Vrije Universiteit Amsterdam

Formal Logic for AI, Project Report

Natural Language Explanations for OWL Reasoning

Author: Saloni Bhandari (2762779)

March 25, 2024

1 Introduction

Ontologies are formal representations of knowledge, usually restricted to a particular application domain [10]. They provide a formal framework for capturing and organizing domain-specific knowledge in a machine readable format. Ontologies have a wide range of applications in diverse fields such as medicine [9], biology [8], the semantic web [7], and banking and finance [4].

One prominent approach to knowledge representation in ontologies leverages Description Logics (DLs). DLs are a family of knowledge representation languages that use concepts to represent individual classes and roles to represent binary relations between concepts [9]. These relationships can specify inheritance hierarchies, constraints, and the overall governance of concepts within the domain. Web Ontology Language (OWL) is a widely used language [13] based on Description Logics that allows for standardized ontology creation. OWL ontologies utilize concepts, roles (relationships), and axioms to formally represent knowledge. Axioms are logical statements that capture constraints and relationships between defined concepts, enabling the creation of rich and expressive knowledge structures.

Reasoning with ontologies allows for automated inference of implicit knowledge, which is crucial for tasks such as automated decision-making and information retrieval. While these formalisms are powerful for automated reasoning, they are often expressed in a symbolic language that is challenging for non-experts to grasp.

Furthermore, it is crucial to understand *why* an entailment is inferred from logical reasoning. In large ontologies with intricate relationships between axioms, the reasoning process behind an entailment can be quite complex. Without explanations, unexpected inferences can arise, causing users to question the ontology's reliability. Understanding *how* the entailment follows from the ontology is important for debugging purposes and allows traceability.

This is where natural language explanations come into play. They bridge the gap between the complex logic and human comprehension by translating the reasoning process into natural language. This project focuses on generating natural language explanations for subclass relationships within OWL ontologies. A subclass relationship specifies that one class (subclass) inherits all the characteristics of another class (superclass). In essence, these explanations aim to make the implicit knowledge captured within the ontology explicit and understandable for human users. We illustrate this using the following example for a subclass axiom from the pizza ontology

Reasoning for Margherita $\sqsubseteq$ CheeseyPizza in Manchester OWL Syntax:

```
Margherita SubClassOf NamedPizza
Margherita SubClassOf hasTopping some MozzarellaTopping
MozzarellaTopping SubClassOf CheeseTopping
CheeseyPizza EquivalentClasses Pizza and hasTopping some CheeseTopping
```

```
NamedPizza SubClassOf Pizza
```

Reasoning for Margherita $\sqsubseteq$ CheeseyPizza in Natural Language:

```
Every Margherita is a NamedPizza
Every Margherita has some hasTopping that is MozzarellaTopping
Every MozzarellaTopping is a CheeseTopping
CheeseyPizza is defined as Pizza and has some hasTopping that is CheeseTopping
Every NamedPizza is a Pizza
```

This example depicts the importance of natural language explanations in bridging the gap between the formal logic of ontologies and human understanding. By providing clear explanations, we can improve the usability and trust in ontology systems, particularly for users who are not experts in Description Logics or OWL syntax. By providing clear explanations for subclass relationships in natural language, we aim to unlock the full potential of OWL ontologies. We envision them not just as powerful knowledge representation tools, but also as transparent and understandable systems that can be effectively utilized by humans, regardless of their technical background in logic or knowledge representation formalisms.

2 Related Work

In recent years, there has been growing interest in improving the interpretability and usability of OWL ontologies through the provision of explanations behind reasonings. Several tools have been developed to visualise entailments and axioms that lead to the specific inference. One such tool is EVONNE, an interactive web-based application for visualising proofs to explain logical inferences derived from OWL ontologies [11, 1]. EVONNE offers users the ability to explore the reasoning process behind ontology entailments through graphical representations. It can be accessed here through a web browser.

