WebSem lab1

Victor MAYAUD, Elliot BOUCHY

8 décembre 2023

1 Q1/ Start Protégé and imagine a good namespace prefix and a namespace URI for modeling the Cycling domain. Provide this couple prefix/uri in your report and explain why you choose them.

For a cycling domain ontology in Protégé, the chosen namespace prefix and URI are as follows :

— **Prefix :** `cycle`
— **URI :** `http://www.example.org/ontology/cycling`

This configuration was chosen for its clarity and direct association with the cycling domain. The prefix `cycle` is succinct and relevant, while the URI combines a standard format with a path that clearly indicates the subject and nature of the ontology.

2 Q2/ In the cycling domain, a Race is either a OneDayRace or a SeveralStages-Race. There are no other types of race. Model these three classes in the ontology and write in the report the logical formula equivalent to this definition.

In the context of modeling an ontology for the cycling domain, we define the following classes :

— Race
— OneDayRace
— SeveralStagesRace

The logical relationship between these classes is :

$$Race \equiv OneDayRace \sqcup SeveralStagesRace \tag{1}$$

$$OneDayRace \sqcap SeveralStagesRace \equiv \emptyset \tag{2}$$

Here, Equation (1) signifies that the class *Race* is equivalent to the union of *OneDay-Race* and *SeveralStagesRace*. Equation (2) states that *OneDayRace* and *SeveralStagesRace* are disjoint classes, meaning that they have no instances in common.

5 a revoir equivalent

3 Q5/ Define with necessary and sufficient conditions (i.e. strict equality) the Prologue class such as the Prologue is exactly always the first Stage of a SeveralStagesRace. You might have to introduce more properties or use the inverse of an existing property to do this modeling. Write in the report the logical formula equivalent to this definition.

We introduced a subclass named **Prologue** within the **Stage** class. To establish the necessary relationship, we created an object property named `isFirstStageOf` with *Prologue* as the domain and *SeveralStagesRace* as the range. This property implies that each instance of *Prologue* is the first stage of a specific *SeveralStagesRace*.

We also defined an inverse property called `hasFirstStage`, setting *SeveralStagesRace* as the domain and *Prologue* as the range. This inverse relationship ensures that a *SeveralStagesRace* has a *Prologue* as its initial stage.

The logical formulae for these definitions in the ontology are as follows :

$$\textit{Prologue} \equiv \texttt{isFirstStageOf}\ \text{some}\ \textit{SeveralStagesRace}$$
$$\textit{SeveralStagesRace} \equiv \texttt{hasFirstStage}\ \text{some}\ \textit{Prologue}$$

These formulas ensure a one-to-one correspondence between a *Prologue* and a *SeveralStagesRace*, thereby satisfying the necessary and sufficient conditions as required.

4 Q6/ Add the concept Person to the ontology. A Person can be a Spectactor, a RacePerson or a TeamPerson among others. A TeamPerson is exactly the union of a Doctor, a Director or a RaceCyclist. A RaceCyclist can himself be considered as a Rider, a Sprinter or a Climber. A Person has a name, an age, and a nationality. A Person participates in Race. Add all these classes and properties and the corresponding axioms. Explain in the report what are the differences between object and datatype properties.

After added the subclasses and union, we created data property named `name`, `age` and `nationality`. We defined the name and nationality as strings and the age as a nonNegativeInteger, then we associated the domains of each data property with person. Then we created an object property named `participatesInRace` with *Person* as the domain and *Race* as the range and we associated this object with the classes.

Object properties in ontology are used to establish relationships between two individual instances of classes. An object property might be `manages`, relating a *Manager* individual to an *Employee* individual. Datatype properties, on the other hand, link an individual instance to a data value. These properties are used to assign attributes that are not objects but primitive data types such as strings or numbers.

In summary, the primary distinction lies in their linkage : object properties connect ontology instances, while datatype properties tie instances to data values.

