Report 1 DL, Group 8

November 25, 2023

Abstract

This study develops an intelligent system for a burger restaurant using knowledge representation and reasoning techniques. A Description Logic Ontology, created with Protege, represents the restaurant's menu with over 25 classes. The project implements an $\mathcal{EL}$ reasoner based on the $\mathcal{EL}$ Completion Algorithm to compute subsumers for classes. The efficiency and capabilities of the $\mathcal{EL}$ reasoner are compared with the ELK reasoner across various ontologies. The paper discusses challenges in ontology development and $\mathcal{EL}$ reasoner implementation, providing an overview of the ontology's class hierarchy and logical structures. The findings highlight the superior efficiency of the ELK reasoner in most cases, but also note instances where ELK underperformed, indicating the need for further optimization of both the $\mathcal{EL}$ and ELK reasoners.

Keywords: Burger Ontology, $\mathcal{EL}$ Reasoning, Knowledge Representation

Contents

Introduction

This project consists of the development of an intelligent system designed for a Burger restaurant, integrating knowledge representation and reasoning. Utilizing Protege as our primary tool, we aimed to construct a description logic ontology that includes the diverse elements of a Burger menu, allowing for intelligent responses to customer queries.

We created a Description Logic Ontology with more than 25 classes, incorporating logical constructors for robust classification results. Our focus is on providing a comprehensive description of the restaurant's offerings, including dishes, ingredients, and relevant properties. After the Ontology creation, we focused on the implementation of an $\mathcal{EL}$ reasoner, using The $\mathcal{EL}$ Completion Algorithm introduced in the course. The reasoner's functionality will extend to computing subsumers for a given class, with command-line usage.

The latter part of the project embarks on a creative exploration, where we assess the performance of our $\mathcal{EL}$ reasoner. The objective of this phase is to juxtapose the capabilities and efficiency of our $\mathcal{EL}$ reasoner with those of the ELK reasoner. Additionally, we compare the number of subsumers computed by each reasoner as a measure of accuracy. To ensure robust and statistically significant results, we employ a variety of ontologies for testing.

1 An Ontology for a Restaurant

In this section, we present an ontology that we have developed for a restaurant, specifically focusing on burgers. The ontology is structured into three main classes: 'Ingredients', 'NamedBurgers', and 'Type'.

1.1 Class Hierarchy

$$\begin{tikzcd}
 & \boxed{\text{Burger}} \arrow[dl] \arrow[d] \arrow[dr] & \\
\boxed{\text{NamedBurgers}} & \boxed{\text{Ingredients}} & \boxed{\text{Types}}
\end{tikzcd}$$

1.1.1 Ingredients

The 'Ingredients' class is a comprehensive collection of all potential components that can be used to assemble a burger. This includes a variety of buns, meats, vegetables, cheeses, and sauces. The class is further divided into subclasses: Buns, Cheese, Patty, Vegetables, Fruit, and Sauces. Some ingredients, such as Egg and Bacon, do not belong to any subclass to avoid unnecessary categorization. The purpose of this class is to offer a wide array of elements that can be mixed and matched to create diverse burgers. It is important to mention that all ingredients are disjointed with the others, this is key for a correct classification of the burgers afterward.

1.1.2 NamedBurgers

The 'NamedBurgers' class is a subclass of 'Burger' and comprises predefined combinations of ingredients, each with a unique name. For instance, the 'Dominican Burger' is a specific combination of ingredients that is both vegetarian and gluten-free. This class has been designed with six distinct burgers to test all the logic classes defined in Types.

Each burger subclass uses the object properties 'hasIngredient', 'hasBun', 'hasVegetable', 'hasPatty', 'hasSauce', and 'hasCheese' to amalgamate the ingredients into a delicious ensemble. These properties help classify all the ingredients and simplify the construction of logic classes.

Additionally, each named burger includes an expression that explicitly lists all its ingredients, like 'hasIngredient some (Bacon)', 'hasPatty some (Beef)', etc. These are known as qualified existential restrictions, stating that there must exist a relationship (in this case, the 'hasIngredient' or 'hasPatty' relationship) with a specific class (in this case, 'Bacon' or 'Beef').

