UNIVERSITY OF OSLO

Faculty of mathematics and natural sciences

Examination in	INF3580/INF4580 — Semantic Technologies
Day of examination:	17 June 2016
Examination hours:	14:30 – 18:30

This problem set consists of 11 pages.

Appendices:	None
Permitted aids:	Any printed or written course material

Please make sure that your copy of the problem set is complete before you attempt to answer anything.

The exam consists of 5 questions with equal weight.

Problem 1 RDF/D2R (20 %)

Given the following D2RQ mapping file:

```
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix d2rq: <http://www.wiwiss.fu-berlin.de/suhl/bizer/D2RQ/0.1#> .
@prefix map: <http://inf3580.no/mapping/> .
@prefix emp: <http://inf3580.no/data/> .

_:Employee a d2rq:ClassMap ;
    d2rq:dataStorage map:EmployeeDB ;
    d2rq:uriPattern "http://inf3580.no/data/emp@@Employee.ID@@" ;
    d2rq:class emp:Employee ;
    d2rq:classDefinitionLabel "Employee"@en, "Ansatt"@no .

_:EmployeeName a d2rq:PropertyBridge ;
    d2rq:belongsToClassMap _:Employee ;
    d2rq:property emp:name ;
    d2rq:column "Employee.Name" .
```

and the following DB table:

(Continued on page 2.)

Employee:

Id	Name	Boss
1	"Green, Eric"	2
2	"Johnson, Carl"	3
3	"Teller, Anne"	3

where each row represents one employee: the ID-column contains the employee's unique ID, the Name-column contains the employee's name, and the Boss-column contains the employee ID of the boss of the employee.

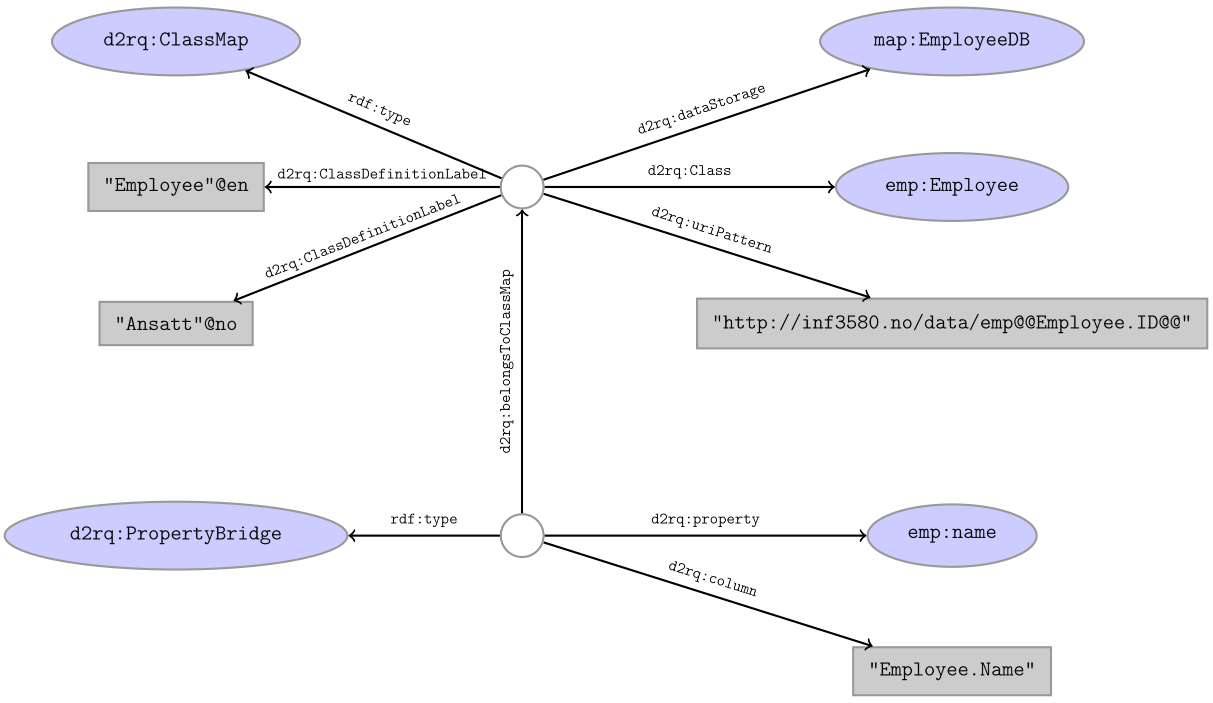
(a) Draw the RDF-graph of the triples that represents the mappings, that is, the triples in the D2RQ-mapping file.