5 Q7/A Team is exactly composed of 1 Doctor, 1 Director and 10 RaceCyclist. Define this class using the belongsTo property. Explain in the report what can you say about this new property with respect to previously defined properties ?

To model the composition of a Team class according to the given requirements, we utilize the object property `belongsTo`. The domain of `belongsTo` is set to the Person class, and its range is set to the Team class. This property inversely relates a Person to a Team they are part of. We then define the Team class with necessary and sufficient

conditions using the cardinality constraints on the `belongsTo` property. Specifically, we assert that a Team is composed of exactly one Doctor, one Director, and ten RaceCyclists. This is articulated in the ontology through the following class expressions :

- `belongsTo exactly 1 Doctor`
- `belongsTo exactly 1 Director`
- `belongsTo exactly 10 RaceCyclist`

These constraints ensure that any individual of the Team class will have these exact membership relationships with individuals of the Doctor, Director, and RaceCyclist classes. It's important to note that the `belongsTo` property is defined in such a way that it can only link members of a Team to their respective roles within that team, differentiating it from any previously defined properties that may relate persons to other concepts or classes.

6 Q9/ Load (import) the FOAF ontology using its http ://xmlns.com /foaf/spec/ and align some of the concepts you have defined such as : the class Person is equivalent to the class Person defined in FOAF and the class Team is a subclass of the class Organisation defined in FOAF. Search for properties in the FOAF ontology to align with your own properties and add the corresponding mapping axioms. Write in the report the mappings you have discovered.

The agent class corresponds to our class person with certaine subclasse in addition such as organization, group. There are also many more data properties to represent a person, such as account name, nickname, gender, birthday. In objects properties, member seems to be associated with our BelongTo class.

7 Q11/ Use the LOV search function at https ://lov.linkeddata.es/dataset/lov/ and identify relevant classes and properties that could be re-used in your ontology. Write them down in your report.

In our ontology, we could add a `race tracker` class, to count the number of laps in a race. We could also add a `location` class, to find out where a race is located. An `identifier` class would also make it easier to identify a cyclist. It would also be interesting to add a `tournament` class to create a competition and then add a `"NumberOfWin"` property to count the number of victories of each competitor, so that they can be compared on each stage.

Property	Value
rdf:type	• owl:Class
rdfs:isDefinedBy	• http://dbpedia.org/ontology/
rdfs:label	• identifier (en) • identifiant (fr)
rdfs:subClassOf	• owl:Thing
owl:equivalentClass	• wikidata:Q6545185
wdrs:describedby	• dbo:data/definitions.ttl
prov:wasDerivedFrom	• http://mappings.dbpedia.org/index.php/OntologyClass:Identifier
is ov:defines of	• http://dbpedia.org/ontology/
is ov:describes of	• dbo:data/definitions.ttl
is rdfs:subClassOf of	• dbo:TopLevelDomain

FIGURE 1 – Identifiant class with LOV search function

Property	Value
rdf:type	• rdf:Property • owl:ObjectProperty
rdfs:comment	• The location of the thing. (en)
rdfs:isDefinedBy	• http://dbpedia.org/ontology/
rdfs:label	• location (en) • emplacement (fr)
rdfs:range	• dbo:Place
rdfs:subPropertyOf	• dul:hasLocation
owl:equivalentProperty	• schema:location
wdrs:describedby	• dbo:data/definitions.ttl
prov:wasDerivedFrom	• http://mappings.dbpedia.org/index.php/OntologyProperty:location
is ov:defines of	• http://dbpedia.org/ontology/
is ov:describes of	• dbo:data/definitions.ttl
is rdfs:subPropertyOf of	• dbo:locationCity • dbo:locationCountry

