UNIVERSITY OF CAPE TOWN

DEPARTMENT OF COMPUTER SCIENCE

CS/IT Honours Final Paper 2020

Title: A Tool for managing the Materialization of Deductions in a Conceptual Data model

Author: Mandisa Baleni

Project Abbreviation: KnowDat

Supervisor(s): Prof. Maria Keet

Category	*Min*	*Max*	Chosen
Requirement Analysis and Design	*0*	*20*	20
Theoretical Analysis	*0*	*25*	0
Experiment Design and Execution	*0*	*20*	0
System Development and Implementation	*0*	*20*	20
Results, Findings and Conclusions	*10*	*20*	10
Aim Formulation and Background Work	*10*	*15*	10
Quality of Paper Writing and Presentation	*10*		10
Quality of Deliverables	*10*		10
Overall General Project Evaluation (*this section allowed only with motivation letter from supervisor*)	*0*	*10*	0
Total marks		**80**	

A Tool for managing the Materialization of Deductions in a Conceptual Data Model

Mandisa Baleni
Computer Science Department
University of Cape Town
blnman002@myuct.ac.za

ABSTRACT

Ontologies are generally used for the semantic web and subject domain applications to capture knowledge in order to facilitate informed decision-making. A proposed architecture KnowID devised a series of steps to manage knowledge and produce some ontology or conceptual data model defined by EER features. The final subprocess in this knowledge management sequence is the materialization of deductions tasked with making implicit knowledge within a model explicit. The proceeding step detects these inferences so the ontology can be edited with them. An expressive EER is expected as output. This report documents an extension of an existing ontology-editing software tool, Owlready, aimed at realizing EER-tailored materialization of deductions on a model and a wrapper function to verify the inferred edits. Its implementation and evaluation yielded a system able to materialize deductions for a range of EER features and provide robust verification services for the resulting inferences to users.

CCS CONCEPTS

• **Theory of computation → Semantics and reasoning**
• *Knowledge-driven databases*

KEYWORDS

OWL 2, description logic, reasoning, EER

1 INTRODUCTION

Ontologies or conceptual data models are used by intelligent systems to manage data more efficiently by several applications [1]. One of the applications is being applied to the Semantic Web, considered as the next generation of the World-Wide Web [2], where data is envisioned as a linked web. The idea is to express knowledge as an ontology or conceptual data model with inter-related concepts done in such a way that the knowledge represented is expressive about the subject domain of which it interprets. Another application of ontology modelling is to aid decision-making in subject domain applications like medicine practice. To do this, the way in which knowledge (the mere representation of a subject domain) is linked to the actual data of a domain has to be considered [1].

A theoretical architecture called KnowID was proposed as seen in figure 1. Its purpose is to carry out a 'knowledge-to-data' pipeline process by using a set of transformation rules instead of using a costly mapping layer [3]. KnowID has a knowledge and information management layer with several components that are responsible for several mechanisms that contribute to expressively outputting an enhanced-entity relationship diagram (a conceptual data model known as EER) which is then used for query formulation as well as transformations to and from an abstract relational model (ARM) in the data management layer. The data management layer handles how the actual data -represented in the knowledge layer- is stored, queried and managed.

We will look broadly at the functionality of the knowledge and information management layer in figure 1. It has four main processes. It starts by (1) taking in either conceptual data models or application ontologies like Web Ontology languages (OWL). For the scope of the project, we will consider OWL models because it is already necessarily formalized in logic. (2) The second component then proposes to formalize a given model into some logic if it is not already so that (3) it can be reasoned over with logic reasoners which is the process of inferring implicit information in the model. This process is achieved by the classification component where classes and relations amongst them in the models can be appropriately classified. (4) The final component, materialization of deductions, is responsible for editing the model to add or remove any of the deductions made from the reasoning step so that the implicit information becomes explicit [1].

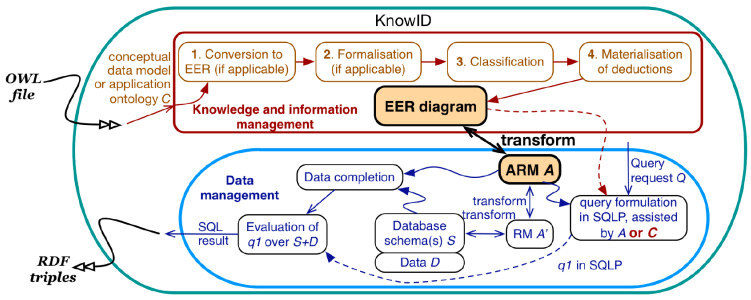

Figure 1: KnowID architecture [3].

The KnowID tool requires the output model of the final materialization of deductions step in the knowledge layer to be an

EER, a graphically-notated diagram. The choice for this model is because its mapping over relational models is popular because it does so well and it is also largely used in the database community [1]. In particular, KnowID maps the EER to an abstract relational model (ARM) which is a more general form of a relational model with object identifiers for entities and associated attributes for relations in order to realize more constructs like disjoint constraint, explicit inheritance and more [1] The ARM or the resulting EER can thus be used to bridge the knowledge and data layer with and aim to provide a graphical knowledge model in a GUI and store it in JSON files so that graphically informed queries can be made [1].

Reasoning tools require models that are formalized in a logic so that the logic can be used by their algorithms to make necessary inferences using description logics (DL) [4], first-order principals [5] or other logic semantics. EER diagrams or ones expressed in a conceptual data model are not formalized in a logic and therefore must be interpreted in terms of some logic. Most available reasoning tools are compatible with OWL constructs and other direct logic semantics. Consequently, existing tools that encapsulate some of the functionality of the materialization of deductions are generally based on OWL constructs. Such existing tools include the Intelligent Conceptual Modelling tool and methodology (ICOM) [6], the OWL API [7], Owlready [8] and previous iterations of Protégé [9]. Although they are able to materialize some deductions, they are not tailored specifically for models that encapsulate EER features. Furthermore, there is little capability for data model validation where users can confirm or refuse deductions based on their sensibility in the real world. Validation also makes models more useful and accurate for its respective subject domain.

1.1 Project Aim

We proposed a software to focus on the implementation of the materialization of deductions component regarding the issues outlined above. The system aimed to address the following:

(1) **Aim:** Materialize deductions that more closely match OWL constructs to EER features.
(2) **Aim:** Provide wrapper functionality to help users validate the ontology edits.

The following section 2 will briefly clarify non-intuitive interchangeable terminology used in ontologies and data models. It will mainly look more closely at related tools and software that use or once used reasoners to materialize deductions as well as the extent to which they did. It will also address the shortcomings of validation amongst them. Section 3 will discuss systems development and design. More specifically, the development methodology and framework, architecture and more. Section 4 will look the system and testing implementation. Section 5 describe result, section 6 will discuss them section 7 will mention ethical and legal issues and sect. Finally, section 8 concludes with main achievements addressed and future work to be considered.

2 RELATED WORKS

2.1 Terminology

The constructs used in ontologies and conceptual data models tend to have differing terminology that express the same concepts. Terminology will also be used interchangeably in this paper. Object properties in OWL are synonymous for relationships and relations in EER whilst OWL's data properties represent EER attributes. Classes and entities express the same concept. Classification and subsumption refer to subclassing and class consistency refers to whether an instance of a class can be instantiated. Lastly inference and deductions will also be used interchangeably.

