A Novel Ontology of an Indonesian Kitchen in Description Logic

November 25, 2023

Abstract

This article explores the application of $\mathcal{EL}$ in creating an ontology for an Indonesian kitchen. The ontology represents various aspects of the kitchen, including different cuisines, ingredients, and their relationships. We employ $\mathcal{DL}$, specifically the $\mathcal{ALC}$s logic, to address challenges in knowledge representation and leverage the expressivity and flexibility it offers. The ontology is implemented using Protégé, and reasoners like HermiT and ELK are compared in terms of their performance and expressivity. The research investigates the running time and inferred subsumers on multiple ontologies, shedding light on the trade-offs between expressivity and computational efficiency in different scenarios.

1 Introduction

Knowledge representation (KR) is a crucial aspect of artificial intelligence (AI) and intelligent systems. It serves as a bridge between the system's understanding of the world and its ability to perform tasks or make decisions based on that understanding. An efficient KR model should be able to represent all the necessary information about the problem domain and should be expressive enough [1]. The design of the KR modal generally faces challenges in such as the intricate nature of the problem domain, the critical role of data curation influencing model performance, and the the difficulty in ensuring explainability and interpretability [2].

We applied Description Logic (DL) to overcome the difficulties in KR model design. Description Logic (DL) is a formal language for expressing knowledge about concepts, roles, and individuals. The fundamental modeling concept of a DL is the axiom — a logical statement relating roles and/or concepts [3]. DL is advantageous in KR model design due to its high expressivity, flexibility, and common usage in AI and Semantic Web, as DL provides a logical formalism for Web Ontology Language (OWL) and its profiles.

In this project, we designed an Ontology of an Indonesian Kitchen, by referencing a menu of an Indonesian Restaurant in Amsterdam [4]. The Ontology is programmed with Protégé, a commonly known OWL editor. We implemented the basic structure of a restaurant menu of an Indonesian Kitchen and defined respective class axioms and

relations of courses of cuisines and their ingredients, where relations between each cuisine and their ingredients were expressively, or implicitly given. For instance, we defined any saté-meal as a meal that has a saté base and some spicy level (e.g.: mild).

Several Ontology reasoners were programmed to comply with Ontology and to assign individuals and (sub)classes to their respective (super)classes based on the predefined logical implication rules and relations. Each reasoner has their own benefits and limitations. For instance, HermiT is an OWL reasoner that is designed for ALC model interpretation, supports all features of the OWL, as well as several specialized reasoning services such as class and property classification, and correctly performs both object and data property classification that are not fully supported by other OWL reasoners [5]; EL is an OWL reasoner that is designed for EL model interpretation [6]. Due to the fact that HermiT can handle more complicated DL rules compared to ELK, we are interested in what extent HermiT Reasoner is more beneficial compared to the ELK Reasoner, in terms of expressivity and time complexity. This project will compare the expressivity of the Indonesian Kitchen Ontology by executing HermiT and ELK reasoners and discuss the performance of both reasoners on Ontology. The experimental design and the results will be discussed in Section 4 and Section 5.

2 The Ontology Of an Indonesian Kitchen

2.1 Introduction

This paper's section deals with the creation of the Indonesian kitchen ontology (version 1) as created by project group 28 of the 2023 Knowledge representation course at the Vrije University Amsterdam. The Ontology is licensed under a Creative Commons Attribution 4.0 International License. Currently the ontology is stored on a private GitHub and is only accessible on request until a need for public access is established. Any version changes will be annotated, and old versions will remain present in this GitHub repository. The ontology was reported under MIRO guidelines (Minimum Information for Reporting an Ontology) [7].

