1	2	3	4	5	6	7	total
40	20	20	20	20	20	20	160

Name:____ UMBC ID: ____________________

UMBC CMSC 491/691 Final Exam, 17 December 2018

Write all of your answers on this exam, using the blank side of pages if you need more room. The exam is closed book but you are allowed one piece of letter-sized paper with notes. You have the two hours to work on this exam. Good luck.

1. True/False (40 points)

T F XML's DTD XML schema language is more expressive than the XSD schema language. False

T F An XML element cannot have two attributes with the same name. True

T F Since an XML document must have a tree structure, it cannot represent an arbitrary graph structure. False

T F In an RDF graph, it's impossible to have a node that is not the subject or object of a triple. True

T F One limitation of RDF is that it can only represent simple binary relations. False

T F An RDF triple cannot have a literal as its subject. True

T *F* *(:A rdfs:subClassOf :B)* and (*:B rdfs:subClassOf :A)* implies that :A and :B are equivalent classes. True

T F The RDFs data model does not allow a class to override information inherited from its super-classes. True

T F A property with identical domain and range values is necessarily a *owl:SymmetricProperty*. False

T F One cannot express the fact that every person has a mother using only RDFS. True

T F (*owl:Nothing rdfs:subClassOf owl:Thing)* is true. True

T F No OWL property can be both a *owl:FunctionalProperty* and *owl:InverseFunctionalProperty*. False

T F OWL2 added the possibility of a term denoting both an OWL class and an individual. True

T F If an OWL reasoner is sound, it will produce all relations that are logically entailed. False

T F The OWL RL profile includes only those features that can be inferred by SWRL rules. True

T F SWRL rules can infer facts from a graph that an OWL-DL reasoner cannot. True

T F A SPARQL SELECT query always returns a sub-graph of the knowledge graph satisfying it. False

T F A SPARQL DESCRIBE query is used to update a knowledge-graph. False

T F Any information that can be embedded in an HTML document using RDFa can also be embedded using Microdata. False
T F The LD in the name JSON-LD stands for *Logical Data*. False

2. Modeling in RDFS and OWL (20 points)

Suppose you manage an apartment building that allows pets, but only if they are dogs, cats or birds. You want to annotate your listings with owl data by completing the description in the box to the right to define a new class, :OkPet, that includes only dogs, cats and birds.

```
:Person a owl:Class.
:Dog a owl:Class.
:Cat a owl:Class.
:Bird a owl:Class.
```

2.1 Explain in a few sentences why it cannot be done in RDFS.

We could try defining OkPet as a super-class of :Dog, :Cat and :Bird, but the open world assumption underlying RDFS would mean that there might be other classes of animals, like :Snake, that are also sub-classes of :OkPet. There is no way to rule this out using only RDF and RDFS schema terms.

2.2 Is this possible in OWL? If so, show the triples you should add in Turtle, if not explain why it is not possible.

Here are three versions, the first two of which will expand to the third in Protege. This defines the :OkPet class to be equal to the union of the classes :Dog, :Cat and :Bird.

```
:OkPet owl:unionOf (:Dog :Cat :Bird).

:OkPet a owl:Class;
  owl:unionOf  (:Dog :Cat :Bird).

:OkPet a owl:Class;
  owl:equivalentClass [a owl:Class ; owl:unionOf ( :Dog :Cat :Bird)].
```

3. Negation in OWL (20)

OWL's ability to express what is **not** true is limited because allowing full negation can make reasoning more complex or even undecidable. Describe how you can specify the following negative statements in OWL, preferably by giving one or more statements in Turtle

```
:Animal a owl:Class.
:Person a Animal.
:Beast a Animal.
:Dog a :Beast.
:Cat a :Beast.
:Bird a :Beast.
:hasPet rdfs:domain :Person;
          rdfs:range :Beast.
:fido a owl:NamedIndividual.
:felix a owl:NamedIndividual.
:john a owl:NamedIndividual.
:mary a owl:NamedIndividual.
```

3.1 No Beast is also a Person

:Beast owl:disjointFrom :Person.

3.2 :fido is an Beast, but is not a Cat.

:NonCat owl:complimentOf :Cat.
:fido a :Beast, NonCat.

Note: this can also be done without naming the complement, e.g.
:fido a :Beast, [a owl:Class; owl:complimentOf :Cat]

3.3 :fido is not the same as :felix.

:fido owl:differentFrom :felix.