2.2 Protégé

One of the most prominent existing tools that performed the process of editing ontologies with inferences was an earlier version of Protégé [9]. An OWL Plugin was extended for Protégé in order to modify OWL ontologies and make use of description logic reasoners. It provided services such as classification and checking for consistency [10] where its owl plugin could directly access DL reasoners like Racer to do so. However, Protégé's original functionality pertaining to materializing deductions is widely inaccessible and the tool was not tailored to suit EER features.

2.3 APIs and object-oriented implementations

2.3.1 OWL API.

The OWL API [7] is a high-level API implemented in Java that allows users to load, manipulate and query ontologies [7]. It has a number of interfaces that examine and edit OWL 2 (structural specification) ontologies using reasoning services [7]. Unlike other APIs, like the Jena and Protégé API which do not support ontologies in RDF syntax, the OWL API does so that it can digest the ontologies as a set of axioms. The manager at its core is what creates, loads, changes and saves ontologies. Providing a central point to do all that allows the user to easily keep track of changes to the ontology [7].

2.3.2 Owlready.

Object programming languages are easier to use to modify an ontology in comparison to APIs like the OWL API [7]. Owlready is a python module designed to use object-oriented programming principals to support OWL 2 ontologies where entities can be accessed like objects in programming languages. Methods are provided to manage classes and their constraints. An algorithm to perform simple closed world reasoning tasks has also been implemented. Closed world reasoning refers to reasoning that infers a logic statement as false if it is unknown [11]. Like the OWL API, Owlready can express OWL 2 constructs such as classes, individuals (instances), disjointness, attributes (object properties) and datatypes. Methods are provided to map OWL constructs to a python object model in Owlready [8]. Owlready is also able to consider equivalence in classes. It can check for subclasses considering equivalent classes. Owlready provides functions to access and edit entities and constraints alike the OWL API [7]. The user needs to call on the reasoner for inferences to be deduced using

a *sync_reasoner()* function. It runs a HermiT reasoner on the file which is an OWL 2 DL reasoner. The function returns the output produced by the reasoner to a specified ontology.

2.3.3 OwlLink.

There are other management mechanisms employed for OWL like OWLLink [12]. OWLLink is an interface that allows one to access OWL reasoner functionality [13]. It provides a client-server protocol between an application as the client and a reasoner as the server [12]. It can communicate with off-the-shelf reasoners [11]. The application can make requests from the server like enquiring about subclasses for a particular class [12].

2.4 ICOM and inference validation

2.4.1 ICOM.

ICOM [6] is a conceptual modelling tool that has a reasoner that is capable of deducing inferences such as class and relationship subsumption and particularly editing with stricter constraints and detecting inconsistencies. It also has a DL reasoning server. Overall, it is able to reason and materialize over modelling features somewhat compatible with EER features. In ICOM, after an ontology is sent to the reasoner, the system queries for deductions and results are given to the ontology editor to edit with the deductions. The editor displays unsatisfiable objects in red, non-explicit deductions are added in green, equivalence found is added using links, and likewise subsumption is shown using specific links and lastly, stricter cardinalities are added. The user can then decide if they want to keep the changes if they are happy with the deductions [6].

2.4.2 Importance of inference validation.

We want to consider inference validation. This involves thinking about how accurately an ontology interprets the reality of the domain. It allows the user to verify changes and reject them if they don't represent the truthfulness of the domain. A problem with this is reversibility which is undoing the changes that have been made which is not always the same as inverting the change. An example is deleting a class in a hierarchy where the subclasses of that class must either also be deleted or reattached to a parent of the class. Reversing a change like this would require readjusting the hierarchy as opposed to just recreating the deleted element [13].

3 SYSTEM DEVELOPMENT & DESIGN

3.1 Software Development Methodology

3.1.1 TDD agile methodology.

A test-first agile development methodology approach was used to implement the Owlready extension. Mainstream software practice recommends the agile approach for development projects that prioritize working software. We hence chose this approach to focus on some getting working software for both aims. The agile approach would enable each stage to contribute a significant feat to realize a robust system that meets the requirements outlined in the introduction. The test-first development approach was used because it is used in best practice when developers want to focus code on specified tests and validate the code's purpose. Defined OWL 2 format is widely used and we wanted to tailor our code specifically to the format so that its reusable. We also want our code's purpose to be as clear as possible so that it can be built on further in the ontology engineering community.

3.1.2 TDD agile approach.

Software development was divided into three iterations. Each iteration was started off by (1) obtaining and modifying (where necessary) test cases of sample models, (2) followed by code implementation, and lastly (3) followed by evaluation to verify working code and progress. The second and third steps were continuously repeated before moving onto further implementation if the code wasn't working to ensure that subsequent code was built on working prototypes.

The first iteration prioritized core EER features with most prominent reasoning services addressed so that a subset of the first aim could be realized. This involved basic subclass constraint edits for entity subsumption; determining inconsistent classes over most of EER features; and appending inconsistent flag comments to those classes that were inconsistent. The EER features encapsulated as OWL constructs considered were binary relationships with varying cardinality restrictions, namely: 1-to-1 mandatory participation, 1-to-1 optional participation, 1-to-N mandatory participation, 1-to-N optional participation, disjointness and covering constraints as well as for equivalent classes.

The second iteration deviated from the original proposal that proposed to implement materialization of deductions for remaining EER features and reasoning tasks and instead focused on the validation wrapper. This was due to an unanticipated delay to achieve proven test case reasoning samples in RDF/XML from the likes of Protégé's guide to building OWL ontologies [14]. The decision to prioritise the wrapper function was so that both the 1st and 2nd aims (as outlined in the introduction) could be decently covered. This second iteration focused on carrying out basic inference acceptance and rejected prompts to the user, and using those decisions to resend the ontology back to the reasoner to be reasoned over again.

The 3rd stage went back to materialization of deductions for unfulfilled EER features and reasoning tasks. This was carrying out relationship subsumption for the ontologies with binary relationships applied with the wrapper. This was done for the cardinality restrictions outlined in the first iteration.

3.2 Development Framework

3.2.1 Programming Language and Framework.

The programming language used was Python. This is because the proposed solution is an extension of the existing Owlready module written in Python. The extended module would need to directly access methods and classes from Owlready's metaclasses. It was developed in JetBrain's PyCharm IDE because of its sophisticated integration with Github Desktop. Github would provide version control throughout the development process.

3.3 Design

3.3.1 Architecture.

The complete architecture of the system is the Owlready architecture with the implemented extension added as seen in figure 2. An extension was implemented because Owlready has extensive functionality that can be used to edit ontologies. Owlready is robust and adding the extension to automate several ontology management functionality makes it easy to integrate at KnowID's knowledge layer.

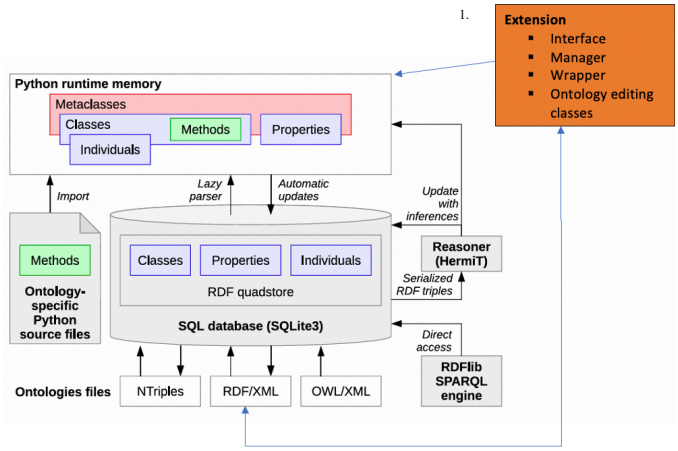

Figure 2: Extension to the Owlready architecture by Lamy [8]

Owlready's original architecture is divided into 5 main parts [8]:

1. RDF quadstore which operates within SQLite3
2. OWL metaclasses which methods that access and edit them that operate in the python runtime memory
3. Optional python source files that can be imported. One can define additional methods for specific OWL classes
4. The HermiT reasoner which performs the reasoning
5. RDFlib module's SPARQL engine

3.3.2 to *3.35* will discuss the functionality of the components with regards to figure 2**.** (3) and (5) are not used in the implementation because it is not in scope.