2.1.1 Motivation and aim

The primary objective of this ontology is to provide comprehensive support for Indonesian restaurants, particularly those catering to takeout services or "tokos" as they are referred to in the Netherlands. The aim was to create an intelligent system that contains an overview of the different types of meals a restaurant offers, as well as knowledge about the ingredients and cooking or serving styles. The ontology should be able to reason about the dietary contents of meals as well as to intelligently answer specific queries from customers that want to order something specific concerning taste or ingredients. The ontology should have a logical structure for Indonesian takeaway so the main focus should be in the meals and the sides. It has to contain a wide variety of sides and a meal is solely classified as any combination of sides. Certain specific combinations of sides should be categorised as specific types of meals. In its creation, a pizza menu ontology was used as the foundation. It can be found at and is one of the example ontologies in Protégé [8]. This ontology indicates the menu of a restaurant concerning pizzas and is in various ways similarly structured to an Indonesian takeaway restaurant as both pizza's

and toko meals have a rather freeform menu where may different types of ingredients can be combined to form a great range of possible meals.

2.1.2 Knowledge acquisitions

The knowledge for the ontology was acquired from the website of Toko Eddo, a classic Indonesian toko [9]. The restaurant has a wide variety of ingredients and meal types that form a viable foundation for an ontology based on Indonesian takeaway culture. The prioritisation of gathered entities was based on a balance between completeness and usability. The chosen entities were solemnly based on the menu of Tokko Eddo. Specifically, their sides and main meals as these two components are the main unifying mark of tokos as discussed before. After analysing their menu and performing a cursory survey of other toko's it was concluded that Toko Eddo contained all primary contents of a toko. These primary entities were supplemented in their menu by less common or specialised elements. These were also added to the project's ontology so it would entail a complete ontology for Toko Eddo. This makes the ontology immediately usable. Furthermore, the creation of the ontology is expedited by the limitation of a singular restaurant. Although, the specialties of sides and meals of other restaurants should in theory be easy to add in the future.

2.2 Ontology content

2.2.1 Metrics

The ontology was created in Protégé version 5.6.2 which uses the OWL 2 Web Ontology Language Manchester Syntax (Second Edition) [10]. The final ontology metrics can be seen in 1. The following types of semantics were used in the ontology and, or, some, only, min, not. Five complicated axioms were present in the final ontology. No individuals were initialised.

Metrics	
Axiom	316
Logical axiom count	221
Declaration axioms count	89
Class count	83
Object property count	6
Data property count	0
Individual count	0
Annotation Property count	3
Class axioms	
SubClassOf	180
EquivalentClasses	16
DisjointClasses	12
GCI count	0
Hidden GCI Count	16

Figure 1: The metrics of the Indonesian kitchen ontology

2.2.2 Upper ontology

The upper ontology consists of a main category called meal which is the heart of the ontology 2. The sides are divided into meat-based sides, vegetarian sides and bases. The meat sides are further defined by their meat type (pork, beef, fish or chicken). The vegetarian based sides were divided by a vegetables class and a meat alternative class. All mentioned classes are disjoint from their sibling classes. The last two side classes are sate_side and spicy_side. These are in essence a list of ingredients that have either have a spiciness of spicy or a style of sate. This would make it easier for customers to decide on their choices of sides and in the definition of meals. Lastly, there is the class value partition which contains the ranges for the relationships descried in the next paragraph.

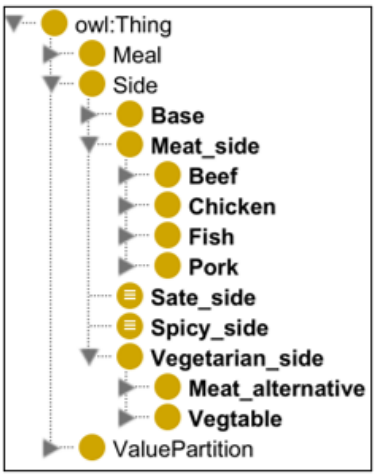


Figure 2: upper ontology of the Indonesian kitchen ontology

2.2.3 Relations

There were four main relationship types in the ontology. Two of them were used to add information to sides. Has_spiciness was utilised to define an ingredient's spiciness as either mild or spicy. Has_style was used to denotes the serving style of a side as being either sate or regular. The third main relation ships is has_ingredient and its inverse is_ingredient_of which denotes what kind of ingredients a meal should contain and in what meals the ingredients are used in respectively. There is also an obsolete relationship called Is_cooked which would give the opportunity to specify in what ways an ingredient is cooked (grilled, fried, boiled, or steamed). This relation was not implemented as toko Eddo doesn't specify their cooking methods. However, it was added as a framework for future additions.