Moreover, each named burger also includes a 'hasIngredient only (ingredient1, ingredient2 ...)' expression, which is a universal restriction. This states that all ingredients of a NamedBurger must be within the specified set (ingredient1, ingredient2, etc.).

This combination of qualified existential restrictions and universal restrictions ensures that the 'NamedBurgers' class is both complete and accurate in its representation of named burgers and their ingredients.

1.1.3 Type

The 'Type' class is another complex class that categorizes burgers based on certain characteristics. For this purpose combines the 'Ingredients' and 'Type' classes. By creating logical descriptions of the different types we can infer whether a burger is of a certain type based on its ingredients. For example, we define the type Vegetarian indicating that it doesn't contain a chicken or beef patty, nor bacon. Some Types are inferred as subtypes of others, like Vegan, which is a subclass of Vegetarian. Or vegetarian a subtype of Pescatarian. This means that all the burgers that fall into the Vegan type are also Vegetarian burgers, and all vegetarian burgers will be Pescatarian, as Pescatarian is less restrictive. In addition to these three types, we have also created a Gluten-Free type, which means that the burger bun is gluten-free and the burger doesn't contain Seitan. This type does not have any disjointness, however, the last type, Meaty, does. We define this class as any burger that contains Chicken or Beef patty, so this is disjoint with Vegan, Vegetarian, and Pescatarian Types.

The following is an example of a complex class, the Vegan type. This would be its logic description in Manchester syntax:

```
Burger
and (not (hasIngridient some (Bacon or Cheese or Egg)))
and (not (hasPatty some (Beef or Chicken or Fish)))
```

As was said before, Vegan type is inferred as a subclass of Vegetarian by the reasoner, which is correct. We can observe that the logic description of Vegetarian is similar to Vegan but less restrictive:

```
Burger
and (not (hasIngridient some Bacon))
and (not (hasPatty some (Beef or Chicken or Fish)))
```

While defining the burger Types, we found some challenges. In the beginning, the $\neg$ statement did not work correctly so we defined the classes without it. However, these definitions were not correct. For example, saying that a Pescatarian burger is a burger with a Fish patty correctly classified our Fish Burger as Pescatarian but was not a true description, as hypothetical burgers without a patty or with a Fish patty and Chicken patty would be misclassified. After some debugging, we found that some ingredients were not disjoint with the others, and therefore, the $\neg$ statements were not working correctly. For example, we forgot to disjoint Beef with the rest of the ingredients, so the restrictions of not having Beef in the Vegetarian, Vegan, and Pescatarian types were not classifying the burgers correctly.

2 Our Own $\mathcal{EL}$ Reasoner

During the development of our $\mathcal{EL}$ reasoner, we encountered several challenges. Despite numerous attempts, we struggled to make it work with the provided 'dl4python' library, mainly due to difficulties in understanding its usage. As a result, we decided to switch to the 'owlready2' library, which is well-documented and easier to use. This change proved to be beneficial, allowing us to overcome the problems we were facing and successfully implement our $\mathcal{EL}$ reasoner.

2.1 How it works

Our $\mathcal{EL}$ reasoner applies a set of inference rules to decide whether a given ontology O entails a subsumption relationship $C_0 \sqsubseteq D_0$. It does not support the conjunction operator $\sqcap$, negation $\neg$, or universal quantification $\forall$, which are outside the scope of $\mathcal{EL}$.

We maintain a mapping of elements to sets of concepts, as well as a structure to track role successors. Our reasoner applies a series of rules iteratively until no more changes occur:

1. T-rule: If the universal concept $\top$ (represented as 'Thing' in 'owlready2') is not in the set of concepts for an element, it is added.
2. Conjunction rule: If a concept is a conjunction of other concepts, those constituent concepts are added to the set.
3. Existential restriction rule: If a concept is an existential restriction of the form $\exists r.C$, where r is a role and C is a concept, the reasoner checks for an existing r-successor. If none exists, a new one is created.
4. Subclass rule: If a concept is a subclass of another concept, the superclass is added to the set.
5. Role successor rule: For each role successor, if the role is not already assigned to the element, it is assigned.