3.3.2 RDF quadstore.

The RDF quadstore is a quadstore relational database that works within a SQL database, specifically SQLLite3. It contains a column table with 4-tuple quads. Each quad contains an object from an ontology and specifies its respective ontology. The object can be a class or an object property and it is specified by its Internationalized Resource Identifier (IRI) or its RDF literal [8]. IRIs are the URL at which an ontology object or more generally an ontology can be accessed on the internet. To dynamically load the objects from the quadstore to metaclasses in python runtime memory, a lazy parser will load the object from the RDF and wrap it into a python object whenever a method accesses it. When a method modifies the wrapped object by updating, adding to it or removing the wrapper object, its automatically updated in the quadstore as well. The wrapper of the object gets destroyed when its no longer accessed and space in the cache runs out but can be created again by reloading it from the RDF if access is needed again [8].

3.3.3 Owlready metaclasses.

Owlready supplies metaclasses in python runtime memory. These classes are classes whose instances are classes as well. The instances of classes encapsulate ontologies so that ontologies become *.py* files. The metaclasses have pre-defined methods that are used to access ontology classes and call the HermiT reasoner on them [8]. These available methods load ontologies from RDF/XML format into python classes, optionally create new python class, perform reasoning on them and export inferences from the reasoner to an ontology and save the inferences to a new RDF/XML. Other capabilities include enquiries about subclasses or parents of a particular class, what object properties are in an ontology and more.

3.3.4 Ontology-specific Python source files.

Owlready also provides the option for users to define additional methods that access OWL classes in addition to the pre-defined methods discussed above. These user-defined methods must be associated to a specific ontology (its respective python class) so that they can be imported along with the class whenever they are accessed and used with the reasoner [8].

3.3.5 HermiT.

Owlready the HermiT reasoner to interact with the metaclasses and RDF quadstore. It needs to be explicitly called by a user to make any inferences on ontologies. The RDF quadstore may have an ontology encapsulated by quad entries for each of the objects inside an ontology. The quads in the RDF quadstore will briefly be interpreted in NTriple format so that HermiT can reason over it. The output of HermiT is then used to update RDF quadstore by adding an inference as an RDF triple *(subject, predicate, object)* and the respective wrapper python object [8]

3.3.6 Extension.

We have added an extension labelled at (1.). It is a module with a user interface, a wrapper manager and other ontology editing classes. In the extension, the user interface is the initial point of interaction with the user. An OWL 2 file in RDF/XML format representing an EER diagram is passed to the extension module via its user interface which requests a path directory of the file. The extension has an ontology manager which is the central point for automating the ontology management functionality provided by Owlready's metaclasses. This includes aggregating the process of loading, reason, save functionality described in *3.3.3* an ontology from RDF/XML to a python class. This is required because HermiT needs to be explicitly invoked.

The extension also has a wrapper which provides further interaction for user validation of inferences. It prompts the users with choices of accepting or rejecting the inferences as well as an option to remove a subclass.

3.3.7 Algorithms and some data structures.

To aggregate and automate the process of the manager given an RDF/XML formatted ontology that represents EER with implicit

inferences, the algorithm in algorithm 1 is followed. Owlready [8] proposes using this functionality to load, reason, and save inferences. Here *sync_reasoner()* calls the reasoner to infer deductions from the *onto* ontology to a temporary *inference_onto* ontology *inference_onto.save()* saves the inferences to an OWL file ready to be amended with to the original OWL file in RDF/XML format. Our implementation uses this algorithm to obtain the inferences in RDF/XML format in order to edit. The original ontology RDF/XML is put into a list. Encapsulating it in a list allows us to dynamically add, remove and alter constraints. Python doesn't have a library to dynamically alter binary files. Generic arrays were used for storing deductions since they did not need to be dynamically processed. This process will be discussed in more depth in the implementation section.

Algorithm 1: inference.py

Input: input owl **I**, inference owl **T**,
Output: inference owl **T**

```
1  onto = get_ontology(I)
2  onto.load()
3  inference_onto = get_ontology(T)
4  with inference_onto:
5        sync_reasoner()
6  inference_onto.save()
```

4 SYSTEM IMPLEMENTATION & TESTING

This section will document the main implementation process to realize materialization of deduction and the wrapper functionality as well as the functional testing used to evaluate the system.

4.1 Implementation

4.1.1 User interface and manager.

The user interface (a python class) in the extension module prompts the user to specify a file path of an ontology that encapsulates EER features in OWL 2 specificaton. It also requests the file path to an empty output owl file in the same location as the input. The input owl file needs to be in RDF/XML format. The interface class then passes these file paths to the manager in the extension. The manager executes the general algorithm presented in **algorithm 1** to combine all steps required to load and call a reasoner on the ontology and save inferences in RDF/XML format to an external owl file. The *append()* function of Owlready adds the file location of an OWL ontology class captured as an object *onto*. *get_ontology()* then fetches the actual ontology by the name passed to it to attach to the *onto* object. When *load()* is called, the RDF/XML ontology is now transformed to a python object wrapped by the onto object. An additional ontology *inference_ontology* is created the same way as *onto* but given an empty RDF/XML file. When *sync_reasoner()* is called, the HermiT reasoner is invoked on the *onto* ontology with the output deductions inserted into *inference_ontology*. *save()* saves the inferences now inside *inference_ontology* into the RDF/XML owl file passed to it. We now have the inferences with the object to which it references in RDF/XML format, ready to be integrated with the original OWL file to a new output file specified by the user.

4.1.2 Materialization.

Algorithm 2: wrapper

Input: input owl **I**, inference owl **T**, output owl **O**
Output: output owl **O**

```
1  prep = Preprocess(I, T, O)
2  prep.find_items()
3
4  material = Materialize (prep.deductions, I, O, prep)
5  material.materialize_deductions()
6  material.write_to_RDFXML()
7
8  wrap = Wrapper(material, prep)
9  wrap.manage()
```

Algorithm 2 starts after writing the explicit isolated inferences to another temporary RDF/XML **T**. To integrate these subsequent inferences in **T** with the original, the following steps in **algorithm 2** is followed. Starting by pre-processing all the deductions in the temporary OWL **T** to identify the object that needs to be edited with the new inference into inference python objects. In line 1, *Preprocess* instantiates an object with the input owl file, a reusable temporary owl file (for the inferences) and an output file. The *find_items()* function will store the deductions from the temporary file in an array as inference objects with an *item_to_edit* and the actual edit itself named *edit*. Both *item_to_edit* and *edit* are in strings to maintain the RDF/XML format from the temporary RDF/XML file they were saved in. This will make it easier to integrate with original ontology. Although the OWL file is binary, Python's *decode* and *encode* methods can be used to convert from binary to string and vice versa respectively.

