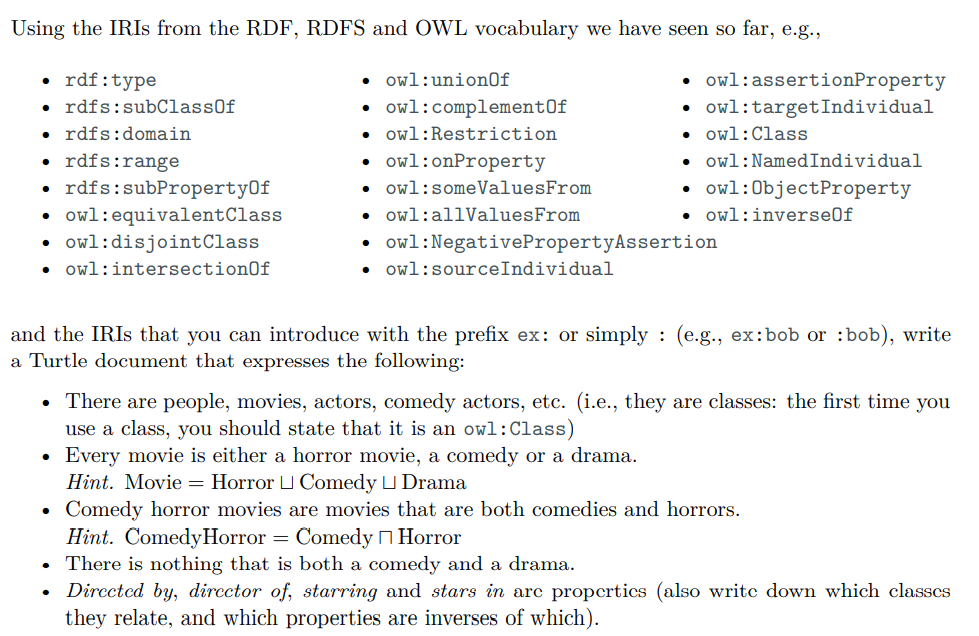
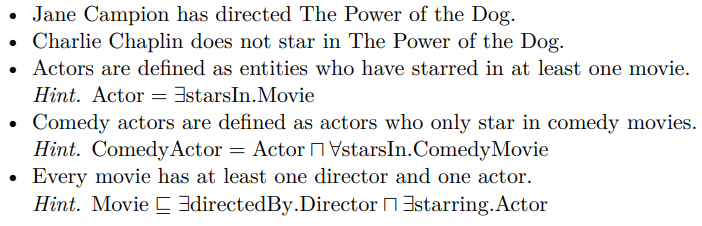
# EXERCISE SESSION 4-5

## E.S. 4-5.1 - OWL Vocabulary





*There are people, movies, actors, comedy actors, etc.*

ex:People rdf:type owl:Class.

ex:Movies rdf:type owl:Class.

ex:Actors rdf:type owl:Class.

ex:ComedyActors rdf:type owl:Class.

*Every movie is either a horror movie, a comedy or a drama.*

ex:Movie owl:equivalentClass [rdf:type owl:Class;

owl:unionOf (ex:HorrorMovie ex:ComedyMovie ex:DramaMovie)].

*Comedy horror movies are movies that are both comedies and horrors.*

ex:ComedyHorrorMovie rdf:type owl:Class;

owl:equivalentClass [rdf:type owl:Class;

owl:intersectionOf (ex:ComedyMovie ex:HorrorMovie)].

*There is nothing that is both a comedy and a drama*

ex:DramaMovie owl:disjointClass ex:ComedyMovie.