2.2.4 Base

Indonesian meals often have a starch ingredient as foundation or base. Examples are rice, yellow rice, mihoen, or bami. All subclasses are disjoint from each other. Since bases don't contain meat, it was considered to let bases become a subclass of vegetarian_side,

but this was not implemented on the basis of the structure of the Indonesian meal. A clear distinction is made in tokos between bases, meat sides and vegetarian sides. It was found that keeping this distinction was beneficial for meal type definitions. This choice does have an effect on the definition of vegetarian meals which will be described under the section "meal".

2.2.5 Meat_side

The meat_side was directed into four categories based on meat types. This was chosen as there is a common preference for meat types for taste and they play a distinct role in some dietary restrictions due to, for example, faith.

2.2.6 Vegetarian_side

Vegetarian_side has two subclasses: meat_alternative and vegetables. It was chosen to make this distinction, because it is a common practise in tokos to allow the swapping of a meat side by a meat alternative for established meal types. This gives the opportunity to make any meal vegetarian. By making this distinction, this common practise could be easier implemented in the ontology.

2.2.7 Meal

The meal subclass is roughly divided in three categories. One part is defined by simple information about the sides that are included in the meal. For example, are there any spicy sides present or are all sides vegetarian. A note should be made for the definitions that are based on exclusion of a class, specifically the non-pork class, the pescetarian class, and the vegetarian class. For any meal to be categorised as a subclass of these meals, the included sides must be designated as the only sides present in the meal. Otherwise, you run into problems with the open world assumption. The second part of the meal class contain four complicated types of meals and is the true heart of the ontology. These meal descriptions are meals that are commonly found in any toko: special, rames, complete, and rijsttafel. These definitions are primarily different in the number of sides that are present and the presence of a sate ingredient. For these definitions cardinality was used, specifically the the min operator was implemented to avoid clashes with the open world assumption. During experiments were the min was replaced by exactly or max as alternatives it required that every meal did not only specify which sides were present but also the number. By avoiding max and exactly we could keep the definitions of meals simpler. Which leads to the final category: signature meals. These are examples of meals. All these signature meals together use all the meal descriptions of the other two categories and it gives an example for each meal type and shows that the ontology correctly classifies meals.

3 The Reasoner

In this section, we provide a high level overview of the what EL is and a brief overview of how we implemented it.

3.1 $\mathcal{EL}$

Description Logics ($\mathcal{DL}$s) define a way to create and reason a knowledge representation. As a subset of First Order Logic, they guarantee a mathematically correct inference of new knowledge from the data it is run on. They are also used to find otherwise easily overlooked contradictions.

$\mathcal{EL}$s are a subset of $\mathcal{DL}$s that provide an (agreeably) optimal trade-off between computability and expressivity. There are some widely used $\mathcal{EL}$ ontologies such as the SNOMED CT—a large medical ontology. The most notable restrictions is the lack of $\neg \sqcap \exists$. While this may seem too restrictive, it guarantees that every $\mathcal{EL}$ ontology is consistent and coherent.

3.2 $\mathcal{EL}$ Subsumption algorithm

Our $\mathcal{EL}$ reasoner follows the algorithm outlined in 1. Please note that this is seriously simplified, and only exists to provide an understanding of the logic.