Materialize instantiates an object of the *Materialize* class responsible for the actual integration of the inferences into the RDF/XML. Its *load_input_file()* loads the original RDF/XML into a list. The most imperative function *materialize_deductions()* actually iterates over the ontology list and edits with the array of deductions accordingly. It checks for subclasses, equivalent classes and inconsistent objects. The *write_to_RDFXML* finally rewrites it back to a binary OWL file **O** in the maintained RDF/XML format.

4.1.3 Wrapper.

The wrapper is called after materialization of deductions is complete which reproduces a user interface and deletes constraints, classes and objects based on the user's selected choice. An object of *Wrapper* class is created with the respective *Preprocess* and *Materialize* objects in order to call its *manage()* function which elicits the wrapper user interface to present the original changes to the ontology and request the user to accept, reject and additionally

remove classes. When changes are accepted nothing further is done to the output ontology. When a request is rejected, the inferred constraint is removed from the ontology. When a class inferred as a subclass is removed, the *remove_subclass()* function deletes the class from the ontology, and makes its subsequent classes inherit all constraints of the removed class. It reparents the subclasses to the removed class' parent. It also removes all subclass references to the object for any other class as we cannot reference a non-existent class. If the class removed has no subclasses, any object properties of which it is the domain will be removed as well since the object property can only be used with the removed class. The resultant object is sent back to the reasoner and any new inferences are sustained in resultant ontology.

When objects in an RDF/XML are put into objects captured as *Item* objects using the *items_original_file()* function, the *inherit*() function defined in *Item* was used to explicitly inherit object property cardinality restrictions from its parents alongside any other constraints like disjoint classes.

4.2 Testing Implementation

Manual functional testing was used to evaluate the outcome of the system. Functional testing bases its tests cases on the functional requirements of a system by providing input and evaluating and verifying the corresponding outcome against such requirements. Our systems broad requirements are the aims defined in the introduction. More specifically the requirements of realizing materialization of deductions can be defined in terms of the edits expected to be made. Our system's wrapper function itself has three overall specifications which is to remove certain constraints, reparent and remove certain classes as well as do nothing. So to determine if the specifications were met, we simply needed to see if the correct OWL files were produced by simply seeing if they matched the output OWL file expected. More specifically, we could check the necessary objects in the file with anticipated change. Hence this black box testing method was suitable to use to compare expected output OWLs to the ones produced over all our requirements. It was done manually so we could fully analyze the changes to the file.

4.2.1 Sample models.

To evaluate the success of the program test sample cases were taken from Protégé's guide to building OWL ontologies [14]. The guide was used because it provides widely-used ontology samples that explore several reasoning cases. The reasoning cases use formal logic to make inferences. This ensures that the expected inferences are verified and reliable. Where needed, we altered and made our own test cases following the principals outlined in the Protégé guide.

The ontology samples use pizza concepts and illustrates them with different constraints. Such constraints include subclasses, minimum and maximum cardinality constraints (qualified and non-qualified), equivalent classes, disjoint classes and union classes (covering). The construction of each ontology example was instructed by the guide where we followed accordingly and used the relevant Protégé 4 Desktop version 5.5.0 tool to easily create them [15]. Once the sample ontology was created, it was saved into RDF/XML syntax. The instruction guide was also used to construct the corresponding output ontology with inferences expected as outlined in the guide. Some of the examples were slightly modified by trimming unneeded concepts so that we could test specific features and simplify the process. Some naming was also changed for more intuitive understanding of the reasoning cases.

4.2.2 Wrapper testing.

The wrapper class was evaluated by analyzing the output to see if

- constraints were sustained or removed
- a deleted subclass had their subclasses reassigned to its parent's class and ensured inheritance of its features
- object properties whose domain was the removed class was removed alongside its domain class. This was only in the case where the removed class did not have any children.
- The removal of subclass references to the object was also inspected.

The table below shows how EER constructs are captured as OWL constructs and can be used to explain the expected output for EER feature descriptions numbered in figure 3.

Table 1: OWL 2 constructs in EER

OWL 2 constructs	EER constraints
Class/subClassOf	Entity/sub-entity
dataProperty	Attributes
qualifiedCardinality rdf:datatype="…#nonNegativeInteger">1	Binary relationship 1 : 1 mandatory
maxQualifiedCardinality rdf:datatype="…#nonNegativeInteger">1	Binary relationship 1 : 1 optional
minQualifiedCardinality rdf:datatype="…#nonNegativeInteger">1	Binary relationship 1 : N mandatory
minQualifiedCardinality rdf:datatype="…#nonNegativeInteger">0	Binary relationship 1 : N optional
disjointWith	Disjointness
intersectionOf/unionOf	Covering
Class/(min/max)qualifiedCardinality	N-ary relationship

4.2.3 System requirements.

The system requirements are captured in figure 3. Figure 3 shows what is expected to be appended or removed depending on the inference made. The following numbered points will refer to the figure to explain the edit: (The numbering in the figure and points correspond)

1. Shows when a class *entity2* is inferred as a subclass of a class *entity*. If the inference is rejected in the wrapper, it is removed

2. When a subclass is removed in the wrapper, the class *entity* is removed

3. To evaluate if a binary relationship with binary 1:1 mandatory participation can be subsumed by *entity* from another class when removing its parent in the wrapper, the red excerpt in 3 figure x should be observed.

4. To evaluate if a binary relationship with binary 1:1 optional participation can be subsumed by *entity* from another class when removing its parent in the wrapper, the red excerpt in 4 figure x should be observed.

5. To evaluate if a binary relationship with binary 1:N mandatory participation can be subsumed by *entity* from another class when removing its parent in the wrapper, the red excerpt in 5 figure x should be observed.

6. To evaluate if a binary relationship with binary 1:N optional participation can be subsumed by *entity* from another class when removing its parent in the wrapper, the red excerpt in 6 figure x should be observed.

7. To evaluate if a class entity subsumed that it is disjoint with another class entity 2 when removing its parent in the wrapper, the red excerpt in 7 figure x should be observed.

8. To evaluate if a class entity is inconsistent, the inconsistent tag show in red in 8 figure x should be observed.

```
1.  Subclass determined (and its rejection)

    <Class> entity
            <subClassOf> entity2
    <Class>

2.  Subclass removal

    <Class> entity
    <Class>

3.  Binary 1:1 mandatory participation subsumed

    <Class> entity
            <subClassOf>
                    <Restriction>
                            <onProperty> property
                                    <qualifiedCardinality> 1
                                    <onClass> entity2
                            <onProperty>
                     <Restriction>
            <subClassOf>
    <Class>
4.  Binary 1:1 optional participation subsumed

    <Class> entity
            <subClassOf>
                    <Restriction>
                            <onProperty> property
                                    <maxQualifiedCardinality> 1
                                    <onClass> entity2
                            <onProperty>
                     <Restriction>
            <subClassOf>
    <Class>

5.  Binary 1: N mandatory participation subsumed
    <Class> entity
            <subClassOf>
                    <Restriction>
                            <onProperty> property
                                    <minQualifiedCardinality> 1
                                    <onClass> entity2
                            <onProperty>
                     <Restriction>
            <subClassOf>
    <Class>

6.  Binary 1: N optional participation subsumed
    <Class>

            <subClassOf>
                    <Restriction>
                            <onProperty> property
                                    <minQualifiedCardinality> 0
                                    <onClass> entity2
                            <onProperty>
                     <Restriction>
            <subClassOf>
    <Class>


7.  Disjointness constraint subsumed

    <Class> entity
            disjointWith: entity2
    <Class>

8.  Inconsistent class

    <Class> entity
            <! --DEDUCTION-->
            <--INCONSISTENT CLASS-->
    <Class>
```