*Directed by, director of, starring and stars in are properties (also write down which classes they relate, and which properties are inverses of which)*

ex:directed\_by rdf:type owl:ObjectProperty;

rdfs:domain ex:Movies;

rdfs:range ex:People;

owl:inverseOf director\_of.

ex:director\_of rdf:type owl:ObjectProperty;

rdfs:domain ex:People;

rdfs:range ex:Movie;

owl:inverseOf directed\_by.

ex:starring rdf:type owl:ObjectProperty;

rdfs:domain ex:Movie;

rdfs:range ex:People;

owl:inverseOf ex:stars\_in.

ex:stars\_in rdf:type owl:ObjectProperty;

rdfs:domain ex:People;

rdfs:range ex:Movie;

owl:inverseOf ex:starring.

*Jane Campion has directed The Power of the Dog*

ex:janeCampion rdf:type owl:namedInvidiual.

ex:tpoftd rdf:type owl:namedInvididual.

ex:janeCampion ex:director\_of ex:tpoftd.

*Charlie Chaplin does not star in The Power of the Dog.*

ex:charlieChaplin rdf:type owl:namedInvidiual;

[ rdf:type owl:NegatiePropertyAssertion;

owl:sourceInvidual ex:charlieChaplin;

owl:assertionProperty ex:stars\_in;

owl:targetIndividual ex:tpotd].

*Actors are defined as entities who have starred in at least one movie.*

ex:Actors owl:equivalentClass [rdf:type owl:Restriction;

owl:onProperty ex:stars\_in;

owl:someValuesFrom ex:Movie].

*Comedy actors are defined as actors who only star in comedy movies.*

ex:ComedyActors owl:equivalentClass [rdf:type owl:Restriction;

owl:onProperty ex:stars\_in;

owl:AllValuesFrom ex:ComedyMovies].

(no need to specify also that ex:ComedyActors are ex:Actors if I state that)

ex:ComedyActors owl:subClassOf ex:Actors.

*Every movie has at least one director and one actor.*

ex:Movie owl:equivalentClass [rdf:type owl:Restriction;

owl:unionOf ([rdf:type owl:Restriction;

owl:onProperty ex:directed\_by;

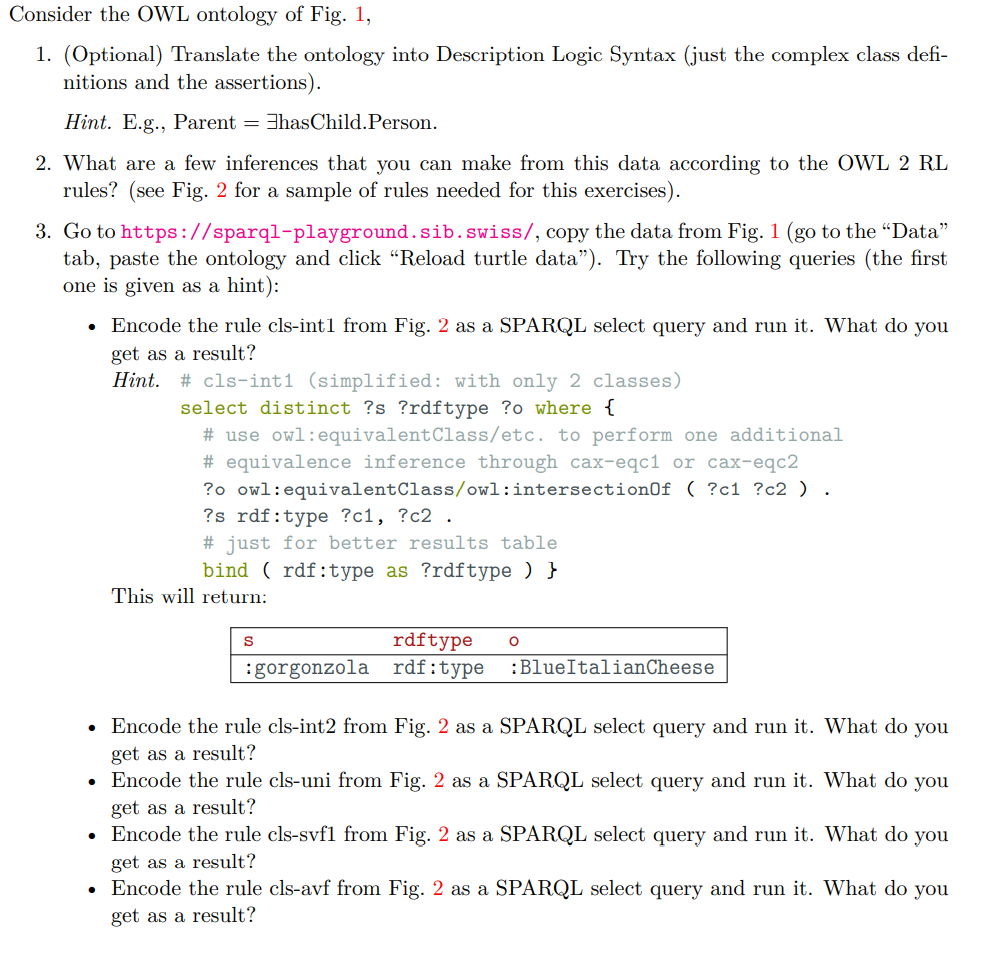
owl:someValuesFrom ex:People]

