TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Norman Kuusik 206275IAIB

ONTOLOGY MERGING IN PROLOG USING THE PRINCIPLE OF SOFT UNIFICATION

Bachelor's Thesis

Supervisor: Jüri Vain
Professor

Tallinn 2024

Author's Declaration of Originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Norman Kuusik

25.05.2024

Abstract

The aim of this thesis is to develop an automatic tool for OWL ontology merging to ensure well-coordinated task completion in the context of collaborative robotics. Ontology is a type of formal knowledge representation that at the very least consists of classes properties and relations between class members. A common language to express ontologies is OWL.

The technique developed for this tool is based on string and semantic matching with additional consideration of structural heterogeneity of concepts. The project is mainly implemented in SWI-Prolog in order to combine string and semantic matching techniques with Prolog's inherent unification mechanism. We refer to this approach as soft unification.

To validate the approach presented in this thesis, experiments were run on OAEI data set with the matching accuracy of 60% across 42 tests. Additionally, we ran the tool on several ontologies from the domain of robotics producing a small, but generally accurate, set of matched concepts. These results clearly show a good capability of the tool to match semantically similar concepts. The results also highlight the challenges related to comparing ontologies without a definite ground truth.

The thesis is written in English and is 56 pages long, including 7 chapters, 2 figures and 9 tables.

Annotatsioon

Ontoloogiate liitmine Prologis pehme unifitseerimise põhimõttel

Selle lõputöö eesmärk on arendada automaatne tööriist OWL-vormingus ontoloogiate liitmiseks et tagada ülesannete hea koordineerimine koostöörobotite kontekstis. Ontoloogia tähendab antud kontekstis mingit tüüpi formaalset teadmiste esitust, mille minimaalseteks komponentideks on klassid, omadused ning relatsioonid klassiliikmete vahel. Üheks üldlevinud ontoloogiate esitluskeeleks on OWL.

Rakenduses kasutatav tehnika põhineb sõne- ja semantilisel sarnasusel, mis võtab ühtlasi arvesse võrreldavate mõistete struktuurset heterogeensust. Projekt on peamiselt implementeeritud programmeerimiskeeles SWI-Prolog eesmärgiga kombineerida omavahel sõne- ja semantilise sarnasuse tehnikad Prologi unifitseerimismehhanismiga. Seda lähenemist nimetame pehmeks unifitseerimiseks.

Jooksutasime lõputöö valideerimiseks jooksutasime programmi OAEI testandmestikul, mille keskmine täpsus 42 testi peale oli 60%. Ühtlasi kasutasime programmi võrdlemaks omavahel mitut robootika valdkonna ontoloogiat, mille tulemuseks õnnestus tuvastada väike, kuid üldjuhul korrektne, hulk omavahel sarnaseid mõisteid. Nende tulemuste põhjal on selge, et arendatud tööriistal on hea võimekus tuvastada semantiliselt sarnaseid mõisteid. Tulemuste analüüs toob välja ka ontoloogiate võrdlemisest johtuvaid väljakutseid olukorras, kus puudub selgelt määratletud alustõde.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 56 leheküljel, 7 peatükki, 2 joonist, 9 tabelit.

List of Abbreviations and Terms

APRS	Agility Performance of Robotic Systems
ARBI	Adapting Robot Behaviour based on Interaction
ARMOR	ROS Multi Ontology Reference framework
CORA	Ontology for Robotics and Automation
CPU	Central processing unit
CSV	Comma-separated values
DOLCE	Descriptive Ontology for Linguistic and Cognitive Engineering
FCA	Formal Concept Analysis
FOAF	Friend of a Friend
IEEE	Institute of Electrical and Electronics Engineers
IEQ	Indoor Environmental Quality
ILP	Inductive Logic Programming
IRI	International Resource Identifier
NLP	Natural Language Processing
OAEI	Ontology Alignment Evaluation Initiative
ORO	OpenRobots Common Sense Ontology
OROSU	Ontology for Robotic Orthopedic Surger
OUR-K	Ontology-based Unified Robot Knowledge
OWL	Web Ontology Language
PMK	Perception and Manipulation Knowledge
RDF	Resource Description Framework
ROA	Robot Architecture Ontology
SLD resolution	Selective Linear Definite clause resolution
SOMA	Socio-physical Model of Activities
SUMO	Suggested Upper Merged Ontology
UFO	Unified Foundational Ontology
W3C	World Wide Web Consortium

Table of Contents

List of Figures

List of Tables

1. Introduction

The topic of this thesis is ontology merging using Prolog and the principle of soft unification. Ontologies in the field of informatics are generally understood as a formal representation of domain-specific knowledge structured into concepts, properties of a concept and relations between concepts. Ontology merging is a type of ontology operation where two or more input ontologies are compared based on some sort of method and similar concepts within these ontologies are matched. In order to decide whether a particular pair of concepts are similar or not, different similarity measures can be applied - similarity based on the unification of terms, similarity as an equivalence relation, similarity based on semantic distance and many others.

The motivation for this thesis is to provide means for interoperability between knowledge-based autonomous decision systems that have individual knowledge built in or acquire this knowledge by learning and adjusting during the course of their earlier operational experience. The prevailing knowledge representations and manipulation techniques in autonomous robot systems rely on ontologies and ontology operations.

The aim of this work is to provide an algorithmic and automatic tool to consolidate the knowledge, remove discrepancies and optimize knowledge bases of robot agents to assure well-coordinated task completion in the context of collaborative multi-robot missions. We provide an integrated approach by combining multiple metrics and exploiting mainstream knowledge representation and manipulation frameworks such as RDF and OWL. This allows easy porting of its results and integration with other knowledge platforms.

The main novelty of this thesis is the elaboration of weak unification principle for ontology operations and its integration with existing RDF/OWL frameworks. The method is implemented as a program that merges ontologies represented in RDF/OWL. The method employs a structural approach where the multi-criterial weighed similarity metrics are used to assess concepts by their structural composition and elements such as relations and their values. The comparison of individual values is based on string and semantic matching, making use of WordNet lexical database and string similarity measures.

While in natural language information can successfully be exchanged between participants even if they have different or incomplete understanding of the same concept, the knowledge representation of the same concept may differ in terms of its structure, terminology, level

of detail which drastically hardens its proper interpretation in artificial systems. As such, the developed approach to ontology merging permits a degree of difference between semantically close concepts even if they substantially differ by syntactic representation. This is achieved by the multi-criterial minimal confidence threshold and context sensitive meta-parameters. The method is completely automated, producing a new merged ontology in RDF/OWL as an output.

The findings of this work are presented from two different perspectives. Firstly, we provide a quantitative validation based on the Ontology Alignment Evaluation Initiative (OAEI) 2016 benchmark test. The test data is designed so that each new test introduces some form of variation to the original reference ontology. Variations include changes in the value names, structure of the concept and the ontology in general. Secondly, we analyze the results received from merging actual ontologies from the domain of robotics. These ontologies vary in terms of scope, level of detail and structure and unlike OAEI benchmark tests, no ground truth is provided in advance.

The rest of the thesis is structured as follows. Chapter 3 highlights related work and state-of-the-art in the domain of ontology operations and positions our approach in that context. In Chapter 2, we introduce the core concepts of ontology theory and ontology operations. Chapter 4 presents the method developed in this work. Chapter 5 describes the implementation of the method as a multi-plaftorm integrated software system. Chapter 6 presents experimental results to validate the method. Chapter 7 concludes the thesis with the discussion on the main results, their validity and open questions for future work.

2. Ontology and Ontology Operations

In this chapter, we describe the theoretical background for ontologies and their operations.

2.1 Definition for Ontology

The word *ontology* has its foundation in philosophy studies and metaphysics denoting "being in general, or of what applies neutrally to everything that is real" [1]. In the context of computer science, however, ontology is typically understood as a type of formal knowledge representation used to hold domain knowledge. This knowledge representation is usually expressed in the form of classes, properties and relations between class members [2]. Furthermore, Euzenat and Shivaiko [3] have provided a formal definition for an ontology as a tuple $o = \langle C, I, R, T, V, \sqsubseteq, \perp, \in, = \rangle$ where

- C is the set of classes;
- I is the set of individuals;
- R is the set of relations;
- T is the set of data types;
- V is the set of values (C, I , R, T , V being pairwise disjoint);
- $\sqsubseteq$ is a relation on $(C \times C) \cup (R \times R) \cup (T \times T)$ called specialisation;
- $\perp$ is a relation on $(C \times C) \cup (R \times R) \cup (T \times T)$ called exclusion;
- $\in$ is a relation over $(I \times C) \cup (V \times T)$ called instantiation;
- $=$ is a relation over $I \times R \times (I \cup V)$ called assignment.

In the terminology used in this thesis, what is named as a *class* in the above definitions will be referred to as a *concept* in later chapters. This is done to avoid confusion with the term *class* in the context of RDF/OWL framework. In addition, *relation* in the context of RDF/OWL is understood as any value in the predicate position of an RDF triple. Interpretation of ontology terms in the context of RDF/OWL are covered in Appendix 2.

Since the motivation of current research is to provide tools for semantic reasoning on knowledge interpretable using ontologies, we consider ontologies as logic theories and apply logic programming and logic constraint programming as a natural framework for representation and reasoning about logic. The language of logic consists of individuals, classes, functions, relations and axioms. The exact list of logic predicates and modalities changes depending on the specific logic language one adopts. Usually an ontology is

specified in First Order Language (FOL) though some logic programming languages such as SWI-Prolog provide means for expressing constructs that raises the abstraction level of ontology based semantic reasoning to the level of higher order constructs such as predicate and function variables, enabling the construction and application of domain agnostic meta-rules for semantic reasoning and meta-interpretive learning.

2.2 Taxonomy of Ontologies

Euzenat and Shivaiko discuss two main approaches to differentiate between types of ontologies: 1) by the degree of formality and 2) by the hierarchy of the ontology [3].

Distinction by Formality

With regards to the degree of formality, the least formal type of ontology could be a collection of tags or folksonomies describing some body of knowledge (e.g., annotating pictures on Flickr). Conversely, the most formal type of ontology would be expressed in some form of description logic, implemented as a special-purpose ontology language (such as OWL) [3]. The motivation to use a formal language for expressing an ontology is that it prescribes the possible ways of interpretation thus permitting unambigous interpretation and automatic reasoning on the basis of the syntactic and semantic rules of that language.

Distinction by Hierarchy

With regards to the hierarchy of the ontology, difference can made between foundational, upper-level and domain ontologies. Additional categorizations such as reference ontology and application-level ontology are also used.

Foundational ontologies are ontologies that define fundamental concepts (e.g., concepts such as *endurant* and *pedurant* that denote concepts we can perceive in their entirety at any given time and concepts that only partially exist at any given time) used in other ontologies. Foundational ontologies are complemented by upper-level ontologies which define non-foundational commonplace concepts (such as vechicles, people, etc.) that are typically also used by other ontologies [3]. For this reason, both foundational and upper-level ontologies can be seen as horizontal ontologies, as they aim to cover a broad spectrum of concepts on an abstract level. The distinction between these two types of ontologies is not always clear. Examples of general-purpose foundational or upper-level ontologies are the Unified Foundational Ontology (UFO) [4] and Suggested Upper Merged Ontology (SUMO) [5].

Domain level ontologies are ontologies that encompass concepts relevant to a specific

domain (e.g., robotics, bibilography, medicine, etc.). For that reason they can be seen as vertical ontologies [3].

2.3 Ontology Operations

The term *ontology operations* refers to a number of manipulations that are conducted to enhance the information contained in the ontologies. Usually this involves a comparison of one or more ontologies though not always. Maroun distinguishes the following ontology operations: *mapping*, *alignment*, *merging*, *annotation*, *matching* and *integration* [6]. However, it should be noted that the definitions of these operations are not entirely univocal. As such, we will also provide an interpretation of operations that are used in our own implementation.

2.3.1 Mapping

Mapping is an general term for "a formal expression describing the semantic relationship between two (or more) concepts belonging to two (or more) different ontologies." [6]. This semantic relationship could be for instance, equivalence, vertical (e.g. *X is a child of Y*) or horizontal (e.g. *X is part of Y*) relationship.

Informally, mapping of an equivalence relationship could be some type of operation that identifies concepts from different ontologies that refer to the same thing. Mapping of a vertical *is a sub-type of* relationship could be some type of operation that identifies, for example the concept of *mammal* from one ontolgoy to specific mammal species in another ontology.

Namyoun *et al* distinguishes additional contexts for ontology mapping: 1) ontology mapping between an integrated global ontology and local ontologies, 2) ontology mapping between local ontologies and 3) ontology mapping in ontology merge and alignment. In the context of this thesis, the third explanation is perhaps the most suitable. That is, "mapping establishes correspondence among source (local) ontologies to be merged or aligned, and determines the set of overlapping concepts, synonyms, or unique concepts to those sources." [7].

In contrast, Euzenat and Shivaiko call this operation (i.e., establishment of a relationship between concepts of different ontologies) as *correspondence* and describe mapping as the directed version of alignment, strictly expressing equivalence relationship. [3]

2.3.2 Alignment

Alignment is a "set of correspondences between two (or more) ontologies in the same domain or in related fields. These correspondences are called mappings" [6]. As a result of ontology alignment, source ontologies "become consistent with each other, but are kept separate" [7]. Similar description is also provided by Euzenat and Shivaiko [3].

2.3.3 Merging

Merging is the combination of mapped concepts into one, creating a new ontology from source ontologies [6]. This merged ontology contains the knowledge from all source ontologies but leaves it unchanged [7], [3].

2.3.4 Annotation

Annotation is process of creating metadata using ontology as a vocabulary [6].

2.3.5 Matching

Matching denotes finding correspondences between linguistically related concepts from different ontologies which can symbolize equivalence or any other semantic relationship [6]. A more abstract description is provided by Doan *et al* as "the problem of finding the semantic mappings between two given ontologies" [8]. Euzenat and Shivaiko call matching the process of "finding relationships or correspondences between entities of different ontologies" [3].

2.3.6 Integration

Maroun provides three definitions for ontology integration. The process of 1) building a new ontology by reusing other already available ontologies, 2) building an ontology by merging several ontologies into one that unifies them all or 3) building an application using one or more ontologies [6]. To differentiate this operation from ontology merging, the focus of integration is on the generation of a new functional ontology, whereas merging entails the mapping of relationships between ontologies and the reproduction of its results. This has been clearly observed by Nyamyoun *et al* [7] (emphasis my own):

Ontology integration is the process of generating a single ontology in one subject from two or more existing and different ontologies in different subjects. The different subjects of the different

ontologies may be related. *Some change is expected in a single integrated ontology*.

The same contrast is also expressed by Euzenat and Shivaiko, noting that in ontology integration "contrary to merging, the first ontology is unaltered while the second one is modified" [3].

2.4 Interpretation of Ontology Operations

The definitions for different ontology operations are not always fixed. Often the terms have several meanings depending on the context of their use (e.g., mapping and integration). The definitions may be interdependent (e.g., mapping is a expression of relationships that are discerned from process of alignment, whereas alignment is a set of correspondences called mappings). There is also interchangeable use of the terms on the general ontology level and on the level of individual concepts (e.g., mapping expresses relationships between concepts, but it can also be said that ontologies that are being mapped). For the sake of clarity, we provide our own interpretation for key ontology operations.

Mapping occurs between individual concepts and solely concerns with the *equivalence* relationship, as we attempt to identify concepts from two different ontologies that are similar to one-another. While inheritance or other relationships are important to determine how similar two concepts are, we do not attempt to map these relations between two concepts of different ontologies.

Matching occurs between individual concepts resulting in a measure of how similar (equivalent) those two concepts are. The methodology for matching in this work is primarily based on semantic and string matching which will be further discussed in Chapter 4.

Alignment occurs between ontologies and is the product of matching every concept in both source ontologies either to an equivalent concept in the other ontology or to none at all if no equivalent concept for a particular concept exists.

Merging occurs between ontologies, after mapping and matching have identified equivalent concepts. As a result of merging, a new ontology is generated, where matched concepts are presented as a single concept. We also introduce an additional step called *specialization* where this single concept only holds those properties and relations that are similar according to some form of metric, whereas separate sub-concepts are generated for those properties and relations that are different. If no equivalence was identified for a particular concept, it is generated as a separate concept with no additional relations.

Annotation and integration operations are out of the scope of this thesis.