Figure 3: Demonstration of inferences made to OWL 2 objects in RDF/XML format

5 RESULTS

5.1 Generated test case sample models

Table 2: A table showing edits to ontologies that were achieved

Sample OWL	OWL edit	Ontology constraint/s determining inference
1	Addition/removal of subClassOf constraint	Binary relationship 1 : 1 mandatory and covering
2	Addition/removal of subClassOf constraint	Binary relationship 1 : 1 mandatory and covering
3	Addition/removal of subClassOf constraint	Binary relationship 1 : N mandatory and covering
4	Addition/removal of subClassOf constraint	Binary relationship 1 : N optional and covering
5	Addition of Inconsistent tag	Disjointness and covering
6	Addition of qualifiedCardinality>1	Subclass
7	Addition of maxQualifiedCardinality>1	Subclass
8	Addition of minQualifiedCardinality>1	Subclass
9	Addition of minQualifiedCardinality>0	Subclass
9	Addition of disjointWith constraint	Subclass
9	Removal of Class	Subclass
10	Removal of Object Property	via Subclass removal

Table 2 shows the edits that were successfully made to 9 sample ontologies. The first column references a specific ontology by number, the second column documents the edit/s that was made to the ontology. The third column identifies which EER constraints were directly responsible for the inferences shown in column 2.

5.1.1 Sample 1.

We will document the results of the first sample from table 2 in depth to make the account of the remaining results more interpretable.

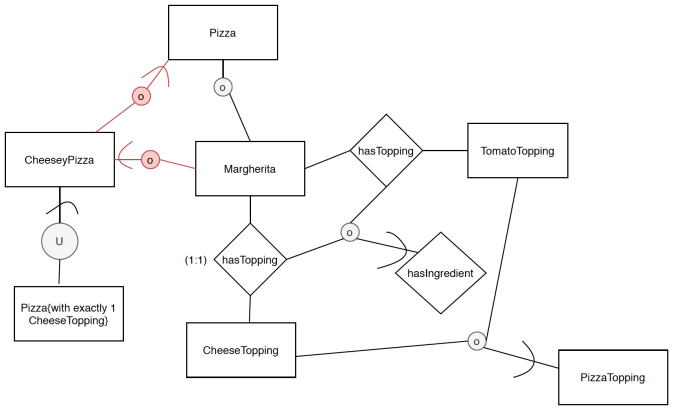

Figure 4: EER diagram of ontology sample 1 (captured in Table 2)

The first sample was an ontology representing the resulting EER diagram show in figure 4 after reasoning. The constraints in red denote the inferred constraints that were added. These were: Margherita is a subclass of CheeseyPizza and CheeseyPizza is a subclass of Pizza.

```
1 <ObjectProperty> hasTopping
2     <subPropertyOf> hasIngredient
3 < ObjectProperty >
```

```
1 <Class> CheeseyPizza
2     <equivalentClass>
3         <Class>
4             <intersectionOf>parseType= "Collection" Pizza
5                 <Restriction>
6                     <onProperty> hasTopping
7                         <qualifiedCardinality> 1
8                             <onClass> CheeseTopping
9                     <onProperty>
10                <Restriction>
11            <intersectionOf>
12        <Class>
13    <equivalentClass>
14 <Class>
```

```
1 <Class> Margherita
2     <subClassOf> Pizza
3     <subClassOf>
4         <Restriction>
5             <onProperty> hasTopping
6                 <qualifiedCardinality> 1
7                 <onClass> CheeseTopping
8             <onProperty>
9         <Restriction>
10    <subClassOf>
11<Class>
```

```
1 <Class> Margherita
2     <subClassOf> CheeseyPizza
3     <subClassOf> Pizza
4     <subClassOf>
5         <Restriction>
6             <onProperty> hasTopping
7                 <qualifiedCardinality> 1
8                 <onClass> CheeseTopping
9             <onProperty>
10        <Restriction>
11    <subClassOf>
12 <Class>
```

Relevant input classes and object properties *Relevant output class*

Figure 5: Input/output owl classes and property before and after reasoning in truncated RDF/XML format for sample 1

Figure 5 is the truncated version of the RDF/XML I/O files produced after running sample 1. The excerpts have also been numbered and slightly graphically altered for ease of interpretation. Relevant naming is highlighted in blue and edits are in red. The complete input/output owl files are attached in the appendix and within the software module along with the remaining sample models to adhere to maximum page constraints.

In OWL 2, a *subClassOf CheeseyPizza* constraint was added to *Margherita* (seen line 2 on the right of figure 5).

5.1.2 Subclass constraint.

subClassOf CheeseyPizza constraint was also added to *Margherita* for samples 2,3 and 4 which had different cardinality restrictions for the binary relationship *hasTopping* as respectively indicated in Table 2.

5.1.3 Wrapper samples removing constraint.

Sample 1, 2, 3, and 4 were also used with wrapper to test the rejection of the added constraint. After running the output file with the wrapper, when reject was selected, *subClassOf CheeseyPizza* constraint was removed from the *Margherita* class in all samples.

5.1.4 Inconsistent.

The inconsistent tag was added to the output RDF/XML in sample 5. It was added to an inconsistent class *ThinandCrispyBase* after reasoning detected it was a subclass of two disjoint classes *PizzaBase* and *PizzaTopping*.

5.1.5 Wrapper samples reparenting, removing references, subsuming relationships & removing subclass.

Samples 6, 7, 8, and 9 were used with the wrapper function. A parent class *Margherita* had two subclasses *GourmetMargherita* and *StandardMargherita* and a parent class *Pizza. GourmetMargherita* and *StandardMargherita* inherited relationships with their cardinality restrictions from the parent class when a parent class was selected to be removed. The cardinality restrictions inherited are once again respectively denoted in the 3rd column of table 2. *Margherita* class was removed. References to Margherita present in only *GourmetMargherita* and *StandardMargherita* were removed. Moreover, any other constraints from *Margherita* were also inherited like in sample 9, *disjointWith BaconPizza* constraint from *Margherita* was inherited by *GourmetMargherita* and StandardMargherita. Also, a new *subClassOf Pizza* constraint was added to *GourmetMargherita* and *StandardMargherita.*

5.1.6 *Wrapper samples removing subclass & object property.*

Margherita has no subclasses. The sample is a case were domain and range of object property *hasHerb* is included. *Margherita* is the domain of object property *hasHerb* and *Basil* is the range of it so that it reads that a *Margherita* has a herb *Basil.* After Margherita is selected for deletion, *hasHerb is deleted as well.*

6 DISCUSSIONS

The aim to materialize deductions on an ontology over EER features was met fairly well. Explicitly adding subclass constraints to the OWL was achieved for ontologies over the binary relationship constraints for all its cardinalities. Moreover, constraints like disjointess, covering and subclass constraints were other EER features reasoned over to add a subclass inference to an ontology. A case of explicitly tagging a detected inconsistent class was satisfied which used disjointness and covering constraint to infer that fact. Amending binary relationship with their cardinalities to subclasses of a parent removed was the extent to which entity subsumption could be achieved. Cardinalities once again include all those specified in table 1. A wrapper was also realized. As discussed, reversibility of some verification is not always as easy as it seems. This system was able to give various options to the user to verify inferences made. It can reparent classes based on user decisions to remove a class subsumed to be a child of another. This was the case of a class which after materialization of deductions was achieved, was now a parent and a child. Removing the class would require reassigning its children to its parent which the system achieved. The achieved reference removal of a non-existent class was also an important feat for a sensible ontology. Other constraints like disjointness were also subsumed to the children when reparenting. The removal of an inconsistent binary relationship also automatically ensues if its domain class is removed and it has no children. The presence of children otherwise has the affect of re-assigning the object property to the children if its cardinality restrictions are subsumed as described above. Alerting the user of an inconsistency is also imperative to materializing deductions and was achieved by the system. Moreover, attribute inferences take place at instance level. This means such inferences would take place in the data layer of KnowID as opposed to the knowledge layer in which the systems is intended to be used. N-ary relationships were not realized for the system as were unable to interpret them in OWL 2. Furthermore, testing was limited to our test cases.