[rdf:type owl:Restriction;

owl:onProperty ex:starring;

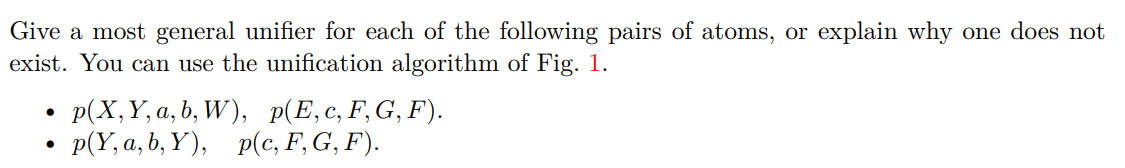
owl:someValuesFrom ex:People] )]

## ⟹ E.S. 4-5.2 - OWL 2 RL Inferences (with SPARQL)



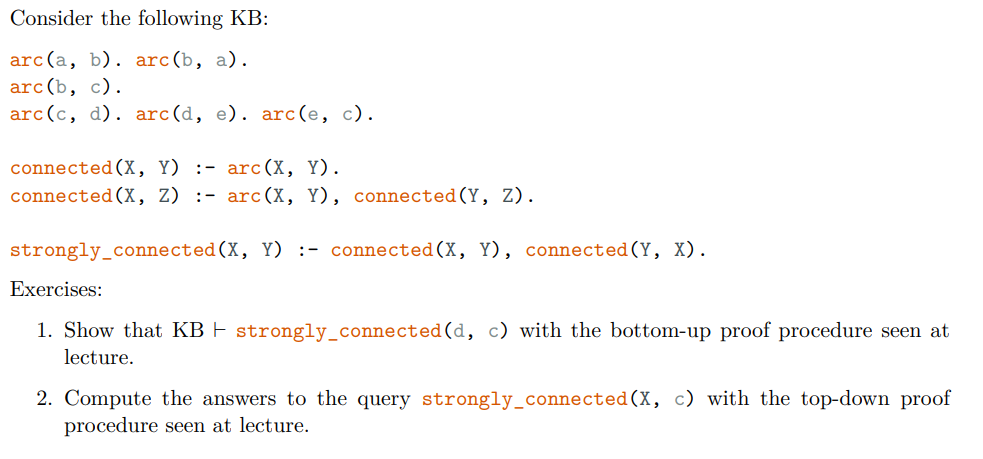
# EXERCISE SESSION 7

## E.S. 7.1 - Unification



* {X/E, Y/c, F/b, W/b}
* Does not exists because Y/F but Y/c and F/a

## E.S. 7.2 - Bottom-up and Top-down



### E.S. 7.2.1

Give a number to all the clause

1. arc(a, b).

2. arc(b, a).

3. arc(b, c).

4. arc(c, d).

5. arc(d, e).