2.5 Heterogeneity of Ontologies

One of the major challenges of implementing ontology operations is the different ways an ontology can be represented. Euzenat and Shivaiko [3] distinguish several types of heterogeneity, among them:

1. **Syntactic heterogeneity** which is caused by modelling the ontology in different ontology languages.
2. **Terminological heterogeneity** which is caused by variations in how concepts are specified. This may occur when ontologies are written on the basis of different natural languages, but also when synonyms or different phrasing is used in the same natural language.
3. **Conceptual heterogeneity** which is caused by differences in how knowledge of the same domain has been modelled in different ontologies. These differences may occur in *coverage* (two ontologies have slightly different domains that partially overlap), *granularity* (two onotologies have a different level of detail of the same domain) and *perspective* (two ontologies describe the same domain with the same level of detail but from a different perspective).
4. **Semiotic heterogeneity** which is caused by the interpretation of concepts by people. Heterogeneity in this context refers to how entities with the same semantic interpretation can be interpreted differently by humans depending on the context of their use.

In this thesis, we are mainly concerned with terminological heterogeneity and conceptual heterogeneity. We assume that concept names and other relation values are either written in English or are formulated on the basis of that. We aim to solve the terminological heterogeneity resulting from differences in the use of words or phrasing. As for conceptual heterogeneity, we aim to mitigate variations in how concepts are structured as well as in their level of detail.

3. Related Work

In this chapter we provide an overview of some of the main approaches used to implement onotology operations. A comprehensive overview of the field falls beyond the scope of this work, but we will outline main techniques used in existing work and how this relates to the methodology proposed in this thesis. Additionally, we provide an overview the use of ontologies in the domain of robotics which is the basis of validation of our own work.

3.1 Existing Work and Approaches

Most ontology operations rely on the notion of concept similarity. Euzenat and Shvaiko [3] propose three broad categories for how to measure the similarity of two concepts.

1. Name-based techniques which analyse the name, comment or other linguistic data of the concept. Difference is made between methods that purely use character strings and those that employ some type of linguistic knowledge.
2. Internal structure-based techniques which additionally take into account the structure of the concepts, such as the value and data type of their properties or the comparison of the concept to other concepts it is related to.
3. Extensional techniques which measure similarity on the basis of individuals (instances) of concepts. For example, if two concepts depicting the notion of a book share an identical set of book titles, we can surmise that these two concepts are equivalent. Extensional techniques are further distinguished into three distinct cases:
   (a) comparing instances that are common between two concepts;
   (b) instance identification in case a common set of instances does not exist;
   (c) disjoint extension comparison where, rather than directly using a common data set for both ontologies, statistical measures and approximation is used to compare concept extensions

3.1.1 String and Semantic Matching

As outlined by Euzenat and Shivaiko, a common approach used in ontology matching is analysing the string values of the concepts that are being compared. As this can be equally done with both concept names and property values, we do not strictly distinguish a name-based approach from an internal structure-based approach.

By *string matching* we mean any kind of technique that analyzes the structural similarity of two strings without considering the meaning of those strings in natural language. This can be purely mechanical (e.g., Levenshtein distance) or on the basis of some larger corpus of text (e.g., cosine distance).

By *semantic matching* we mean that the similarity of two strings is evaluated based on the natural language meaning of those words (for example, by comparing whether two English-language words are synonyms). This is typically done with the help of some lexical database, such as WordNet for English, that links words based on their meaning, semantic and grammatical relations [9].

String and semantic matching is extensively used, for example, in Robin and Uma [10] which employs a mixed strategy of both, choosing the best outcome from several metrics. Single-metric approaches also exist, such as Stoilos *et al.* that focuses solely on developing a single string-matching technique for ontology alignment [11]. String or semantic matching can also serve as a complementary technique to some other approach, as for example in Karimi and Kamandi [12] where trigram word-similarity is used as part of an inductive logic programming approach to ontolgy alignment.

An obvious drawback of using string and semantic matching is that it assumes that the names and property values of a concept are in some form meaningful in natural language. Unconstrained use of string matching is prone for false positives (e.g., words such as *correct* and *incorrect* are structurally very similar, despite being semantically opposite to one-another). Semantic matching may require extensive pre-processing of the string (such as identifying and separating individual words from a longer string) and be further complicated by identical words having different meanings.

3.1.2 Formal Concept Analysis

Formal Concept Analysis (FCA) is a data analysis technique to study the relationship between a set of objects and a set of attributes. Input data of FCA is typically represented as a cross table where the set of objects and the set of attributes are the dimensions and the presence of a property in an object is marked in the cross table. Using this data, FCA produces groups that 1) represent "natural" concepts based on the attributes of the data and 2) a collection of implications describing specific dependencies that exist in the data [13]. FCA was first proposed by Rudolf Willie in 1982 [14]. Note that *concept* in the context of FCA is not necessarily the same as in the discussion on ontologies.

Euzenat and Shivaiko position FCA as an extensional ontology matching technique for

when a common set of instances already exists. In such a case, attributes of the FCA lattice could be seen as the concepts where instances are known to belong, regardless of their source ontology. FCA would allow to identify not only equivalence relationship but also sub-type correspondences [3].

An example of FCA used in such a way can be found in the FCA-Merge approach proposed by Stumme and Maedche [15]. FCA-Merge takes ontologies and a set of domain-specific text documents as input. Natural language processing techniques are used to extract instances from domain-specific texts. FCA is used to derive a concept lattice from this input data. Finally a new merged ontology is generated on the basis of the concept lattice and human domain specialist interaction.

3.1.3 Background Knowledge and Inductive Logic Programming

Inductive Logic Programming (ILP) is a branch of machine learning with a focus on learning from examples (*induction*). ILP is used for instance to learn concept descriptions from instances of those concepts or to find regularities from large amounts of data. ILP typically makes use of *background knowledge* - some kind of data that is relevant for the task at hand in the form of positive and negative examples [16]. A typical representation of data and rules used in ILP is in first-order predicate logic.

ILP is used by Karimi and Kamandi by Karimi *et al* for ontology alignment. This approach consists of 1) generating background from concept taxonomies, 2) explicit addition of domain-specific background knowledge, 3) generation positive and negative examples from a set of instances, 4) induction using an algorithm developed by the authors, 5) interpretation of the result of induction as ontology mapping [12].

3.1.4 Our Approach

Based on the techniques described above, the approach developed in this thesis can be seen as a structure-based analysis of all the relations attributed to a concept directly and inherited from their ancestors. The values of these relations are compared for their similarity on the basis of string and semantic matching. Additional techniques are adopted for asymmetric structures, where the number of values a relation has does not match between the two concepts (the application of weak unification principle). Additionally we use FCA to pre-process the data on the relations of each concept and identify identical relations in advance.

Similarly, weak unification is applied for the aggregate similarity estimate, where several input parameters in the range of [0,1] are used to determine the weight of different similarity measures or the threshold of what constitutes a positive match. The method is intended to be automatic without the need for human intervention.

3.2 Ontologies in Robotics

Ontologies may be used in any possible domain, ranging from a taxonomy of proteins to describing the structure of an organisation. In the domain of robotics, ontologies are particularly useful in the context of autonomous social robotic systems where knowledge representation regarding the objects and the environment can be achieved through the use of ontologies. Domain knowledge represented in ontologies improves the flexibility, re-usability and adaptability of various robotic tasks [17].

To mitigate the heterogenity of ontologies developed for robotics, IEEE has proposed a standard for knowledge representation and reasoning in autonomous robotics or autonomous robot architecture ontology (ROA) which provides a conceptual framework for sharing information about robot architectures [18]. ROA is built on top of Suggested Upper Merged Ontology (SUMO) and Ontology for Robotics and Automation (CORA) that pass on some of their concepts to ROA. CORA is an upper-level ontology specific to the field of robotics, providing a knowledge representation of concepts such as *robot*, *robotic system*, *robot part* among others [19]. Other upper-level ontologies relevant for robot missions are Descriptive Ontology for Linguistic and Cognitive Engineering (DOLCE) [20], Ontology-based Unified Robot Knowledge (OUR-K) [21], OpenRobots Common Sense Ontology (ORO) [22].

We will not attempt to provide a comprehensive overview of domain-level ontologies as these are numerous and specific to their domain or application. Rather we will use the survey conducted by Manzoor *et al* [17] that describes a number of domain ontologies related to robots with social roles. These are the KnowRob [23] project and related SOMA ontology [24], Ontology for Robotic Orthopedic Surgery (OROSU) [25], CARESSES [26], Perception and Manipulation Knowledge (PMK) [27], search and rescue SARbot [28], indoor environmental quality (IEQ) [29], SmartRules [30], Adapting Robot Behaviour based on Interaction (ARBI), Worker-cobot [31], Agility Performance of Robotic Systems (APRS) [32].

A majority of these systems use OWL for knowledge representation. Prolog, the language of choice in this thesis, is also used in KnowRob, PMK and ARBI. A common tool for creating or editing ontologies was Protege [33].

Work has also been done to develop a universal approach for linking an OWL ontology with ROS middleware. One ongoing effort in this regard is the Java-based ROS Multi Ontology Reference (ARMOR) framework [34] that also has a publicly available repository with installation instructions and other tutorials [35]. Unfortunately, the project has not been updated over several years and is dependent on a version of ROS that is by now outdated.

4. Developed Methodology

In this chapter, we describe in greater detail the methodology we developed in our approach to ontology merging. It is assumed that the ontologies are represented in the RDF/OWL framework. A thorough overview of RDF/OWL framework can be found in Appendix 2.

On the most abstract level and using the terminology introduced in Chapter 2, we interpret ontology merging as the comparison of every concept C_i in ontology $\mathcal{O}_1$ to every concept C_j in ontology $\mathcal{O}_2$ and matching the concept pairs that exceed some predefined similarity threshold.

The degree of how similar two concept are is referred to as the *confidence value* of each concept pair. The minimal degree of similarity to declare two concepts as being similar is expressed as a parameter in the range of [0,1] and is referred to as *confidence threshold*. A concept from ontology is *matched* to a concept from another ontology if neither of the concepts is already matched, their confidence value exceeds the confidence threshold and it is also the highest available confidence value for both concepts in the concept pair. If a concept is not paired to any other concept, it is considered to be *mismatched*.

$$\text{Let } C_i \in \mathcal{O}_1, C_j \in \mathcal{O}_2$$

The signature of the merging operation could formally be expressed as

$$merge : \mathcal{O} \leftarrow \begin{cases} \mathcal{O} \cup match(C_i, C_j) & \text{if} \quad max_{k\in[1,m]}(f_k) \geq f^U \wedge match(C_i, .) \notin \\ & \quad \mathcal{O} \wedge match(., C_j) \notin \mathcal{O} \quad \forall C_i \in \mathcal{O}_1 \wedge \\ & \quad \forall C_j \in \mathcal{O}_2 \\ \mathcal{O} \cup \{C_i\} \cup \{C_j\} & \text{otherwise} \end{cases} \tag{4.1}$$

where

- C_i and C_j are input concepts;
- $\mathcal{O}_1$ and $\mathcal{O}_2$ are input ontologies;
- $\mathcal{O}$ is the merged output ontology;
- $match(C_i, C_j)$ is the match of concepts C_i and C_j;
- $max_{k\in[1,m]}(f_k)$ is the maximum confidence value from all possible comparison pairs $[1, m]$;

- f^U is the confidence threshold.

The matched concept pairs are further refined by assessing the confidence value of each relation value pair in the matched concept pair. If the confidence value of a particular value pair falls below the confidence threshold f^U, sub-concepts of the matched concept pair are created and the diverging relation type and value are added as a distinct sub-concept. This could be considered the *specialization* of matched concepts denoted as $\sqsubseteq$. The new merged ontology is the return value after the completion of the specialization operation.

Let $rels$ be an operation that return the set of all relation values of a concept:

$$rels : C \rightarrow \{rel_1, ..., rel_n\}$$

then

$$rels(match(C_i, C_j)) = rels(C_i) \cup rels(C_j).$$

and the specialization operation $\sqsubseteq$ is expressed as

$$\sqsubseteq (rels(match(C_i, C_j))) = (\{rels(C_i) \cap_f rels(C_j)\}) \cup \{rels(C_i^{'})\} \cup \{rels(C_j{}^{'})\} \tag{4.2}$$

where

- $\cap_f$ operator signifies the intersection of relation value pairs whose confidence value exceeds the confidence threshold;
- $C_i^{'}$ and $C_j^{'}$ are the new subconcepts created as a result of specialization;
- $rels(C_i^{'}) = rels(C_i) \backslash rels(C_j)$ and $rels(C_j^{'}) = rels(C_j) \backslash rels(C_i)$.

4.1 Similarity Matching as a Form of Weak Unification

In the context of logic programming, unification of terms is the process of transforming logic clauses using mgu-algorithm [36] to a form where some of their literals become equivalent modulo negation, allowing the elimination of these literals from the derived clause by the SLD-resolution rule. It is used in Prolog to evaluate the truth value of a goal on the basis of clauses with the instantiation of their argument terms that are provided in the knowledge base. In order to make better use of Prolog unification mechanism, we apply the encapsulation of clauses [37], i.e., we transform all concepts to a single uniform representation where relation types that were found in both input onotologies are taken as arguments to be used for grounding of concepts. Therefore, the value of each argument is represented as a list of relation values of that type. If a concept does not have a particular

relation, the argument value is an empty list. The fundamental difference between strong syntactic unification and the weak unification applied in our method for term comparison and clause merging is following: while in syntactic unification, the terms are transformed to syntactically equivalent form, weak unification contrary enables some difference in the structure of terms and values up to predefined threshold in some well-defined similarity metrics. The relation values are then calculated for their similarity.

Hence, in the case of strong unification, two concepts are matched only when their argument values are either identical or they are variables that can be uniformly instantiated on the basis of Prolog knowledge base. We call the current approach *weak* unification because we use a similarity metric (*confidence*) instead of Prolog's internal unification mechanism.

For example, the following two concepts named *android* would not be unified using strong unification due to the asymmetry of relations and a differences in the values these relations hold. However, they could be unified in the sense of weak unification if the semantic connection between *robot* and *automaton* is made (using, for example, WordNet) and the absence of a value in the relation `equivalentClass` is permitted.

```
android(equivalentClass([]), subClassOf(['robot']).

android(equivalentClass(['humanoid']), subClassOf(['automaton'])).
```

4.2 Calculating the Confidence of a Concept Pair

The similarity confidence of a concept pair is the calculated average of the confidences calculated over each relation value pair of compared concepts. If a relation has more than one value, the values are matched using a best-first approach and each value pair is considered separately. The two concepts need not have an identical structure: either one of the concepts has a relation that the other does not or the two concepts have a different number of values for the same relation. We use an additional input parameter referred to as *confidence value of an absence* which determines what confidence value is attributed to a relation that does not have a counterpart in the concept it is compared with.

Relation values of any ancestral concept are also used in similarity calculation, if the concept has an ancestral relation (e.g., in RDF expressed either as `rdfs:subClassOf` or `rdfs:subPropertyOf`) and there exists knowledge of the ancestor concept (e.g., either it is defined in the same ontology or is provided as background knowledge).

OWL class descriptions (e.g., `owl:Restriction`, `owl:intersectionOf`, `owl:unionOf`, `owl:complementOf`) are compared on same principles as concepts, but are constrained to matching relation types. If a pair of class descriptions does not have identical relation types, their similarity is considered to be 0, regardless of relation values.

The formal expression of calculating the similarity confidence $\bar{S}(C_1, C_2)$ of a concept pair (C_1, C_2) is

$$\bar{S}(C_1, C_2) = \frac{\Sigma_{i=1}^{n}(s_i)}{n} \tag{4.3}$$

where

- n is the number of relation value pairs existing in both input concepts;
- s_i is the similarity confidence of the *i*-th relation value pair.

Additional examples on calculating concept pair confidence can be found in Appendix 4.

4.3 Calculating the Confidence of a Relation value pair

In this section we describe the method for comparing individual relation value pairs that make up the confidence of the concept pair. Several comparison methods are used in parallel, the weights of which are determined as input parameters. From a more abstract perspective, any number of methods may exist, each with their own weight. However, in the current thesis, we specifically use *semantic similarity* and *string similarity* to calculate the similarity of relation value pairs which were introduced in 3.1.1. The weight of either method is implemented as an input parameter of the matching algorithm. Additionally, we use an exception measure in situations where the weight of one metric has a high value (i.e., greater than parameter S^u) that is distinctly dominating over values of other metrics.