(b) Write down the triples generated by the mappings when applied to the Employee database. You can use the prefixes defined in the mappings, so you do not have to write out the full URIs.

(c) Define a mapping that generates triples `x emp:boss y` whenever `x` has `y` as boss according to the Employee table. Note that `emp:boss` is a property between the individuals generated by the mapping `_:Employee`.

Answer:

(a) The graph is drawn below:



(Continued on page 3.)

(b) The triples generated are

```
emp:Employee rdfs:label "Employee"@en, "Ansatt"@no .
emp:emp1 a emp:Employee ;
    emp:name "Green, Eric" .
emp:emp2 a emp:Employee ;
    emp:name "Johnson, Carl" .
emp:emp3 a emp:Employee ;
    emp:name "Teller, Anne" .
```

(c) The mapping that makes `emp:boss` is:

```
_:EmployeeToBoss a d2rq:PropertyBridge ;
    d2rq:belongsToClassMap _:Employee ;
    d2rq:property emp:boss ;
    d2rq:refersToClassMap _:Employee ;
    d2rq:join "Employee.Boss=>Employee.ID" .
```

or

```
_:EmployeeToBoss a d2rq:PropertyBridge ;
    d2rq:belongsToClassMap _:Employee ;
    d2rq:property emp:boss ;
    d2rq:uriPattern "http://inf3580.no/data/emp@@Boss.ID@@" .
```

Problem 2 SPARQL (20 %)

Assume we have the following RDFS classes

- `hs:House` - The class of all houses.
- `hs:Sale` - The class of house sales.
- `hs:Distance` - The class of distances, each distance has an integer value (in meters), and is related to at least two houses, representing the distance between the two.

and properties

- `hs:house` - The relationship between a sale and the house for sale.

(Continued on page 4.)

- `hs:price` - The relationship between a sale and a positive integer literal representing the price (in NOK).
- `hs:address` - The relationship between a house and the a string denoting that house's address.
- `hs:openHouseDate` - The relationship between a sale and the date (`xsd:dateTime`) of the open house for that sale.
- `hs:distanceValue` - The relationship between a distance individual and the integer literal denoting the value of the distance in meters.
- `hs:between` - The relationship between a distance instance and a house instance. Every distance is related to at least two houses via this relation, such that if `:d hs:between :h1, :h2.` then `:d` describes the distance between `:h1` and `:h2`.

Below are some example triples:

```
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix hs: <http://inf3580.no/houses/> .

hs:house1 a hs:House ;
          hs:address "Problemveien 7, 0123 Oslo" .

hs:house2 a hs:House ;
          hs:address "Moldegata 1, 1234 Oslo" .

# The distance between house1 and house2 is 2500m
[] a hs:Distance ;
   hs:between hs:house1, hs:house2 ;
   hs:distanceValue "2500"^^xsd:int .

# house1 is for sale with a price of 1500000 NOK
# with open house 17.06.16.
[] a hs:Sale ;
   hs:house hs:house1 ;
   hs:price "1500000"^^xsd:int ;
   hs:openHouseDate "2016-06-17"^^xsd:dateTime .

[] a hs:Sale ;
   hs:house [a hs:House;
             hs:address "Nygata 4, 2345 Oslo"] ;
   hs:openHouseDate "2015-01-02"^^xsd:dateTime .
```

(a) Write a SPARQL query that lists the price and date of all sales with an open house date in 2016.