6. arc(e, c).

7. connected(X, Y) :- arc(X, Y).

8. connected(X, Z) :- arc(X, Y), connected(Y, Z).

9. stronlgy\_connected(X, Y) :- connected(X, Y), connected(Y, X).

C = {}

C = {arc(a, b). arc(b, a). arc(b, c). arc(c, d). arc(d, e). arc(e, c).}

Select clause n. 7 and rename {X/e. Y/c}

C = {arc(a, b). arc(b, a). arc(b, c). arc(c, d). arc(d, e). arc(e, c). connected(e, c).}

Select clause n.8 and rename {X/d, Y/e, Z/c}

C = {arc(a, b). arc(b, a). arc(b, c). arc(c, d). arc(d, e). arc(e, c). connected(e, c). connected(d,c)}

Select clause n.7 and rename {X/c, Y/d}

C = {arc(a, b). arc(b, a). arc(b, c). arc(c, d). arc(d, e). arc(e, c). connected(e, c). connected(d,c), connected(c,d)}

Select clause n.9 and rename {X/d, Y/c}

C = {arc(a, b). arc(b, a). arc(b, c). arc(c, d). arc(d, e). arc(e, c). connected(e, c). connected(d,c), connected(c,d), strongly\_connected(d,c)}

### E.S. 7.2.2

1. arc(a, b).

2. arc(b, a).

3. arc(b, c).

4. arc(c, d).

5. arc(d, e).

6. arc(e, c).

7. connected(X, Y) :- arc(X, Y).

8. connected(X, Z) :- arc(X, Y), connected(Y, Z).

9. stronlgy\_connected(X, Y) :- connected(X, Y), connected(Y, X).

We want to find the answer to the query strongly\_connected(X, c).

First of all create che answer clause (ac)

yes(X) :- strongly\_connected(X,c).

Select strongly\_connected(X,c) and choose the clause [9] and rename it such that it becomes stronlgy\_connected(X0, Y0) :- connected(X0, Y0), connected(Y0, X0). Then apply the mgu {X/X0 Y0/c}. We get:

yes(X0) :- connected(X0, c), connected(c, X0).

Select connected(c, X0) and choose clause [7] and rename the variables such that it becomes connected(X1, Y1) :- arc(X1, Y1). Then apply the mgu {X1/c, X0/Y1}. We get

yes(Y1) :- connected(Y1, c), arc(c, Y1).

Select connected(Y1, c) and choose [8] and rename the variables such that it becomes connected(X2, Z2) :- arc(X2, Y2), connected(Y2, Z2). Then apply the mgu {Y1/X2, Z2/c}. We get:

yes(X2) :- arc(X2, Y2), connected(Y2, c), arc(c, X2).

Select connected(Y2, c) and choose [7] and rename the variables such that it becomes connected(X3, Y3) :- arc(X3, Y3). Then apply the mgu {Y2/X3, Y3/c}. We get:

yes(X2) :- arc(X2, X3), arc(X3, c), arc(c, X2).

Select arc(X2, X3) and choose [5] and apply the mgu {X2/d, X3/e}. We get:

yes(d) :- arc(d, e), arc(e, c), arc(c, d).

We can now remove arc(d,e) because we have [5], we get:

yes(d) :- arc(e, c), arc(c, d).

