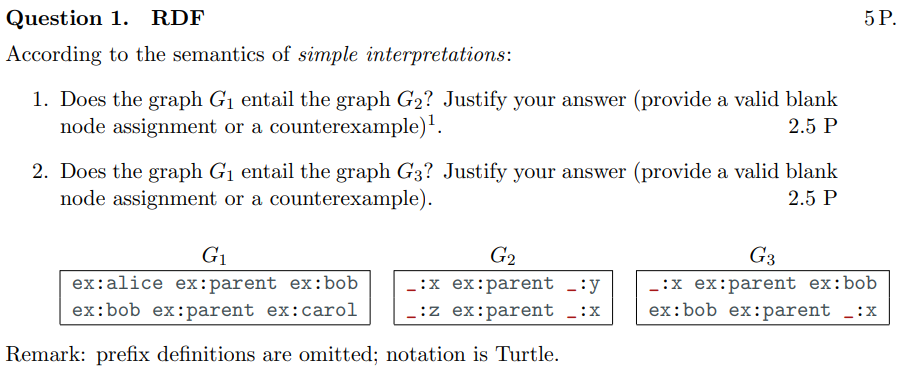


# Question 1. RDF (tot 5 P.)



## 1.1

In the graph *G2* there are three blank nodes \_:x, \_:y and \_:z. To check if G1 entails G2 we need to try to make some substitutions. If we substitute \_:x with ex:bob, \_:y with ex:carol and \_:z with ex:alice we get G1. So we can conclude that G1 entails the graph G2

**To be more specific also add**

α’(\_:x) = ex:bobi

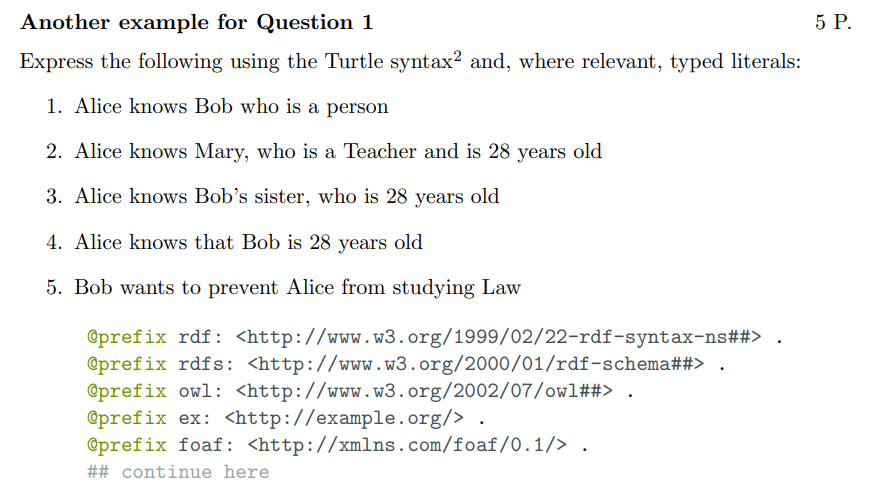
α’(\_:y) = ex:caroli

α’(\_:z) = ex:alicei

## 1.2

The graph G1 does not entail G3. This is because even if we try all the possible substitutions of the blank node \_:x we can not reach the same interpretation of the graph G1

# Another Question 1 (tot 5 P.)



## 1b.1

ex:Person rdf:type owl:Class.

ex:alice foaf:know ex:bob.

ex:bob rdf:type ex:Person.

## 1b.2

ex:Job rdf:type owl:Class.

ex:teacher rdf:type ex:Job.

ex:occupation rdf:type owl:ObjectProperty;

rdfs:domain ex:Person;

rdfs:range ex:Job.

ex:yearsOld rdf:type owl:ObjectProperty;

rdfs:domain ex:Person;

rdfs:range xsd:integer.

ex:alice foaf:know ex:mary.

ex:mary ex:occupation ex:teacher.

ex:mary ex:yearsOld “28”^^xsd:integer

**Oppure (più easy)**

ex:Teacher rdf:type owl:Class;

rdfs:subClassOf ex:Person.

ex:alice foaf:know ex:mary.

ex:mary rdf:type ex:Teacher;

ex:yearsOld “28”^^xsd:integer.