(Continued on page 5.)

(b) Assume that not all sales have a price set. Write a SPARQL query that for all sales lists the address of the house to be sold, such that if there is a price, the price should be less than 1,000,000 NOK.

(c) Write a SPARQL query that returns true if there are two sales for the same house but with different prices, and false otherwise.

(d) Write a SPARQL query that constructs an RDF-graph of triples `x rdf:type hs:ExpensiveHouse` for every house `x` that either has a price greater than 10,000,000 NOK or where the house is closer than 500 meters to a house that has a price greater than 10,000,000 NOK.

(e) Let two houses be close neighbors if the distance between them is less than 100 meters. Write a SPARQL query that for all houses that have less than 5 close neighbors, lists the address of the house and the number of close neighbors. The list should be ordered by the number of close neighbors from lowest to highest.

Answer:

(a)
```
 SELECT DISTINCT ?date, ?price
 WHERE {
   [] a hs:Sale ;
      hs:openHouseDate ?date ;
      hs:price ?price .
   FILTER (?date >= "2016-01-01"^^xsd:dateTime &&
           ?date < "2017-01-01"^^xsd:dateTime)
}
```

(b)
```
 SELECT DISTINCT ?address
 WHERE {
   _:sale a hs:Sale ;
          hs:house [a hs:House ;
                    hs:address ?address] .
   FILTER NOT EXISTS {
     _:sale hs:price ?price .
     FILTER (?price > "1000000"^^xsd:int)
   }
 }
```

(c)
```
 ASK
 WHERE {
   [] a hs:Sale ;
      hs:price ?price1 ;
```

(Continued on page 6.)

```
             hs:house ?house .

          [] a hs:Sale ;
             hs:price ?price2 ;
             hs:house ?house .
         FILTER (price1 != price2);
      }
```

(d)

```
      CONSTRUCT { ?x rdf:type hs:ExpensiveHouse }
      WHERE {
        {
          ?x a hs:House .

          [] a hs:Sale ;
             hs:price ?price ;
             hs:house ?x .
          FILTER (?price > 10000000)
        }
        UNION
        {
          ?x a hs:House .

          [] a hs:Sale ;
             hs:price ?price ;
             hs:house ?o .

          ?o a hs:House .

          [] a hs:Distance ;
             hs:between ?o, ?x ;
             hs:distanceValue ?distVal .
          FILTER (?price > 10000000 && ?distVal < 500)
        }
      }
```

(e)

```
        SELECT ?address, count(?close) AS ?count
        WHERE {
          ?house a hs:House ;
                 hs:address ?address .

          ?close a hs:House .

          [] a hs:Distance ;
             hs:between ?house, ?close ;
```

(Continued on page 7.)

```
                  hs:distanceValue ?distVal .

                FILTER (distVal < 100)
              } GROUP BY ?address HAVING (?count < 5)
                ORDER BY ASC(?count)
```

Problem 3 RDFS Reasoning (20 %)

Consider the following triples about companies, and their roles as being each others suppliers and/or customers:

```
     @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
     @prefix rdfs:<http://www.w3.org/2000/01/rdf-schema#> .
     @prefix : <http://www.ifi.uio.no/companies#> .
 (1) :Customer rdfs:subClassOf :Company .
 (2) :Supplier rdfs:subClassOf :Company .
 (3) :hasCustomer rdfs:domain :Supplier .
 (4) :hasCustomer rdfs:range :Customer .
 (5) :hasSupplier rdfs:domain :Customer .
 (6) :hasSupplier rdfs:range :Supplier .
 (7) :hasCustomer rdfs:subPropertyOf :hasAssociate  .
 (8) :hasSupplier rdfs:subPropertyOf :hasAssociate  .
 (9) :maxMusli :hasCustomer :coop .
(10) :maxMusli :hasSupplier :allNuts .
(11) :coop  :hasSupplier _:1 .
(12) _:1  :hasSupplier :allNuts .
```

For the triples in (a)–(c), and the set of triples in (d), either give a derivation using the rules of RDFS and simple entailment, or give a short explanation of why such a derivation does not exist. If no derivation exists, also indicate whether the statement is entailed or not (under the simplified RDF/RDFS semantics used in the course).