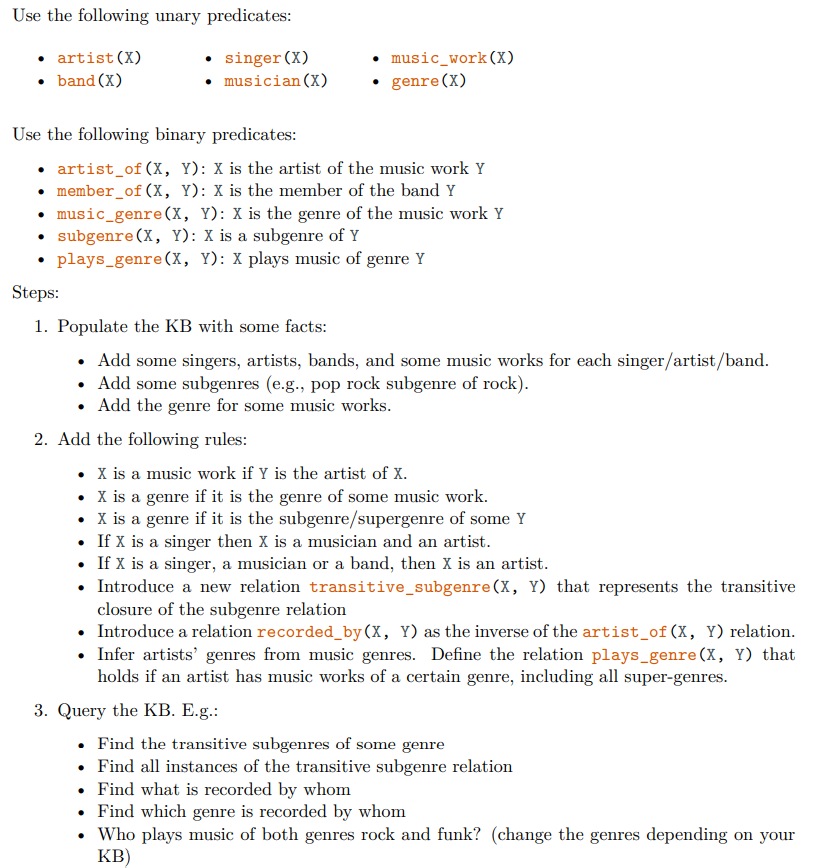
We can now remove arc(e, c) because we have [6], we get:

yes(d) :- arc(c, d).

We can now remove arc(c, d) because we have [4], we get:

yes(d) :-

## E.S. 7.3 - DLV – Music Knowledge Base



musician(bocelli).

singer(eminem).

singer(deandre).

artist(caparezza).

artist(cattelan).

band(beatles).

band(dpg).

music\_work(lose\_your\_self).

music\_work(still\_dre).

subgenre(pop\_rock, rock).

subgenre(trap, rap).

genre(rap).

genre(rock).

*X is a music work if Y is the artist of X*

music\_work(X) :- artist\_of(Y, X).

*X is a genre if it is the genre of some music work*

genre(X) :- music\_genre(X, Y).

*X is a genre if it is the subgenre/supergenre of some Y*

genre(X) :- subgenre(X, Y).

genre(X) :- subgenre(Y, X).

*If X is a singer then X is a musician and an artist*

musician(X) v artist(X):- singer(X).

*If X is a singer, a musician or a band, then X is an artist*

artist(X) :- singer(X).

artist(X) :- musician(X).

artist(X) :- band(X).

*Introduce a new relation transitive\_subgenre(X, Y) that represents the transitive closure of the subgenre relation*

music\_genre(X, Z) :- music\_genre(X, Y), subgenre(X, Z).

music\_genre(X, Z) :- music\_genre(X, Y), subgenre(Z, X).

*Introduce a relation recorded\_by(X, Y) as the inverse of the artist\_of(X, Y) relation.*