7 ETHICAL, PROFESSIONAL, AND LEGAL ISSUES

Further development on Owlready is permissible. It is available under GNU GPL v3 license which is a free and copyleft license for software. This system extension was also developed under GNU. It is open-sourced so that others are free to use and develop further.

8 CONCLUSIONS & FUTURE WORK

Overall, an extension to materialize deductions on an EER was successful and so was a wrapper to aid an effective user-verification process. For future work, one can implement a transformation algorithm from RDF/XML files to JSON files. It has been proven that it can be achieved [1]. If this is realized, it can be integrated with this system to facilitate query formulation in the data layer. It requires a JSON file to be used with React [1] to graphically display the EER in a GUI and have users formulate queries over it. Some ontology variations with the reasoning cases and constraints explored in this report can be tested over these three components merged.

ACKNOWLEDGMENTS

I would like to thank my project partner, Bradley Malgas, for his support and being an exemplar in navigating software development and project work. I would also like to greatly thank my supervisor, Maria Keet for all of her advice, constant support, guidance, and pushing me to think open-mindedly.

REFERENCES

[1] Pablo R. Fillotrani, S. Jamieson and C. Maria Keet. 2020. Connecting knowledge to data through transformations in KnowID: system description. In KI - Künstliche Intelligenz (KI '20), Springer, 373–379. DOI: https://doi.org/10.1007/s13218-020-00675-6.

[2] Natalya F. Noy, Monica Crubézy, Ray W. Fergerson, Holger Knublauch, Samson W. Tu, Jennifer Vendetti and Mark A. Musen. 2003. Protégé-2000: An Open-Source Ontology-Development and Knowledge-Acquisition Environment. In *AMIA Annual Symposium Proceedings (AMIMA '03).*

[3] Pablo R. Fillotrani and C. Maria Keet. 2020. KnowID: An architecture for efficient Knowledgedriven Information and Data access.

[4] Franz Baader, Ian Horrocks and Ulrike Sattler. 2007. Description Logics. In *Reasoning Web. Semantic Technologies for Information Systems* (Reasoning Web '09). Springer, Berlin, Heidelberg. DOI: https://doi.org/10.1007/978-3-642-03754-2_1

[5] Ian Horrocks and Andrei Voronkov. 2006. Reasoning Support for Expressive Ontology Languages Using a Theorem Prover. In Diz J., Hegner S.J. (eds) *Foundations of Information and Knowledge Systems 4th International Symposium, Lecture Notes in Computer Science* 3861 *(FolKS 2006)*, February 14 – 17, 2006, Budapest, Hungary. Springer, Berlin, Heidelberg, 201-218.

[6] Pablo R. Fillottrani, Enrico Franconi, and Sergio Tessaris. 2012. The ICOM 3.0 Intelligent Conceptual Modelling tool and methodology. Semantic Web 3, 3 (2012), 293-306.

[7] Sean Bechhofer and Matthew Horridge. 2009. The OWL API: A Java API for Working with OWL 2 Ontologies. In *Proceedings of the 6th International Conference on OWL: Experiences and Directions (OWLED 2009)*, October 23 - 24, 2009, Aachen, Germany, 49-58.

[8] Jean-Baptiste Lamy. 2017. Owlready: Ontology-oriented programming in Python with automatic classification and high-level constructs for biomedical ontologies. *Artificial Intelligence in Medicine* 80, 11-28.

[9] John H. Gennari, Mark A. Musen, Ray W. Fergerson, William E. Grosso, Monika Crubézy, Henrik Eriksson, Natalya F., Noy and Samson W. Tu. The evolution of Protégé: An environment for knowledge-based systems development. In *International Journal of HumanComputer Studies*, 58, 1 (2003), 89–123. DOI:10.1016/S1071-5819(02)00127-1.

[10] Holger Knublauch, Ray W. Fergerson, Natalya F. Noy, and Mark A. Musen. 2004. The Protégé OWL Plugin: An Open Development Environment for Semantic Web Applications. In *Third International Semantic Web Conference (ISWC 2004)*, November 7 – 11, 2004, Hiroshima, Japan. Springer, Berlin, Heidelberg, 229-243.

[11] Oren Etzioni, Keith Golden and Daniel Weld. 1994. Tractable Closed World Reasoning Updates. In *Proceedings of the Conference on Principles of Knowledge Representation and Reasoning (KR-94* 1994).

[12] Thorsten Liebig, Marko Luther, Olaf Noppens and Michael Wessel. 2011. *OWLlink. Semantic Web*, January 2011.

[13] Alexander Maedche, Boris Motik, Ljiljana Stojanovic, Rudi Studer, and Raphael Volz. 2003. Ontologies for Enterprise Knowledge Management. *IEEE Intelligent Systems 18, 2 (March-April 2003),* 26-33.

[14] Matthew Horridge and Sebastian Brandt. 2011. A Practical Guide To Building OWL Ontologies Using Protége 4 and CO-ODE Tools Edition 1.3. The University of Manchester, Manchester, UK.

[15] Protégé. 2016. Protégé: https://protege.stanford.edu/products.php

APPENDIX

Relevant excerpts from each sample taken and shown

Sample 1 input

```
  <!-- http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-27#CheeseyPizza -->

  <owl:Class rdf:about="http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-27#CheeseyPizza">
    <owl:equivalentClass>
      <owl:Class>
        <owl:intersectionOf rdf:parseType="Collection">
          <rdf:Description rdf:about="http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-27#Pizza"/>
          <owl:Restriction>
            <owl:onProperty rdf:resource="http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-27#hasTopping"/>
            <owl:qualifiedCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInteger">1</owl:qualifiedCardinality>
            <owl:onClass rdf:resource="http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-27#CheeseTopping"/>
          </owl:Restriction>
        </owl:intersectionOf>
      </owl:Class>
    </owl:equivalentClass>
  </owl:Class>


  <!-- http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-27#Margherita -->

  <owl:Class rdf:about="http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-27#Margherita">
    <rdfs:subClassOf rdf:resource="http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-27#Pizza"/>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-27#hasTopping"/>
        <owl:someValuesFrom rdf:resource="http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-27#TomatoTopping"/>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-
```

```
27#hasTopping"/>
            <owl:qualifiedCardinality
rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInteger">1</owl:qualifiedCardinalit
y>
            <owl:onClass
rdf:resource="http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-
27#CheeseTopping"/>
         </owl:Restriction>
      </rdfs:subClassOf>
   </owl:Class>
```

