on a set of 9 suitable ontologies from the COLORE repository, the approximations captured class and property hierarchies where present in FOL ontologies as well as additional OWL axioms.

Our implementation was applied to the COLORE set of ontologies to benchmark the practical usability and scalability of the approach. On average it was possible to create an OWL approximation of a FOl ontology in under a second. With exception of a single outlier it was possible to create an approximations of 1,332 ontologies in a total time of under 5 seconds per ontology using a single core of a modern CPU and a negligible amount of memory.

Future Work Having seen the promising outcomes of approximating FOL ontologies in OWL additional work can be undertaken to leverage this in order to provide even greater benefit to knowledge engineers. During the debugging efforts, it became clear that it was possible to use

the Pellet and Fact++ OWL provers backfeed implicit axiomatizations to a source ontology. This effort would aim to make explicit any axiomatizations derived automatically and translate those axioms back to FOL. Doing so would reduce future runtime of the provers and provide knowledge engineers another method to inspect the corpus of knowledge contained in their ontology.

Currently there are limited development environments available to work with FOL ontologies in CLIF or other formats. With the new OO implementation and the LR parser Macleod as a framework could also be further extended. For example, leveraging the OO libraries and LR parser a visualization and editing environment could be incorporated to benefit ontology engineers.

An expanded evaluation could also be conducted on approximated OWL ontologies. In particular, a larger set of approximations could be compared against natively developed OWL ontologies using the metrics used in Section 5.4. Doing so would give further insight in how OWL ontologies can be developed and what, if anything, could be done to better approximate FOL ontologies using OWL. This extended evaluation may also identify common ontology types that are better suited to be developed natively in FOL vs. OWL.

Another type of evaluation of the approximated ontologies could be conducted by converting the approximated ontologies back to FOL. Once the approximated ontology has been returned to it's original format, it could be more readily compared against the original ontology. In particular, there is the possibility to apply native FOL competency questions as shown by [GF95] against the approximation to measure what may have been lost during approximation.

Additional effort could also be devoted to extending the range of approximated OWL constructs. This thesis targeted the most commonly used constructs and left some of the more complex constructs such as property chains and numeric support for future work. By applying more effort to these areas it may be possible to capture better approximations of FOL ontologies in OWL.

APPENDIX

Pattern A.1 (SubclassOf) *A universally quantified sentence with a single variable that consists of only unary predicates where at least one of them is negated. Negated unary predicates represent the subclass whereas the non-negated predicates represent the superclass. If there exist more than a single predicate in either category, then all members of that category must be joined together in a union in the generated approximation.*

Pattern A.2 (DisjointClass) *A universally quantified sentence with a single variable that consists of only two negated unary predicates.*

Pattern A.3 (SubObjectPropertyOf) *A universally quantified sentence with a two variables that consists of only binary predicates where at least one of them is negated. Negated binary predicates represent the subproperty whereas the non-negated predicates represent the superproperty. If there exist more than a single predicate in either category, then all members of that category must be joined together in a union in the generated approximation. Placement of variables in each predicate will determine if the `InverseOf` construct should be applied.*

Pattern A.4 (DisjointObjectPropertyOf) *A universally quantified sentence with a two variables that consists of only two negated binary unary predicates. Placement of variables in each predicate will determine if the `InverseOf` construct should be applied.*

Pattern A.5 (ObjectPropertyDomain) *A universally quantified sentence with two variables that consists of a single negated binary predicate and one or more unary predicates. Unary predicates that use the variable from the domain of the binary predicate corre- spond to this pattern. For variables covering the range of the binary predicate see the Pattern A.6. If there exists more than a single unary predicate then they must be joined in a union in the generated approximation.*

Pattern A.6 (ObjectPropertyRange) *A universally quantified sentence with two variables that consists of a single negated binary predicate and one or more unary predicates. Unary predicates that use the variable from the range of the binary predicate correspond to this pattern. For variables covering the domain of the binary predicate see Pattern A.5. If there exists more than a single unary predicate then they must be joined in a union in the generated approximation.*

Pattern A.7 (InverseObjectProperty) *This pattern is indicated by two instances of inverted Pattern A.3 occuring.*

Pattern A.8 (ReflexiveObjectProperty) *A universally quantified sentence with a single variable and a single positive binary predicate. The single universally quantified variable must appear in both the domain and range of the binary predicate.*

