# Assignment 5

Course Lecturer: Yizheng Zhao

June 3, 2022

⋆ **This assignment, due on 26th June, contributes to 10% of the total mark of the course.**

**Question 1. Warm-up — CW3: Populating Family History**

A dataset describing aspects of family history constitutes the content of this question. It contains information about people and the occupations or roles they played. Each individual described has associated information:

- Surname
- Married Surname (if known)
- Birth Year (if known)
- Given Name (first name)

Along with this, there will be a number (possibly zero) of roles/occupations played. For each of these there will be

- Year (if known)
- Source
- Occupation (if known — this may be "none given", stating that the role is unknown)

Data is provided as CSV (comma separate values). Code is given that will parse this CSV file into a collection of Bean Objects with appropriate get and set methods. The parser returns a collection of Beans, representing each row in the spreadsheet. You can assume that given name, surname and date of birth are sufficient to distinguish different individuals. If no year, source or occupation is given in a row, this row can be ignored.

For the question, you will write a Java program that populates an ontology with this data using the OWL API version 5.0.0. The ontology containing class and property definitions is given in the archive. You should not need to provide any additional classes, properties or definitions to the ontology — all your additions should concern individuals. Each person identified in the data should be represented as a named individual (of type Person) in the ontology. Each role played should result in an instance of RolePlayed, with appropriate relationships to a role instance, a source and a year (if known). The individuals representing roles played may be named, while some may be left as anonymous individuals; the choice is yours. The names of role classes in the CSV file should match those given in the role hierarchy in the ontology, but note that you may have to do a small amount of processing to map roles/occupations in the data to roles in the ontology. Hint: you may need to think carefully about how to handle the special case "none give", which is not the name of a role(!), but is used to indicate that no explicit role was named.

Your submission will be tested by loading a set of unseen data and then evaluating queries against this data. A sample set of data is provided in test/test.csv. The ontology for population is provided in resources/ontology.owl.

You will need to implement three methods in the owlapi.khkb.fspopulation.CW3 class:

- loadOntology(OWLOntologyManager manager, IRI ontologyIRI) Loading ontology from physical IRI
- populateOntology (OWLOntologyManager manager, OWLOntology ontology, Collection<JonDataBean> beans) Populating the ontology using the collection of Java beans holding the data from CSV
- saveOntology (OWLOntologyManager manager, OWLOntology ontology, IRI locationIRI) Saving the ontology back to the disk

You can add additional methods to this class if you wish, but do not change the signature of the existing methods or of the no-argument constructor. You will embed all the required functionality in CW3.java. As this is the only file you will submit, please do not implement any essential functionality outside of CW3.java. The harness is configured as a maven project, and you can do your testing through 3 implemented Junit tests. In order to set your project up, do the following:

1. Extract the cw3.zip somewhere on your computer
2. Open your Eclipse, and select File->Import->Existing Maven Project and select the directory you just unpacked.
3. Go to the Build Path settings and make sure that the JRE selected is in fact the one from your JDK.
4. Right click on the project: Maven clean.
5. Right click on the project: Maven install

You will see your project failing to build because it does not pass the Junit tests. Now you are good to go. As a suggestion, run the Junit tests frequently within Eclipse to monitor your progress. The tests will take the input from resources/test.csv, the ontology from resources/ontology.owl and will produce a populated ontology in fhkb-ai.owl. Once you have finished the assignment, you should submit a zip archive of the CW3.java file in your project directory.

**Question 2. Warm-up — CW5: querying implementing**

CW5 is all about querying implementing and OWL based querying system. Your task will be to query the reasoner for sub-classes, equivalent classes and instances, and store the results in a QueryResult object. You are asked to implement the following methods in the Java class CW5 (project layout the same as CW3!). Do not alter any class other than CW5, which is the only one you are (allowed) to submit! The classes App.java and PizzaOrderingSystemApp.java provide a, hopefully fun, way to test what you are doing is correct. You can run them in the same way as you ran the unit tests in CW3, simply by hitting the small white and green arrow icon in Eclipse when the class is open. If you are not interested in having fun, then you can, in the same way as you did last time, use the implemented unit tests to figure out whether you are on the right track. Please note that this time passing the unit tests does not automatically mean full marks. We will use some of our own tests to further validate your solution. Once you are done, please zip and submit CW5.java to the PedagogySquare platform. Good Luck!

**Tasks:**

1. public Set<QueryResult> performQuery (OWLClassExpression exp, Query Type type) {...}
   This method takes in an arbitrary expression in OWL, like "Person and hasBirthYear value 1964" or "hasTopping some MeatTopping" and queries the ontology for the following three types of knowledge:
   ⋆ EquivalentClasses: you are asked to return all those (named) classes that are equivalent to the expression (exp), using the (already fully initialized) reasoner.