Sample 1 Output

```
   <!-- http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-
27#CheeseyPizza -->

   <owl:Class rdf:about="http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-
ontology-27#CheeseyPizza">

  <rdfs:subClassOf
rdf:resource="http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-
27#Pizza"/>
      <owl:equivalentClass>
         <owl:Class>
            <owl:intersectionOf rdf:parseType="Collection">
               <rdf:Description
rdf:about="http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-
27#Pizza"/>
               <owl:Restriction>
                  <owl:onProperty
rdf:resource="http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-
27#hasTopping"/>
                  <owl:qualifiedCardinality
rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInteger">1</owl:qualifiedCardinalit
y>
                  <owl:onClass
rdf:resource="http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-
27#CheeseTopping"/>
               </owl:Restriction>
            </owl:intersectionOf>
         </owl:Class>
      </owl:equivalentClass>
   </owl:Class>


   <!-- http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-
27#Margherita -->

   <owl:Class rdf:about="http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-
ontology-27#Margherita">
  <rdfs:subClassOf
rdf:resource="http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-
27#CheeseyPizza"/>
      <rdfs:subClassOf
rdf:resource="http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-
```

```
27#Pizza"/>
      <rdfs:subClassOf>
        <owl:Restriction>
          <owl:onProperty
rdf:resource="http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-
27#hasTopping"/>
          <owl:someValuesFrom
rdf:resource="http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-
27#TomatoTopping"/>
        </owl:Restriction>
      </rdfs:subClassOf>
      <rdfs:subClassOf>
        <owl:Restriction>
          <owl:onProperty
rdf:resource="http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-
27#hasTopping"/>
          <owl:qualifiedCardinality
rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInteger">1</owl:qualifiedCardinalit
y>
          <owl:onClass
rdf:resource="http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-
27#CheeseTopping"/>
        </owl:Restriction>
      </rdfs:subClassOf>
    </owl:Class>
```

Sample 2 Input

```
<!-- http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-27#SomePizza
-->

<owl:Class rdf:about="http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-
ontology-27#SomePizza">
    <owl:equivalentClass>
      <owl:Class>
        <owl:intersectionOf rdf:parseType="Collection">
          <rdf:Description
rdf:about="http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-
27#Pizza"/>
          <owl:Restriction>
            <owl:onProperty
rdf:resource="http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-
27#hasTopping"/>
            <owl:maxQualifiedCardinality
rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInteger">1</owl:maxQualifiedCardi
nality>
            <owl:onClass
rdf:resource="http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-
27#CheeseTopping"/>
          </owl:Restriction>
        </owl:intersectionOf>
      </owl:Class>
    </owl:equivalentClass>
</owl:Class>
```

Sample 2 output

Same as sample 1

Sample 3 Input

```
<!-- http://www.co-ode.org/ontologies/pizza/pizza.owl#CheeseyPizza -->

<owl:Class rdf:about="http://www.co-ode.org/ontologies/pizza/pizza.owl#CheeseyPizza">
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <rdf:Description rdf:about="http://www.co-ode.org/ontologies/pizza/pizza.owl#Pizza"/>
        <owl:Restriction>
          <owl:onProperty rdf:resource="http://www.co-
ode.org/ontologies/pizza/pizza.owl#hasTopping"/>
          <owl:minQualifiedCardinality
rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInteger">1</owl:minQualifiedCardin
ality>
          <owl:onClass rdf:resource="http://www.co-
ode.org/ontologies/pizza/pizza.owl#CheeseTopping"/>
        </owl:Restriction>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
  <rdfs:comment xml:lang="en">Any pizza that has at least 1 cheese topping.</rdfs:comment>
  <rdfs:label xml:lang="pt">PizzaComQueijo</rdfs:label>
</owl:Class>
```

Sample 3 output

Same as sample 1

Sample 4 input

```
<!-- http://www.co-ode.org/ontologies/pizza/pizza.owl#SomePizza -->

<owl:Class rdf:about="http://www.co-ode.org/ontologies/pizza/pizza.owl#SomePizza">
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <rdf:Description rdf:about="http://www.co-ode.org/ontologies/pizza/pizza.owl#Pizza"/>
        <owl:Restriction>
          <owl:onProperty rdf:resource="http://www.co-
ode.org/ontologies/pizza/pizza.owl#hasTopping"/>
          <owl:minQualifiedCardinality
rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInteger">0</owl:minQualifiedCardin
ality>
          <owl:onClass rdf:resource="http://www.co-
ode.org/ontologies/pizza/pizza.owl#CheeseTopping"/>
        </owl:Restriction>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
  <rdfs:subClassOf rdf:resource="http://www.co-ode.org/ontologies/pizza/pizza.owl#Pizza"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="http://www.co-
ode.org/ontologies/pizza/pizza.owl#hasTopping"/>
      <owl:someValuesFrom rdf:resource="http://www.co-
ode.org/ontologies/pizza/pizza.owl#TomatoTopping"/>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
```

```
        <owl:onProperty rdf:resource="http://www.co-
ode.org/ontologies/pizza/pizza.owl#hasTopping"/>
        <owl:minQualifiedCardinality
rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInteger">0</owl:minQualifiedCardin
ality>
        <owl:onClass rdf:resource="http://www.co-
ode.org/ontologies/pizza/pizza.owl#CheeseTopping"/>
      </owl:Restriction>
   </rdfs:subClassOf>
   <rdfs:comment xml:lang="en">Any pizza that has at least 1 cheese topping.</rdfs:comment>
   <rdfs:label xml:lang="pt">PizzaComQueijo</rdfs:label>
</owl:Class>
```

Sample 4 input

Same as sample 1

Sample 5 input

```
<!-- http://www.co-ode.org/ontologies/pizza/pizza.owl#ThinAndCrispyBase -->

<owl:Class rdf:about="#ThinAndCrispyBase">
   <rdfs:label xml:lang="pt"
      >BaseFinaEQuebradica</rdfs:label>
   <rdfs:subClassOf rdf:resource="#PizzaBase"/>
   <rdfs:subClassOf rdf:resource="#PizzaTopping"/>
</owl:Class>
<!-- http://www.co-ode.org/ontologies/pizza/pizza.owl#PizzaBase -->

<owl:Class rdf:about="#PizzaBase">
   <rdfs:label xml:lang="pt">BaseDaPizza</rdfs:label>
   <rdfs:subClassOf rdf:resource="#Food"/>
   <owl:disjointWith rdf:resource="#PizzaTopping"/>
   <owl:disjointWith rdf:resource="#Pizza"/>
</owl:Class>

<!-- http://www.co-ode.org/ontologies/pizza/pizza.owl#PizzaTopping -->

<owl:Class rdf:about="#PizzaTopping">
   <rdfs:label xml:lang="pt"
      >CoberturaDaPizza</rdfs:label>
   <rdfs:subClassOf rdf:resource="#Food"/>
</owl:Class>
```

Sample 5 output

```
   <!-- http://www.co-ode.org/ontologies/pizza/pizza.owl#   dCrispyBase -->

   <owl:Class rdf:about="#ThinAndCrispyBase">
<!--DEDUCTION-->
<!--INCONSISTENT CLASS-->
      <rdfs:label xml:lang="pt"
         >BaseFinaEQuebradica</rdfs:label>
      <rdfs:subClassOf rdf:resource="#PizzaBase"/>
      <rdfs:subClassOf rdf:resource="#PizzaTopping"/>
      <owl:disjointWith rdf:resource="#DeepPanBase"/>
   </owl:Class>
```