## 1b.3

ex:sisterOf rdf:type owl:ObjectProperty;

rdfs:domain ex:Person;

rdfs:range ex:Person.

ex:bob rdf:type ex:Person.

\_:x rdf:type ex:Person.

ex:alice foaf:know \_:x.

\_:x ex:sisterOf ex:bob;

ex:yearsOld “28”^^xsd:integer.

## 1b.4

ex:know rdf:type owl:ObjectProperty;

rdfs:domain ex:Person;

rdfs:range rdf:Statement.

ex:alice ex:know [rdf:type rdf:Statement;

rdf:subject ex:bob;

rdf:predicate ex:yearsOld;

rdf:object “28”^^xsd:integer]

## 1b.5

ex:wantTo rdf:type owl:ObjectProperty;

rdfs:domain ex:Person;

rdfs:range ex:Person.

ex:prevent rdf:type owl:ObjectProperty;

rdfs:domain ex:Person;

rdfs:range rdf:Statement.

ex:Subject rdf:type owl:Class.

ex:law rdf:type ex:Subject.

ex:study rdf:type owl:ObjectProperty;

rdf:domain ex:Person;

rdf:range ex:Subject.

ex:bob ex:wantTo [rdf:type rdf:Statement;

rdf:subject ex:bob;

rdf:predicate ex:prevent [rdf:type rdf:Statement;

rdf:subject ex:bob

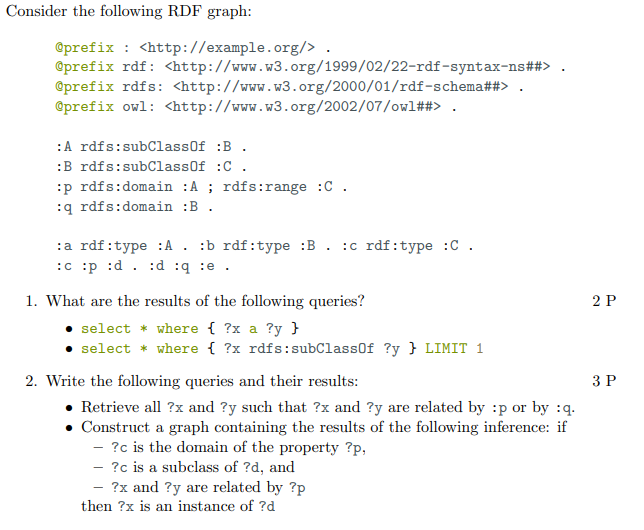
rdf:predicate ex:study

rdf:object ex:law ] ;

rdf:object ex:alice]

# Question 2. SPARQL (tot 5 P.)

## 



## 2.1

### 2.1.1

| ?x | ?y |
| --- | --- |
| :a | :A |
| :b | :B |
| :c | :C |

### 2.1.2

| ?x | ?y |
| --- | --- |
| :A | :B |

## 2.2

### 2.1

SELECT \*

WHERE { {?x :p ?y} UNION {?x :q ?y} }

### 2.2

CONSTRUCT { ?x rdf:type ?d}

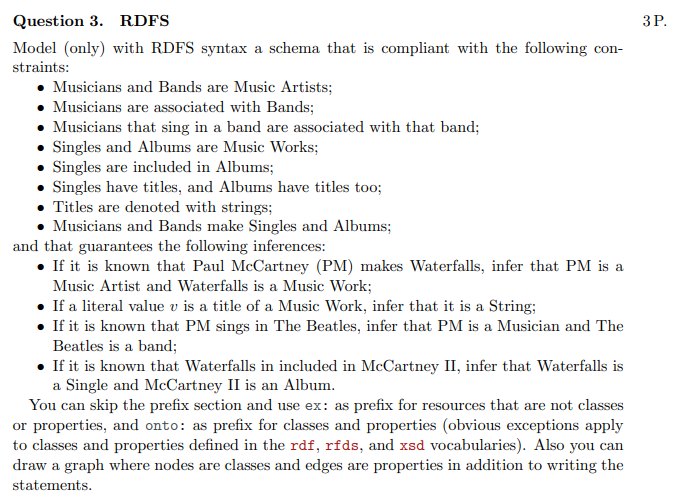
WHERE {?p ~~rdf:type owl:ObjectProperty;~~

rdf:domain ?c.