⋆ Sub-classes: return all those classes that are (named) sub-classes of the expression, using the reasoner.
⋆ Instances: return all those individuals that are instances of the expression, using the reasoner.
Query results are stored in query result objects. These are created for example as follows:
QueryResult qr = new QueryResult(ind, false, type);
Note that you need to include the information whether the inference is direct or indirect. Example: Given three classes with A subclass B subclass C, then A is a direct subclass of B and B is a direct subclass of C, but A is an indirect subclass of C (through B!). In order to solve this problem, you will query the reasoner for direct and indirect sub-class separately. After creating the QueryResult, add it to the provided set.

2. public Boolean isValidPizza (OWLClassExpression exp) {...};
   In this method, you check whether the supplied class expression exp is a valid pizza expression, i.e., whether it is inferred to be a Pizza.

3. public Set<QueryResult> filterNamedPizzas (Set<QueryResult> results) {...};
   This question is similar to the one before, only that you are asked to filter from a set of results those that correspond to NamedPizza's, such as Margherita or AmericanHot.

4. public Set<OWLClassExpression> getAllSuperClassExpressions (OWLClass ce) {...};

This question requires a bit of thinking. You are asked to query the ontology for all information you can find about the class ce. In order to get an idea of what "AL" means, you can open Protégé and look at the "Classe" tab. If you click on a class, you will find that it often has not only super-classes and equivalent-classes, but also inherited super-classes (Anonymous Ancestors), and of course indirect super-classes. Unfortunately the reasoner will not easily give you access to those. In order to get full marks for this task, it is sufficient to query for all super-classes (direct and indirect, using the reasoner), and somehow find a way to obtain the anonymous super-classes using the ontology directly (that will need a bit of thought, do not despair too quickly). A perfect solution will test, for all sub-expressions in the ontology, whether ce is a sub-concept of it. Attempt this only if you are really confident with the OWL API by now.

**Question 3. Tracking semantic changes over SNOMED CT's evolution**

E-health vendors[1] such as ALERT,[2] Babylon Health,[3] are seeking to make healthcare affordable and accessible for everyone through an AI doctor system, which relies on a large knowledge base formed by the merger and alignment of healthcare ontologies from different sources. These ontologies are, however, in a continuous state of being refined, improved and adapted, and new versions incorporating the evolutionary changes are periodically released. An important task at these vendors is to identify the difference in the meaning of the medical terms in different ontology versions so as to monitor and understand the evolution of the ontologies and make the process of migrating existing data and services faster and more reliable.

Dealing with multiple versions of the same information unit is a standard operation in computer and information sciences, and *version control* is a well-established technology. Modern version control systems are based on the *diff* operation that has been extensively used in text processing and software engineering to track the changes in versions of text files and code blocks. However, such a notion, referred to as *syntactic difference* in the context of ontology versioning, cannot underlie the task of identifying the differences in the meaning of terms in different versions of an ontology, because it ignores all meta-information about the ontology and is only concerned with its text serialization (e.g., *the order of ontology axioms*). For this reason,

[1] http://www.snomed.org/our-customers/vendors/marketplace
[2] https://www.alert-online.com/
[3] https://www.babylonhealth.com/

a new notion of *structural difference* has been proposed, which extends the syntactic notion by taking into account the meta-information about the structure of ontologies. Structural difference regards ontologies as structured objects such as a concept hierarchy, where changes to ontologies are described in terms of structural operations (e.g, the addition or deletion of a concept or an axiom). For example,

$$\mathcal{T}_1 = \{\mathsf{Father} \sqsubseteq \mathsf{Human} \sqcap \exists \mathsf{hasSex}.\mathsf{Male} \sqcap \exists \mathsf{hasChild}.\top\}$$

is regarded *structurally different* from

$$\mathcal{T}_2 = \{\mathsf{Father} \sqsubseteq \mathsf{Human} \sqcap \exists \mathsf{hasSex}.\mathsf{Male}, \mathsf{Father} \sqsubseteq \exists \mathsf{hasChild}.\top\},$$

though they are *logically (semantically) equivalent* — the term Father has the same meaning in $\mathcal{T}_1$ and $\mathcal{T}_2$. While structural difference is a step forward towards capturing meta-information about distinct versions of ontologies, it does not provide an unambiguous semantic foundation and is not syntax independent.