These rules form the basis of the $\mathcal{EL}$ completion algorithm, a method used in description logic to compute subsumption relationships between concepts. The algorithm works by iteratively applying these rules to an ontology until no more changes occur. It constructs a representation of the minimal model of the knowledge base and is efficient, with a computation time that is polynomial in the size of the ontology [1].

We also include a method to compute the subsumers of a given class, which are the classes that it is a subclass of. This involves finding the class in the ontology, applying the rules iteratively, and then collecting the resulting concepts. Because of the use of 'owlready2' library, we are formatting the output so that it matches the ELK output e.g.: 'Thing' concept is mapped to '$\top$' concept.

2.2 Differences with ELK Reasoner

The ELK reasoner, which supports the OWL 2 $\mathcal{EL}$ profile and the more expressive $\mathcal{EL}+$ description logic, provides a point of comparison for our $\mathcal{EL}$ reasoner [2]. When tested with the pizza ontology, the ELK reasoner was able to retrieve more information for the 'Margherita' class. Specifically, it identified 'Margherita' as a 'CheesyPizza', a detail our reasoner did not retrieve due to lack of support for equivalence axioms.

3 The Creative Part

For this part, we are going to compare the performance of our $\mathcal{EL}$ reasoner with the ELK reasoner. In order to compare them, we will run both reasoners on 8 different ontologies including the Burger ontology previously designed. In addition, we will also test the accuracy of our $\mathcal{EL}$ reasoner as well as ELK in finding subsumes of all classes of said ontologies and compare the results to HermiT reasoner results (ground truth).

3.1 Efficiency

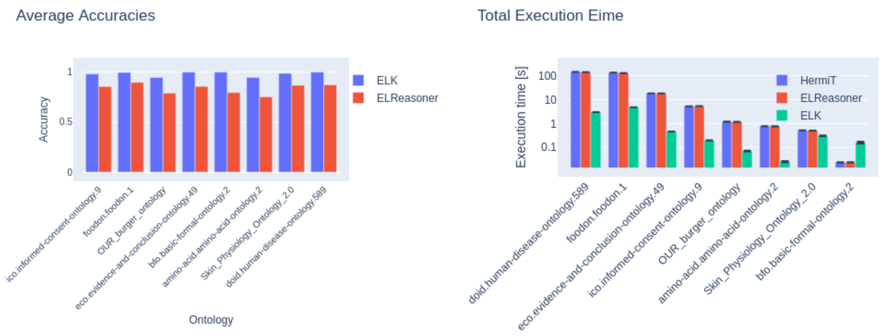

Figure 1: Log normalized total execution time

In the analysis presented in Figure 1, it is evident that the $\mathcal{EL}$ reasoner exhibits reduced performance speed across the majority of the test set's ontologies.

This suboptimal speed can primarily be attributed to the absence of advanced optimization processes within the $\mathcal{EL}$ reasoner. In contrast, the ELK reasoner, benefiting from extensive optimization, establishes a high benchmark in performance, rendering it a formidable competitor in this domain. Additionally, the programming languages utilized for each reasoner—Python for $\mathcal{EL}$ and Java for ELK—may contribute to this disparity. Java is generally recognized for its superior execution speed relative to Python, which could further explain the observed performance differences.