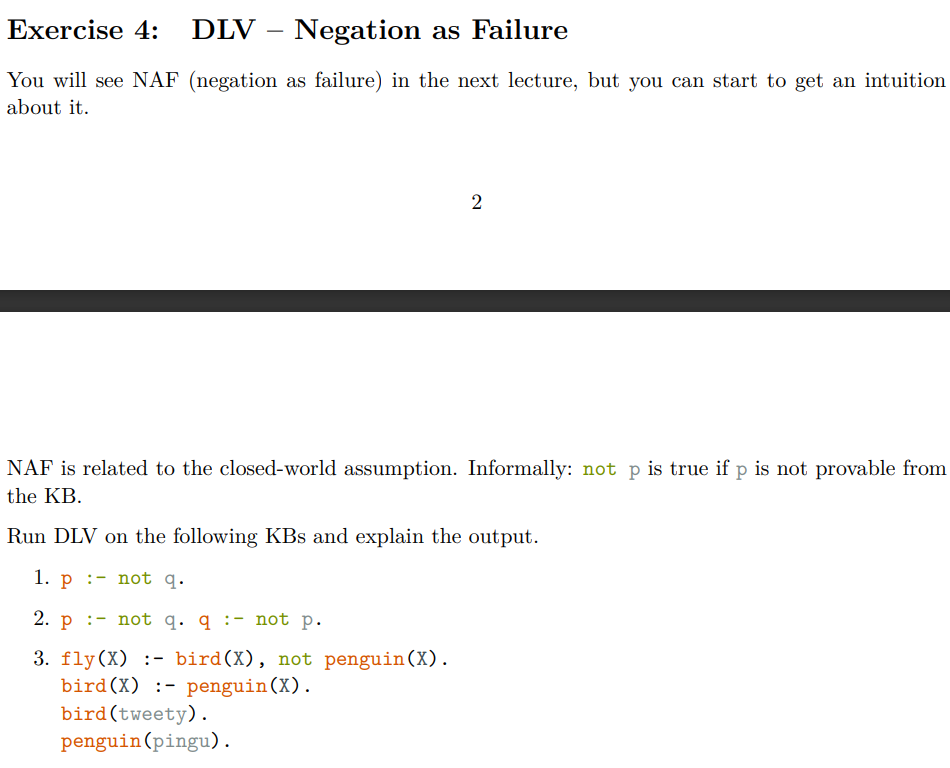
recorded\_by(X,Y) :- artist\_of(Y,X).

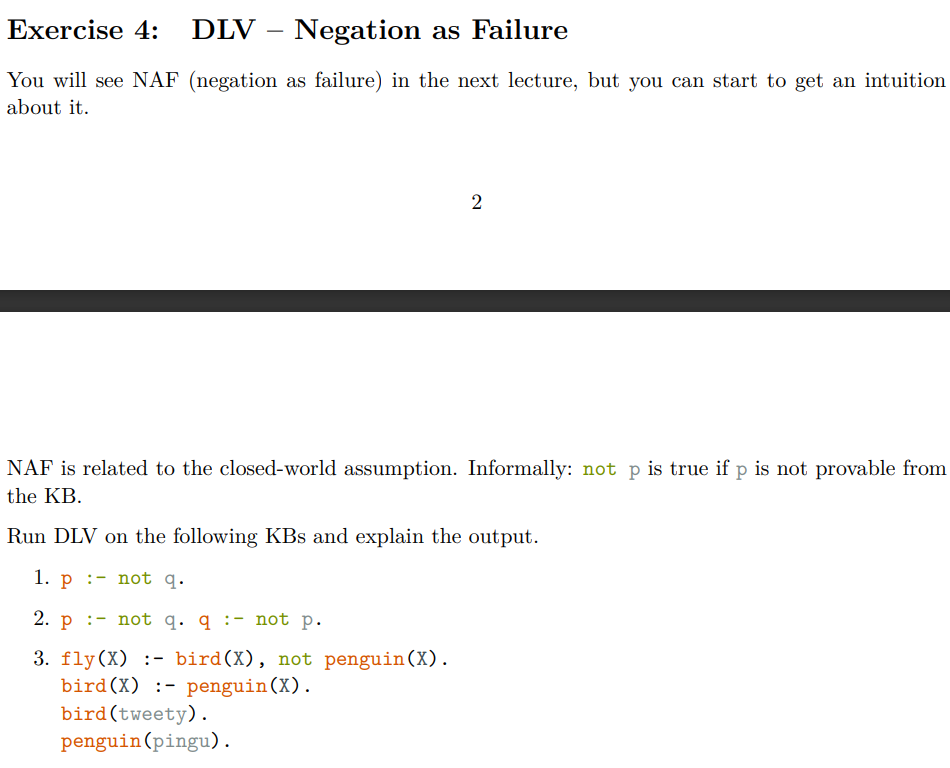
artist\_of(X,Y) :- recorded\_by(Y,X).