Another tool that for proof generation is the Evee-libs (EVincing Expressive Entailments), a Java library that forms the basis of Evee-protege, a collection of Protégé plugins for interactively inspecting proofs [3, 2]. Evee-protege allows users to interactively inspect proofs generated by Evee-libs. In contrast to EVONNE's web-based visualization approach, Evee-protege integrates seamlessly within the familiar Protégé environment, potentially making it easier for existing Protégé users to explore proofs directly within their ontologies. The source code and installation instructions for Evee-libs and Evee-protege can be found here (version 0.2) [2], where also the plugins can be downloaded.

3 Preliminaries

Description Logics (DLs) are a family of knowledge representation languages that model concepts and relationships. We focus specifically on the DL $\mathcal{ALC}$. A concept is an entity that represents a class of objects (e.g Pizza, NamedPizza) by combining other concepts and role names using connectives of the DL, such as conjunction ($\sqcap$) or disjunction ($\sqcup$). A DL ontology is a set of axioms that express relationships between concepts. Lets illustrate using an example of an axiom that states that cheesey pizzas are defined as pizzas that have some cheese topping:

$$CheeseyPizza \equiv Pizza \sqcap \exists hasTopping.CheeseTopping.$$

Here *CheeseyPizza*, *Pizza* and *CheeseTopping* are concept names describing sets of objects and *hasTopping* is a role name that defines a relationship between those objects. A more detailed description of the semantics and syntax of Description Logics can be found in [5] and [6]. A DL reasoner can automatically infer information that logically follows from the given ontology. For example, a common reasoning task is classification, that is, the inference of the subsumption heirarchy. For example, a reasoner can infer that *CheeseyPizza* is subsumed by (or a subclass of) *Pizza*, by using the definition of *CheeseyPizza*, as described above.

The most common way of explaining an entailment, is through *justifications.* Justifications are minimum set of axioms from the ontology that lead to an entailment, which are very useful when working with large ontologies. Formally we can define as justification $\mathcal{O}$ for an ontology $\mathcal{O}$ and axiom α, where $\mathcal{O} \models \alpha$ as:

A justification for α in $\mathcal{O}$ is a subset-minimal $\mathcal{J} \subseteq \mathcal{O}$ such that $\mathcal{J} \models \alpha$ [14].

In the worst case, there can be exponentially many justifications for an entailment.

4 Natural Language Explanations

While OWL ontologies provide a structured framework for representing domain knowledge, their abstract nature and the use of formal logic can present comprehension barriers for users unfamiliar with the syntactical intricacies of OWL or DLs. This restricts the use of OWL ontologies and significantly narrows their scope. Our system addresses this challenge by generating natural language explanations for OWL axioms, allowing users, regardless of their expertise in formal logic or ontologies, to grasp the underlying concepts and relationships more easily. For instance, consider the following axiom:

$$CheeseyPizza \equiv Pizza \sqcap \exists hasTopping.CheeseTopping.$$

This axiom, written in $\mathcal{ALC}$ sntax gives an equivalence axiom for a CheeseyPizza. An easier way to represent this axiom would be:

A CheeseyPizza is defined as a Pizza that has some hasTopping that is CheeseTopping.

By using similar natural language translations, we make the use of ontologies and reasoners more accessible and understandable to a wider audience. Moreover, natural language explanations enhance the interpretability and trustworthiness of ontology-based systems. By understanding the reasoning behind inferred relationships and classifications, users can confidently utilize ontology-driven applications for decision-making and information retrieval tasks. In decision-making scenarios, users can trust the recommendations or suggestions provided by the ontology system because they understand the justification behind those recommendations. Similarly, in information retrieval tasks, users can have confidence in the retrieved information knowing that it is relevant and accurate based on the reasoning performed by the ontology.

