Oblig 5 – IN3060

1: Model Semantics

Exercise 1.1: Interpretation

1. First, let's define things.

   Our domain

   $\triangle^I$ = {Tweety, JollyJumper, Bruce, b}

   Assigning classes to the domain

   $Tweety^I$ = Tweety, $JollyJumper^I$ = JollyJumper, $Bruce^I$ = Bruce

   $Animal^I$ = {Tweety, JollyJumper, Bruce}

   $Food^I$ = {Bruce, b}

   $Vegetable^I$ = {b},

   $Fish^I$ = {Bruce},

   $Horse^I$ = {JollyJumper, Bruce},

   $Penguin^I$ = {Tweety} = $Bird^I$,

   $hasNickname^I$ = {<JollyJumper, "JJ">, <Bruce, "Alonso">}

   $favouriteFood^I$ = {<JollyJumper, b>} = $eats^I$

   $likes^I$ = {<JollyJumper, Tweety>}

   This is a valid interpretation L_1 such that $L_1 \vDash \Gamma_1$

2. To make the interpretation we just made not valid/not true, such that $L_1 \nvDash \Gamma_1$ holds; we only need to remove something from the interpretation. Removing for instance Tweety from Animal, such that $Animal^I$ = {JollyJumper, Bruce}, then :likes is not valid, since :likes has domain:Animal.

Exercise 1.2: Entailment

1. :Tweety is an animal.

:Tweety rdf:type :Animal

:Tweety rdf:type :Penguin - P

:Penguin rdfs:subClassOf :Bird - P

:Bird rdfs:subClassOf :Animal - P

:Tweety rdf:type :Animal – rdfs11

2. :Tweety likes :JollyJumper.

From :Tweety can only derive :favouriteFood or :Penguin.

:JollyJumper likes :Tweety only goes one way and not both ways, so not possible to entail.

A countermodel is simply using the one from task 1.1.1, which in fact states that $likes^I$ = {<JollyJumper, Tweety>}, where <Tweety, JollyJumper> $\notin$ $likes^I$. Hence, proven that :likes is only valid from :JollyJumper to :Tweety.

3. :Food is the range of :favouriteFood.

:favouriteFood rdfs:subPropertyOf :eats - P

:eats rdfs:range :Food - P

:Food rdfs:range :favouriteFood – rdfs5

4. :Bruce has some favourite food.

:Bruce does not have the property :favouriteFood in it's domain. Therefore we cannot say :Bruce has some favourite food. But, it could be stated that in a OWA assumption we could say :Bruce has some favourite food, but it has not been added to the model yet.

A countermodel is simply just using the model from 1.1.1 which states that $favouriteFood^I$ = {<JollyJumper, b>}, and the tuple <Bruce, b> $\notin$ $favouriteFood^I$, hence proven.

5. :Bruce is a vegetable.

:Bruce rdf:type :Fish

:Fish rdfs:subClassOf :Food

:Vegetable rdfs:subClassOf :Food

:Bruce is not a :Vegetable, but :Bruce is a :Fish which is a subclass of :Food, and :Vegetable is also a subclass of :Food, but that does not mean that it is possible to entail that :Bruce is a :Vegetable.

Countermodel:

For a valid countermodel we could use L_1 from 1.1.1 since it states that $:Bruce^I \notin Vegetable^I$, hence proven.

6. :Bruce is a horse

:Bruce :hasNickname "Alonso" - P

:hasNickName rdfs:domain :Horse - P

:Bruce is a horse – rdfs2

7. :Bruce is a fish.

:Bruce rdf:type :Fish – rdf1

:Bruce is a fish

2: Semantic web and reasoning

1. Closed world assumption also known as “CWA” means the knowledge base is complete concerning all relevant knowledge. In contrast, we have the open world assumption “OWA”, which states that a knowledge base is incomplete.

   For example, if you are constructing a “CWA” system for finding out if a country exists which is relying on a database, then the system only knows the given knowledge in the database. If you ask the system for a country which is not in the database, then the system returns something like “could not find” or “No”. Which is the expected answer considering CWA paradigm. In contrast, if the system was built on the “OWA” paradigm, then the system would have instead returned “I don’t know”, or that it is unknown if “Norway” is a country. Instead of relying on a CWA system which states everything that is not in the knowledge base as false, OWA instead assumes the system has incomplete information or does not know yet (wants to seek more information).

   Let’s say the system assumes the statement “a person can only have one citizenship” is true in a CWA system. If we add new knowledge to a CWA system, like for instance a person having citizenship in both Norway and Sweden, then it will give an error, because we stated that a person can only have one citizenship and we are considering the countries as being different. In contrast, if the system was built on OWA, then it would infer a new statement like “if a person can only hold one citizenship, and if the person has citizenship in both Sweden and Norway, then Norway and Sweden have to be the same thing”. We do not assume the countries are different in OWA.

   Semantic Web is using the OWA assumption. This is because the web is incomplete, we need to have the possibility to add and infer new information consistently. This gives us the possibility to assume things, and later add logic which falsifies the assumptions or makes them correct.
2. Unique name assumption assumes that different named individuals are in fact different. But if we add declarations as in OWL: owl:SameAs, then we can declare that two different named individuals are the same thing. In contrast, non-unique name assumption is the exact opposite. We cannot assume that individuals with different

names are different from each other unless it stated otherwise. The Semantic Web is based upon the non-unique assumption. Let's say two people are creating an identifier for the person "Kong Harald", and the identifiers are different. Then if we are using non-unique name assumption, we can assume that the identifiers can in fact be identifying the same person, even though the identifiers are different. Using this assumption makes it easier to add knowledge about individuals, without having to always coordinate on having one identifier about an individual. In short, there are many things in the universe known by many names!

3. Forward rule chaining is reasoning from premises to conclusions of rules. This means that we add facts corresponding to the conclusion of rules. In short, we can entail statements from these facts, which means removing or changing facts are in fact going to change the entailed statements. This is better for data, which is going to be accessed many times, may be expensive to compute and does not change very often. Backward chaining is the opposite. We add facts from conclusions of rules, by adding what needs to be true for the conclusion to hold. In short, we add facts from entailed statements. This is better for data that does not need to be computed many times and ensures that the answers can come from multiple dynamic sources. It also ensures to not to use more storage space than needed.
4. RDFS entailment are sound with respect to the RDFS semantics if B can be derived from A, then A entails B. Soundness ensures that there's is no wrong answers, where if something is entailed then it is really entailed.
5. RDFS entailment rules are not complete with respect to RDF semantics when there exists a statement in which we cannot derive using the given rules in our semantics. To make it complete we need to add new rules, which in short means that we have enough rules to find any entailment.