As a consequence, a semantic notion has been proposed as a way of capturing the difference in the meaning of terms that is independent of the representation of ontologies. In particular, the *logical (semantic) difference* between two ontologies are the axioms entailed by one ontology but not the other, reflecting the information gain and information loss between them. Formally, given two ontologies $\mathcal{T}_1$ and $\mathcal{T}_2$, the *information gain* $\mathsf{Diff}(\mathcal{T}_1, \mathcal{T}_2)$ from $\mathcal{T}_1$ to $\mathcal{T}_2$ (the *information loss* from $\mathcal{T}_2$ to $\mathcal{T}_1$) are the axioms entailed by $\mathcal{T}_2$ but not $\mathcal{T}_1$.

**Definition 1 (Logical Difference)** *Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be two ontologies specified in the description logic $\mathcal{L}$. The* logical difference between $\mathcal{T}_1$ and $\mathcal{T}_2$ *is the set* $\mathsf{Diff}(\mathcal{T}_1, \mathcal{T}_2)$ *of all $\mathcal{L}$-axioms $\alpha$ such that:*

*(i)* $\alpha \in \mathcal{T}_2$, *but*

*(ii)* $\mathcal{T}_1 \not\models \alpha$.

*An axiom $\alpha$ satisfying the above conditions is called a* witness *of a difference in $\mathcal{T}_2$ w.r.t. $\mathcal{T}_1$.*

As ontologies are dynamic entities that are constantly evolving, computing the logical difference between two versions of an ontology can be a critical task — to track what has changed in a new version of an ontology, to ensure that the changes are safe in the sense of the new version being a *conservative extension* of a preceding version, and to identify unexpected consequences in new versions of an ontology. This provides effective means for discovering issues and enhances quality control during the ontology evolution process.

SNOMED CT [4] is now the most comprehensive, multilingual clinical healthcare terminology in the world; it has been integrated into the knowledge base of many e-health vendors. SNOMED International[5] owns and maintains SNOMED CT, issuing releases of its international edition at the end of January and July each year, where revisions have been made to meet users' needs and reflect the results of quality assurance activities.

- At this stage, your task is to implement a diff(OWLOntology onto_1, OWLOntology onto_2, String path) method to compute the logical difference between two ontologies. The input of the method are two ontologies, namely onto_1 and onto_2 to be compared which must be specified as OWL/XML files, or as URLs pointing to OWL/XML files, as well as a file path specifying the location where the output returned by the method is going to be saved. The output are a set of witnesses, which is saved as standard ontologies (OWL/XML files) that can be used for in-depth analysis or further processing. You should submit a zip archive of the entire project including everything needed to run your method.

[4] https://www.nlm.nih.gov/healthit/snomedct/international.html
[5] https://www.snomed.org/

- You are given a dataset containing six consecutive international releases of SNOMED CT. Your task is to compute the logical difference between each release and its immediate follower (and vice versa), i.e., the information gain and information loss, using the implementation you have just developed. The output of each computation is an OWL/XML file containing the witnesses computed by the method between the two compared releases. These OWL/XML files are what you will submit as deliverables for marking.

The *signature* $\text{sig}(\mathcal{T})$ of an ontology $\mathcal{T}$ is the set of all concept and role names used in $\mathcal{T}$. A new version of an ontology may introduce new names or discard existing names in or from its signature on the grounds of adaptation, refinement and update necessity of its content. As a consequence however, some of the computed witnesses may become less interesting; consider two versions of a snippet of a family ontology:

$$\begin{aligned}\mathcal{T}_1 &= \{\mathsf{Father} \equiv \mathsf{Human} \sqcap \exists \mathsf{hasSex}.\mathsf{Male} \sqcap \exists \mathsf{hasChild}.\top\}\\ \mathcal{T}_2 &= \{\mathsf{Father} \equiv \mathsf{Parent} \sqcap \exists \mathsf{hasSex}.\mathsf{Male},\\ &\qquad \mathsf{Parent} \equiv \mathsf{Human} \sqcap \exists \mathsf{hasChild}.\top\},\end{aligned}$$

where Parent is a newly-introduced concept in $\mathcal{T}_2$. It is not difficult to verify that both axioms in $\mathcal{T}_2$ are not entailed by $\mathcal{T}_1$, and thus should, in principle, count as witnesses of $\text{Diff}(\mathcal{T}_1, \mathcal{T}_2)$. We however think of these witnesses as "less interesting", because Parent is only defined in $\mathcal{T}_2$ but not in $\mathcal{T}_1$, i.e., $\mathcal{T}_1$ is completely ignorant of what Parent stands for, leading to the absence of a definition serving as a reference (the baseline) for comparison. In addition, the introduction of Parent does not interfere with the meaning of the other names in $\mathcal{T}_2$ — the definition of Father remains unchanged from $\mathcal{T}_1$ from $\mathcal{T}_2$, with the newcomer Parent serving as the "go-between" to underpin the interaction between the two definitions.