Additionally, natural language explanations make the reasoning process behind inferences transparent. This allows users to see the chain of logic that leads to a specific conclusion. By examining the explanation, users can identify potential errors or inconsistencies in the ontology's structure. Once identified, these errors can be rectified, leading to a more accurate and reliable knowledge base as well as a smoother debugging process.

Table 1 depicts DL expressions supported by the system and their corresponding natural language explanations:

DL syntax	Natural Language Explanations
$C \sqcup D$	either C or D
$C \sqcap D$	C and D
$\neg C$	does not C
$\forall r.C$	has r that is C
$\exists r.C$	has some r that is C
$\{a_1, a_2, ..., a_n\}$	is one of: a1, a2, ..., an
$\geq n\ r.C$	has at least n r that are C
$C \equiv D$	C is defined as D
$C \sqsubseteq \neg D$	C and D do not overlap.
$\geq 1 r \sqsubseteq C$	r is in the domain C
$\top \sqsubseteq \forall r.C$	r has the range C
$C \sqsubseteq D$	Every C is a D
Functional(r)	r can only have one value
$r_1 \sqsubseteq r_2$	Every r1 is a r2
$r_1 \equiv r_2^{-1}$	r1 is the inverse of r2

Table 1: OWL DL Expressions and their Natural Language Explanations

5 Implementation

The implementation presented in this project offers a solution for interactively exploring and explaining logical inferences within OWL ontologies in Java by leveraging theOWL API to interact with ontologies and the HermiT reasoner to perform reasoning tasks.

The implementation focuses on subclass axioms which define a relationship between a superclass (a more general concept) and a subclass (a more specific concept that inherits the characteristics of the superclass). The user interacts with the program through a text-based (command line) interface. On launch, the user is prompted with a list of classes in the ontology and asked to choose a subclass. Lets illustrate this through an example from the pizza ontology

```
  Classes in the ontology (in alphabetical order):
1. American
2. AmericanHot
3. AnchoviesTopping
4. ArtichokeTopping
5. AsparagusTopping
.
.
.
96. VegetarianPizza
97. VegetarianPizzaEquivalent1
98. VegetarianPizzaEquivalent2
99. VegetarianTopping
100. Veneziana

Choose a subclass, or enter 'q' to quit:
```

On choosing a subclass, it's corresponding superclasses are displayed. Once again, the user is prompted to choose the superclass they would like to explore. Users can navigate back and forth between subclass and superclass selections as needed.

```
  Choose a subclass, or enter 'q' to quit:
> 40

Chosen subclass is: Margherita
Superclasses of Margherita (in alphabetical order):
1. CheeseyPizza
2. NamedPizza
3. VegetarianPizzaEquivalent1
4. VegetarianPizzaEquivalent2

Choose a superclass, or enter 'b' to go back, or 'q' to quit:
```

The system offers users two forms of explanations to aid in understanding the logical inferences between subclasses and superclasses. Users can opt for either a a minimal set explanations (a justification) or a more comprehensive set of explanations.

```
  Choose a superclass, or enter 'b' to go back, or 'q' to quit:
> 1
Chosen superclass is: CheeseyPizza

Choose the type of explanation:
1. Minimal Set of explanations
2. Full Set of explanations
Or press 'b' to go back
```

Once both the superclass and subclass are selected, explanations are generated for the selected subclass axioms. These explanations are generated by implementing visitor patterns. Visitor patterns are a design pattern commonly used in object-oriented programming to implement a definition of a new operation on an object structure without changing the type of the objects [12]. In OWL, vistor patterns are used to traverse the complex hierarchy of ontology entities, such as classes, properties, and axioms.