3.2 Accuracy

In evaluating the accuracy as shown in Figure 1 of the custom ELReasoner in comparison with the ELK reasoner, several factors emerge that may contribute to the observed phenomenon where the ELReasoner identifies a greater number of subsumers. Firstly, the rule application and interpretation within the ELReasoner, particularly concerning existential restrictions and role successors, appear to be more inclusive than those in the ELK implementation, potentially leading to a broader identification of subsumers. Additionally, the treatment of equivalents and complex expressions in the ELReasoner, as evidenced in the method *initialize_elements_with_class_and_equivalents*, encompasses a wider range of elements at the initialization stage, which could result in an expanded set of inferred subsumers. The Python and Owlready2 framework, with its unique nuances in processing OWL ontologies, contrasts with ELK's Java-based approach, potentially affecting the outcome of the reasoning process. Furthermore, the less optimized nature of the ELReasoner might contribute to a more exhaustive search for subsumers. This is contrasted by ELK's highly optimized processing, which may exclude certain relations under specific conditions. The handling of complex class expressions and restrictions in the ELReasoner also indicates a comprehensive approach that may identify more intricate subsumption relations. Given these factors, it is possible that the custom ELReasoner, through its distinct implementation and reasoning strategy, finds a more extensive set of subsumers compared to the ELK reasoner.

3.3 Interesting Cases

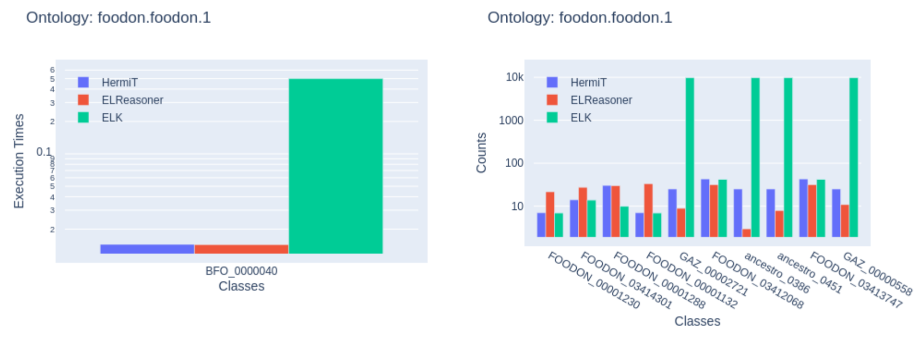

Figure 2: Log normalized total execution time

We identified classes with significant discrepancies in subsumer counts and computation times across the three reasoners. These discrepancies are summarized below.

3.3.1 Food Ontology (FOODON)

In figure 3 (left), we observe that ELK took significantly longer than the other two reasoners. The table below provides a detailed comparison of the subsumed results for each reasoner. Despite taking much longer, ELK returned only the searched class and $\top$, which means no inferred subsumers were found. On the other hand, ELReasoner and HermiT returned pretty much the same set of subsumers, except ELReasoner does not produce $\top$ because it's not a subsumer per se and we filter it out.

```
Ontology: foodon.foodon.1
Results for BFO_0000040:
+------------+----------------------------------------+-------+
| Reasoner   | Subsumers                              | Count |
+------------+----------------------------------------+-------+
| HermiT     | BFO_0000001, BFO_0000002, BFO_0000004, | 5     |
|            | BFO_0000040, T                         |       |
+------------+----------------------------------------+-------+
| ELReasoner | BFO_0000001, BFO_0000002, BFO_0000004, | 4     |
|            | BFO_0000040                            |       |
+------------+----------------------------------------+-------+
| ELK        | BFO_0000040, T                         | 2     |
+------------+----------------------------------------+-------+
```