Algorithm 1 $\mathcal{EL}$ Reasoner pseudocode

```
1: concept ← GetConcept(< userInput >)
2: allConcepts ← GetAllConcepts(< ontology >)
3: Box ← CreateNewInterprentation()
4: Box.add(concept)
5: for Each concept d in Box do
6:     for Each concept c in allConcepts do
7:         conceptType ← c.getConceptType()
8:         if conceptTypeis⊓ then
9:             if d has L ⊓ R then
10:                Add L to d
11:                Add R to d
12:            else if (d has L) & (d has R) then
13:                Add L ⊓ R to d
14:            end if
15:        else if conceptType is ∃ then
16:            if d has ∃r.C & there is no r-successor with C assigned then
17:                Add a new r-successor to d and assign C to it
18:            else if d has an r-successor with C assigned then
19:                Add ∃r.C to d
20:            end if
21:        else if conceptType is ⊑ then
22:            if d has C assigned and C ⊑ D then
23:                Assign D to d
24:            end if
25:        else if conceptType is T then
26:            Add T to d
27:        end if
28:    end for
29: end for
```

3.3 Implementation Details

To realize this reasoner, some key data structures had to be made. Existential relations were represented as *Relations*. Relations are a class that hold a name and point to some *Individual*. An *Individual* stores *Relations*, concepts, and some information about itself. All *Individual*s are stored in a class *Box*. Each of these classes have helper methods such as overload operators and getters/setters to help facilitate modifications to the *Box* object.

3.3.1 Class Definitions:

This section quickly overviews the purpose of each class and their functionality:

Concept: Represents a concept and provides a Python wrapper for complex Java objects. It includes methods for string representation, equality, hashing, and attribute forwarding to the Java object.

Individual: Represents an individual (instance/node) with concepts, relations, and a name. It includes methods for adding concepts and relations, checking concept existence, and getting individuals based on relations and concepts.

Relation: Represents a relation from an individual to other individuals. It includes methods for string representation, equality, hashing, and attribute definition.

Box: Stores a collection of individuals to create a CGI (Conceptual Graph Interchange). It includes methods for adding individuals, getting individuals based on concepts, and generating a new random individual.

4 Research Question and Methodology

This project aims to answer the following research question:

- "Given multiple Ontologies randomly that both follow the ALC logic and purely EL Logic, to what extent the HermiT Reasoner is more beneficial in expressivity and in time complexity, compared to the ELK Reasoner?"

The objective of the research is to discover how much more implication the HermiT Reasoner could infer, and how much the running time could differ, compared to the ELK. First, we execute the HermiT Reasoner on Indonesian Restaurant Ontology, and on multiple Ontologies found on the web that vary in size and expressivity. Second, we execute the ELK Reasoner on the same Ontologies. Subsequently, we compare the results of the inferences, specifically, the number of subsumers returned for ecah class axiom, and the running time of each ontology executed by both Reasoners. Table 1 illustrates the size and the type of Ontologies applied in this research, note that Ontology No.1 is the Indonesian Kitchen Ontology programmed by us.

In this research, we used Protégé to measure the running time of each Ontology and used a Python-based Reasoner to calculate the total number of subsumers inferred by both ELK and HermiT. Two versions of ELK Reasoners were built within the Protégé: the *ELK 0.4.3* and the *ELK 0.5.0*. Theoretically, *ELK 0.5.0* Reasoner has identical performance in expressivity and a faster running time, compared to *ELK 0.4.3*. Only Ontologies that are not purely EL are used for expressivity comparison.