Sample 6 input

```
   <!-- http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-27#GourmetMargherita -->

   <owl:Class rdf:about="http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-27#GourmetMargherita">
      <rdfs:subClassOf rdf:resource="http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-27#Margherita"/>
   </owl:Class>



   <!-- http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-27#Margherita -->

   <owl:Class rdf:about="http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-27#Margherita">
      <rdfs:subClassOf rdf:resource="http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-27#Pizza"/>
      <rdfs:subClassOf>
         <owl:Restriction>
            <owl:onProperty rdf:resource="http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-27#hasTopping"/>
            <owl:someValuesFrom rdf:resource="http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-27#TomatoTopping"/>
         </owl:Restriction>
      </rdfs:subClassOf>
      <rdfs:subClassOf>
         <owl:Restriction>
            <owl:onProperty rdf:resource="http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-27#hasTopping"/>
            <owl:qualifiedCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInteger">1</owl:qualifiedCardinality>
            <owl:onClass rdf:resource="http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-27#CheeseTopping"/>
         </owl:Restriction>
      </rdfs:subClassOf>
   </owl:Class>


   <!-- http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-27#StandardMargherita -->

   <owl:Class rdf:about="http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-27#StandardMargherita">
      <rdfs:subClassOf rdf:resource="http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-
```

```
27#Margherita"/>
```

Sample 6 output

```
   <!-- http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-27#GourmetMargherita -->

   <owl:Class rdf:about="http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-27#GourmetMargherita">
      <rdfs:subClassOf rdf:resource="http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-27#Pizza"/>
      <rdfs:subClassOf>
         <owl:Restriction>
            <owl:onProperty rdf:resource="http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-27#hasTopping"/>
            <owl:someValuesFrom rdf:resource="http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-27#TomatoTopping"/>
         </owl:Restriction>
      </rdfs:subClassOf>
      <rdfs:subClassOf>
         <owl:Restriction>
            <owl:onProperty rdf:resource="http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-27#hasTopping"/>
            <owl:qualifiedCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInteger">1</owl:qualifiedCardinality>
            <owl:onClass rdf:resource="http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-27#CheeseTopping"/>
         </owl:Restriction>
      </rdfs:subClassOf>
   </owl:Class>



   <!-- http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-27#StandardMargherita -->

   <owl:Class rdf:about="http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-27#StandardMargherita">
      <rdfs:subClassOf rdf:resource="http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-27#Pizza"/>
      <rdfs:subClassOf>
         <owl:Restriction>
            <owl:onProperty rdf:resource="http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-27#hasTopping"/>
            <owl:someValuesFrom rdf:resource="http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-27#TomatoTopping"/>
         </owl:Restriction>
      </rdfs:subClassOf>
```

```
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty
rdf:resource="http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-
27#hasTopping"/>
        <owl:qualifiedCardinality
rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInteger">1</owl:qualifiedCardinalit
y>
        <owl:onClass
rdf:resource="http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-
27#CheeseTopping"/>
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>
```

Samples 7, 8 and 9 are nearly similar qualified cardinalities inherited differ according to table showing binary cardinalities.

Sample 9 has more inferencing done.
Sample 9 input, margherita removed

```
<!-- http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-
27#StandardMargherita -->

<owl:Class rdf:about="http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-
ontology-27#StandardMargherita">
  <rdfs:subClassOf
rdf:resource="http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-
27#Margherita"/>
</owl:Class>
<!-- http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-
27#GourmetMargherita -->

<owl:Class rdf:about="http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-
ontology-27#GourmetMargherita">
  <rdfs:subClassOf
rdf:resource="http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-
27#Margherita"/>
</owl:Class>
```

Sample 9 input cont.

```
<!-- http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-27#Margherita
-->

<owl:Class rdf:about="http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-
ontology-27#Margherita">
  <owl:disjointWith
rdf:resource="http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-
27#BaconPizza"/>
  <rdfs:subClassOf
rdf:resource="http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-
27#Pizza"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty
```

```
rdf:resource="http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-27#hasTopping"/>
        <owl:someValuesFrom rdf:resource="http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-27#TomatoTopping"/>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-27#hasTopping"/>
        <owl:qualifiedCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInteger">1</owl:qualifiedCardinality>
        <owl:onClass rdf:resource="http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-27#CheeseTopping"/>
      </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
```

Sample 9 output

```
<!-- http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-27#GourmetMargherita -->

<owl:Class rdf:about="http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-27#GourmetMargherita">
    <rdfs:subClassOf rdf:resource="http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-27#Pizza"/>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-27#hasTopping"/>
        <owl:someValuesFrom rdf:resource="http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-27#TomatoTopping"/>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-27#hasTopping"/>
        <owl:qualifiedCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInteger">1</owl:qualifiedCardinality>
        <owl:onClass rdf:resource="http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-27#CheeseTopping"/>
      </owl:Restriction>
    </rdfs:subClassOf>
    <owl:disjointWith rdf:resource="http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-
```

```
27#BaconPizza"/>
</owl:Class>
```

```
<!-- http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-27#StandardMargherita -->

<owl:Class rdf:about="http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-27#StandardMargherita">
    <rdfs:subClassOf rdf:resource="http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-27#Pizza"/>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-27#hasTopping"/>
            <owl:someValuesFrom rdf:resource="http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-27#TomatoTopping"/>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-27#hasTopping"/>
            <owl:qualifiedCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInteger">1</owl:qualifiedCardinality>
            <owl:onClass rdf:resource="http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-27#CheeseTopping"/>
        </owl:Restriction>
    </rdfs:subClassOf>
    <owl:disjointWith rdf:resource="http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-27#BaconPizza"/>
</owl:Class>
```

Sample 10 input

```
<!-- http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-27#hasHerb -->

<owl:ObjectProperty rdf:about="http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-27#hasHerb">
    <rdfs:subPropertyOf rdf:resource="http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-27#hasIngredient"/>
    <rdfs:domain rdf:resource="http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-27#Margherita"/>
    <rdfs:range rdf:resource="http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-27#Basil"/>
</owl:ObjectProperty>

<!-- http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-27#Margherita -->
```

```
<owl:Class rdf:about="http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-27#Margherita">
   <rdfs:subClassOf rdf:resource="http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-27#Pizza"/>
   <rdfs:subClassOf>
      <owl:Restriction>
         <owl:onProperty rdf:resource="http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-27#hasHerb"/>
         <owl:someValuesFrom rdf:resource="http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-27#Basil"/>
      </owl:Restriction>
   </rdfs:subClassOf>
   <rdfs:subClassOf>
      <owl:Restriction>
         <owl:onProperty rdf:resource="http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-27#hasTopping"/>
         <owl:someValuesFrom rdf:resource="http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-27#TomatoTopping"/>
      </owl:Restriction>
   </rdfs:subClassOf>
   <rdfs:subClassOf>
      <owl:Restriction>
         <owl:onProperty rdf:resource="http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-27#hasTopping"/>
         <owl:qualifiedCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInteger">1</owl:qualifiedCardinality>
         <owl:onClass rdf:resource="http://www.semanticweb.org/mandisabaleni/ontologies/2020/8/untitled-ontology-27#CheeseTopping"/>
      </owl:Restriction>
   </rdfs:subClassOf>
</owl:Class>
```

Sample 10 Output

Both hasHerb and Margherita were removed