The calculation of the similarity confidence of a relation value pair $\bar{S}(R_1, R_2)$ could be formally expressed as

$$\bar{S}(R_1, R_2) = \begin{cases} \Sigma_{i=1}^{n}(w_i \times S_i(R_1, R_2)) & \text{if } \forall i : S_i(R_1, R_2) < S^u \\ S_k(R_1, R_2) & \text{if } \exists k \in [1, n] : max(S_k(R_1, R_2)) \geq S^u \end{cases} \tag{4.4}$$

where

- R_1 and R_2 are relation values
- n is the number of value matching techniques;
- w_i is the weight of method i

- $S_i(R_1, R_2)$ is the similarity confidence of a relation value pair using method i;
- S^u is the single dominance threshold of similarity confidence.

4.3.1 Semantic Similarity

WordNet is used to examine the meaning of word(s) that are found within the relation value. Each relation value is discerned into a list of words and that are reduced to their root form for normalized comparison. For instance, relation values `rdf:about="iri1#isGridPowered"` and `rdf:about="iri2#has-grid-power"` would be processed into following list of words: `[be, grid, power]` and `[have, grid, power]` respectively.

Every word in one list is then compared with every word in the other list for evaluating the similarity in meaning. Three types of similarities are distinguished:

1. the words are identical, meaning that the similarity value of that word pair is 1.
2. the words are synonyms the value of which is taken from an input parameter. We refer to this input parameter as the *confidence value of a synonym*.
3. the words express any other positive semantic relation in which case the similarity value of the word pair is half of the synonym input parameter value. We refer to this input parameter as the *confidence value of other semantic relation*. Any other positive semantic relation could be for instance, hypernymy (i.e., the words have a subordinate-superordinate relationship, such as *cat* is a *mammal* and *mammal* is an *animal*), meronymy (i.e., the words have part-of relationship, such as *hand* is part of a *body*), etc. The full description of semantic relations expressed in WordNet can be found on [38].

The semantic similarity confidence of a relation value pair is the sum of all identified similarities divided with the total number of words in both compared relations. The formal representation of calculating semantic similarity is essentially the same as defined in Formula 4.3.

For example, given 0.9 as confidence value of a synonym and 0.45 as confidence value of other semantic relation, the similarity value for the list of words `[bike]` and `[electric, bike]` would be 2÷3= 0.667. For `[bike]` and `[electric, bicycle]`, this would be 1.8÷3=0.6, as *bike* and *bicycle* are identified as synonyms. For `[velocipede]` and `[electric, bicycle]`, this would be 0.9÷3=0.3, as *bicycle* is a hyperonym of *velocipede*.

Additional Heuristics

Some additional constraints have been introduced to improve the accuracy of semantic similarity. These were employed on the basis of patterns exhibited in false positives and can thus be seen as constraints to the grammatic context of where words occur. These will discussed in greater detail in Chapter 5.

4.3.2 String Similarity

String similarity characterizes relation values without considering possible meaning of words in those values. Relation values are similarly separated into a list individual words which are then concatenated into a single string and then compared.

Different methods exist on how to measure the similarity of two strings. In our approach, we are using a string metric developed by Stoilos *et al.* [11]. The method is based on the length of common substrings, similar to Levenshtein distance which measures the minimum number of changes needed to convert one string to another. The smaller the number of changes, the more similar two strings are [39].

Reasons for choosing Stoilos *et al.* as the key metric are mostly implementation-related. As possible alternative, we also considered cosine similarity which is a method of calculating the similarity of two vectors by taking the dot product and dividing it by the magnitudes of each vector [40]. The *vector* in this case is a body of text in vectorized form. Various methods and tools exist to vectorize text, however, in its essence, it entails converting the text into numerical representation (i.e., counting frequency of each word in text). The closer the angle between the two vectors is to zero, the more similar these texts are.

One of the principal difference between Stoilos *et al.* and cosine similarity is that the method proposed by Stoilos *et al.* is a purely mechanical approach, providing one singular result. Conversely, the result of cosine similarity depends on an additional corpora of text (NLP model(s)). The more attuned this model is to the specific topic of the texts that are compared, the more accurate result one can expect.

In order to better determine the suitability of either string similarity method, we compared the results provided by Stoilos *et al.* and cosine similarity with a number of ready-to-use NLP models on the set of actual data which revealed that there was no overwhelming advantage to using the more implementation-heavy cosine similarity. Results of this comparison can be found in Appendix B.1.1.

4.3.3 String Similarity of Comments

Concepts' descriptions in ontologies sometimes include natural language comments, represented as the relation `rdfs:comment`. As the occurrence of comments is largely ontology-specific and does not describe an additional property of a concept in itself, it can rather be seen as an annotation on the concept. As such, they are not compared as a separate relation, but as an supplementary value to the name of the concept. As comments may range from a few words to several paragraphs, only string similarity method is used. The weight of comment similarity is determined as an input parameter, however, if one of the compared concepts does include a comment, this weight is ignored entirely.

4.3.4 Determining the Threshold Values of Similarity Metrics

Both semantic and string similarity method have a weight that is determined as an input parameter of the matching algorithm and the similarity of a relation value pair is determined as a weighed average of the results from each method. However, when the result of a single method is greater or equal to an upper threshold (0.9 by default), only that particular method is exclusively used to assess the similarity of the relation value pair. This approach ensures reinforcement of more decisive matching result when other similarity methods provide a low score.

For example, when comparing `Robot` to `Automaton`, semantic similarity method returns a similarity score of 0.9, as the two words synonymous in WordNet (assuming 0.9 is the input argument for synonym weight). However, string similarity method provides a similarity score of 0. Clearly, providing a high similarity value on the basis of semantic similarity alone is preferable to providing a considerably lower value as a weighed average of both similarity values.

The opposite also applies when semantic matching either fails to identify words in the relation value (e.g., the relation value is an abbrevation) or the underlying semantic connection, while string matching is able to recognize the similarity of the two values. For example, the comparison of `relation` and `relational`, semantic matching returns a similarity score 0, because even though both words are documented in WordNet, there is no explicit syntactic relation between them. String matching on the other hand identifies the connection between these words and provides a confidence value of 0.96.

4.3.5 Confidence Pre-Value

So far we have considered relation values as strings. However, when two relation values both refer to a concept in their ontology, the last known confidence value of the already compared concepts is used instead of the string comparison. We refer to this as *confidence pre-value* as it is the confidence value of the concept pair that is known from the previous comparison iteration and to exclude repeated evaluations, existing estimate is used.

In the following example, the relation `rdfs:subClassOf` values `#Electric-Vechicle` and `#WheeledVechicle` are not calculated on the basis of their lexical or string similarity, but rather the confidence value of the respective concepts is used instead.

Example A

```
<owl:Class
    rdf:about="iri1#ElectricBike">
    <rdfs:subClassOf
        rdf:resource="iri1#Bike"/>
    <rdfs:subClassOf
        rdf:resource=
            "iri1#ElectricVechicle"/>
</owl:Class>

<owl:Class
    rdf:about=
        "iri1#ElectricVechicle">
    <rdfs:subClassOf
        rdf:resource="iri1#Object"/>
</owl:Class>
```

Example B

```
<owl:Class
    rdf:about="iri2#ElectricBike">
    <rdfs:subClassOf
        rdf:resource="iri2#Transport"/>
    <rdfs:subClassOf
        rdf:resource=
            "iri2#WheeledVechicle"/>
</owl:Class>

<owl:Class
    rdf:about=
        "iri1#WheeledVechicle">
    <rdfs:subClassOf
        rdf:resource=iri1#Object"/>
</owl:Class>
```

4.4 Validation

We employ two approaches to validate our results: 1) a quantitative approach on the basis of test data from the Ontology Alignment Evaluation Initiative OAEI, 2) a qualitative approach based on the results from comparing robotics-related ontologies outlined in [17]. The results of both approaches will be discussed in Chapter 6.

4.4.1 OAEI Based Evaluation

OAEI is an international initiative that hosts a number of tests sets as well as annual competitions and workshops for ontology matching. We use the 2016 benchmark test

[41] [42] that comprises of a base ontology and 110 test ontologies from the domain of bibliography. The base ontology consists of 33 named classes, 24 object properties, 40 data properties, 56 named individuals and 20 anonymous individuals. Each new test introduces some form of variation to the original reference ontology. These variations include changes in the value names (e.g., different naming conventions, use of synonyms, scrambling), structure of the concept (e.g., omission of restrictions, comments or other relations) and the ontology in general (e.g., removal or addition of concepts). In addition, four tests are based on real bibilographic ontologies that only partially correspond to the reference ontology.

4.4.2 Merging Robotics-Related Ontologies

In addition to OAEI test set, we analyze the results gathered by comparing selected ontologies from the domain of robotics. More specifically, we compared three ontologies introduced in Section 3.2: DOLCE, PMK, SOMA.

In terms of ontology taxonomy, PMK and SOMA are both domain level ontologies that differ in terms of subdomain and scope. PMK focuses on autonomous robot perception and manipulation [43] whereas SOMA on the characterization of physical and social activity context [44]. DOLCE is a foundational ontology that aims to "model a commonsense view of reality" [20]. SOMA is partially based on DOLCE foundational framework, whereas PMK has no direct connection to DOLCE. As such, we can expect SOMA and DOLCE to have the biggest and PMK and DOLCE to have the smallest overlap. The specific version of DOLCE we used for the comparison is version 397 of Dolce-Lite [45]. In terms of size SOMA ontology is the biggest, consisting of 839 defined concepts and 6,471 triplets. DOLCE ontology consists of 107 defined concepts and 872 triplets, whereas PMK has 90 concepts and 306 triplets. This clearly shows how ontologies differ not only in the number of concepts but also in their level of detail (average number of triplets/relations per concept).

5. Implementation

In this chapter, we cover the implementation of the methodology described in Chapter 4. We start by describing the general architecture and set of technologies used and then discuss specific sub-parts of the program.

The main programming language we use is SWI-Prolog. The choice of language is primarily due to the unification mechanism unique to Prolog that we use to our advantage. Furthermore, SWI-Prolog provides an extensive library for RDF manipulation [46] that we use for reading the input ontologies and creating the output ontology as well as other useful libraries. Additionally we use Python as a supporting language to conduct FCA and natural language processing. Python was chosen over other Programming languages due to its relative ease of implementation as well as the useful libraries provided in that language. Python scripts are launched directly from Prolog as additional processes, CSV file format is used to communicate larger quantities of data between Prolog and Python.

Conceptually, the program can be divided into three distinct phases:

1. *preprocessing* phase that reads the input files in RDF/OWL format, transforms the concepts to Prolog's internal representation and runs the FCA;
2. *comparison* phase that reads the results of FCA from `.csv` files, compares the concepts using the methodology described in Chapter 4, and produces the set of matched concept pairs and the set of mismatched concepts;
3. *refinement* phase that further analyzes matched concept pairs on the basis of individual relation values and produces the merged ontology in RDF/OWL as output.

A visual representation of these phases can be found in Figure 1

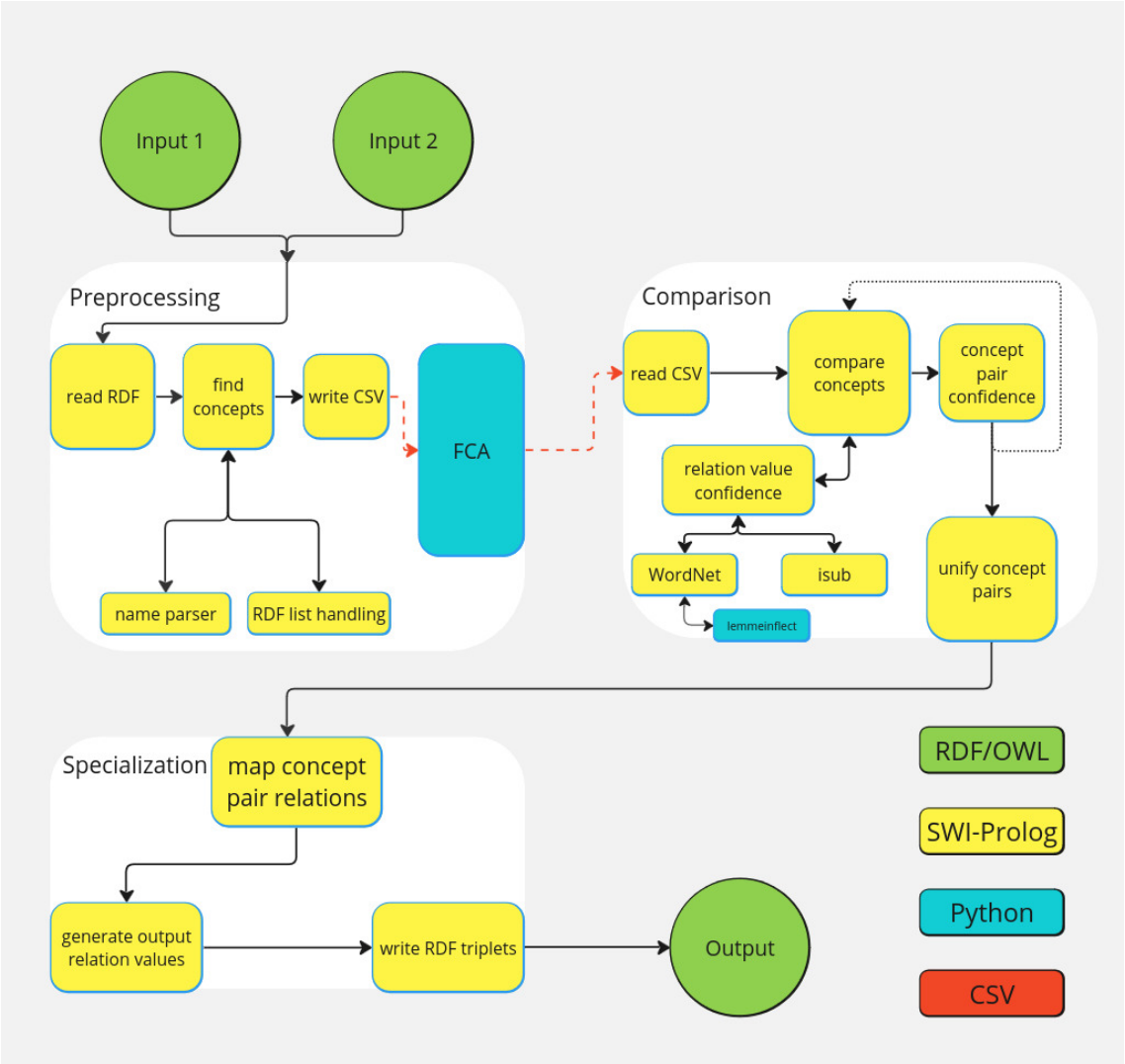

Figure 1. Conceptual architecture

5.1 Preprocessing

Preprocessing handles the input reading using Prolog's RDF manipulation library, identifies what concepts exist in the input ontologies and transforms them to Prolog's internal representation as dynamic facts and runs the FCA. In addition, preprocessing handles parsing relation values from various different naming conventions into a single uniform representation and transforms `rdf:List` into Prolog lists. We will illustrate these transformations with examples in Prolog-based pseudo-code.

5.1.1 Internal Representation of Concepts in Prolog

As described in Appendix 2, all data in RDF is represented as graph of triplets that consists of interlinked subject, predicate and object elements. Any concept can be present both as a subject as well as an object of any number of RDF triplets, whereas the predicate indicates the type of relation that exists between the subject and the object.

In Prolog, we represent concepts as dynamic facts that are structured such that all the relation types that were identified in both input ontologies (in other words, all the predicate values identified in the RDF triplets) are taken as arguments of a concept. The value of each argument is a tuple, consisting of the type of relation with a list of relation values as the argument value. If a concept does not have particular relation, the tuple for that relation takes an empty list as its argument value. Relation types are sorted beforehand so as to ensure that the same relation type is present in the same argument position for every concept.