Pattern A.9 (IrreflexiveObjectProperty) *A universally quantified sentence with a single variable and a single negated binary predicate. The single universally quantified variable must appear in both the domain and range of the binary predicate.*

Pattern A.10 (SymmetricObjectProperty) *A universally quantified sentence with two variables and a single binary predicate that appears twice, both positive and negated. The variables between negated and positive predicates must reverse their positions.*

Pattern A.11 (AsymmetricObjectProperty) *Similiar to Pattern A.10 only both binary predicates must be negated.*

Pattern A.12 (TransitiveObjectProperty) *A universally quantified sentence with three variables in which a single predicate appears three times, negated twice and positive once. The two negated predicates must share the same variable for one of their domains and one of their ranges. The single positive binary predicate's domain must match the domain of one of the negated predicates and the range must match the range of one of the other negated predicate.*

Pattern A.13 (FunctionalObjectProperty) *A universally quantified sentence with three variables and three predicates in which a single predicate appears negated twice. The other predicate must be the equality relation* $=$*. The negated predicates must share the same variable for one their domain and must cover a different variable for their range. The equality relation's domain must match the range of one of the negated predicates and the equality relations range must match the range of one of the other negated predicate.*

Pattern A.14 (InverseFunctionalObjectProperty) *A universally quantified sentence with three variables and three predicates in which a single predicate appears negated twice. The other predicate*

must be the equality relation =. A negated predicate must share the same variable for its domain as the equality relation but each should have different ranges. The other negated predicate must have a domain that matches the other negated predicate or the equality relation. The other negated predicate must have a range that matches the other negated predicate or the equality relation.

Pattern A.15 (SubClassOf ObjectSomeValuesFrom) *A sentence* $\forall x \exists y$ *which contains two conjunctive terms of disjunctions. One term will consist of a conjunction over two different unary predicates where one of them is negated. The negated unary predicate will be covering the universally quantified variable and the positive predicate will cover the existentially quantified variable. The other term will have the same negated unary predicate covering the universal variable and a positive binary predicate. The binary predicate will have the universally quantified variable in it's domain and the existentially quantified variable in its range.*

Pattern A.16 (ObjectSomeValuesFrom SubClassOf) *A sentence* $\forall x \exists y$ *which contains three predicates. A positive unary predicate covering the existentially quantified variable. A negated unary predicate covering the universally quantified variable. A negated binary predicate who domain is the universally quantified variable and whose range is the existentially quantified variable.*

Pattern A.17 (SubClassOf ObjectAllValuesFrom) *A universally quantified sentence with two variables and three predicates. A negated unary predicate, a positive unary predicate, and a negated binary predicate. The domain of the binary predicate must be the same variable as the negated unary predicate. The range of the binary predicate must be the same variable as the positive unary predicate.*

Pattern A.18 (ObjectAllValuesFrom SubClassOf) *A universally quantified conjunction of two disjunctions with two variables having three predicates. In one disjunction are two different unary predicates of different variables where one predicate is negated. The other disjunction must have the same positive predicate over the same variable and a positive binary predicate. The domain of the binary predicate must match the positive unary predicate. The range of the binary predicate must match the variable of the negated unary predicate in the other disjunction.*

Pattern A.19 (SubClassOf ObjectHasSelf) *A universally quantified sentence with a single variable and both a positive binary predicate and negated unary predicate.*

Pattern A.20 (ObjectHasSelf SubclassOf) *A universally quantified sentence with a single variable and both a negated binary predicate and positive unary predicate.*

BIBLIOGRAPHY

[BHS04] Franz Baader, Ian Horrocks, and Ulrike Sattler. “Description Logics”. In: *Studies In Health Technology And Informatics* 101 (2004), pp. 137–41. arXiv: `1112.3142v1` (cit. on pp. 11, 14).

[BLHL01] Tim Berners-Lee, James Hendler, and Ora Lassila. “The semantic web”. In: *Scientific American* 284.5 (2001), pp. 34–43 (cit. on p. 1).

[Bor96] Alex Borgida. “On the relative expressiveness of description logics and predicate logics”. In: *Artificial Intelligence* 82.1-2 (1996), pp. 353–367 (cit. on pp. 4, 14).

[CJB] B Chandrasekaran, John R Josephson, and V Richard Benjamins. “What Are Ontologies , and Why Do We Need Them ?” In: () (cit. on pp. 5, 7).