Figure 3 (right) shows significant count differences between the reasoners in several classes. Table 2 provides a detailed comparison of the subsumed results for each reasoner. Again, both ELReasoner and HermiT produce similar results (again, except $\top$), while ELK fails to compute the NCBITaxon classes.

```
Ontology: foodon.foodon.1
Results for FOODON_00001288:
+------------+---------------------------------------+-------+
| Reasoner   | Subsumers                             | Count |
+------------+---------------------------------------+-------+
| HermiT     | FOODON_00000000, FOODON_00001002,     | 31    |
|            | FOODON_00001015, FOODON_00001057,     |       |
|            | FOODON_00001150, FOODON_00001152,     |       |
|            | FOODON_00001153, FOODON_00001288,     |       |
|            | FOODON_03400361, FOODON_03411140,     |       |
|            | FOODON_03411283, FOODON_03411347,     |       |
|            | FOODON_03411471, FOODON_03411564,     |       |
|            | FOODON_03412067, FOODON_03414224,     |       |
|            | NCBITaxon_1003877, NCBITaxon_1437183, |       |
|            | NCBITaxon_1437201, NCBITaxon_2759,    |       |
|            | NCBITaxon_33090, NCBITaxon_3398,      |       |
|            | NCBITaxon_3650, NCBITaxon_3655,       |       |
|            | NCBITaxon_3656, NCBITaxon_58024,       |       |
|            | NCBITaxon_71240, NCBITaxon_71275,      |       |
|            | NCBITaxon_91835, OBI_0100026, T        |       |
+------------+----------------------------------------+-------+
| ELReasoner | FOODON_00000000, FOODON_00001002,      | 30    |
|            | FOODON_00001015, FOODON_00001057,      |       |
|            | FOODON_00001150, FOODON_00001152,      |       |
|            | FOODON_00001153, FOODON_00001288,      |       |
|            | FOODON_03400361, FOODON_03411140,      |       |
|            | FOODON_03411283, FOODON_03411347,      |       |
|            | FOODON_03411471, FOODON_03411564,      |       |
|            | FOODON_03412067, FOODON_03414224,      |       |
|            | NCBITaxon_1003877, NCBITaxon_1437183,  |       |
|            | NCBITaxon_1437201, NCBITaxon_2759,     |       |
|            | NCBITaxon_33090, NCBITaxon_3398,       |       |
|            | NCBITaxon_3650, NCBITaxon_3655,        |       |
|            | NCBITaxon_3656, NCBITaxon_58024,       |       |
|            | NCBITaxon_71240, NCBITaxon_71275,      |       |
|            | NCBITaxon_91835, OBI_0100026           |       |
+------------+----------------------------------------+-------+
| ELK        | FOODON_00000000, FOODON_00001002,      | 10    |
|            | FOODON_00001015, FOODON_00001057,      |       |
|            | FOODON_00001150, FOODON_00001152,      |       |
|            | FOODON_00001153, FOODON_00001288,      |       |
|            | FOODON_03400361, T                     |       |
+------------+----------------------------------------+-------+
```

The discrepancies observed could be due to the different levels of expressivity of the reasoners. ELK is based on the $\mathcal{EL}+$ description logic, while HermiT is based on the more expressive ALC description logic. ELReasoner, being our own code, is not as advanced as ELK or HermiT. This could explain why ELK and ELReasoner fail to compute certain classes that HermiT can handle.