For example, these two concepts expressed in RDF

Concept A

```
<owl:ObjectProperty
    rdf:about=
        "iri1#isGridPowered">
    <rdfs:domain
        rdf:resource=
            "iri1#device"/>
    <rdfs:domain
        rdf:resource=
            "iri1#airplanes"/>
    <rdfs:subPropertyOf
        rdf:resource=
            "iri1#powerTypes"/>
</owl:ObjectProperty>
```

Concept B

```
<owl:ObjectProperty
    rdf:about=
        "iri2#has-grid-power">
    <rdfs:domain
        rdf:resource=
            "iri1#airplanes"/>
    <rdfs:range
        rdf:resource=
            "iri2#battery"/>
</owl:ObjectProperty>
```

would have the following internal representation in SWI-Prolog:

Concept A

```
'iri1#isGridPowered'(domain(['iri1#airplanes', 'iri1#device']),
    range([]), subPropertyOf(['iri1#powerTypes']),
    type(['#ObjectProperty']).
```

Concept B

```
'iri2#has-grid-power'(domain(['iri1#airplanes']),
    range(['iri2#battery']), subPropertyOf([]),
    type(['#ObjectProperty']).
```

The advantage of this representation is that it is universal regardless of the relation types that exist in the input ontologies or what relation types have a value in the case of a specific concept.

5.1.2 Parsing of Relation Values

As noted in Appendix 2, RDF does not prescribe any particular convention, how RDF resources should be named. As such, the naming convention of relation values depends on the creators or the tool used for generating ontology files. Typically, the IRI is separated from the resource name by `#` and either `CamelCase` or `dash-case` is used for the name. Different capitalization conventions may also be used depending on the type of the resource (e.g., `FullCamelCase` is used for `rdfs:Class` concepts, whereas `lowerCamelCase` is used for concepts that are of type `rdf:Property`).

The current implementation handles `CamelCase` and `dash-case` regardless of capitalization. The IRI separator is expected to be either the last `#` or `/` character of the full string.

Several representations of relation values are held as dynamic facts: 1) the full name with the IRI for unique identification, 2) relation value split as per naming convention into list of words as that is used for semantic similarity calculation, 3) atomic version of the parsed word list which is used for string similarity calculation, 4) shorthand version of the unparsed relation value with the IRI omitted that is used for referencing relation values during the comparison phase.

The concept names of the previous example would have the following representations:

Concept A

```
'iri1#isGridPowered', [is, grid, powered], 'is grid powered',
    isGridPowered
```

Concept B

```
'iri2#has-grid-power', [has, grid, power], 'has grid power',
    'has-grid-power'
```

5.1.3 Handling of rdf:List Resources

`rdf:List` is essentially a linked list where each element in the list refers to a the next one. Unfortunately, this makes it extremely cumbersome to compare elements in `rdf:List` as only the first element in the list is linked to the top level relation that applies to every element in the list, while the actual order of elements in the list is irrelevant form a semantic point of view. To ease later manipulation, all values in `rdf:List` are directly linked to whatever relation lies immediately above the `rdf:List`.

We will use the following example, representing the statement "*John is a friend of Alice, Bob and anyone who is a friend of Bob*".

```
<Person rdf:about="#John">
    <isFriendOf>
        <owl:Class>
            <owl:unionOf rdf:parseType="Collection">
                <Person rdf:about="#Alice"/>
                <Person rdf:about="#Bob"/>
                <owl:Restriction>
```

```
                    <owl:onProperty
                        rdf:resource="#isFriendOf" />
                    <owl:hasValue rdf:resource="#Bob" />
                </owl:Restriction>
            </owl:unionOf>
        </owl:Class>
    </isFriendOf>
</Person>
```

There are three `rdf:List` resources (Alice, Bob and a property restriction) encapsulated into the anonymous class with a `owl:unionOf` relation which in return is an object of the relation `isFriendOf`. In the Prolog implementation, elements of the `rdf:List` are directly linked to the anonymous class using the following representation:

```
'#anonymous_class_ref',
    [unionOf(['#Alice', '#Bob',
        '#property_restriction_ref'])]
```

This can be in return referred to from general representation of the concept, e.g.:

```
#John(isFriendOf(['#anonymous_class_ref']),
    type(['#Person']))
```

Note that it is important also retain the relation type (in this case `owl:unionOf` referring to the list. Though rare, it is possible that a single RDF resource encapsulates more than one collection of `rdf:List` with different relation types. For that reason relation types with `rdf:List` resources are held as a list themselves, rather than single elements.

An exception is made if the immediate relation referring to the `rdf:List` is `owl:intersectionOf`. In that case, the values of the `rdf:List` are lifted to the next immediate relation as described in Section B.1 since multiple values of the same relation type can be interpreted as an intersection of those values.

To illustrate this, we will use the example "*John is a friend of anyone who is a friend of both Alice and Bob*".

```
<Person rdf:about="#John">
    <isFriendOf>
        <owl:Class>
            <owl:intersectionOf rdf:parseType="Collection">
```

```
                <owl:Restriction>
                    <owl:onProperty
                        rdf:resource="#isFriendOf" />
                    <owl:hasValue rdf:resource="#Alice" />
                </owl:Restriction>
                <owl:Restriction>
                    <owl:onProperty
                        rdf:resource="#isFriendOf" />
                    <owl:hasValue rdf:resource="#Bob" />
                </owl:Restriction>
            </owl:intersectionOf>
        </owl:Class>
    </isFriendOf>
</Person>
```

In this case there are two `rdf:List` resources, both property restrictions. Rather than linking them to the anonymous class with the `owl:intersectionOf` relation, we replace the anonymous class with a (Prolog) list containing all the `rdf:List` resources as an object of the `isFriendOf` relation. The internal representation of the concept would therefore be:

```
#John(isFriendOf(['#property_resctriction_ref_1',
    '#property_resctriction_ref_2']), type(['#Person']))
```

Note that this transformation is not possible when the anonymous class with `owl:intersectionOf` is itself nested in another `rdf:List` or when there are other relations in the same anonymous class. In such cases, the `rdf:List` resources are handled in the same way as in the case of any other relation type.

5.1.4 FCA

Once the concepts have identified and their contents has been transformed into Prolog's internal representation, the next step is conducting FCA. As the input of FCA is a matrix consisting of objects and attributes that they possess, every concept is interpreted as an object and every single relation value is interpreted as an attribute in FCA. In addition to relation values that are explicitly declared to a concept, the object is also attributed with all the relation values they inherit from their ancestors. We consider concept types to be strictly disjoint which is why FCA is conducted separately for each value of `rdf:type`.

To illustrate this, we expand our example of `iri1#isGridPowered` and `iri2#has-grid-power` from Section 5.1.1 with an ancestor to `iri1#isGridPowered`.

Concept C

```
<owl:ObjectProperty rdf:about="iri1#powerTypes">
    <rdfs:domain rdf:resource="iri1#aviation"/>
    <rdfs:range rdf:resource="iri1#battery"/>
</owl:ObjectProperty>
```

The merged input data for FCA of `owl:ObjectProperty` type of concepts would be following:

Table 1. Example of FCA input data

object	domain (iri1 #air-planes)	domain (iri1 #device)	subPropertyOf (iri1 #power-Types)	type (Object-Property)	domain (iri1 #avi-ation)	range (iri1 #battery)	range (iri2 #battery)
iri1 #pow-erTypes				X	X	X	
iri2 #has-grid-power	X			X			X
iri1 #isGrid-Powered	X	X	X	X	X	X	

External Python library [47] is used to perform the FCA. While a number of programs exist for FCA [48], we chose in favour of this particular implementation due to its lightweights nature and ease of use that does not rely on graphical interface or user interaction. We use CSV as the input and output data format for the FCA because it is robust enough to transmit the relatively sizeable input data without considerable overhead.

The output of FCA is a set of concept pairs where each concept pair is mapped for identical (i.e., where IRI completely matches) and differing relation values.

5.2 Comparison

The comparison phase of the program handles the evaluation of each concept pair produced by FCA for their similarity. This entails transforming the results of FCA from CSV back to internal representation in Prolog, comparing differing relation values of a concept pair for semantic similarity, calculating the overall similarity value of a concept and finally unifying concept pairs that exceed the confidence threshold.

5.2.1 Handling of FCA Output

As mentioned in Section 5.1.4, FCA produces a set of concept pairs where identical and differing relation values are counted and identified. This output is written as CSV files and used in Prolog to generate pruned versions of the concepts specific to that concept pair. The pruning maintains only those relation values that differ within the concept pair and discards all those relation types where neither concept has a relation value.

5.2.2 Comparison of Relation Values

The similarity of a concept pair comprises of the similarity of individual relation value pairs. A concept can have any number of relation values for a particular relation type, so this is essentially a comparison of two lists for best matching values. As the relation types have been sorted in pre-processing and further pruned in FCA, lists of relation values can be compared recursively, regardless of their type.

The comparison is conducted separately for both directions. This means that the comparison of relation values from concept A to B is conducted separately from the comparison of relation values from concept B to A. Relation values from the source value list are compared in the order of their appearance to the entire target relation value list and are matched using the best first principle. If a source relation value no longer has any value in the target value list, it is assigned absence confidence value instead. The final similarity of a relation value pair is the average of both confidence values from both directions.

During the comparison of a specific relation value pair, it is checked whether both source and target relation value refer to concepts within the same ontology. If this is true, pre-value confidence of that concept pair is used instead of the lexical relation value pair. By default, a concept pair is given the pre-value confidence of 0 which is then reassigned once all the concept pairs have been compared and their similarity calculated. It is therefore necessary to run several iterations of the comparison to stabilize the pre-values of concepts pairs. If one of the elements a relation value pair does not refer to a concept, lexical similarity is used instead. Unlike pre-value confidence, lexical similarity is a static value which needs to be calculated only once.

5.2.3 Calculation of Lexical Similarity

Lexical similarity expresses how similar is a specific relation value pair using the methodology described in Section 4.3. This entails calculating similarity value using both semantic

and string matching as well as the string similarity of comments (if applicable) and providing a combined similarity value using the weights provided as input parameters.

String Matching

The implementation of string similarity is relatively straightforward, making use of native the *isub* library provided in SWI-Prolog [11]. When comparing relation values, the input for string matching are the atomic versions of the parsed word list. When comparing comments, it is the value of `rdfs:comment` in atomic form.

Semantic Matching

The WordNet-based semantic similarity is implemented using the `prologdb(5WN)` library [38] which permits querying the WordNet database directly in Prolog. Alternatively, it would be possible to query WordNet's web API hosted by Princeton University, but given the large amount of queries, we considered the direct and Prolog-native integration to be more preferable.

A total of 14 different query types are run against the Prolog database for possible semantic relations between words or phrases. The number of identified semantic relations are stored and the semantic similarity value calculated as described in Section 4.3.1.

One of the major drawback of the WordNet database in Prolog is that it accepts only exact case-sensitive matches. This means that for maximum effect, queries need run for both individual words in the relation value (obtained during the parsing of relation values in pre-processing) as well as a combination of words and a combination of capitalization. For example, given a relation value `#TheUnitedStatesOfAmerica`, an exact combination of words and capitalization is needed to match the phrase `United States of America` found in WordNet database.

In addition, relation value may include inflectional word forms or derivatives that need to be transformed to their lemma (dictionary form) before they can be matched with a WordNet entry. Typical English inflectional forms include 3. person singular verb form (e.g., *takes* $\rightarrow$ *take*, *has* $\rightarrow$ *have*), various past tense verb forms (*reached* $\rightarrow$ *reach*, *spoken* $\rightarrow$ *speak*) or plural noun forms (*queries* $\rightarrow$ *query*, *oxen* $\rightarrow$ *ox*). In addition, certain English words may simultaneously be both lemmas and inflectional forms depending on the meaning or grammatical context (e.g., *found* is both a dictionary form of a verb as well as past participle for *find*).

For maximum effect, if a WordNet query yielded no result (meaning that there was no

entry for that particular combination of characters), we attempted to reduce the input word to the lemma and repeat the query until either a WordNet provided a result or there were no additional lemmas for that word. We used an external Python module *Lemminflect* to determine all possible dictionary forms of a word [49]. Querying Lemminflect for the root form requires initiating the Python script as a new process for each query which is significantly resource-consuming in case of larger input data.

SWI-Prolog does come with an implementation of *Snowball* stemming algorithms [50] which are used to identify the root of a word. Using Snowball instead of Lemminflect would significantly speed up querying WordNet. However, as this is a purely mechanical approach that does not take into account whether the input is actually a valid inflected form, using Snowball deemed unsuitable for our purposes.

To minimize resource-heavy queries, both Lemminflect and WordNet are queried only once for each input and the results are stored in Prolog's internal memory for reuse.

Additional Heuristics

In order to improve the accuracy of lexical similarity calculation, some additional constraints have been introduced. These constraints are purely intuitive based on the patterns exhibited in false positives. As such we can see these as heuristic techniques employed on grammatic context of where words occur. Two such constraints have been developed.

The first constraint excludes a predefined set of English prepositions and auxiliary verbs from the similarity calculation of a relation value. However, if such preposition or auxiliary verb does exist in the source relation value, it is required that it also exists in the target relation value. If this does not apply, both string and semantic similarity default to 0. This constraint was implemented on the grounds that while auxiliary verbs and prepositions are common in relation values, they do not have semantic value of their own, but rather describe the grammatic context of words with semantic value. Implementation of this constraint removes the occurrence of false positives between relation values such as `#isPart`, `#isPartOf` and `#hasPart` which are similar in their construction, but are clearly different in meaning. Of course, it is possible to express the same meaning in many different ways (for example, we can intuitively surmise that `#isPartOf` is the same as `#belongsTo`). However, given the relatively formulaic approach to how relation values tend to be expressed within a single ontology, adding such structural constraint does benefit the overall accuracy of the method.

The second constraint excludes that the name of a concepts of types `rdfs:Class`,

`owl:Class` and `owl:NamedIndividual` start with a verb. As concepts of these types always refer to an object, we can assume that the first word in the concept's name is never a verb. This in return enables to exclude those WordNet queries that occur between verbs in that position. For example, given two concepts of type `owl:Class` named `#Head` (referring to the body part) and `#Lead` (referring to the type of metal), this constraint excludes the false positive match as a verb with the meaning "to travel in front of" or "be in charge of".

5.2.4 Calculation of Concept Pair Similarity

As described in Section 4.2, similarity of a concept pair is the calculated average of the confidence of all related relation value pairs. During the comparison of relation values, the similarity of each relation value pair is stored to a temporary list. Once all relation value pairs have been compared, the similarity of the concept pair can be calculated by dividing the sum of all values in the temporary list with the length of the list.

5.2.5 Unification of Similar Concepts

Once the similarity of all concept pairs have been calculated, similar concepts can be matched by iterating through all the concept pairs and choosing the concept pair that has the highest similarity value for a particular concept.

We use a top-down approach where the first iteration through the concept pairs matches concept pairs with the confidence value 1, then decrementing this by 0.01 for the next iteration (0.99, 0.98 and so on). If a concept pair is matched all other concept pairs with either of those concepts are also removed thus ensuring that no concept is matched multiple times. Once the confidence value falls below the minimal confidence threshold, all remaining concept pairs are discarded and any concept that was not yet matched is considered a mismatch.

Best results are achieved when the unification of similar concepts occurs after several iterations of concept pairs being compared. This is due to the fact that the pre-value of a concept pair is always based on the previous comparison iteration (defaulting to lexical similarity on the first iteration). Since the pre-value of a concept pair is being used as the similarity measure of a relation value pair where both values refer to such concepts that are defined in either of the input ontologies, the similarity of these relation value pairs changes from iteration to iteration. Due to time constraints, this behaviour was not extensively observed, but the intuition is that the variance in the similarity is greater during the first

few iterations and then stabilizes by the third or fourth iteration of the comparison process.

5.3 Specialization and Output Writing

As described in Chapter 4, specialization operation further refines the matched concepts, generating, if needed, new sub-concepts for relation values that diverged (i.e., the relation value pair fell below the minimal confidence threshold). In this phase of the program, confidence of individual relation value pairs is mapped and output relation values with necessary modifications are generated. Finally, RDF triplets are generated and written into an output file.

5.3.1 Mapping of Concept Pair Relations

In this phase, every relation value pair of every matched concept pair is mapped to a specific confidence value. This confidence value can be either the pre-value of a concept pair when both relation values refer to a concept or lexical similarity value or value 1 in case of an identical IRI. In addition, this phase is used to filter out values from ignored relations that are not desired in output ontology and transform various compound values, such as comments, into a more uniform representation.

5.3.2 Generation of Output Relation Values

Once relation value pairs of matched concept pairs have been mapped for their confidence, data for the output ontology can be generated.