3.4 :john has no pets.

```
:john a [ rdf:type owl:Restriction ;
          owl:onProperty :hasPet ;
          owl:cardinality "0"] .
```

3.5 :fido is not a pet of :mary

```
[ owl:NegativePropertyAssertion ;
  owl:sourceIndividual  :mary ;
  owl:assertionProperty :hasPet ;
  owl:targetIndividual  :fido ] .
```

4. Inferences in OWL (20 points)

Suppose we have the knowledge graph shown in the box to the right. Show all additional triples that can be inferred by a DL-reasoner from the graph that have either :google, :spichai or :sundarPichai as a subject or object.

```
:Company a owl:Class .
:Person a owl:Class .

:worksFor  a owl:ObjectProperty;
   rdfs:domain :Person;
   rdfs:range :Company;
   owl:inverseOf :hasEmployee.

:ceoOf rdfs:subPropertyOf :worksFor;
   rdfs:domain :Person;
   rdfs:range :Company;
   a owl:InverseFunctionalProperty;
   owl:inverseOf :hasCEO.

:google :hasCEO :spichai.
:sundarPichai :ceoOf :google.
```

```
:google a :Company
:google :hasCEO :spichai .
:google :hasEmployee :spichai .
:google :hasCEO :sundarPichai.
:google :hasEmployee :sundarPichai.

:spichai a :Person.
:spichai :worksFor :google.
:spichai :ceoOf :google.
:spichai owd:sameIndividualAs :sundarPichai.

:sundarPichai a :Person.
:sundarPichai :worksFor :google.
:sundarPichai :ceoOf :google.
:sundarPichai owl:sameIndividualAs :spichai.
```

5. Finding co-workers with SPARLQ query (20 points)

Assume we have the knowledge graph shown in in the box to the right with instances of *:Person* and *:Company* and two relations, *:partOf* that links a company to another company it is part of, and *:worksFor*, that links a person to the company they (directly) work for.

```
:partOf a owl:ObjectProperty;
  rdfs:domain :Company;
  rdfs:range :Company.
:coworker a owl:ObjectProperty;
  rdfs:domain :Person;
  rdfs:range :Person.

:loon   :partOf :google.
:google :partOf :alphabet.
:waymo  :partOf :alphabet.
:john :worksFor :loon.
:mary :worksFor :waymo.
```

Write a SPARQL query to find co-workers, i.e. pairs of people who work for the same company or for companies are ultimately part of the same company (e.g., :loon and :waymo in the example). In this example, :john and :mary are co-workers. Your query should **not** return a pair where both people are the same (e.g., :mary and :mary)

HINT: use property chains and the * or + operators and a FILTER condition.

```
PREFIX  : <http://ex.org/q5#>
SELECT * WHERE {
  ?p1 :worksFor/:partOf* ?c1.
  ?p2 :worksFor/:partOf* ?c1.
  filter (?p1 != ?p2)
 }
```

6. Finding co-workers with rules (20 points)

Suppose we want to enable a triple store to infer **:coworker** relations that holds for two people if they work for the same company or for companies are ultimately part of the same company. The box on the right shows the new relations we want to infer in bold. As in the previous problem, the :coworker relation should not hold between a person and herself.

Show additional owl assertions and rules in using N3 or SWRL notation that will allow a system with a owl-DL reasoner and rule-based engine to add **:coworker** relations.

HINT: You can do this with just rules, or with a combination of additional OWL axioms and rules.

```
:partOf a owl:ObjectProperty;
  rdfs:domain :Company;
  rdfs:range :Company.
:coworker a owl:ObjectProperty;
  rdfs:domain :Person;
  rdfs:range :Person.

:loon   :partOf :google.
:google :partOf :alphabet.
:waymo  :partOf :alphabet.
:john :worksFor :loon.
:mary :worksFor :waymo.

:mary :coworker :john.
:john :coworker :mary.
```

Add additional OWL axioms in Turtle syntax

```
:partOf a owl:TransitiveProperty

You might also include :partOf a owl:reflexiveProperty
```

Rules in N3 or SWRL notation

(1) worksFor(?p1, ?c1), worksFor(?p2, ?c2), partOf(?c1, c), part of(?c2, ?c) -> coworker(?p1, ?p2)
company(?x) -> partOf(?x, ?x)

(2) if you do include the owl axiom that :partOf is reflexive:

worksFor(?p1, ?c1), worksFor(?p2, ?c2), partOf(?c1, c), part of(?c2, ?c) -> coworker(?p1, ?p2)

(3) if you don't define part of as transitive in owl, you can do it with a rule

partOf(?x, ?y), partOf(?y, ?z) -> partOf(?x, ?z)

7. Default reasoning in OWL and SPARQL (20)

Default reasoning allows a system to make assumptions in the absence of contradictory evidence. Suppose the :Student class has three subclasses, :FullTimeStudent, :PartTimeStudent and :SpecialStudent, and we follow a policy of assuming an instance of a :Student is a :FullTimeStudent if we don't know what subclass she belongs to. Such reasoning is not supported in OWL, but can be achieved using SPARQL.

```
:Student a owl:Class.
:FullTimeStudent
  rdfs:subClassOf :Student.
:PartTimeStudent
  rdfs:subClassOf :Student.
:SpecialStudent
  rdfs:subClassOf :Student.
```

Write a SPARQL Select query that finds students who are either known to be instances of :FullTimeStudent or can be assumed to be a :FullTimeStudent using the policy above.

```
Select ?s WHERE {
  {?s a :FullTimeStudent}
  UNION
  {?s a :Student
     FILTER NOT EXISTS {?s a :PartTimeStudent}
     FILTER NOT EXISTS {?s a :SpecialStudent}
  }
}
```