*Infer artists’ genres from music genres. Define the relation plays\_genre(X, Y) that holds if an artist has music works of a certain genre, including all super-genres.*

artist\_genre(X,Y) :- artist(X), genre(Y), artist\_of(X, Z), music\_genre(Y, Z).

## E.S. 7.4 - Negation as Failure





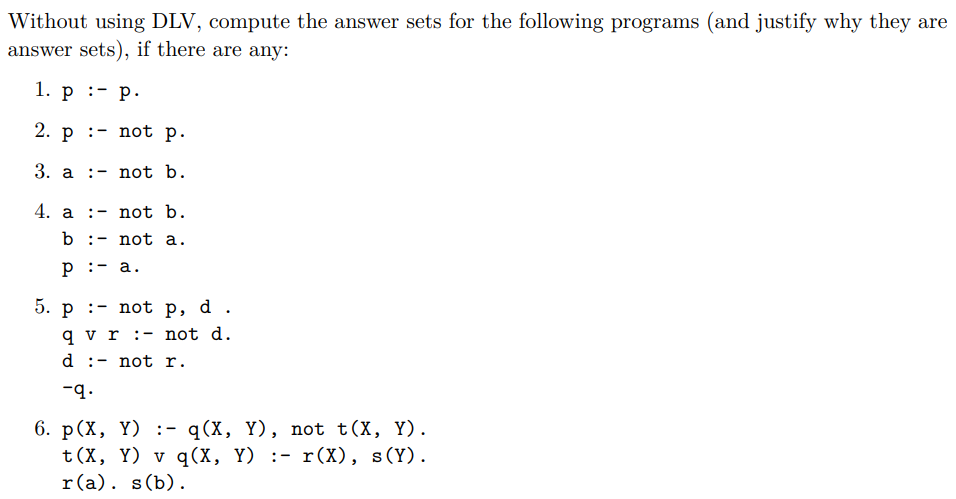
# EXERCISE SESSION 8

## E.S. 8.1 - DLV SEMANTICS

**❓ - How to do it**

If you have {p} remove the entire rule where you have *not p* in the body

If you have {p, e} remove all the *nop p*, *not e* that you have in all the body



p :- p.

{}

Is a an answer set

{p}

p :- p.

Is a model, but is not minimal

p :- not p.

{}

p :- ~~not p~~.

Is not an answer set

{p}

~~p :- not p~~. ⇐ I removed the entire clause because I have *p* true, so I can not have *not p*

a :- not b.

{}

a :- ~~not b~~. ⇐ I removed only *not b* because I don’t have nothing that contradict it

{a}

a :- ~~not b~~. ⇐ I removed only *not b* because I don’t have nothing that contradict it

a is an answer set

{b}

~~a :- not b~~. I remove the entire law because I have *b*

{a,b}

~~a :- not b~~. I remove the entire law because I have *b*

a :- not b.

b :- not a.

p :- a.

{}

a :- ~~not b~~.

b :- ~~not a~~.

p :- a.

Not an answer set

{a}

a :- ~~not b~~.

~~b :- not a~~.

p :- a.

Not an answer set, should have also p

{b}

~~a :- not b~~.

b :- ~~not a~~.

p :- a.

Is an answer set

{p}

a :- ~~not b~~.

b :- ~~not a~~.

p :- a.

Not an answer set, should have also a, b

{a,b}

~~a :- not b~~.

~~b :- not a~~.

p :- a.

Not an answer set, should have also p

{a,p}

a :- ~~not b~~.

~~b :- not a~~.

p :- a.

In an answer set

{b,p}

~~a :- not b~~.

b :- not a.

p :- a.