The general format for output relation values is `ConceptList - RelationType - RelationValue` which can be easily transformed into RDF triplets. When generating new sub-concepts from diverging relation values of a matched concept pair, naming pattern `ConceptName-from-OntologyName` is used to distinguish it from the parent concept and an additional ancestral relation (either `rdfs:subClassOf` or `rdfs:subPropertyOf`) referencing to the parent concept is generated. The same naming pattern is also used for mismatched concepts.

As the values in the relation value pair are typically not identical, the following principles are used to modify relation values:

- if neither or both of the relation values refer to a defined concept, the first relation value is used;

- if only one relation value refers to a defined concept, that relation value is used;
- if a relation value refers to a defined concept that is mismatched, the same naming pattern is used as for the concept name;
- if a relation value refers to an RDF list resource, it is replaced with the contents of that particular RDF list represented as a Prolog list;
- if a relation value is a compound value, it is cast into an atomic form.

Exception is made for anonymous concept when handling diverging relation values. As anonymous concepts cannot be further divided into sub-concepts, a single diverging relation value will result the entire anonymous concept to be considered diverging with regards to the relation(s) that refer to them. This applies to any type of anonymous concepts: property restrictions, anonymous class descriptions, Axioms as well as RDF lists used outside anonymous class descriptions.

To illustrate this, let us assume that the following property restriction pair was successfully matched as the overall similarity of the pair that exceeds the minimal confidence threshold. However, in the case of one relation value pair (`owl:someValuesFrom`), the similarity falls below the minimal confidence threshold. As such, the entire property restriction pair will be treated as diverged. Consequently, if a diverging anonymous concept pair is nested in additional anonymous concept pairs, these will also be treated as diverged.

Example A

```
<owl:Restriction>
    <owl:onProperty
        rdf:resource="#common" />
    <owl:someValuesFrom
        rdf:resource="#Vechicle" />
</owl:Restriction>
```

Example B

```
<owl:Restriction>
    <owl:onProperty
        rdf:resource="#common" />
    <owl:someValuesFrom
        rdf:resource="#Carriage" />
</owl:Restriction>
```

5.3.3 Writing RDF Triplets

In this phase the generated output relation values are transformed into output file consisting of RDF triples. This also involves generating new RDF lists from relation values that are Prolog lists and transforming any compound values in atomic form into generic literals.

5.4 Testing

A total of 47 automatic tests are implemented using SWI-Prolog's native testing library. The majority of these tests follow the same general structure that validates the functionality of the code by assessing the correctness of the following data: 1) matched concept pairs and their confidence, 2) mismatched concepts, 3) concept pairs generated by FCA and 4) generated output relation values. In that sense, we can see these more like integration tests than typical unit tests.

The input data for tests is manually composed to consider the variety of different syntactic structures possible in RDF/OWL. Unfortunately it is not possible to provide a definite value for code coverage: while SWI-Prolog does provide a library to analyse code coverage, the current implementation only provides code coverage of a single run, as each test requires the program to be restarted. Running the whole test suit comes with a considerable time cost in comparison to typical unit tests.

5.5 Other Topics

Multi-Threading

The program employs dynamic multi-threading as a separate thread is created for each `rdf:type` value present in the input ontologies during the FCA and concept comparison phase. Since multi-threading caused problems with the SWI-Prolog testing library, the program can be run in a single-threaded mode, but this should be limited for running tests, as it seems to generate sub-optimal results during normal operations.

Bash Scripts

In order to facilitate repeatedly running the program, two bash scripts have been created for a more streamlined initialization: one for running the program with on two input ontologies and the other for running tests.

Implemented Input Parameters

The following input parameters have been implemented for added customizability:

- minimal confidence threshold;
- confidence value of a synonym;
- confidence value of an absence;
- weights for semantic and string similarity;

- weights for lexical and comment similarity (if comments exist in concept pair);
- boolean whether multi-threaded or single-threaded mode be used;
- boolean whether output ontology should be generated;
- paths for input ontologies.

Performance and profiling

It should be noted that the current implementation does come with a considerable computational expense and lengthier comparisons may potentially last for hours or even more than a day. A thorough performance assessment falls out of the scope of this thesis, but profiling does reveal that the main bottlenecks lie in handling dynamic facts that contain large amounts of data. To illustrate that, we profiled three different comparisons: a small comparison with few matches (PMK-DOLCE), a large comparison with few matches (PMK-SOMA) and a medium-sized comparison with 100% matches (OAEI test 101). Note that due to the limitations of the SWI-Prolog profiler, single-thread mode was used and that results vary somewhat even under identical conditions. The below figures should therefore not be taken for exact values, but rather as a general estimate.

Table 2. Key CPU Time per Comparison

Predicate name	PMK-DOLCE	PMK-SOMA	OAEI 101
config:foundConcepts/5	25.6%	23.2%	0.3%
garbage_collect_atoms/0	14.6%	4.8%	1.0%
config:uncheckedConcepts/5	11.3%	33.7%	4.6%
config:pairedValues/3	10.1%	17.4%	42.5%
retractall/1	7.3%	6.3%	20.5%
runtime (seconds)	357	15473	65142

Predicate `garbage_collect_atoms/0` and `retractall/1` are system predicates. The first is used to reclaim unused atoms, whereas the second to delete all data found in a dynamic fact. The other predicates all refer to dynamic facts that contain large amounts of data and are also frequently queried. The dynamic fact `config:foundConcepts/5` holds data on the concepts that were identified from input ontologies, `config:uncheckedConcepts/5` holds data on which concepts have not yet been compared during a single iteration of the comparison phase and `config:pairedValues/3` holds data on which relation values have been paired for every possible concept pair. The most notable variation across comparisons is with `config:pairedValues/3` which took more than twice the share during OAEI 101 than during other comparisons. This shows that perhaps one of the key factors determining runtime is how similar the input ontologies are.

6. Results

In this chapter, we discuss the results of the experiments run on the OAEI data set and the three robotics-related ontologies. For comparability, identical input parameter values were used for both experiments. The choice of input parameter values aimed to achieve the best balance between minimizing false positives and permitting structural flexibility.

- minimal confidence threshold = 0.7;
- confidence value of a synonym = 0.9;
- confidence value of other semantic relation = 0.675;
- confidence value of an absence = 0.5;
- semantic similarity weight = 0.5;
- string similarity weight = 0.5;
- multi-threaded mode was used.

6.1 Results of OAEI Test Set

We ran the program on 42 base tests and 4 additional tests which can be found in Table 3. The summed average of correct matches was 60% across the 42 base tests, including 17 tests 100% match and 8 tests with 0% matches. Better results were achieved in tests that simplified the original ontology in some way: e.g., simplification of the OWL language (tests 102, 103, 104), removal of subclass assertions (test 221), individuals (test 224), property restrictions (test 225), properties (test 228), general simplification of hierarchy (test 222). Test changing concept names to semantically irrelevant strings was quite successful with 93% accuracy (test 201). However, when identical comments were additionally removed, accuracy dropped considerably to 23% (test 202). Marginal loss of accuracy also occurred when the hierarchy was increased with intermediate classes (test 223, 97%). From the perspective of semantic and string matching, test 209 is perhaps the most relevant, as this introduces the use of synonyms while removing the comments. The accuracy in this test was 58% which is slightly below the summed average.

We can distinguish two types weaknesses of the method in the matching process. The first type is the incorrect categorization of a concept as mismatched due to the concept not having any available pair over the minimal confidence threshold. The second type is the occurrence of false positives due to matching incorrect concept in relation to ground truth. Lowering the minimal confidence threshold would reduce occurrence of mistakes of the

first type, but increase the occurrence of mistakes of the second type. With the current input parameters the number of false positives remained relatively low in all tests with the highest share being 8% for test 209.

Table 3. Results from OAEI Test Data

Test	C/T	False pos	%	Comment
101	97/97	0	100%	Compares the ontology to itself.
102	0/0	0	100%	Compares the ontology to a totally irrelevant one.
103	96/97	0	99%	Compares the ontology with its generalisation in OWL Lite
104	97/97	0	100%	Compares the ontology with its restriction in OWL Lite (where unavailable constraints have been discarded).
201	90/97	2	93%	Names and labels replaced by a random string.
202	22/97	0	23%	Based on 201, comments have been removed.
203	97/97	0	100%	Labels and comments have been removed.
204	97/97	0	100%	Names and labels are written in a variety of different naming conventions.
205	93/96*	2	97%	Names and labels are replaced by synonyms.
208	96/97	0	99%	Based on 204, comments have been removed.
209	56/96	8	58%	Based on 205, comments have been removed.
221	97/97	0	100%	No class hierarchy: all subclass assertions to named classes are removed.
222	93/93	0	100%	Reduced class hierarchy.
223	94/97	3	97%	Extended class hierarchy: numerous intermediate classes are introduced.
224	97/97	0	100%	All individuals have been removed.
225	97/97	0	100%	All local restrictions on properties have been removed.
228	33/33	0	100%	Properties and relations between objects have been completely removed.
232	97/97	0	100%	Combines 221 and 224
233	33/33	0	100%	Combines 221 and 228
236	33/33	0	100%	Combines 224 and 228
237	93/93	1	100%	Combines 222 and 224
238	94/97	3	97%	Combines 223 and 228
239	29/29	0	100%	Combines 222 and 228
240	9/33	2	27%	Combines 223 and 228

Test	C/T	False pos	%	Comment
241	33/33	0	100%	Combines 232, 233 and 236
246	29/29	0	100%	Combines 236, 237 and 239
247	10/33	2	30%	Combines 236, 238 and 240
248	6/97	0	6%	Combines 202 and 221
249	24/97	0	25%	Combines 202 and 224
250	0/33	0	0%	Combines 202 and 228
251	12/93	1	13%	Combines 202 and 222
252	3/97	3	3%	Combines 202 and 223
253	6/97	0	6%	Combines 202, 221 and 224
254	0/33	0	0%	Combines 202, 221, 225
257	0/33	0	0%	Combines 202, 224 and 228
258	12/93	1	13%	Combines 202, 222 and 224
259	13/97	3	13%	Combines 202, 223 and 224
260	0/29	0	0%	Combines 202, 222 and 228
261	0/33	0	0%	Combines 202, 223 and 228
262	0/33	2	0%	Combines 202, 221, 224 and 228
265	0/29	0	0%	Combines 202, 222, 224 and 225
266	0/33	0	0%	Combines 202, 223, 224 and 225
301	9/52	1	17%	Real ontology: BibTeX/MIT
302	0/35	0	0%	Real ontology: BibTeX/UMBC
303	0/40	0	0%	Real ontology: Karlsruhe
304	44/73	1	60%	Real ontology: INRIA

6.2 Results of Robotics-Related Ontologies

Unlike the OAEI data set, there is no quantifiable ground truth for these experiments. In order to have a better understanding of the matched concept pairs, we look at the following meta-properties of the concept pairs:

- the number of explicit relations of a concept;
- the number of inherited relations of a concept;
- the number of relations in the concept pair that were considered similar in the specialization process;
- the matching confidence of the concept pair.

The number of relations, both explicit and inherited, provides general characterization of the

matched concepts: a considerable difference in the number of relations indicates asymmetry in their structural definitions. Likewise, the number of similar relations indicates to what extent did the specialization process consolidate the knowledge from each concepts. It should be noted, that at the very minimum, each matched concept pair has at least two similar relations (type and name).

6.2.1 Comparison of PMK and DOLCE

As it was expected due to their different nature, the comparison of PMK and DOLCE yielded very small overlap. There is only a single matched concept pair and 195 concepts were mismatched. In this case, the names of the matching concepts are identical in both ontologies, the level of detail of the concepts differs considerably. Based on the identical lexical content, we can assess that this is a valid match.

Table 4. Matched Concepts from PMK-DOLCE Comparison

PMK	Relations		DOLCE	Relations			
Name	Explicit	Inherited	Name	Explicit	Inherited	Similar	Confidence
#Region	3	0	#region	4	5	2	0.72

6.2.2 Comparison of PMK and SOMA

The comparison between PMK and SOMA produced 12 matched concept pairs and 905 mismatched concepts. In several cases, the concept name is an identical match. However, we there are also matches that could be seen as a match between a specification and a general concept (e.g., `#hasSensingComponent-#hasComponent`, `#ActionClass-#Action`) and false positives (e.g., `#PhysicalEnvironment-#PhysicalAgent` and `#WspaceClass-#SpaceRegion`). The number of relations is relatively low in most of matched concepts, even though the concepts in SOMA generally have a larger number of relations.

Table 5. Matched Concepts from PMK-SOMA Comparison

PMK	Relations		SOMA	Relations			
Name	Explicit	Inherited	Name	Explicit	Inherited	Similar	Conf
#has-Sensing-Component	2	0	#has-Component	2	0	2	0.96
#is-RelatedTo	2	0	#is-RelatedTo-Concept	2	0	2	0.96

#Region	3	0	#Region	2	0	2	0.91
#Task	3	0	#Task	2	0	2	0.91
#Action-Class	2	0	#Action	3	2	2	0.875
#objectType	3	1	#Object	2	0	2	0.84
#Quality-Aggregation	3	0	#Quality	2	0	2	0.82
#Physical-Environment	3	0	#Physical-Agent	2	0	2	0.79
#Attributes	3	0	#Physical-Attribute	2	0	2	0.79
#Situation	3	0	#Situation-Transition	3	0	2	0.76
#Wspace-Class	2	0	#Space-Region	2	0	2	0.75
#Context-Reasoning-Class	2	0	#Reasoning	3	5	2	0.7

6.2.3 Comparison of DOLCE and SOMA

The comparison between DOLCE and SOMA produced 13 positive matches and 920 mismatched concepts. In most cases, the concept name is an identical match, although the number of relations typically differs. We can explain this contrast as the same concept being described in full detail in DOLCE, but being considerably abridged in SOMA. Exceptions to this observation are clear false positives `#particular-#Item` and `#physical-quality-#PhysicalAttribute`. With regards to the latter, SOMA ontology seemingly does have more accurate concept named `#PhysicalQuality` which is categorized as a mismatch instead.

Table 6. Matched Concepts from DOLCE-SOMA Comparison

DOLCE	Relations		SOMA	Relations			
Name	Explicit	Indirect	Name	Explicit	Indirect	Similar	Conf
#quality-space	4	0	#Quality	2	0	2	0.84
#overlaps	6	1	#overlaps	2	0	2	0.82

#has-quality	6	2	#hasQuality	2	0	2	0.82
#particular	2	0	#Item	3	2	2	0.82
#region	4	5	#Region	2	0	2	0.81
#event	3	9	#Event	2	0	2	0.79
#process	3	10	#Process	2	0	2	0.79
#time-interval	3	10	#Time-Interval	2	0	2	0.79
#physical-object	5	17	#Physical-Object	2	0	2	0.77
#space-region	5	13	#Space-Region	2	0	2	0.78
#has-quale	6	8	#hasQuale	6	2	4	0.74
#physical-quality	7	6	#Physical-Attribute	2	0	2	0.73
#state	3	10	#State	3	0	2	0.71

6.3 Discussion

As seen from the results of comparing PMK, DOLCE and SOMA ontologies, the number of matched concepts in all comparisons is remarkably low in relation to the total number of concepts in those ontologies. This could be either due to the inherent dissimilarity of the ontologies, the overly high accuracy of the matching process or both. Determining the ground truth regarding the similarity of semantically close concepts in large weakly related onotlogies is a challenge even for a human expert. This is not only because of the complexity of ontological structures, but also because of the variety of contexts the meaning of a concept pair can exhaustively be interpreted. As such, it is a challenge to determine the extent of semantically equivalent concepts that could have been matched but were not, as this is specific to each ontology and often even ground truth cannot be assessed with full certainty.

The comparison between DOLCE and SOMA provides some insight in this regard, as SOMA incorporates parts of DOLCE into their ontology. There are 15 concepts in total with identical names between DOLCE and SOMA. Nine of those concepts were matched correctly and one was matched incorrectly as discussed above. The remaining five concepts (*#Set, #Feature, #DependentPlace, #TimeInterval, #RelevantPart*) were not matched at all. Intuitively, this seems to be caused by differences in the relations these concepts have. It should also be noted that the DOLCE concepts that were incorporated into SOMA, were

re-defined within SOMA, meaning they are provided a new IRI as a SOMA concept. This considerably limits the potential usability of common IRI-s as background knowledge.