FIGURE 2 – Location class with LOV search function

Property	Value
rdf:type	• owl:Class
rdfs:isDefinedBy	• http://dbpedia.org/ontology/
rdfs:label	• tournament (en) • tournoi (fr)
rdfs:subClassOf	• dbo:SportsEvent
owl:equivalentClass	• wikidata:Q500834
wdrs:describedby	• dbo:data/definitions.ttl
prov:wasDerivedFrom	• http://mappings.dbpedia.org/index.php/OntologyClass:Tournament
is ov:defines of	• http://dbpedia.org/ontology/
is ov:describes of	• dbo:data/definitions.ttl
is rdfs:range of	• dbo:continentalTournament • dbo:nationalTournament • dbo:olympicGames • dbo:worldTournament
is rdfs:subClassOf of	• dbo:GolfTournament • dbo:SoccerTournament • dbo:TennisTournament

FIGURE 3 – Tournament class with LOV search function

8 Q12/ Validate your ontology by clicking on Reasoner and Start reasoner where HermiT is selected as the reasoning service. What is the result ? Did you get the message “your ontology is coherent and consistent” ?

At the end of the log, the sentence "Ontologies processed in 249078 ms by HermiT" is displayed, but there is no explicit message indicating whether “your ontology is coherent and consistent.” However, as there are no errors, we can infer that our ontology is likely coherent and consistent.

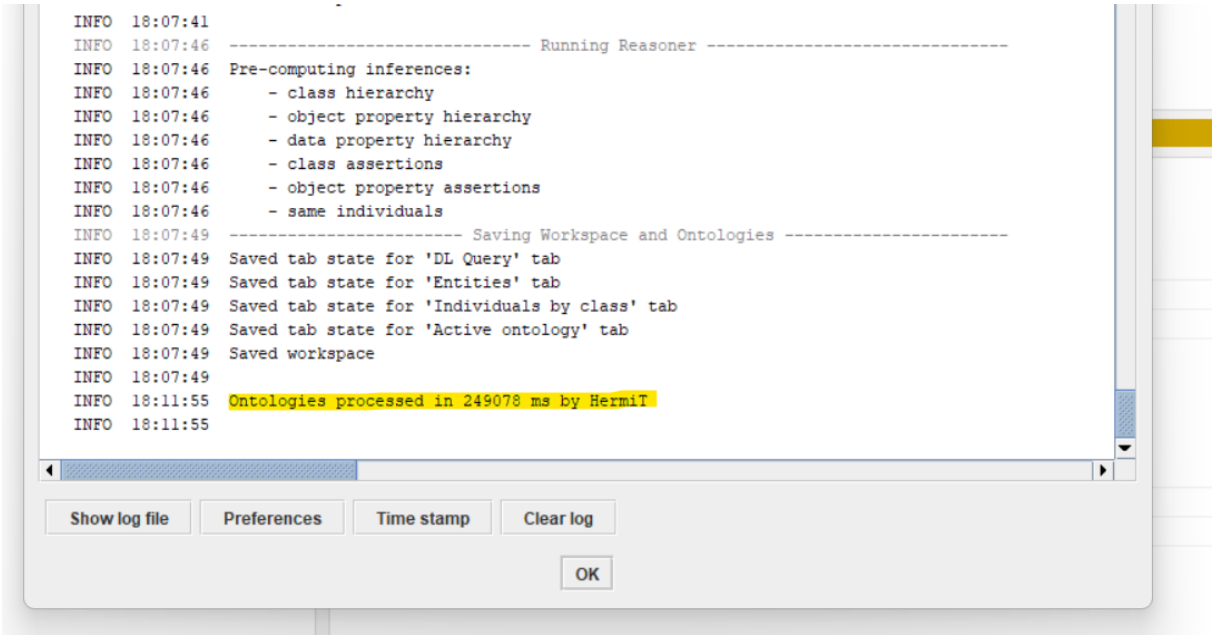

FIGURE 4 – The result of the reasoner.

9 Difficulties

We had a few problems debugging our ontology. We had trouble understanding whether an object could take on several distinct domains, and if so, how. So it was difficult to find the right domain and the right range. This made it difficult to make our ontology consistent. What’s more, it took us a while to get used to the software’s various tabs. Despite these difficulties, we managed to create a functional and coherent ontology in the allotted time, and in the process learned how an ontology works.