To make each witness count, a key step leading up to the computation of $\text{Diff}(\mathcal{T}_1, \mathcal{T}_2)$ is to unify the signatures of $\mathcal{T}_1$ and $\mathcal{T}_2$ in a way that the meanings of the names in the unified signature are preserved. This ensures a fair comparison between the definitions of the same names in distinct ontology versions. In this case, one may unify the signatures of $\mathcal{T}_1$ and $\mathcal{T}_2$ by replacing the Parent in the first axiom of $\mathcal{T}_2$ by its definition $\mathsf{Human} \sqcap \mathsf{hasChild}.\top$, which is specified in the second axiom, yielding the same definition of Father as in $\mathcal{T}_1$, i.e., $\mathsf{Father} \equiv \mathsf{Human} \sqcap \exists \mathsf{hasSex}.\mathsf{Male} \sqcap \exists \mathsf{hasChild}.\top$. Therefore, a more meaningful notion of logical difference should be defined upon the common signature of two ontologies (or subsets thereof).

A promising way to unify the signatures of $\mathcal{T}_1$ and $\mathcal{T}_2$ is to compute a view $\mathcal{V}_2$ of $\mathcal{T}_2$ in such a way that the semantics of the names in $\mathcal{V}_2$ is preserved, where $\text{sig}(\mathcal{V}_2) \subseteq \text{sig}(\mathcal{T}_1)$.

**Definition 2 (View)** *Let $\mathcal{V}$ and $\mathcal{T}$ be two ontologies specified in the description logic $\mathcal{L}$. We say that $\mathcal{V}$ is a* view *of $\mathcal{T}$, analogous to a view of a relational database, is an ontology obtained from $\mathcal{T}$, where*

*(i) $sig(\mathcal{V}) \subseteq sig(\mathcal{T})$, and*

*(ii) for any $\mathcal{L}$-axiom (GCI) $\alpha$ with $sig(\alpha) \subseteq sig(\mathcal{V})$, $\mathcal{V} \models \alpha$ iff $\mathcal{T} \models \alpha$.*

Observe that $\mathcal{V}$ is a view of $\mathcal{T}$ iff $\mathcal{T}$ is a conservation extension of $\mathcal{V}$; they can be defined in terms of $\equiv_{\sqsubseteq}$-*inseparability* — $\mathcal{V}$ and $\mathcal{T}$ are $\equiv_{\sqsubseteq}$-*inseparable* w.r.t. $\text{sig}(\mathcal{V})$. Intuitively, this means that $\mathcal{V}$ and $\mathcal{T}$ behave the same when referring to $\text{sig}(\mathcal{V})$, or $\mathcal{V}$ and $\mathcal{T}$ speak the same on the topic of $\text{sig}(\mathcal{V})$.

Therefore, a more meaningful notion of logical difference should be defined upon the common signature of two ontologies (or subsets thereof).

**Definition 3 (View-based logical Difference)** *Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be two ontologies specified in the description logic $\mathcal{L}$, and $\Sigma$ be a subset of the common signature of $\mathcal{T}_1$ and $\mathcal{T}_2$. The* view-based logical difference *between $\mathcal{T}_1$ and $\mathcal{T}_2$ is the set View-Diff$_\Sigma(\mathcal{T}_1, \mathcal{T}_2)$ of all $\mathcal{L}$-axioms $\alpha$ such that:*

*(i)* $sig(\alpha) \subseteq \Sigma$,

*(ii)* $\alpha \in \mathcal{V}_2$, *and*

*(iii)* $\mathcal{T}_1 \not\models \alpha$, *where* $\mathcal{V}_2$ *is a* $\Sigma$*-view of* $\mathcal{T}_2$.

*An axiom* $\alpha$ *satisfying the above conditions is called a* view-witness *of View-Diff*$_\Sigma(\mathcal{O}_1, \mathcal{O}_2)$.

Computing views of ontologies is a computationally extremely hard problem — a finite view does not always exist for the description logic $\mathcal{EL}$, and if it exists, then there exists one of at most triple exponential size in terms of the original ontology, and that, in the worst case, no smaller view exists. You are given a Java interface compute_view(Set<OWLClass> c_sig, Set<OWLObjectProperty> r_sig, OWLOntology onto) to compute a $\Sigma$-view of an ontology onto, where $\Sigma =$ c_sig $\cup$ r_sig.

- At this stage, your task is to compute the logical difference between each release and the $\Sigma$-view of its immediate follower (and vice versa), where $\Sigma = \mathsf{sig}(\mathsf{onto\_1}) \cap \mathsf{sig}(\mathsf{onto\_2})$, using the implementation you have developed previously. The output of each computation is an OWL/XML file containing the view-witnesses computed by the method between the two compared releases. These OWL/XML files are what you will submit as deliverables for marking.