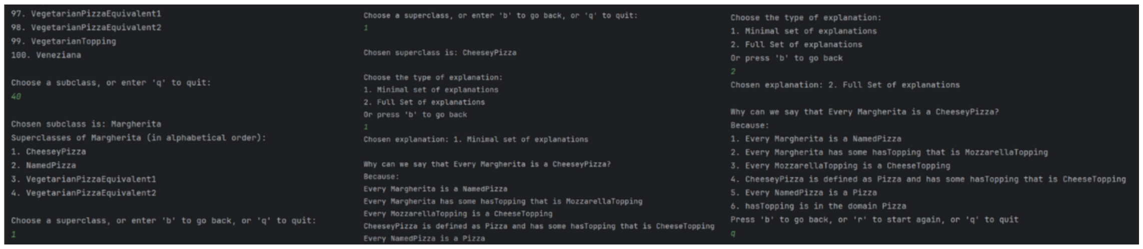
Here, the visitor pattern is used to extend existing interfaces for class expressions, objects, and axioms. This enables the definition of specialized visitor methods that handle specific types of axioms, class expressions or objects such as "OWLIntersectionOf", "OWLEquivalentClassesAxiom", "OWLClass", etc. When a user selects a subclass axiom for explanation, the program employs a corresponding visitor object. This visitor object traverses the axiom's components, including the superclass, subclass, and any subcomponents (e.g., restrictions within class expressions). By visiting each component, the visitor extracts relevant information and constructs a human-readable explanation in natural language. This structure gives us a flexible and maintainable framework. New types of explanations can be introduced by implementing additional visitor classes, without modifying the core logic for processing ontology entities. This modular approach simplifies future extensions and reduces the risk of introducing errors during code modifications.

For this example, lets consider both a justification and a set of explanations:

```
  > 1
Chosen explanation: 1. Minimal Set of explanations

Why can we say that Every Margherita is a CheeseyPizza?
Because:
Every Margherita is a NamedPizza
Every Margherita has some hasTopping that is MozzarellaTopping
Every MozzarellaTopping is a CheeseTopping
CheeseyPizza is defined as Pizza and has some hasTopping that is CheeseTopping
Every NamedPizza is a Pizza

Press 'b' to go back, or 'r' to start again, or 'q' to quit
```

```
  > 2
Chosen explanation: Full Set of explanations

Why can we say that Every Margherita is a CheeseyPizza?
Because:
1. Every Margherita is a NamedPizza
2. Every Margherita has some hasTopping that is MozzarellaTopping
3. Every MozzarellaTopping is a CheeseTopping
4. CheeseyPizza is defined as Pizza and has some hasTopping that is CheeseTopping
5. Every NamedPizza is a Pizza
6. hasTopping is in the domain Pizza

Press 'b' to go back, or 'r' to start again, or 'q' to quit
```

Figure 1 illustrates the same example interaction with the application's user interface.



```
97. VegetarianPizzaEquivalent1
98. VegetarianPizzaEquivalent2
99. VegetarianTopping
100. Veneziana

Choose a subclass, or enter 'q' to quit:
40

Chosen subclass is: Margherita
Superclasses of Margherita (in alphabetical order):
1. CheeseyPizza
2. NamedPizza
3. VegetarianPizzaEquivalent1
4. VegetarianPizzaEquivalent2

Choose a superclass, or enter 'b' to go back, or 'q' to quit:
1
```

```
Choose a superclass, or enter 'b' to go back, or 'q' to quit:
1

Chosen superclass is: CheeseyPizza

Choose the type of explanation:
1. Minimal set of explanations
2. Full Set of explanations
Or press 'b' to go back
1
Chosen explanation: 1. Minimal set of explanations

Why can we say that Every Margherita is a CheeseyPizza?
Because:
Every Margherita is a NamedPizza
Every Margherita has some hasTopping that is MozzarellaTopping
Every MozzarellaTopping is a CheeseTopping
CheeseyPizza is defined as Pizza and has some hasTopping that is CheeseTopping
Every NamedPizza is a Pizza
```

```
Choose the type of explanation:
1. Minimal set of explanations
2. Full Set of explanations
Or press 'b' to go back
2
Chosen explanation: 2. Full Set of explanations

Why can we say that Every Margherita is a CheeseyPizza?
Because:
1. Every Margherita is a NamedPizza
2. Every Margherita has some hasTopping that is MozzarellaTopping
3. Every MozzarellaTopping is a CheeseTopping
4. CheeseyPizza is defined as Pizza and has some hasTopping that is CheeseTopping
5. Every NamedPizza is a Pizza
6. hasTopping is in the domain Pizza
Press 'b' to go back, or 'r' to start again, or 'q' to quit
q
```