The number of relations of matched concepts provides another useful insight. Clearly, most matches occur between concepts with a low number of relations (often only the name and type of the concept) whereas concepts with a large number relations tend to have a lower similarity value in general. Conversely, having a match between concepts with a high number of relations could be seen as an additional validation that these concepts are indeed similar beyond simply having a similar name. One such example from our results is the match between `#has-quale` and `#hasQuale` from DOLCE and SOMA comparison. It is notable that not a single OWL class description exceeded the confidence threshold in matched concepts, further highlighting the inherent differences of ontologies not created for ontology merge benchmarking purposes.

7. Further Work and Summary

7.1 Conclusions

In this thesis, we have introduced the motivation, principles and methodology of a new algorithmic tool for ontology merging that relies on semantic and string matching combined with structure-based analysis and Prolog built-in syntactic unification mechanism together called weak unification.

We have provided several experimental results to validate the accuracy of the merging process. Experiments run on the OAEI data set reveal relatively strong results of tests focusing on structural variations and synonyms while being generally unsuccessful on tests that removed all semantically meaningful labels.

Experiments run on real life robotics ontologies produced a relatively small number of matched concepts. Given that, no ground truth could have been reliably established for this type of test. More comprehensive set of comparisons are needed to asses whether this is due to the dissimilarity of ontologies or the accuracy of the matching process. However, the concept pairs that were matched, are generally of good quality, typically linking concepts that could be considered either semantically equivalent or specification/generalization of one another. In the comparison between PMK and DOLCE ontologies, the single match is clearly between semantically equivalent concepts. In the comparison between PMK and SOMA ontologies, 2 matches out of 12 (16.6%) are between semantically equivalent concepts, 8 matches (66,6%) are between concepts that could be considered specialization-generalization of one another and 2 matches (16.6%) that are clear false positives. In the comparison between DOLCE and SOMA ontologies, 10 matches out of 13 (77%) are between concepts that are clearly equivalent, one match (8%) can be considered to be between specialization-generalization and 2 matches (15%) are clear false positives. As a future improvement of the method, a number of relations attributed to a matched concepts and other metadata can be used to better analyze the nature of a particular match.

7.2 Proposed Further Work

Several aspects of the tool can be improved to further enhance its accuracy, in addition to improving the heuristics of the matching process.

First and foremost, the best-first approach in relation value matching should be replaced with a more comprehensive search that always ensures the best result. The current best-first search is primarily a concern when the concept has many relation values of the same type and can easily produce sub-optimal results.

Secondly, it could be beneficial to permit a more flexible type comparison. The current implementation restricts concept matching to strictly by type. However, more flexibility could be achieved by permitting comparison of concepts between a sub-type and its generalization. This particularly pertinent in the context of OWL vocabulary that introduces a hierarchical system of property types (e.g., `owl:InverseFunctionalProperty`, `owl:TransitiveProperty` and `owl:SymmetricProperty` are all defined as sub-classes of `owl:ObjectProperty`).

Furthermore, the current tool works the best under the assumption that input ontologies have a common set of relations defined in a shared vocabulary (RDF, OWL, FOAF, etc.). Relation types that are declared within the context of an input onotology introduce an additional layer of heterogenity and need to be transformed to a more universal representation in order to be efficiently comparable. Some work was already done in this regard essentially by treating all such locally declared relation types and their values as tuples that that were compared under the same principles as anonymous concepts. However, results of this approach proved inconclusive.

With regards to the speed and efficiency of the tool, it has to be noted that depending on the size and structure of input ontologies, a single comparison can take considerable amount of time (in the experiments run as part of this thesis, sometimes more than a day). The most immediate solution to increasing the speed of individual comparisons is a aggressive pruning of irrelevant possible concept pairs on the FCA phase. However, it is easy to exclude valid matches and a well working technique for pruning in FCA was not developed at the current state.

From a broader perspective, it is also worth to consider re-structuring the entire project so that each phase of the program (pre-processing, comparison and specialization) is an isolated module embedded in Python. In such case, each phase would operate independently and any intermediary data they generate could be maintained, and manipulated if needed, in a Python wrapper thus improving the modularity and robustness of the program. As an added benefit, it would make it possible to implement any additional Python libraries in a more seamless manner.

References

[1] *Encyclopædia Britannica. Ontology*. https://www.britannica.com/topic/ontology-metaphysics. Accessed: 2024-01-19. 2024.

[2] Tom Gruber. *Ontology*. http://web.dfc.unibo.it/buzzetti/IUcorso2007-08/mdidattici/ontology-definition-2007.htm. Accessed: 2024-01-19. 2007.

[3] Pavel Euzenat Jérôme;Shvaiko. *Ontology Matching*. 2nd ed. 2013. Springer Berlin Heidelberg, 2013. ISBN: 9783642387203,9783642387210,2013952732,3642387209.

[4] Giancarlo Guizzardi et al. "UFO: Unified Foundational Ontology". In: *Applied Ontology* (Jan. 2022). DOI: 10.3233/AO-210256.

[5] *Suggested Upper Merged Ontology (SUMO)*. https://www.ontologyportal.org/. Accessed: 2024-02-29.

[6] Mario Maroun. "A Survey On Ontology Operations Techniques". In: 7 (Nov. 2021), pp. 7–28. DOI: 10.5281/zenodo.5716095.

[7] Namyoun Choi, Il-Yeol Song, and Hyoil Han. "A Survey on Ontology Mapping". In: *SIGMOD Record* 35 (Sept. 2006), pp. 34–41. DOI: 10.1145/1168092.1168097.

[8] AnHai Doan et al. "Ontology Matching: A Machine Learning Approach". In: *Handbook on Ontologies*. 2004. URL: https://api.semanticscholar.org/CorpusID:14213246.

[9] Princeton University. *About WordNet*. https://wordnet.princeton.edu/. Accessed: 2023-10-27. 2010.

[10] Rene Robin and G. Uma. "A Novel Algorithm for Fully Automated Ontology Merging Using Hybrid Strategy". In: *European Journal of Scientific Research* 47 (Nov. 2010), pp. 074–081.

[11] Giorgos Stoilos, Giorgos Stamou, and Stefanos Kollias. "A string metric for ontology alignment". In: *The Semantic Web–ISWC 2005: 4th International Semantic Web Conference, ISWC 2005, Galway, Ireland, November 6-10, 2005. Proceedings 4*. Springer. 2005, pp. 624–637.

[12] Hamed Karimi and Ali Kamandi. "Ontology alignment using inductive logic programming". In: Apr. 2018, pp. 118–127. DOI: 10.1109/ICWR.2018.8387247.

[13] Claudio Rocco, Elvis Hernandez-Perdomo, and Johnathan Mun. “Introduction to Formal Concept Analysis and Its Applications in Reliability Engineering”. In: *Reliability Engineering System Safety* 202 (May 2020), p. 107002. DOI: 10.1016/j.ress.2020.107002.

[14] Rudolf Wille. “RESTRUCTURING LATTICE THEORY: AN APPROACH BASED ON HIERARCHIES OF CONCEPTS”. In: *Formal Concept Analysis*. Ed. by Sébastien Ferré and Sebastian Rudolph. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 314–339. ISBN: 978-3-642-01815-2.

[15] Gerd Stumme and Alexander Maedche. “FCA-Merge: Bottom-up merging of ontologies”. In: (May 2001).

[16] “What is inductive logic programming?” In: *Foundations of Inductive Logic Programming*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1997, pp. 162–177. ISBN: 978-3-540-69049-8. DOI: 10.1007/3-540-62927-0_9. URL: https://doi.org/10.1007/3-540-62927-0_9.

[17] Sumaira Manzoor et al. “Ontology-Based Knowledge Representation in Robotic Systems: A Survey Oriented toward Applications”. In: *Applied Sciences* 11 (May 2021), p. 4324. DOI: 10.3390/app11104324.

[18] Joanna Isabelle Olszewska et al. “Ontology for autonomous robotics”. In: *2017 26th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*. 2017, pp. 189–194. DOI: 10.1109/ROMAN.2017.8172300.

[19] Edson Prestes, Sandro Fiorini, and Joel Carbonera. “Core Ontology for Robotics and Automation”. In: Sept. 2014.

[20] Stefano Borgo et al. *DOLCE: A Descriptive Ontology for Linguistic and Cognitive Engineering*. Aug. 2023.

[21] Gi Hyun Lim, Il Hong Suh, and Hyowon Suh. “Ontology-Based Unified Robot Knowledge for Service Robots in Indoor Environments”. In: *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on* 41 (June 2011), pp. 492–509. DOI: 10.1109/TSMCA.2010.2076404.

[22] Séverin Lemaignan et al. “ORO, a knowledge management platform for cognitive architectures in robotics”. In: Oct. 2010, pp. 3548–3553. DOI: 10.1109/IROS.2010.5649547.

[23] Michael Beetz et al. “Know Rob 2.0 — A 2nd Generation Knowledge Processing Framework for Cognition-Enabled Robotic Agents”. In: May 2018, pp. 512–519. DOI: 10.1109/ICRA.2018.8460964.

[24] Daniel Beßler et al. *Foundations of the Socio-physical Model of Activities (SOMA) for Autonomous Robotic Agents*. 2020. arXiv: 2011.11972 [cs.RO].

[25] Paulo J.S. Gonçalves and Pedro M.B. Torres. “Knowledge representation applied to robotic orthopedic surgery”. In: *Robotics and Computer-Integrated Manufacturing* 33 (2015). Special Issue on Knowledge Driven Robotics and Manufacturing, pp. 90–99. ISSN: 0736-5845. DOI: https://doi.org/10.1016/j.rcim.2014.08.014. URL: https://www.sciencedirect.com/science/article/pii/S0736584514000751.

[26] Barbara Bruno et al. “The CARESSES EU-Japan Project: Making Assistive Robots Culturally Competent”. In: *Ambient Assisted Living*. Ed. by Niccolò Casiddu et al. Cham: Springer International Publishing, 2019, pp. 151–169. ISBN: 978-3-030-04672-9.

[27] Mohammed Diab et al. “PMK-A Knowledge Processing Framework for Autonomous Robotics Perception and Manipulation”. In: *Sensors* 19 (Mar. 2019). DOI: 10.3390/s19051166.

[28] Xiaolei Sun, Yu Zhang, and Jing Chen. “High-Level Smart Decision Making of a Robot Based on Ontology in a Search and Rescue Scenario”. In: *Future Internet* 11 (Oct. 2019), p. 230. DOI: 10.3390/fi11110230.

[29] Patrizia Ribino et al. “A Humanoid Social Robot Based Approach for Indoor Environment Quality Monitoring and Well-Being Improvement”. In: *International Journal of Social Robotics* 13 (Apr. 2021). DOI: 10.1007/s12369-020-00638-9.

[30] Lyazid Sabri et al. “An integrated semantic framework for designing context-aware Internet of Robotic Things systems”. In: *Integrated Computer-Aided Engineering* 25 (Dec. 2017), pp. 1–20. DOI: 10.3233/ICA-170559.

[31] Ahmed R. Sadik and Bodo Urban. “An Ontology-Based Approach to Enable Knowledge Representation and Reasoning in Worker-Cobot Agile Manufacturing”. In: *Future Internet* 9 (2017), p. 90. URL: https://api.semanticscholar.org/CorpusID:31034167.

[32] Zeid Kootbally et al. “Implementation of an Ontology-Based Approach to Enable Agility in Kit Building Applications”. In: *International Journal of Semantic Computing* 12 (Mar. 2018), pp. 5–24. DOI: 10.1142/S1793351X18400019.

[33] Mark A. Musen. “The protégé project: a look back and a look forward”. In: *AI Matters* 1.4 (June 2015), pp. 4–12. DOI: 10.1145/2757001.2757003. URL: https://doi.org/10.1145/2757001.2757003.

[34] Luca Buoncompagni, Alessio Capitanelli, and Fulvio Mastrogiovanni. “A ROS multi-ontology references services: OWL reasoners and application prototyping issues”. In: (June 2017).

[35] European Master on Advanced Robotics Lab in Genoa. *ARMOR*. https://github.com/EmaroLab/armor. Accessed: 2024-02-29.

[36] Franz Baader et al. "Unification Theory". In: Dec. 2001, pp. 445–533. ISBN: 9780444508133. DOI: 10.1016/B978-044450813-3/50010-2.

[37] Andrew Cropper and Stephen H. Muggleton. "Logical Minimisation of Meta-Rules Within Meta-Interpretive Learning". In: *Inductive Logic Programming*. Ed. by Jesse Davis and Jan Ramon. Cham: Springer International Publishing, 2015, pp. 62–75. ISBN: 978-3-319-23708-4.

[38] Princeton University. *Format of Prolog-loadable WordNet files*. https://wordnet.princeton.edu/documentation/prologdb5wn. Accessed: 2024-03-07. 2024.

[39] Kristin H. Huseby. *How to improve the performance of a machine learning model with post processing employing Levenshtein distance*. https://towardsdatascience.com/how-to-improve-the-performance-of-a-machine-learning-model-with-post-processing-employing-b8559d2d670a. Accessed: 2024-03-09. 2020.

[40] Ben Chamblee. *What is Cosine Similarity? How to Compare Text and Images in Python*. https://towardsdatascience.com/what-is-cosine-similarity-how-to-compare-text-and-images-in-python-d2bb6e411ef0. Accessed: 2024-03-09. 2022.

[41] Ontology Alignment Evaluation Initiative. *Benchmark test*. https://oaei.ontologymatching.org/2016/benchmarks/index.html. Accessed: 2023-10-27. 2016.

[42] Ontology Alignment Evaluation Initiative. *Ontology Alignment Evaluation Initiative - Test library*. https://oaei.ontologymatching.org/2016/benchmarks/index.html. Accessed: 2023-04-15.

[43] Mohammed Diab. *PMK*. https://github.com/MohammedDiab1/PMK. Accessed: 2024-03-15. n.d.

[44] IAI University Bremen. *SOMA*. https://ease-crc.github.io/soma/. Accessed: 2024-03-15. n.d.

[45] Aldo Gangemi. *DOLCE-Lite*. https://github.com/iddi/sofia/blob/master/eu.sofia.adk.common/ontologies/foundational/DOLCE-Lite.owl/. Accessed: 2024-04-05. n.d.

[46] Jan Wielemaker. *SWI-Prolog Semantic Web Library 3.0*. https://www.swi-prolog.org/. Accessed: 2023-03-15.

[47] *Concepts. A simple Python implementation of Formal Concept Analysis*. https://concepts.readthedocs.io/en/stable/index.html. Accessed: 2023-10-27. 2023.

[48] *FCA Software*. https://upriss.github.io/fca/fcasoftware.html. Accessed: 2024-04-12. 2024.

[49] Brad Jascob. *LemmInflect*. https://github.com/bjascob/LemmInflect. Accessed: 2024-04-18. n.d.

[50] Martin Porter et al. *Snowball*. https://snowballstem.org/. Accessed: 2024-04-18. n.d.

[51] W3C. *RDF Schema 1.1*. https://www.w3.org/TR/rdf11-schema/. Accessed: 2024-01-31. 2023.

[52] Dan Brickley and Libby Miller. *RDF Schema 1.1*. http://xmlns.com/foaf/spec/. Accessed: 2024-01-31. 2014.

[53] Vladislav Ogorodnik. *Veebiliides relatsiooniliste andmete teisendamiseks RDF kujule*. https://www.w3.org/TR/rdf11-primer/. 2021.

[54] W3C. *OWL 2 Web Ontology Language Primer (Second Edition)*. https://www.w3.org/TR/owl2-primer/. Accessed: 2024-02-15. 2012.

[55] W3C. *OWL Web Ontology Language Overview*. https://www.w3.org/TR/owl-features/. Accessed: 2024-02-09. 2004.

[56] W3C. *OWL 2 Web Ontology Language Document Overview (Second Edition)*. https://www.w3.org/TR/owl2-overview/. Accessed: 2024-02-09. 2012.

[57] W3C. *OWL Web Ontology Language Reference*. https://www.w3.org/TR/owl-ref/. Accessed: 2024-02-09. 2004.

[58] Radim Řehůřek. *Gensim*. https://radimrehurek.com/gensim/index.html. Accessed: 2024-05-03. 2022.

[59] *Gensim-data*. https://github.com/piskvorky/gensim-data. Accessed: 2024-05-03. 2018.