Onntology No.	Topic	Size(KB)	Number Of Concepts	Purely EL
1	Indonesian Kitchen	111	191	No
2	Animo Acid	58	111	No
3	Basic Formal Ontology (BFO)	14	35	Yes
4	Ecosystem Ontology (ECSO)	2946	8010	No
5	Human Disease	2181	8236	Yes
6	Skin Physiology	241	584	No
7	Food	3836	11782	No
8	Clinical Trials	144	376	Yes
9	Evidence and Conclusion (ECO)	562	1213	Yes
10	Informed Consent Ontology (ICO)	240	739	No
11	Kinetic Simulation Algorithm Ontology (KiSAO)	175	517	No
12	ontoDom	572	842	No

Table 1: A Summary of Ontology applied in this study

The running time of each Reasoner is not static and is dependent on various factors. For instance, the Operation System, the time of executions, and the complexity of an Ontology. The running time of the first execution of a Reasoner is generally longer and slightly decreases when executing again. Therefore, we took the average running time of 10 executions as the running time result to cancel the possible confounded variables that could impact the running time.

5 Results and Discussion

5.1 Running time

Figure 3 illustrates the result of running test of each Reasoner. The first impression of the result was that the running time of HermiT on *Indonesian Kitchen*, *ECSO*, *FoodOn*, *ICO*, *Skin Physiology* and *OntoDom*, was significantly higher than the ELK (we took the result of ELK0.5.0 as representative for two reasons: 1. this is a later version of the ELK reasoner; 2. the performance among two versions vary, but none of the version significantly outperforms another). This was due to the fact that these six Ontologies either had a significantly larger size and number of concepts or more complicated inference rules that were out of the limitation of the ELK. For instance, HermiT inferred 109 more rules that were out of the EL limitations on *ontoDom*, but only 5 more on*ICO*; and 25 more on *Indonesian Kitchen* but only 1 more on *Animo Acid*.

An interesting finding was that in most of the cases, HermiT had a longer running time than the ELK; however, HermiT outperformed the ELK in certain cases, these cases were found on *Animo Acid* and *BFO*, while both Ontologies had a relatively small size

and fewer number of concepts, *BFO* was a purely EL ontology, and *Animo Acid* with only 1 inference difference. Similarly, the performance of Hermit on *Climate Trials*, which was a purely EL Ontology, and the size and the number of concepts were nearly two times as the *BFO*, was observed as fast as the ELK reasoner - the running time difference is 7.2 ms which was neglectable. In other cases observed in this research, when the Ontology did not have a significantly large size, or when an Ontology had a significantly large size but was purely EL (e.g.: the *Human Disease Ontology*), the running time of HermiT was generally two times longer than the ELK Reasoner.

5.2 Expressivity

Figure 4 and figure 5a illustrates the difference of expressivity on non-purely-EL Ontologies (due to the limitation of the Reasoner, the Ontology *KiSAO* and *ECSO* were excluded from the result of expressivity). 6 Ontologies were used for expressivity study, respectively: *Indonesian Kitchen*, *Animo Acid*, *Skin Physiology*, *ICO*, *FoodOn* and *OntoDom*. On the sample of 6 cases overall, HermiT was able to return 199506 (183559 from *FoodOn*) inferences in total, and 199536 (183526 from *FoodOn*) inferences by ELK. HermiT returned 183 more inferences than ELK, meaning that the inference result of HermiT was observed approximately 1% more expressive than the inference result of ELK.