Figure 1: Screenshot of the application's user interface

5.1 Supported Expressions

This implementation provides natural language explanations for the following OWL Constructs:

5.1.1 OWL Class Expressions

1. **OWLClass:** Handles basic OWL classes, returning their name (e.g., "Margherita").
2. **OWLObjectComplementOf:** Represents the complement of a class (e.g., "does not have AnchoviesTopping").
3. **OWLObjectAllValuesFrom:** Represents a restriction on a class, stating that all instances must have a value for the specified property (e.g., "hasTopping that is CheeseTopping").
4. **OWLObjectSomeValuesFrom:** Represents a restriction on a class, stating that all instances must have at least one value for the specified property (e.g., "has some Topping that is VegetableTopping").

5. **OWLObjectUnionOf:** Represents the disjunction (or) of multiple class expressions (e.g., "is either VegetarianPizza or CheeseyPizza").

6. **OWLObjectIntersectionOf:** Represents the conjunction (and) of multiple class expressions (e.g., "hasTopping that is CheeseTopping and hasTopping that is TomatoTopping").

7. **OWLObjectHasValue:** Represents a restriction on a class, stating that all instances must have a specific value for the specified property (e.g., "hasSize with value Large").

8. **OWLObjectOneOf:** Represents a class that consists of only a specific set of enumerated individuals (e.g., "is one of: VegetarianPizza, CheeseyPizza").

9. **OWLObjectMinCardinality:** Represents a restriction on a class, stating that all instances must have at least a certain number of values for the specified property that belong to another class (e.g., "has at least 2 Toppings that are CheeseTopping").

5.1.2 OWL Axioms

1. **OWLFunctionalObjectPropertyAxiom:** Represents an axiom stating that an object property can only have one value for each instance (e.g., "hasSpiciness can only have one value").

2. **OWLSubObjectPropertyOfAxiom:** Represents an axiom stating that one object property is a sub-property of another, meaning it inherits all the characteristics of the super-property (e.g., "Every hasIngredient is a hasTopping").

3. **OWLInverseObjectPropertiesAxiom:** Represents an axiom stating that two object properties are inverses of each other (e.g., "hasPart is the inverse of isPartOf").

4. **OWLObjectPropertyDomainAxiom:** Represents an axiom stating that the domain (allowed source) for a specific object property is restricted to a certain class (e.g., "hasTopping has domain Pizza").

5. **OWLObjectPropertyRangeAxiom:** Represents an axiom stating that the range (allowed target) for a specific object property is restricted to a certain class (e.g., "hasTopping has range Topping").

6. **OWLEquivalentClassesAxiom:** Represents an axiom stating that two class expressions are equivalent, meaning they have exactly the same instances (e.g., "VegetarianPizza is defined as Pizza").

7. **OWLDisjointClassesAxiom:** Represents an axiom stating that two class expressions are disjoint, meaning they cannot have any instances in common (e.g., "VegetarianPizza and MeatPizza are disjoint").

8. **OWLSubClassOfAxiom:** Represents a subclass axiom. It states that every instance of the subclass is also an instance of the superclass (e.g., "Every VegetarianPizza is a Pizza").

6 Conclusion and Future Work

Natural language explanations for entailments in OWL ontologies bridge the gap between the machine-readable formal logic of DLs and human understanding. This project introduced a tool that generates natural language explanations for subclass relationships within ontologies. We can generate both, a minimal set of explanations and a full set of explanations behind an entailment. While this work focuses on subclass relationships, there is still room for improvement.