3.3.2 Skin Physiology Ontology

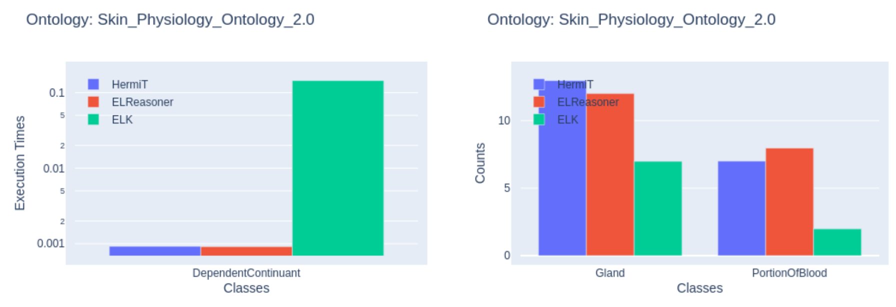

Figure 3: Log normalized total execution time

The table below provides a detailed comparison of the subsumed results for the class 'DependentContinuant' for each reasoner. All three reasoners returned the same set of subsumers, except ELReasoner does not produce $\top$ by design. Despite taking much longer, ELK returned the same set of subsumers as HermiT and ELReasoner. This could be due to the complexity of the ontology or the expressivity of the $\mathcal{EL}+$ description logic that ELK is based on.

```
Ontology: Skin_Physiology_Ontology_2.0
Results for DependentContinuant:
+------------+------------------------------------------+-------+
| Reasoner   | Subsumers                                | Count |
+------------+------------------------------------------+-------+
| HermiT     | Continuant, DependentContinuant, Entity, | 4     |
|            | T                                        |       |
+------------+------------------------------------------+-------+
| ELReasoner | Continuant, DependentContinuant, Entity  | 3     |
+------------+------------------------------------------+-------+
| ELK        | Continuant, DependentContinuant, Entity, | 4     |
|            | T                                        |       |
+------------+------------------------------------------+-------+
Ontology: Skin_Physiology_Ontology_2.0
Results for Gland:
+------------+------------------------------------------+-------+
| Reasoner   | Subsumers                                | Count |
+------------+------------------------------------------+-------+
| HermiT     | AnatomicalStructure, ApocrineGland,      | 13    |
|            | CardinalTissuePart, Continuant, Entity,  |       |
|            | Gland, GlandOfEpidermis,                 |       |
|            | IndependentContinuant, MerocrineGland,   |       |
|            | PhysicalObject, SubdivisionOfEpidermis,  |       |
|            | Sudoriferous_gland, T                    |       |
+------------+------------------------------------------+-------+
| ELReasoner | AnatomicalStructure, ApocrineGland,      | 12    |
|            | CardinalTissuePart, Continuant, Entity,  |       |
|            | Gland, GlandOfEpidermis,                 |       |
|            | IndependentContinuant, MerocrineGland,   |       |
|            | PhysicalObject, SubdivisionOfEpidermis,  |       |
|            | Sudoriferous_gland                       |       |
+------------+------------------------------------------+-------+
| ELK        | AnatomicalStructure, Continuant, Entity, | 7     |
|            | Gland, IndependentContinuant,            |       |
|            | PhysicalObject, T                        |       |
+------------+------------------------------------------+-------+
```

Similarly, for the 'Gland' class, as shown in the above table, ELReasoner and HermiT again produce closely matching results, barring the $\top$. However, ELK appears to encounter difficulties with certain classes, failing to compute them. This highlights the varying performance of the reasoners when faced with different classes and ontologies.

4 Conclusion

This research project has presented an in-depth exploration of knowledge representation and reasoning within the context of a Burger restaurant ontology. Throughout this study, the development and comparative analysis of an $\mathcal{EL}$ reasoner and the ELK reasoner has provided significant insights into the intricacies of ontology reasoning.

Our custom $\mathcal{EL}$ reasoner, despite its less optimized nature and implementation in Python, has shown a notable capability in finding a broader range of subsumers in various ontologies when compared to the ELK reasoner. This is attributed to its inclusive approach to rule application, particularly with existential restrictions and role successors, and its comprehensive handling of equivalents and complex expressions. However, the performance evaluation revealed that the ELK reasoner, with its advanced optimization and implementation in Java, markedly surpasses our reasoner in execution speed.

The project also underscores the challenges and learning opportunities in developing and refining ontological systems. The initial hurdles with the 'dl4python' library and the subsequent transition to 'owlready2' highlight the importance of selecting appropriate tools in ontology development.

In conclusion, while the $\mathcal{EL}$ reasoner developed in this project demonstrates a promising foundation, it necessitates further refinement to enhance its efficiency and accuracy. The insights gained from this comparative study contribute to the broader understanding of description logic reasoners and pave the way for future advancements in the field of ontology reasoning and knowledge representation.

References

[1] R. Penaloza and A.-Y. Turhan, "Towards Approximative Most Specific Concepts by Completion for EL with Subjective Probabilities."

[2] Y. Kazakov, M. Krötzsch, and F. Simančík, "The Incredible ELK," *Journal of Automated Reasoning*, vol. 53, no. 1, pp. 1–61, Jun. 2014. [Online]. Available: https://doi.org/10.1007/s10817-013-9296-3