Semantic Web: Assignment 1

Report

Note: you can find my ontology in owl attached file.

1. For the namespace prefix and the namespace URI, I decided to choose **consistent** and **explicit** names. The names I chose are:

- "cyc" for the prefix
- http://www.semanticweb.org/thomas/ontologies/2020/cycling for the URI

2. I created a class `Race` and two subclasses of this class: `OneDayRace` and `SeveralStagesRace`. To make the machine understand that a race can be a one-day race or a several-stages race only, I added an axiom to the class `Race` called `Disjoint Union of OneDayRace, SeveralStagesRace`. The logical formula equivalent to this definition of the class is a **XOR**:

(OneDayRace OR SeveralStagesRace) AND NOT (OneDayRace AND SeveralStagesRace)

3. I created a class `Stage` and the object property `composedOf` with domain `SeveralStageRace` and range `Stages`. In the class `SeveralStagesRace`, I added an axiom of type "Equivalent To" called `composedOf some Stages`.
The property `composedOf` is an object property because an instance of the class `SeveralStagesRace` is composed of some instances of the class `Stages`.

4. In the class `Stages`, I created the following three subclasses: `MountainStage, FlatStage, TimeTrial`. I also added a `Disjoint With` axiom between these three classes because the disjoint union of these three classes is not equal to the class `Stage`. For the `TimeTrial` class, I created two subclasses (`IndividualTimeTrial` and `TeamTimeTrial`). I added a `Disjoint Union Of` axiom to the `TimeTrial` class because a time trial stage can be a team time trial or an individual time trial only.

5. For the prologue, I built a Data Property called `stage_number`. The domain of this property is the class `Stage` and the range is the data type literal. Then, I created the class `Prologue` as a subclass of the class `Stage`, and I added an axiom to the class `Prologue` called `stage_number value 1`. It means that the prologue is always the stage with the number 1.

6. Same as before, I created the class `Person`, and then the three following subclasses `Spectactor, RacePerson, TeamPerson`. I added a `Disjoint With` axiom between these three classes.
I created three subclasses of the class `TeamPerson`: `Doctor, Director, RaceCyclist`. I added an axiom to the class TeamPerson: `Disjoint Union of Doctor, Director, RaceCyclist`.
Moreover, the class `RaceCyclist` is the `Disjoint Union of Climber, Rider, Sprinter`.

Three data properties are created:

- `age`: domain `Person` and range literal
- `name`: domain `Person` and range literal
- `nationality`: domain `Person` and range literal

Finally, I created the object property `participatesIn`. I added the axiom `participatesIn some Race` to the class `Person`.

Reminder

Object properties are used when a property links two objects. For example, the property `participatesIn` is an object property because an instance of the class `Person` participates in an instance of the class `Race`.

Data properties are used when a property links an object to a datatype. For example, the property `age` is a data property because an instance of the class `Person` has an age which is a literal datatype and not a class or an instance of a class.

7. For this question, I defined the `belongsTo` property which domain is `TeamPerson` and range is `Team`. We can observe that the property `composedOf` can't be the inverse of `belongsTo` because the domain and the ranges are not inversed between these two properties.

Then, I added the three following axioms to `Team`:

- `team inverse(belongsTo) exactly 1 Director`
- `team inverse(belongsTo) exactly 1 Doctor`
- `team inverse(belongsTo) exactly 10 RaceCyclist`

8. For this question, I used again the `belongsTo` property. I created `GoodTeam` class as a subclass of the class `Team`. Then, `GoodTeam` inherits the previous axioms of the class `Team`. To define well and precisely the `GoodTeam` class, I added the following two axioms:

- `GoodTeam inverse(belongsTo) exactly 1 Rider`
- `GoodTeam inverse(belongsTo) min 3 Climber`

9. For the data properties, I aligned: `firstName`, `lastName`, `surName`.

For the object properties, I aligned: `currentRace`, `knows`, `pastRace`, `races`.

10. **Instances** and related **properties** are trivially added (cf. individuals in the ontology).

11. For this question, I used the LOV Search function. After searching "cycling", it appears that there are two relevant classes re-usable for our ontology: **CyclingCompetition** and **CyclingRace.**

Then, for these two classes, the dbpedia mapping shows different properties that could be re-used for my ontology:

- champion: winner of the competition
- silverMedalist: 2nd of the podium
- bronzeMedalist: 3rd of the podium
- mostWins; the rider who won the most important number of races
- raceResult
- recentWinner

12. The validation of the ontology is done on the following website:
    http://visualdataweb.de/validator/.

Running the reasoner gives the following screen:

```
INFO  23:46:43  ------------------------------ Running Reasoner ------------------------------
INFO  23:46:43  Pre-computing inferences:
INFO  23:46:43      - class hierarchy
INFO  23:46:43      - object property hierarchy
INFO  23:46:43      - data property hierarchy
INFO  23:46:43      - class assertions
INFO  23:46:43      - object property assertions
INFO  23:46:43      - same individuals
INFO  23:51:02  Ontologies processed in 258826 ms by HermiT
INFO  23:51:02
```

The reasoner checks each step in this order and tries to detect errors in the ontology. If there is no error (like here), the ontology is processed. The result here is successful and the reasoner used was **HermiT**.

• • • END • • •