The tool currently focuses on explaining subclass relationships, a single type of reasoning task within ontologies. In the future, the implementation can be extended to support multiple reasoning tasks. For instance, the tool could explain why a property holds between classes (property assertions) or why two classes are guaranteed to have no members in common (disjointness). Moreover, we currently support a limited number of OWL constructs. Since the implementation utilizes visitor patterns, it can easily be extended to support more OWL constructs that are not accounted for in this version.

Lastly, the tool currently employs a command line based text interface, which is functional, but cumbersome and not very user friendly. Developing a user-friendly graphical interface would significantly enhance accessibility. Users could then visualize the ontology structure, select elements for explanation through menus, and receive explanations presented in a clear and formatted manner. This would once again, broaden the scope of use by providing explanations in a more user-friendly manner.

References

[1] Alrabbaa, C., Baader, F., Borgwardt, S., Dachselt, R., Koopmann, P., and Méndez, J. Evonne: Interactive proof visualization for description logics (system description). In *International Joint Conference on Automated Reasoning* (2022), Springer, pp. 271–280.

[2] Alrabbaa, C., Borgwardt, S., Friese, T., Koopmann, P., and Kotlov, M. Why not? explaining missing entailments with evee. In *36th International Workshop on Description Logics, DL 2023* (2023), CEUR-WS. org, pp. 1–13.

[3] Alrabbaa, C., Borgwardt, S., Friese, T., Koopmann, P., Méndez, J., and Popovic, A. On the eve of true explainability for owl ontologies: Description logic proofs with evee and evonne. In *Description Logics* (2022).

[4] Altinok, D. An ontology-based dialogue management system for banking and finance dialogue systems, 2018.

[5] Baader, F., Horrocks, I., Lutz, C., and Sattler, U. *An Introduction to Description Logic.* Cambridge University Press, 2017.

[6] Baader, F., Horrocks, I., and Sattler, U. Chapter 3 description logics. In *Handbook of Knowledge Representation*, F. van Harmelen, V. Lifschitz, and B. Porter, Eds., vol. 3 of *Foundations of Artificial Intelligence.* Elsevier, 2008, pp. 135–179.

[7] Grimm, S., Abecker, A., Völker, J., and Studer, R. *Ontologies and the Semantic Web.* Springer Berlin Heidelberg, Berlin, Heidelberg, 2011, pp. 507–579.

[8] Hoehndorf, R., Schofield, P. N., and Gkoutos, G. V. The role of ontologies in biological and biomedical research: a functional perspective. *Briefings in Bioinformatics 16*, 6 (04 2015), 1069–1080.

[9] Ivanović, M., and Budimac, Z. An overview of ontologies and data resources in medical domains. *Expert Systems with Applications 41*, 11 (2014), 5158–5166.

[10] Lehmann, J., and Voelker, J. An introduction to ontology learning. *Perspectives on Ontology Learning 18* (2014).

[11] Méndez, J., Alrabbaa, C., Koopmann, P., Langner, R., Baader, F., and Dachselt, R. Evonne: A visual tool for explaining reasoning with owl ontologies and supporting interactive debugging. In *Computer Graphics Forum* (2023), vol. 42, Wiley Online Library, p. e14730.

[12] Palsberg, J., and Jay, C. B. The essence of the visitor pattern. In *Proceedings. The Twenty-Second Annual International Computer Software and Applications Conference (Compsac'98)(Cat. No. 98CB 36241)* (1998), IEEE, pp. 9–15.

[13] Pan, J., and OWL Working Group. Owl 2 web ontology language document overview: W3c recommendation 27 october 2009, 2009.

[14] Schlobach, S., and Cornet, R. Non-standard reasoning services for the debugging of description logic terminologies. pp. 355–362.