[CS03] Koen Claessen and Niklas Sorensson. “New techniques that improve MACE-style finite model finding”. In: *CADE-19 Workshop: Model* (2003), pp. 11–27. URL: `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.58.653{\&}rep=rep1{\&}type=pdf` (cit. on p. 30).

[Cod82] E. F. Codd. “Relational database: a practical foundation for productivity”. In: *Communications of the ACM* 25.2 (1982), pp. 109–117 (cit. on p. 4).

[DMQ05] Dejing Dou, Drew McDermott, and Peishen Qi. “Ontology translation on the Semantic Web”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 3360 LNCS (2005), pp. 35–57 (cit. on pp. 2, 14).

[Fal+13] Ricardo A Falbo, Giancarlo Guizzardi, Aldo Gangemi, and Valentina Presutti. “Ontology Patterns: Clarifying Concepts and Terminology”. In: *Workshop on Ontology and Semantic Web Patterns (WOP 2013).* 2013 (cit. on p. 7).

[Fei+07] Lee Feigenbaum, Ivan Herman, Tonya Hongsermeier, Eric Neumann, and Susie Stephens. “The semantic web in action”. In: *Scientific American* (2007) (cit. on p. 1).

[GKV97] Erich Grädel, Phokion G. Kolaitis, and Moshe Y. Vardi. “On the decision problem for two-variable first-order logic”. In: *The Bulletin of Symbolic Logic* 3.1 (1997), pp. 53–69 (cit. on p. 8).

[GS04] Pierre Grenon and Barry Smith. “{SNAP} and {SPAN}: towards dynamic spatial ontology”. In: *J. Spat. Cogn. Comput.* 4.1 (2004), pp. 69–104 (cit. on p. 7).

[Gru93] Thomas R Gruber. “A translation approach to portable ontology specifications”. In: *Knowledge Acquisition* 5.2 (1993), pp. 199–220. URL: `http://www.sciencedirect.com/science/article/pii/S1042814383710083` (cit. on p. 4).

[GF95] Michael Grüninger and Mark S. Fox. “The Role of Competency Questions in Enterprise Engineering”. In: (1995), pp. 22–31 (cit. on p. 52).

[Grü+10] Michael Grüninger, Torsten Hahmann, Ali Hashemi, and Darren Ong. “Ontology Verification with Repositories.” In: *FOIS* (2010). URL: `http://www.spatial.maine.edu/{~}torsten/publications/MGruninger{_}FOIS-2010.pdf` (cit. on pp. 2, 7, 33).

[Grü+12] Michael Grüninger, Torsten Hahmann, Ali Hashemi, Darren Ong, and Atalay Ozgovde. “Modular first-order ontologies via repositories”. In: *Applied Ontology* 7.2 (2012), pp. 169–209 (cit. on p. 14).

[GS98] Nicola Guarino and Barry Smith. “{F}ormal ontology and information systems”. In: *FOIS*. IOS Press, 1998, pp. 3–15 (cit. on pp. 5, 7).

[Gui05] Giancarlo Guizzardi. *Ontological Foundations for Structural Conceptual Models*. January 2005. 2005. URL: `http://www.loa.istc.cnr.it/Guizzardi/SELMAS-CR.pdf` (cit. on p. 7).

[Hah14] Torsten Hahmann. “Ontology Repositories: A Treasure Trove for Content Ontology Design Patterns”. In: *Proc. of the Workshop on Modular Ontologies (WoMO 2014)*. 2014 (cit. on p. 7).

[HS18] Torsten Hahmann and Shirly Stephen. “Using a hydro-reference ontology to provide improved computer-interpretable semantics for the groundwater markup language (GWML2)”. In: *International Journal of Geographical Information Science* (2018) (cit. on p. 30).

[HSB16] Torsten Hahmann, Shirly Stephen, and Boyan Brodaric. “Semantically Refining the Groundwater Markup Language (GWML2) with the Help of a Reference Ontology”. In: (2016) (cit. on p. 7).

[Hen11] James Hendler. “Introduction to the Semantic Web Technologies”. In: *Handbook on Semantic Web Technologies*. 2011 (cit. on pp. 4, 14).

[Her+06] Heinrich Herre et al. “General Formal Ontology (GFO) - A Foundational Ontology Integrating Objects and Processes [Version 1.0]”. In: 8 (2006) (cit. on p. 7).