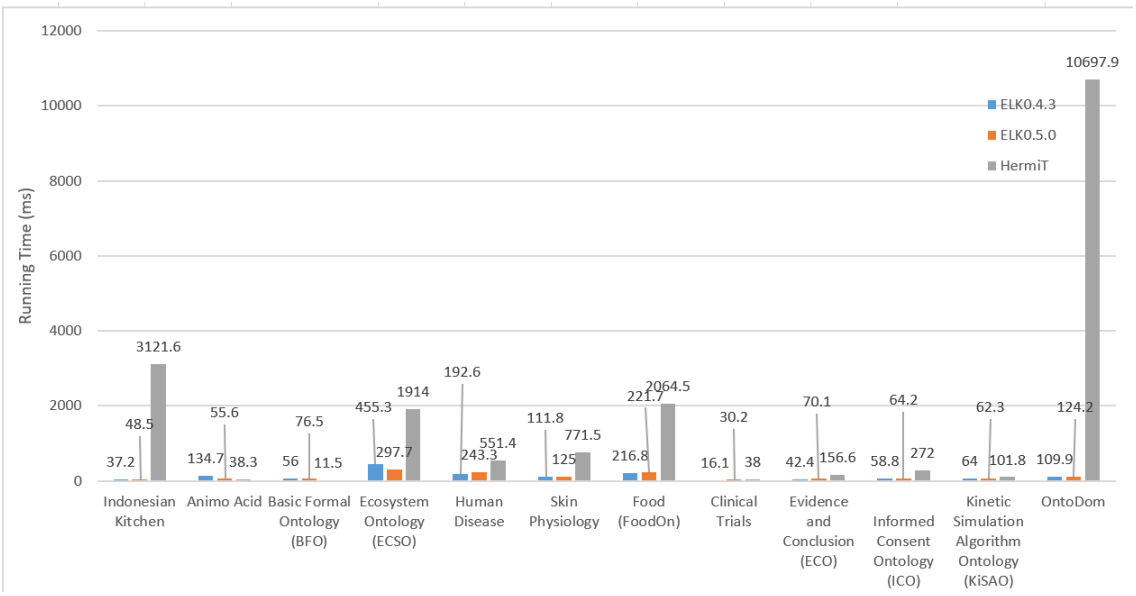


Figure 3: The running time of ELK and HermiT on each Ontology

For instance, in our *Indonesian Kitchen* Ontology, Hermit was able to return the following class hierarchies that ELK was not able to do: the class *Pescatarian meal* was inferred as a subsumer of *vegetarian meal* by HermiT, as *vegetarian meal* was defined as a meal that had no meat side and *Pescatarian meal* was defined as a meal that had no specific type of meat except fish. Hence, *Pescatarian meal* had a broader but included

definition than *Vegetarian meal*. *Rames* was a meal that had at least 1 Base, 1 vegetable, and 1 meat; a *Complete* was a meal that had at least 1 Base, 2 vegetables, and 2 meats. Therefore, *Rames* was inferred as a subsumer of *Complete* (A complete meal with 1 base, 2 vegetables, and 2 meats was considered as a Rames, but a Rames that had 1 base, 1 vegetable, and 1 meat was not necessarily a Complete meal). Similarly, *Rijsttafel* was a meal that had some sate side and had a minimum of 2 bases, 2 ingredients, and a minimum of 3 vegetables and 3 meats; *Special* was a meal that had some base and some Sate. Therefore, the *Special* class was considered a subsumer of *Rijsttafel* due to its broader coverage.

Figure 5b summarizes the number of Subsumers of purely-EL Ontologies that both Reasoners returned. It was observed that on the Ontologies selected in this study, when the complexity of inference grew, HermiT heavily sacrificed the running time in return for a more expressive result. From a macro perspective, Hermit took 16326.6 ms longer to return 183 more inferences in total, meaning that in order to process one more expressive inference, HermiT would consume 89.2 ms more. On the contrary, when Ontologies were purely EL, both Reasoners returned 79501 inferences in total, and HermiT took solely 337.4 ms longer than ELK, meaning that Hermit would consume only less than 0.004 ms more on each inference than ELK.