Appendix 1 – Non-Exclusive License for Reproduction and Publication of a Graduation Thesis[1]

I Norman Kuusik

1. Grant Tallinn University of Technology free licence (non-exclusive licence) for my thesis "Ontology Merging in Prolog Using the Principle of Soft Unification", supervised by Jüri Vain
   1.1. to be reproduced for the purposes of preservation and electronic publication of the graduation thesis, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright;
   1.2. to be published via the web of Tallinn University of Technology, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright.
2. I am aware that the author also retains the rights specified in clause 1 of the non-exclusive licence.
3. I confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights, the rights arising from the Personal Data Protection Act or rights arising from other legislation.

25.05.2024

[1]The non-exclusive licence is not valid during the validity of access restriction indicated in the student's application for restriction on access to the graduation thesis that has been signed by the school's dean, except in case of the university's right to reproduce the thesis for preservation purposes only. If a graduation thesis is based on the joint creative activity of two or more persons and the co-author(s) has/have not granted, by the set deadline, the student defending his/her graduation thesis consent to reproduce and publish the graduation thesis in compliance with clauses 1.1 and 1.2 of the non-exclusive licence, the non-exclusive license shall not be valid for the period.

Appendix 2 – RDF/OWL Framework

This appendix covers the RDF and OWL frameworks that are commonly used together for representing ontologies and how it is interpreted in the context of this thesis.

A. RDF

RDF stands for *Resource Description Framework* and is a standard developed by World Wide Web Consortium (W3C) for expressing information about resources (e.g. information on documents, people, physical objects or abstract concepts). The key motivation for RDF is to express information on the Web so that it could be seamlessly processed and exchanged between applications without loss of meaning.

A.1 Key RDF Concepts

RDF Triples

At the core of RDF is principle of semantic triples expressing some type of relationship between two resources. This triple consists of a *subject*, *predicate* and *object* where subject and object are the resources concerned and the predicate is the relationship between them.

The following pseudo-code example on RDF triples can be found in W3C website which well exemplifies the variety of information that an RDF triplet can hold.

```
<Bob> <is a> <person>.
<Bob> <is a friend of> <Alice>.
<Bob> <is born on> <the 4th of July 1990>.
<Bob> <is interested in> <the Mona Lisa>.
<the Mona Lisa> <was created by> <Leonardo da Vinci>.
<the video 'La Joconde a Washington'> <is about>
    <the Mona Lisa>.
```

International Resource Identifier

To differentiate between resources with identical names, a unique identifier named *International Resource Identifier* (IRI) is used, similar to the URL of a web address. To expand on the previous example, the subject *Bob* refers to two different resources whereas the object *person* is the same resource in the following example.

```
<http://unique1.org/Bob> <is a>
    <http://common.org/person>.
<http://unique2.org/Bob/Bob> <is a>
    <http://common.org/person>.
```

It is also important to note that RDF itself does not impose any uniform standard for the format of the IRI, but organisations or tools using RDF usually have their own naming conventions. This also has practical implications in the context of this project as a semantic analysis of any resource assumes it is correctly extracted from the IRI.

Literals

Literals are any type of value that is not a resource themselves (i.e, they do not have an IRI). They are associated with a datatype that enables correct parsing and interpretation of of that value. In the SWI-Prolog RDF parser, literals are encapsulated as a tuple `literal(...)` and are handled as a compound datatype.

Blank nodes

Blank nodes are RDF resources not have any value (be it an IRI or a literal) at all. These typically occur when the value of a relation is nested inside some additional resource, most notably `owl:Restriction` that is discussed in Chapter B. In the SWI-Prolog RDF parser blank nodes are provided a generated value that can be used to refer to the blank node.

A.2 RDF vocabulary

RDF vocabulary refers to a collection of pre-defined resources. Phrased in another way, these are building blocks for describing the information about other resources. Difference is made between *classes* that classify a resource and *properties* that describe the resource. More specifically, *property* is a relation between the resource and some other resources [51]. Additionally *classes* themselves are defined through a specific property called the *type* property (`rdf:type`). In other words, a *class* is an instance of the *type* property.

The following table outlines some of the more common classes and properties.

Table 7. Common RDF Classes and Properties

classes	comment	properties	comment
rdfs:Resource	All other classes are subclasses of this class	rdfs:range	States that the values of a property are instances of one or more classes
rdfs:Class	Resources that are RDF classes	rdfs:domain	States that any resource that has a given property is an instance of one or more classes
rdfs:Datatype	Class of RDF datatypes	rdf:type	States that a resource is an instance of a class
rdfs:Property	Class of RDF properties	rdfs:subClassOf	States that all the instances of one class are instances of another
rdf:List	Class of RDF Lists	rdfs:subPropertyOf	States that all resources related by one property are also related by another
		rdfs:comment	Used to provide a human-readable description of a resource
		rdfs:label	Used to provide a human-readable version of a resource's name

Of these resources, `rdf:List` deserves some additional explanation as it often requires different handling than all other classes. `rdf:List` is essentially an entry in a linked list with two properties: `rdf:first` that holds the value of the element and `rdf:rest` that references to the next entry in the linked list (itself an `rdf:List` type resource). Value of `rdf:first` can be any class. In practical terms this means that values of `rdf:List` relation `rdf:first` need to be 'lifted' out of the `rdf:List` element and be directly linked to the overlying property. As the order of elements does not matter for our purposes, the `rdf:List` elements can then be discarded.

A number of other vocabularies exist in addition to RDF vocabulary and can be used in combination with one another. In the scope of this work, the most important of those vocabularies is that of OWL (*Ontology Web Language)* and will be discussed in greater length. In the above examples, the source of the vocabulary is indicated by the prefixes *rdf/rdfs* which both refer to RDF. In this notation, a resource from OWL vocabulary would

be prefixed with *owl* (e.g. `owl:Class`, `owl:ObjectProperty`) and similarly from any other vocabulary (for example, `foaf:Person` would be a resource from a more specific vocabulary called FOAF (*Friend of a Friend*) [52].

The definitions or explanations provided to resources in RDF and OWL vocabularies have an important impact to how the similarity of two concepts is measured. For instance, properties `rdfs:subClassOf` and `rdfs:subPropertyOf` express, unsurprisingly, child-parent relationship which is interpreted such that each relation existing for a parent concept should also be considered when measuring the similarity of the corresponding child concept. A more in-depth description can be found in the methodolgy chapter.

It is also worth noting that the project is primarily focused on comparing concepts which are instances of `rdfs:Class`, `rdf:Property` and `rdfs:Datatype` or some of their sub-types described in the OWL vocabulary. Depending on the composition and scope of the ontology, it is possible that a concept, an instance of some RDF class, is defined within the ontology and is then used to define additional concepts using the `rdf:type` property. Using the informal example from above, the statement *Bob is a person* could be expressed using two resources: first that there exists an instance of `rdfs:Class` called *Person* and then that there exists an instance of *Person* called *Bob*. In the methodology chapter such occurrences are referred to as either *custom types* or *custom relations*.

A.3 RDF Data Formats

In lieu of the pseudo-code used in previous examples, a number of data formats have been developed to represent RDF resources. Common formats listed in the W3C website are 1) Turtle family of RDF languages, 2) JSON-LD which uses JSON-based RDF syntax, 3) RDFa for HTML and XML embedding and 4) RDF/XML with XML syntax for RDF. We will not be going into the technical description of these formats, but a more in-depth analysis, including possible advantages or disadvantages of each format, has been covered by Vladislav Ogorodnik in [53].

In the implementation, we limit input and output of ontology manipulation to the RDF/XML due to the fact that this is the most common of the available options. Ultimately, the choice of formats can be expanded relatively easily, as the tools used to parse RDF graphs into Prolog internal form are also capable of handling other data formats. This would, however, require extensive testing of alternative or mixed data formats in the pipeline and remains out of scope for this project.

In the following, we reproduce the statements *Bob is a person*, *Bob is a friend of Alice* and

Bob is born on the 4th of July 1990 in the RDF/XML format.

```
<rdf:Class rdf:about="http://unique1.org#Person"/>

<rdf:Property rdf:about="http://unique1.org#isFriendOf">
    <rdfs:domain rdf:resource="http://unique1.org#Person"/>
    <rdfs:range rdf:resource="http://unique1.org#Person"/>
</rdf:Property>

<rdf:Property rdf:about="http://unique1.org#isBornOnDay">
    <rdfs:domain rdf:resource="http://unique1.org#Person"/>
    <rdfs:range>
        <rdfs:Datatype
            rdf:about="http://www.w3.org/2001/XMLSchema#integer"/>
    </rdfs:range>
</rdf:Property>

<rdf:Property rdf:about="http://unique1.org#isBornOnMonth">
    <rdfs:domain rdf:resource="http://unique1.org#Person"/>
    <rdfs:range>
        <rdfs:Datatype
            rdf:about="http://www.w3.org/2001/XMLSchema#string"/>
    </rdfs:range>
</rdf:Property>

<rdf:Property rdf:about="http://unique1.org#isBornOnYear">
    <rdfs:domain rdf:resource="http://unique1.org#Person"/>
    <rdfs:range>
        <rdfs:Datatype
            rdf:about="http://www.w3.org/2001/XMLSchema#integer"/>
    </rdfs:range>
</rdf:Property>

<Person rdf:about="Alice"/>

<Person rdf:about="Bob">
    <isFriendOf>Alice</isFriendOf>
    <isBornOnDay>4</isBornOnDay>
    <isBornOnMonth>July</isBornOnMonth>
    <isBornOnYear>1990</isBornOnYear>
```

```
</Person>
```

Looking at the example, it is clear that formally expressing even simple knowledge quickly grows verbose and hard to follow with limited vocabulary. In addition, it is not possible express certain notions that are semantically relevant (such as exactly expressing what is a valid date of birth or not). To mitigate this, we can use the more expressive vocabulary of OWL.

B. OWL

OWL stands for Web Ontology Language and is a formal language developed by W3C. It is built upon the structure described in RDF and the ontologies described in OWL are typically exchanged in RDF documents [54]. As such, the data formats described for RDF also apply for OWL and we will continue using RDF/XML format in our examples.

Essentially OWL expands the formal vocabulary of RDF and specifically geared towards expressing ontologies. Several sublanguages of OWL exist, respectively *OWL Lite*, *OWL DL* (Description Logic) and *OWL Full*, each with increased degree of expressiveness [55]. In addition, there exists an updated and backwards-compatible version of the language, informally called OWL 2, that further expands the expressiveness and simplifies the syntax [56].

We will not attempt to provide a comprehensive overview of all the features and differences between versions of OWL. Rather, this chapter will introduce some key vocabulary elements that are common in actual ontologies.

B.1 Anonymous Concepts

The main addition of OWL is a greatly extended set of syntactic structures that do not have a direct value of its own, but express some form of limitation or logic relation and hold one or more values nested in themselves. In RDF terms these are all blank nodes, usually differentiated by their type relation. In this subsection, we will provide an overview of the most common such constructs and how they are interpreted in the implementation. We will use the umbrella term *anonymous concept* to refer to all such elements.

Property Restrictions

Property restrictions are perhaps the most common new vocabulary element. A blank node is a restriction when the relation `rdf:type` has been attributed with the value `owl:Restriction`. Property restriction is commonly either the value of a `rdfs:subClassOf` relation of some class or the value of an anonymous class restriction. Semantically, property restrictions describe an anonymous class of all individuals that satisfy some kind of restriction. An `owl:Restriction` element typically contains two relations. The first relation is `owl:onProperty` and refers to the property resource related to the restriction. The second relation depends on the type of restriction. There are

two types of property restrictions: *value constraints* and *cardinality constraints* [57].

Value constraints restrict the range of the property when this property is applied to a particular class description. There are three relation types: `owl:hasValue`, `owl:someValuesFrom` and `owl:allValuesFrom`. `owl:hasValue` indicates a specific value that the property should have, whereas `owl:someValuesFrom` corresponds to existential ($\exists$) and `owl:allValuesFrom` to universal ($\forall$) quantifier from predicate logic [57].

Cardinality constraints restrict the number of values a particular property can have per instance of a class. There are three possible relation types: `owl:minCardinality`, `owl:maxCardinality` and `owl:cardinality`. These express the minimum, maximum and specific number of values an instance of a class can have[57].

Using property restrictions, we can improve the above example by stating that the Person class needs to have exactly one date of birth, month and year. To improve readability, we also assume our IRI is unique across all concepts from now on

```
<owl:Class rdf:about="#Person">
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#isBornOnDay" />
            <owl:cardinality
                rdf:datatype="&xsd;nonNegativeInteger">1
            </owl:cardinality>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#isBornOnMonth" />
            <owl:cardinality
                rdf:datatype="&xsd;nonNegativeInteger">1
            </owl:cardinality>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#isBornOnYear"/>
            <owl:cardinality
                rdf:datatype="&xsd;nonNegativeInteger">1
```

```
            </owl:cardinality>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
```

Similarly, we can express "Bob is a friend of Alice" using property restrictions.

```
<Person rdf:about="Bob">
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#isFriendOf" />
            <owl:hasValue rdf:resource="#Alice" />
        </owl:Restriction>
    </rdfs:subClassOf>
</Person>
```

In the implementation phase of this project, property restrictions are handled as separate elements (i.e. every restriction from ontology A is compared to every restriction from ontology B). As the choice of relations a restriction can have is relatively limited, strict matching of relation types is used. For example, if a restriction with relation `owl:someValuesFrom` is compared to a restriction that has a relation any other than `owl:someValuesFrom`, their similarity will always default to 0, regardless of what is the value of that relation.

Restrictions Used with Axioms

Typically a `owl:Restriction` resource is only referenced once. However, in the OWL 2, there may exist an additional reference from an `owl:Axiom` resource that annotates the restriction with additional information [54]. In the implementation phase, to separate this from typical `owl:Restriction` use case, we refer to such `owl:Axiom` resources as *Axioms* and `owl:Restriction` resources *Nodes* respectively.

Anonymous Class Descriptions

In addition to restrictions, there exist another type of blank nodes that express AND, OR and NOT set-operations on classes - `owl:intersectionOf`, `owl:unionOf`, and `owl:complementOf` respectively. These class descritions are always nested in `owl:Class` relation, typically without any value of its own, which is why we will refer to these as *anonymous classes* as a syntactic element. Unlike restrictions, anonymous classes may occur in any other relation.

As operations on a collection of sets, the relations `owl:intersectionOf` and `owl:unionOf` always contain more than one value, it is nested in an `rdf:List`.

Below are some examples of `owl:intersectionOf`, `owl:unionOf` and `owl:complementOf` relations in context.

John is a friend of Alice, Bob and anyone who is a friend of Bob.

```
<Person rdf:about="#John">
    <isFriendOf>
        <owl:Class>
            <owl:unionOf rdf:parseType="Collection">
                <Person rdf:about="#Alice"/>
                <Person rdf:about="#Bob"/>
                <owl:Restriction>
                    <owl:onProperty
                        rdf:resource="#isFriendOf" />
                    <owl:hasValue rdf:resource="#Bob" />
                </owl:Restriction>
            </owl:unionOf>
        </owl:Class>
    </isFriendOf>
</Person>
```

John is a friend of anyone who is a friend of both Alice and Bob.

```
<Person rdf:about="#John">
    <isFriendOf>
        <owl:Class>
            <owl:intersectionOf rdf:parseType="Collection">
                <owl:Restriction>
                    <owl:onProperty
                        rdf:resource="#isFriendOf" />
                    <owl:hasValue rdf:resource="#Alice" />
                </owl:Restriction>
                <owl:Restriction>
                    <owl:onProperty
                        rdf:resource="#isFriendOf" />
                    <owl:hasValue rdf:resource="#Bob" />
                </owl:Restriction>
```

```
            </owl:intersectionOf>
        </owl:Class>
    </isFriendOf>
</Person>
```

John is a friend of Bob and anyone who is not a friend of Alice.

```
<Person rdf:about="#John">
    <isFriendOf>
        <owl:Class>
            <owl:unionOf rdf:parseType="Collection">
                <Person rdf:about="#Bob"/>
                <owl:Class>
                    <owl:complementOf>
                        <owl:Restriction>
                            <owl:onProperty
                                rdf:resource="#isFriendOf" />
                            <owl:hasValue
                                rdf:resource="#Alice" />
                        </owl:Restriction>
                    </owl:complementOf>
                </owl:Class>
            </owl:unionOf>
        </owl:Class>
    </isFriendOf>
</Person>
```

Like with property restrictions, only those anonymous classes that have matching a relation type are compared. Values that are encapsulated inside `rdf:List` are transformed to a Prolog list. Furthermore, as the two forms are equivalent, values of a `owl:intersectionOf` relation are no longer presented as an `rdf:List` in the generated output and simply as relations of the parent resource. The statement *John is a friend of anyone who is a friend of both Alice and Bob* would therefore be expressed as following:

```
<Person rdf:about="#John">
    <isFriendOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#isFriendOf" />
            <owl:hasValue rdf:resource="#Alice" />
        </owl:Restriction>
```

```
    </isFriendOf>
    <isFriendOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#isFriendOf" />
            <owl:hasValue rdf:resource="#Bob" />
        </owl:Restriction>
    </isFriendOf>
</Person>
```

The motivation for this transformation is to ease the manipulation and readability of such elements as `rdf:List` elements are cumbersome to work with and are not split into smaller elements in output writing.