[Hin05] P. Hinman. *Fundamentals of Mathematical Logic.* A K Peters, 2005 (cit. on p. 10).

[Hit+09] Pascal Hitzler, Bijan Parsia, Peter F Patel-schneider, and Sebastian Rudolph. "OWL 2 Web Ontology Language Primer". In: *W3C Recommendation* December (2009), pp. 1–123. URL: `https://www.w3.org/TR/owl2-primer/` (cit. on pp. 11, 12).

[Hod01] Wilfrid Hodges. "Classical logic I: first-order logic". In: *The Blackwell Guide to Philosophical Logic. ...* (2001), pp. 9–32 (cit. on pp. 4, 18).

[Hoe15] R Hoekstra. *Hoekstra, R.. Ontology Representation. Amsterdam, NLD: IOS Press, 2009. ProQuest ebrary. Web. 27 January 2015. Copyright © 2009. IOS Press. All rights reserved.* January. IOS Press, 2015 (cit. on p. 4).

[ISO18] ISO. *Information technology Common Logic (CL): a framework for a family of logic-based languages.* 2018. URL: `https://www.iso.org/standard/66249.html` (cit. on pp. 30, 33).

[LB85] Hector J. Levesque and Ronald J Brachman. "A Fundamental Tradeoff in Knwoledge Representation and Reasoning (Revised Version)". In: *Readings in Knowledge Representation* (1985), pp. 41–70 (cit. on pp. 4, 5, 8).

[Mas+03] Claudio Masolo, Stefano Borgo, Aldo Gangemi, Nicola Guarino, and Alessandro Oltramari. "WonderWeb Deliverable D18". In: *Communities* 2003 (2003), p. 343. URL: `http://wonderweb.semanticweb.org/deliverables/documents/D18.pdf` (cit. on p. 7).

[Men97] Elliott Mendelson. *Introduction to Mathematical Logic.* 4th Edition. Springer, 1997, pp. 50–56,106 (cit. on pp. 10, 22).

[Men03] Christopher Menzel. "Reference Ontologies - Application Ontologies: Either/Or or Both/And?" In: *Proceedings of the KI2003 Workshop on Reference Ontologies and Application Ontologies* (2003), pp. 1–10 (cit. on pp. 5, 7, 8).

[Mos17] Till Mossakowski. "The distributed ontology, model and specification language DOL". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 10644 LNCS (2017), pp. 5–10 (cit. on pp. 14, 15).

[MK11] Till Mossakowski and Oliver Kutz. "The onto-logical translation graph". In: *Frontiers in Artificial Intelligence and Applications* 230 (2011), pp. 94–109 (cit. on pp. 14, 15).

[MML07] Till Mossakowski, Christian Maeder, and Klaus Lüttich. "The heterogeneous tool set". In: *CEUR Workshop Proceedings* 259 (2007), pp. 119–135 (cit. on p. 15).

[Mus15] Mark A. Musen. "The protégé project". In: *AI Matters* 1.4 (2015), pp. 4–12 (cit. on p. 15).

[PB10] Adrian Paschke and Harold Boley. "Rule markup languages and semantic web rule languages". In: *Rule Markup Languages and Semantic Web Rule Languages* (2010), pp. 1–24 (cit. on p. 2).

[RV02] Alexandre Riazanov and Andrei Voronkov. "The design and implementation of VAMPIRE". In: *AI Communications* 15.2 (2002), pp. 91–110 (cit. on p. 30).

[Rus+] Stuart J Russell et al. *Artificial Intelligence: A Modern Approach* (cit. on p. 11).

[SK] Kamruzzaman Sarker and Adila Krisnadhi. "Rule-based OWL Modeling with ROWLTab Proteg e Plugin". In: () (cit. on pp. 2, 15).

BIOGRAPHY OF THE AUTHOR

Robert Powell was born a local to Maine in 1993. While attending Mount View Highschool in Thorndike, Maine he also attended the Waldo County Technical Center where he took courses on computer and networking technology. He enrolled at the University of Maine in 2011 and went on to obtain his B.S. in Computer Science. From there he enrolled in the Masters program in Computer Science at the University of Maine.

Robert Wesley Powell II is a candidate for the Master of Science in Computer Science from the University of Maine in December 2020.