Figure 4: The difference of expressivity of non-purely-EL Ontologies

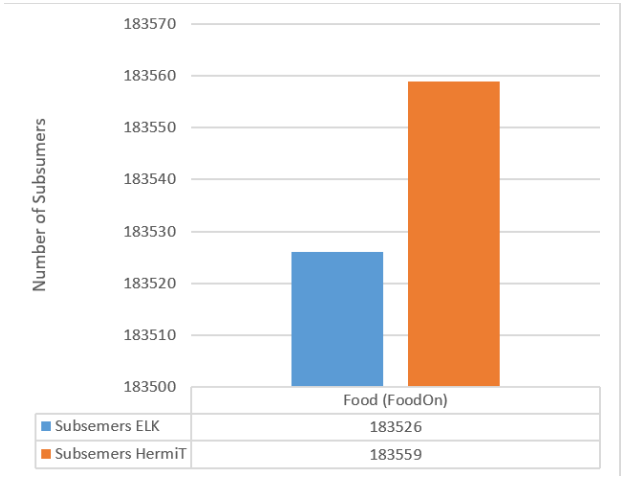
6 Conclusion

To conclude, this paper investigated the application of $\mathcal{EL}$ in creating an ontology for an Indonesian kitchen. The ontology was implemented using Protégé, and HermiT and ELK reasoners were compared in terms of their performance and expressivity.

In terms of running time, HermiT generally showed longer execution times compared to ELK, especially on larger and more complex ontologies. However, there were cases, such as with smaller and purely EL ontologies, where HermiT's performance was comparable to ELK. The running time difference varied across ontologies, suggesting that the choice between HermiT and ELK may depend on the specific characteristics of the ontology being processed.

In terms of expressivity, HermiT returned slightly more inferences compared to ELK on non-purely EL ontologies. This indicates that HermiT may have a slightly higher level of expressivity in certain scenarios, but this comes at the cost of longer running times.

Ultimately, the choice between HermiT and ELK depends on the specific requirements and characteristics of the ontology. If the ontology is relatively small, purely EL or the rich expressivity is not considered a priority, and the focus is on running time efficiency, ELK may be a suitable choice. On the other hand, if the ontology is larger, has more complex inference rules, and the priority is on expressivity, HermiT could be considered. The trade-off between running time and expressivity should be carefully evaluated based on the specific needs of the application.



(a) Expressivity comparison on FoodOn

Onto Name	Subsemers ELK and HermiT
Basic Formal Ontology (BFO)	186
Human Disease	68926
Clinical Trials	3550
Evidence and Conclusion (ECO)	6839

(b) Overall number of Subsumer returned by both Reasoners

Figure 5: Expressivity of foodOn and the number of Subsumers of purely-EL Ontologies

References

[1] "Knowledge representation in ai," https://www.javatpoint.com/knowledge-representation-in-ai, accessed: Insert Date Here.

[2] J. Borrego-Díaz and J. Galán Páez, "Knowledge representation for explainable artificial intelligence," *Complex Intell. Syst.*, vol. 8, pp. 1579–1601, 2022. [Online]. Available: https://doi.org/10.1007/s40747-021-00613-5

[3] F. Badie, "A description logic based knowledge representation model for concept understanding," in *Agents and Artificial Intelligence*, J. van den Herik, A. P. Rocha, and J. Filipe, Eds. Cham: Springer International Publishing, 2018, pp. 1–21.

[4] Toko eddo afhaal menu. [Online]. Available: https://www.tokoeddo.nl/afhaal-menu

[5] B. Glimm, I. Horrocks, B. Motik, and et al., "Hermit: An OWL 2 reasoner," *Journal of Automated Reasoning*, vol. 53, pp. 245–269, 2014. [Online]. Available: https://doi.org/10.1007/s10817-014-9305-1

[6] Y. Kazakov, M. Krötzsch, and F. Simančík, "The incredible ELK," *Journal of Automated Reasoning*, vol. 53, pp. 1–61, 2014. [Online]. Available: https://doi.org/10.1007/s10817-013-9296-3

[7] M. J. M. C. e. a. Matentzoglu, N., "Miro: guidelines for minimum information for the reporting of an ontology." *MJ Biomed Semant*, vol. 9, p. 6, 2018. [Online]. Available: https://doi.org/10.1186/s13326-017-0172-7

[8] "Pizza.owl," https://protege.stanford.edu/ontologies/pizza/pizza.owl, accessed: 2023-11-13.

[9] "toko eddo," https://www.tokoeddo.nl/, accessed: 2023-11-15.

[10] M. M. P. Team, "he protégé project: A look back and a look forward. ai matters." 2015.