Summary of Anonymous Concepts

To sum up, OWL introduces a number of vocabulary elements that do have a direct value of its own, but convey some kind of semantic meaning. In our thesis, we refer to these as anonymous concepts because they are compared on similar principles for their similarity as (defined) concepts. We distinguish two major sub-types of anonymous concepts: 1) property restrictions add some type of constraint to an `owl:Class` element and 2) anonymous classes that express AND, OR and NOT set-operations. In addition there are axioms that are typically linked to a property restriction and some relation of a defined concept with the aim of further annotating that relation.

Similar to anonymous concepts is `rdf:List` from the RDF vocabulary. Like anonymous concepts, `rdf:List` elements do not have a direct value of their own, but we do not consider them anonymous concepts and do not compare them for their similarity as these elements need to be considered in the context where they occur (i.e., the overlying relation). However, it is important to note that `rdf:List` elements often occur nested inside an anonymous class.

In Figure 2 we present a diagrammatic breakdown of various sub-types of anonymous concepts.

B.1.1 OWL Property Categories

In addition to various constructs using blank nodes, OWL also extends the use of properties, distinguishing between a number of different property categories. The two main categories are `owl:ObjectProperty` that links individuals to individuals and

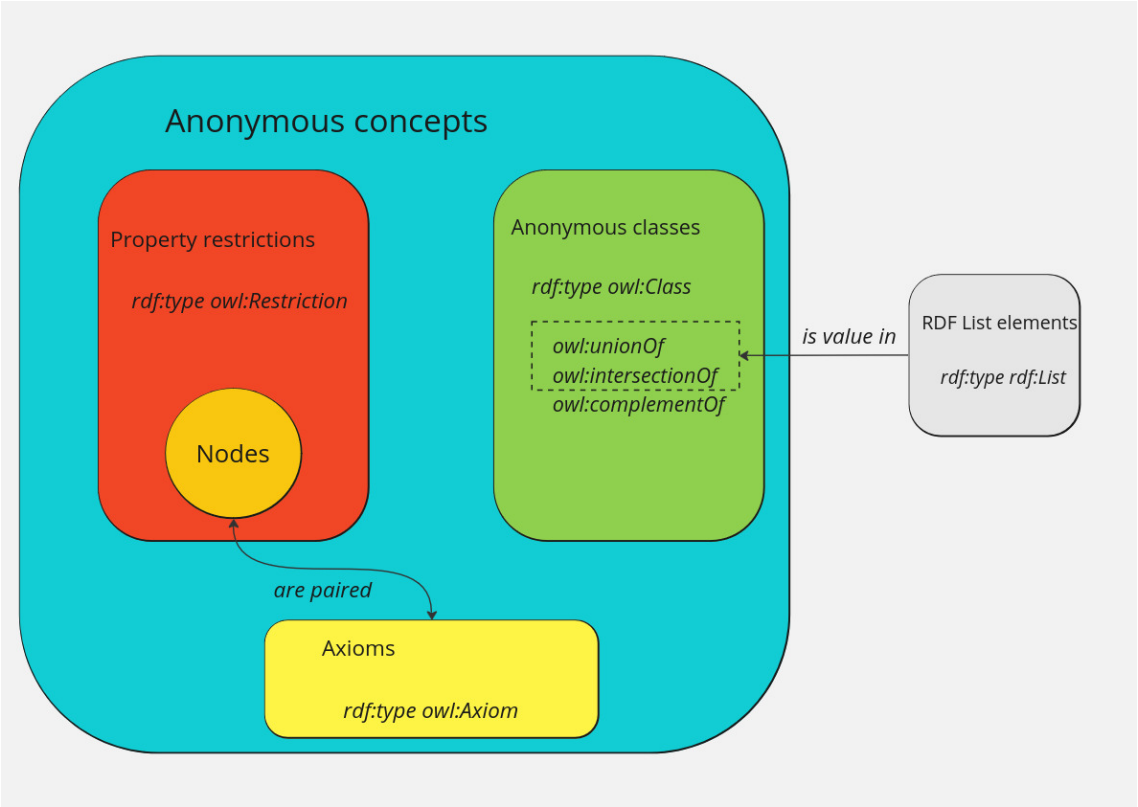

Figure 2. Taxonomy of Anonymous Concepts

`owl:DatatypeProperty` that links individuals to data values. In addition, there are less common categories `owl:AnnotationProperty` and `owl:OntologyProperty`. All these new property categories are subclasses of `rdf:Property`. OWL property categories make use of the same key relations as `rdf:Property`, most notably `rdfs:subPropertyOf`, `rdfs:domain` and `rdfs:range`, but also relations specific to OWL vocabulary, such as `owl:equivalentProperty` and `owl:inverseOf` [57].

Both `owl:ObjectProperty` and `owl:DatatypeProperty` can be constrained such that only one unique value of relation `rdfs:domain` is allowed for each value of relation `rdfs:range`. This can be achieved by using `owl:FunctionalProperty` which is also constructed a subclass of `rdf:Property`. In the following example `owl:FunctionalProperty` *husband* is used to state that a woman can have only one man as a husband.

```
<owl:ObjectProperty rdf:ID="husband">
  <rdf:type rdf:resource="&owl;FunctionalProperty" />
  <rdfs:domain rdf:resource="#Woman" />
  <rdfs:range rdf:resource="#Man" />
</owl:ObjectProperty>
```

Similarly owl:ObjectProperty can be constrained such that only one unique value of relation rdfs:range is allowed for each value of relation rdfs:domain by using owl:InverseFunctionalProperty. Unlike owl:FunctionalProperty, it cannot be used with owl:DatatypeProperty which is why it is constructed as a subclass of owl:ObjectProperty. Below is an example of owl:InverseFunctional-Property *biologicalMotherOf* which states that every human can have only one mother.

```
<owl:InverseFunctionalProperty rdf:ID="biologicalMotherOf">
  <rdfs:domain rdf:resource="#Woman"/>
  <rdfs:range rdf:resource="#Human"/>
</owl:InverseFunctionalProperty>
```

Lastly owl:ObjectProperty has two additional subclasses that further refine the logical characteristics of the property: owl:TransitiveProperty and owl:Symmetric-Property.

As indicated from the name, owl:TransitiveProperty expresses transitivity: if pairs (x,y) and (y,z) are instances of P, then (x,z) is also an instance of P. For example, we can use owl:TransitiveProperty to state that since Bob is a friend of Alice and Alice is a friend of John, so is Bob also a friend of John without explicitly stating it. [57]

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

<owl:TransitiveProperty rdf:ID="isFriendOf">
  <rdfs:domain rdf:resource="#Person"/>
  <rdfs:range rdf:resource="#Person"/>
</owl:TransitiveProperty>

<Person rdf:about="#John"/>

<Person rdf:about="#Bob">
    <isFriendOf rdf:resource="#Alice"/>
</Person>

<Person rdf:about="#Alice">
    <isFriendOf rdf:resource="#John"/>
</Person>
```

Conversely, owl:SymmetricProperty expresses symmetry, i.e. if pair (x,y) is an instance of P, then (x,y) is also an instance of P. Thus we can use owl:SymmetricProperty

to state that since Bob is a friend of Alice, Alice is also friend of Bob without explicitly stating it [57].

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

<owl:SymmetricProperty rdf:ID="isFriendOf">
  <rdfs:domain rdf:resource="#Person"/>
  <rdfs:range  rdf:resource="#Person"/>
</owl:SymmetricProperty>

<Person rdf:about="#Bob">
    <isFriendOf rdf:resource="#Alice"/>
</Person>

<Person rdf:about="#Alice" />
```

As different OWL property categories express distinctly different meaning, they are not compared with one-another in the implementation. It is worth considering comparing `owl:ObjectProperty` to its subtypes as a future development.

Appendix 3 – Examples on Calculating Concept Pair Confidence

This appendix illustrates the formal description of the methodology described in Chapter 4 with a number of examples represented in RDF/OWL format.

The simplest case would be when the concepts have no additional relations other than their name. The confidence of a concept pair would in that case be the similarity of their names.

Example A

```
<rdf:Description
    rdf:about="iri1#Car" />
```

Example B

```
<rdf:Description
    rdf:about="iri2#Vehicle" />
```

Basic Example

Typically concepts do have some relations other than their name. In the following example, both concepts have three relations in addition to their name: `rdf:type`, `rdfs:domain` and `rdfs:range`. Note that both concept have two values for the relation `rdfs:domain`.

Example A

```
<owl:ObjectProperty
    rdf:about=
        "iri1#isGridPowered">
    <rdfs:domain
        rdf:resource=
            "iri1#device"/>
    <rdfs:domain
        rdf:resource=
            "iri1#airplanes"/>
    <rdfs:range
        rdf:resource=
            "iri1#battery"/>
</owl:ObjectProperty>
```

Example B

```
<owl:ObjectProperty
    rdf:about=
        "iri2#has-grid-power">
    <rdfs:domain
        rdf:resource=
            "iri2#machine"/>
    <rdfs:domain
        rdf:resource=
            "iri2#airplanes"/>
    <rdfs:range
        rdf:resource=
            "iri2#battery"/>
</owl:ObjectProperty>
```

Let us suppose that the semantic and string similarity metric calculated the following confidence relation values:

Table 8. Relation value pairs of basic example

Relation type	Value A	Value B	Conf
rdf:type	ObjectProperty	ObjectProperty	1
rdfs:domain	airplanes	airplanes	1
rdfs:range	battery	battery	1
name	isGridPowered	has-grid-power	0.8
rdfs:domain	device	machine	0.6

In that case, calculated similarity of the entire concept pair is $\frac{6*1+2*0.8+2*0.6}{2*5} = 0.88$.

Example with Absent Relation Values

It is common that two concepts do not have an identical structure: either one of the concepts has a relation that the other does not or the two concepts have a different number of values for the same relation. In such case, the confidence value of an absence is used instead. In the following example, the concepts have a different number of values in both rdfs:range (0 value vs 1 value) and rdfs:domain (2 values vs 1 value). The relation values that end up without a counterpart are assigned the absence confidence value.

Example A

```
<owl:ObjectProperty
    rdf:about=
        "iri1#isGridPowered">
    <rdfs:domain
        rdf:resource=
            "iri1#device"/>
    <rdfs:domain
        rdf:resource=
            "iri1#airplanes"/>
</owl:ObjectProperty>
```

Example B

```
<owl:ObjectProperty
    rdf:about=
        "iri2#has-grid-power">
    <rdfs:domain
        rdf:resource=
            "iri1#airplanes"/>
    <rdfs:range
        rdf:resource=
            "iri2#battery"/>
</owl:ObjectProperty>
```

Let us suppose that confidence value of an absence is set at 0.5 and we have the following confidence relation values:

Table 9. Relation value pairs of example with absences

Relation type	Value A	Value B	Conf
rdf:type	ObjectProperty	ObjectProperty	1
rdfs:domain	airplanes	airplanes	1
rdfs:range	-	battery	0.5
name	isGridPowered	has-grid-power	0.8
rdfs:domain	device	-	0.5

In that case, calculated similarity of the entire concept pair is $\frac{4*1+2*0.8+2*0.5}{2*4} = 0.825$. Note that the total number of relations is 8 and both concepts are missing asymmetrically different relations in comparison to the previous example.

Example with Inherited Relations

If the concept has an ancestral relation (`rdfs:subClassOf` or `rdfs:subPropertyOf` respectively) to another defined concept that exists in the same ontology, the relation values of the ancestral concept are also used when calculating confidence value of the descendant. This process is recursive, meaning that the relation values of all ancestors on any level are considered. In the following example, calculating the confidence of the concept `#ElectricBike` would also consider the relation values inherited by `#Bike` and `#Vehicle` respectively.

```
<owl:Class rdf:about="http://iri1#ElectricBike">
    <rdfs:subClassOf rdf:resource="http://iri1#Bike"/>
</owl:Class>

<owl:Class rdf:about="http://iri1#Bike">
    <rdfs:subClassOf rdf:resource="http://iri1#Vehicle"/>
</owl:Class>

<owl:Class rdf:about="http://iri1#Vehicle">
    <rdfs:subClassOf rdf:resource="http://iri1#Object"/>
</owl:Class>
```

Example with Anonymous Concepts

Comparison of anonymous concept pairs is based on same principles, but is further constrained to a specific set of relations. In other words, since there is a limited combination of relations that can exist within an anonymous concept, only those pairs are considered valid where relation types match completely.

For example, the following anonymous concept pairs are not considered valid (i.e., the confidence of the pair is always 0) due to a differing relation type.

Restriction A

```
<owl:Restriction>
    <owl:onProperty
        rdf:resource="#isFriendOf" />
    <owl:hasValue
        rdf:resource="#Alice" />
</owl:Restriction>
```

Restriction B

```
<owl:Restriction>
    <owl:onProperty
        rdf:resource="#isFriendOf" />
    <owl:allValuesFrom
        rdf:resource="#Alice" />
</owl:Restriction>
```

Anonymous class A

```
<owl:Class>
    <owl:unionOf
        rdf:parseType="Collection">
        <Person rdf:about="#Alice"/>
        <Person rdf:about="#Bob"/>
    </owl:unionOf>
</owl:Class>
```

Anonymous class B

```
<owl:Class>
    <owl:intersectionOf
        rdf:parseType="Collection">
        <Person rdf:about="#Alice"/>
        <Person rdf:about="#Bob"/>
    </owl:intersectionOf>
</owl:Class>
```

It is worth noting that relations expressing ancestry can not exist within an anonymous concept. As such, inherited relations never occur within comparison of anonymous concept pairs.

Appendix 4 – String Similarity Comparison Table

This appendix covers the results of an experiment to observe a the performance SWI-Prolog's *isub* library to cosine similarity. Open-source Python library Gensim [58] was used for calculating cosine similarity. Two different pre-trained models available at [59] were used as datasets for cosine similarity: *glove-wiki-gigaword-300* based on the 2014 version of Wikipedia and *word2vec-google-news-300* based on Google News

The relation value pairs are a subset from PMK-Dolce comparison where semantic or string similarity exceed 0.4.

Value 1	Value 2	isub	cosine (Wiki)	cosine (Google News)
accomplishment	ActionClass	0.1	0.13	0.13
achievement	ActionClass	0.1	0.28	0.17
ActionClass	process	0	0.31	0.13
set	ActionClass	0	0.39	0.1
ActionClass	state	0	0.32	0.12
hasSensingComponent	part	0	0.48	0.36
ActionClass	abstract	0.44	0.21	0.08
ActionClass	abstract-region	0.58	0.35	0.13
hasMemory	has-quale	0.64	0.48	0.47
hasMemory	has-t-quality	0.59	0.6	0.32
ActionClass	proposition	0.45	0.25	0.12
ActionClass	region	0.46	0.27	0.11
has-quale	hasSensingComponent	0.54	0.34	0.47
hasSensingComponent	has-quality	0.48	0.54	0.34
hasSensingComponent	partly-compresent	0.53	0.3	0.22
QualityAggregation	quale	0.72	-0.08	0.21
QualityAggregation	quality-space	0.73	0.49	0.58
isRelatedTo	Immediate-relation	0.51	0.61	0.41
isRelatedTo	mediated-relation	0.51	0.31	0.35
isRelatedTo	mediated-relation-i	0.49	0.52	0.38
Situation	spatio-temporal-particular	0.41	0.27	0.13
SpatialContext	spatio-temporal-particular	0.64	0.69	0.62
TemporalContext	spatio-temporal-particular	0.56	0.9	0.84
TemporallyExtendedStuff	spatio-temporal-particular	0.41	0.29	0.35
WSpaceClass	quality-space	0.57	0.6	0.53
WSpaceClass	space-region	0.6	0.56	0.49