(a) `:allNuts a :Company .`

(b) `:hasAssociate rdfs:range :Company .`

(c) `:coop :hasSupplier :maxMusli .`

(Continued on page 8.)

(d) `:coop :hasAssociate _:a .`
`_:a a :Customer .`

(e) Give an example of a triple that is entailed (under the simplified RDF/RDFS semantics used in the course) by this RDF graph, but that cannot be derived by the RDFS and simple entailment rules.

(f) Intuitively, when one company is a customer of another, that other company is a supplier for the first one, and *vice versa.* Is it possible in RDFS to specify that relationship between `:hasSupplier` and `:hasCustomer`? Are there any other ways to achieve this?

Answer:

(a)

(a1) `:allNuts a :Supplier` (by rule rdfs3 on (6), (10) or (12))

(a2) `:allNuts a :Company` (by rule rdfs9 on (2), (a))

(b) The only information given about `:hasAssociate` is that `:hasCustomer` and `:hasSupplier` are subproperties. So `:hasAssociate` could relate many other resources, that are not in `:Company`. Therefore, this triple is not entailed, and consequently it cannot be derived.

(c) This triple is not entailed, and therefore cannot be derived either. In particular, `:maxMusli :hasCustomer :coop` does not entail this triple, since the RDFS axioms don't express anything about the relationship between `:hasCustomer` and `:hasSupplier`. And furthermore, triples (11) and (12) don't entail this one, since the blank node can be instantiated by some other resource that is a supplier of `:coop` and that in turn has `:allNuts` as supplier.

(d) (d1) `:coop :hasSupplier _:a` (by rule se1 on (11) assigning `_:1` to the new blank node `_:a`)

(d2) `_:a :hasSupplier :allNuts` (by rule se2 on (12) reusing the assignment for `_:a`)

(d3) `:coop :hasAssociate _:a` (by rule rdfs7 on (8) and (d1))

(d4) `_:a a :Customer` (by rule rdfs2 on (5) and (d2))

(e) Two examples are
`:hasSupplier rdfs:range :Company .`
`:hasSupplier rdfs:domain :Company .`
These, and corresponding triples with `:hasCustomer` are entailed semantically, since the ranges and domains of `:hasCustomer` and `:hasSupplier` are subclasses of `:Customer`. But they are not derivable, since there are no RDFS rules which produce range or domain statements.

(Continued on page 9.)

(f) The relationship to be expressed is that `:hasCustomer` is the *inverse* of `:hasSupplier`. This *cannot* be expressed in RDFS. However, It can be expressed using an OWL axiom. It is also possible to use a rule engine or simply a Java program to add the corresponding inverse relationship for every `:hasCustomer` and `:hasSupplier` triple in a data store.

Problem 4 Description logics/OWL (20 %)

We use the following vocabulary:

Classes: Student, Delivery, Assignment, PendingDelivery, PassedDelivery, FailedDelivery.

Properties: hasDelivery, forAssignment, hasAttempted.

The idea is that hasDelivery connects students to their deliveries. forAssignment connects deliveries to the assignments (e.g. mandatories) the delivery was for. And hasAttempted connects students to the assignments they have attempted.

The Delivery class has several subclasses to indicate whether the delivery was graded as passed or failed, or whether it is still "pending," i.e. it is hasn't been graded.

(a) Write DL axioms that express that every delivery is either pending, or passed, or failed, but only one of these.