?c rdfs:subClassOf ?d.

?x ?p ?y}

# Question 3. RDFS (tot 3 P.)



## 3.1

onto:Musicians rdfs:subClassOf onto:MusicArtist.

onto:Bands rdfs:subClassOf onto:MusicArtist.

## 3.2

onto:associated rdf:type rdf:predicate;

rdfs:domain onto:Musicians;

rdfs:range onto:Bands.

~~onto:Musicians onto:associated onto:Bands.~~

## 3.3

onto:sing rdf:type rdf:property;

rdfs:domain onto:Musicians;

rdfs:range onto:Bands.

Or onto:sing rdfs:subClassOf onto:associated.

## 3.4

onto:Singles rdfs:subClassOf onto:MusicWork.

onto:Albums rdfs:subClassOf onto:MusicWork.

## 3.5

onto:included rdf:type rdf:property;

rdfs:domain onto:Singles;

rdfs:range onto:Album.

## 3.6 - 3.7

onto:haveTitle rdf:type rdf:property;

rdfs:domain onto:MusicWork;

rdfs:range ex:title.

ex:title rdf:type xsd:strings.

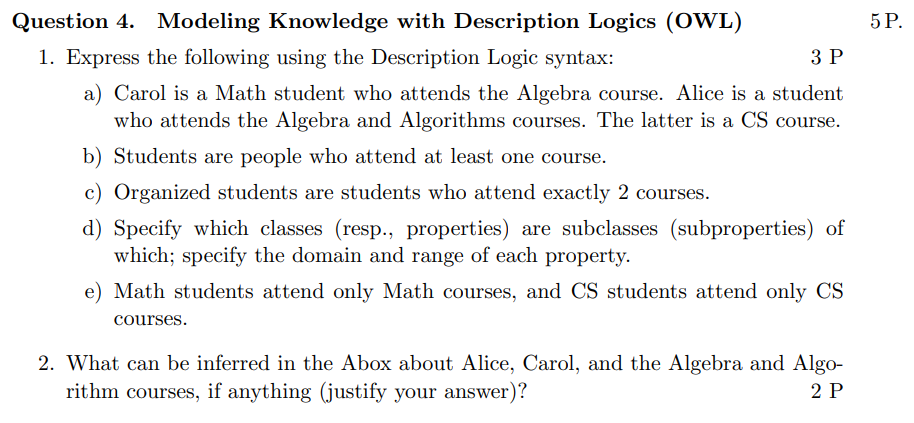
## 3.8

onto:make rdf:type rdf:property;

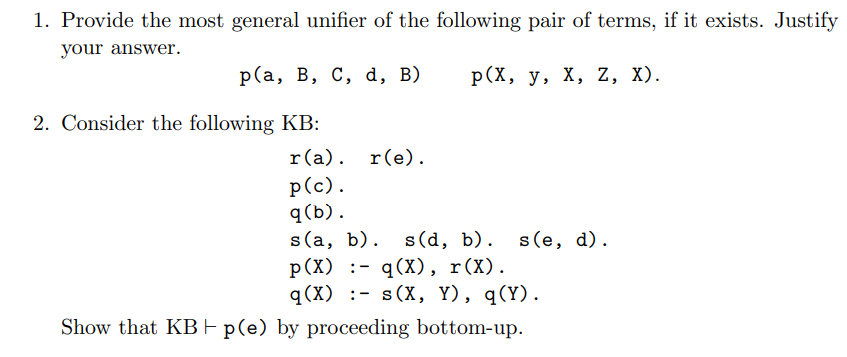
rdfs:domain onto:MusicArtist;

rdfs:range onto:MusicWork.

# !! Question 4. Modelling Knowledge with Description Logic (OWL) (tot 5 P.)



# Question 5. Datalog (tot 5 P.)



## 5.1 with y

p(a, B, C, d, B) p(X, y, X, Z, X)

The most general unifier does not exist. This is because the variable X needs to be substituted with a, the variable B needs to be substituted with y but X and B are equivalent.

## 5.1 with Y

p(a, B, C, d, B) p(X, Y, X, Z, X)

{X/a, Y/a, C/a, Z/d, B/a}

## 5.2

First of all number the clause, we get

1. r(a).
2. r(e).
3. p(c).
4. q(b).
5. s(a,b).
6. s(d,b).
7. s(e,d).
8. p(X) :- q(X), r(X).
9. q(X) :- s(X, Y), q(Y).