(b) Write a DL axiom that expresses that every delivery is for exactly one assignment

(c) Write a DL axiom that expresses that anything that delivers a delivery is a student

(d) Write a DL concept expression for "Student who has at least 5 passed deliveries"

(e) Write a DL concept expression for "Assignment for which there is no pending delivery"

(f) Write a DL axiom that expresses that when a student has handed in a delivery for an assignment, then the student has attempted that assignment.

Answer:

(Continued on page 10.)

(a)
$$\begin{gathered}\mathsf{Delivery} \sqsubseteq \mathsf{PendingDelivery} \sqcup \mathsf{PassedDelivery} \sqcup \mathsf{FailedDelivery}\\ \mathsf{PendingDelivery} \sqcap \mathsf{PassedDelivery} \equiv \bot\\ \mathsf{PendingDelivery} \sqcap \mathsf{FailedDelivery} \equiv \bot\\ \mathsf{PassedDelivery} \sqcap \mathsf{FailedDelivery} \equiv \bot\end{gathered}$$

(b)
$$\mathsf{Delivery} \sqsubseteq {=_1}\, \mathsf{forAssignment}.\mathsf{Assignment}$$

(c)
$$\exists \mathsf{hasDelivery}.\top \sqsubseteq \mathsf{Student}$$

(d)
$$\mathsf{Student} \sqcap {\geq_5}\, \mathsf{hasDelivery}.\mathsf{PassedDelivery}$$

(e)
$$\mathsf{Assignment} \sqcap \neg\exists \mathsf{forAssignment}^{-}.\mathsf{PendingDelivery}$$

(f)
$$\mathsf{hasDelivery} \circ \mathsf{forAssignment} \sqsubseteq \mathsf{hasAttempted}$$

Problem 5 RDF and OWL semantics (20 %)

Consider the following set of description logic axioms:

$$\begin{aligned} A &\sqsubseteq \exists R.B\\ A &\sqsubseteq \exists R.C\\ A &\sqsubseteq {\leq_1}\, R.\top \end{aligned}$$

(a) Provide a DL-interpretation $\mathcal{I}_1$ with $A^{\mathcal{I}_1} \neq \emptyset$ that satisfies all of these axioms, or explain why none exists.

(b) To the first three axioms, add a fourth one:

$$B \sqcap C \sqsubseteq \bot$$

Provide a DL-interpretation $\mathcal{I}_2$ with $A^{\mathcal{I}_2} \neq \emptyset$ that satisfies all of these axioms, or explain why none exists.

Answer:

(Continued on page 11.)

(a) We define

$$\begin{array}{rcl} \Delta^{\mathcal{I}_1} & = & \{a, bc\} \\ A^{\mathcal{I}_1} & = & \{a\} \\ B^{\mathcal{I}_1} & = & \{bc\} \\ C^{\mathcal{I}_1} & = & \{bc\} \\ R^{\mathcal{I}_1} & = & \{\langle a, bc\rangle\} \end{array}$$

So a is R-related to only one domain element, which belongs to the interpretation of both B and C.

(b) We show that these four axioms together cannot be satisfied by an interpretation with $A^{\mathcal{I}_2} \neq \emptyset$.

Assume $a \in A^{\mathcal{I}_2}$. Then due to the first axiom, there must be a domain element $b \in B^{\mathcal{I}_2}$ with $\langle a, b\rangle \in R^{\mathcal{I}_2}$. Due to the second axiom, there must also be a $c \in C^{\mathcal{I}_2}$ with $\langle a, c\rangle \in R^{\mathcal{I}_2}$.

The fourth axiom tells us that $B^{\mathcal{I}_2}$ and $C^{\mathcal{I}_2}$ have no common elements, so $b \neq c$. Therefore $\{x \mid \langle a, x\rangle \in R^{\mathcal{I}_2}\}$ has at least two elements, and there fore the third axiom cannot be satisfied.