Start with an empty set

C = {}

Add the clause from number 1 to number 7

C = { r(a). r(e). p(c). q(b). s(a,b). s(d,b). s(e,d).}

Select clause n.9 and substitute {X/d Y/b}, we get:

C = { r(a). r(e). p(c). q(b). s(a,b). s(d,b). s(e,d). q(d)}

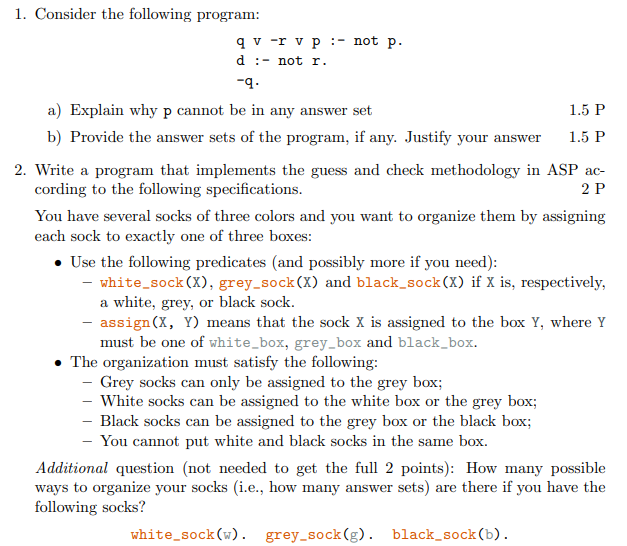
Select clause n.9 and substitute {X/e, Y/d}, we get:

C = { r(a). r(e). p(c). q(b). s(a,b). s(d,b). s(e,d). q(d), q(e).}

Select clause n.8 and substitute {X/e}, we get:

C = { r(a). r(e). p(c). q(b). s(a,b). s(d,b). s(e,d). q(d), q(e). p(e).}

# !! Question n.6 (tot 5 P.)



## 6.1

### 6.1a

p cannot be in any answer set because the only clause with p in the head is the first one and to have it satisfied we need to have the body satisfied, and so we must have *not p* true.

**Also**

p should never be contained by any interpretation because as soon as p is in the interpretation you get rid of the rule q v -r v p :- not p but p does not appear anywhere else in the model

### !! 6.1b

q v -r v p :- not p.

d :- not r.

-q.

{-q, d}

q v -r v p :- ~~not p~~.

d :- ~~not r~~.

-q.

We get:

q v -r v p :-

d :-.

-q.

{-q, d} is not a model

{-q, d, -r}

q v -r v p :- ~~not p~~.

d :- ~~not r.~~

-q.

We get:

q v -r v p :-

d :-

-q.

{-q, d, -r} is a model

## 6.2

### 6.2.1

*Grey socks can only be assigned to the grey box*

aka: grey socks can not go in the black or white box

-assign(x, black\_box) v -assign(x, white\_box) :- grey\_sock(x).

### 6.2.2

*White socks can be assigned to the white box or the grey box*

aka: white socks can not be assigned to the black box

-assign(x, black\_box) :- white\_sock(x).

Grey

### 6.2.3

*Black socks can be assigned to the grey box or the black box*

aka: black socks can not be assigned to the white box

-assigned(x, white\_box) :- black\_sock(x).

### CHECK 6.2.4

*You cannot put white and black socks in the same box*

assign(x, z) :- white\_sock(x), black\_sock(z).

-assign(k, z) :- white\_sock(x), black\_sock(z).

assign(x, z) :- white\_sock(z), black\_sock(x).

-assign(k, z) :- white\_sock(z), black\_sock(x).

Correct: :-assign(X, Y) , assign (XX, Y), white\_sock(X), black\_sock(XX)

# Question 7. Open question (tot 3 P.)



**Lexical ontologies**: they are also called tesauri. Concepts are defined as sets of words that can have similar meaning in similar contexts

**Taxonomies**: hierarchical organisation of objects structure

**Axiomatic ontologies**: used to organise knowledge by defining individuals, classes relations and their properties. Organise knowledge by defining symbols and axioms