In an answer set

{a,b,p}

~~a :- not b~~.

~~b :- not a~~.

p :- a.

In an answer set

p :- not p, d.

q v r :- not d.

d :- not r.

-q.

(here we must always have -q)

{}

p :- not p, d.

q v r :- not d.

d :- not r.

-q.

Not an answer set, should also have -q

{-q}

p :- ~~not p~~, d.

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

d :- ~~not r~~.

-q.

Not an answer set, should also have d

{-q, d}

p :- ~~not p~~, d.

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

d :- ~~not r~~.

-q.

Not an answer set, should also have p

{-q, d, p}

~~p :- not p, d~~.

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

d :- ~~not r~~.

-q.

Not an answer set, because it is not minimal

{-q, r}

p :- ~~not p~~, d.

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

~~d :- not r~~.

-q.

Is an answer set

{-q, r, d}

p :- ~~not p~~, d.

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

~~d :- not r~~.

-q.

Not an answer set, should also have p

{-q, r, p}

~~p :- not p, d~~.

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

~~d :- not r~~.

-q.

Is an answer set

p(X, Y) :- q(X, Y), not t(X, Y).

t(X, Y) v q(X, Y) :- r(X), s(Y).

r(a).

s(b).

First of all we need to ground all the rules

p(a, a) :- q(a, a), not t(a, a).

p(a, b) :- q(a, b), not t(a,b).

p(b,a) :- q(b,a), not t(b,a).

p(b,b) :- q(b,b), not t(b,b).

t(a, a) v q(a, a) :- r(a), s(a).

t(a, b) v q(a, b) :- r(a), s(b).

t(b, a) v q(b, a) :- r(b), s(a).

t(b, b) v q(b, b) :- r(b), s(b).

r(a).

s(b).

{r(a). s(b)}

p(a, a) :- q(a, a), ~~not t(a, a)~~.

p(a, b) :- q(a, b), ~~not t(a,b)~~.

p(b,a) :- q(b,a), ~~not t(b,a)~~.

p(b,b) :- q(b,b), ~~not t(b,b)~~.

t(a, a) v q(a, a) :- r(a), s(a).

t(a, b) v q(a, b) :- r(a), s(b).

t(b, a) v q(b, a) :- r(b), s(a).

t(b, b) v q(b, b) :- r(b), s(b).

r(a).

s(b).

Not a model, need to have or t(a,b) or q(a,b)

{r(a). s(b), t(a,b)}

p(a, a) :- q(a, a), ~~not t(a, a)~~.

~~p(a, b) :- q(a, b), not t(a,b)~~.

p(b,a) :- q(b,a), ~~not t(b,a)~~.

p(b,b) :- q(b,b), ~~not t(b,b)~~.

t(a, a) v q(a, a) :- r(a), s(a).

t(a, b) v q(a, b) :- r(a), s(b).

t(b, a) v q(b, a) :- r(b), s(a).

t(b, b) v q(b, b) :- r(b), s(b).

r(a).

s(b).

Is a model

{r(a). s(b), q(a,b)}

p(a, a) :- q(a, a), ~~not t(a, a)~~.

p(a, b) :- q(a, b), ~~not t(a,b)~~.

p(b,a) :- q(b,a), ~~not t(b,a)~~.

p(b,b) :- q(b,b), ~~not t(b,b)~~.

t(a, a) v q(a, a) :- r(a), s(a).

t(a, b) v q(a, b) :- r(a), s(b).

t(b, a) v q(b, a) :- r(b), s(a).

t(b, b) v q(b, b) :- r(b), s(b).

r(a).

s(b).

Not a model, should also have p(a,b)

{r(a). s(b), q(a,b), p(a,b)}

p(a, a) :- q(a, a), ~~not t(a, a)~~.

p(a, b) :- q(a, b), ~~not t(a,b)~~.

p(b,a) :- q(b,a), ~~not t(b,a)~~.

p(b,b) :- q(b,b), ~~not t(b,b)~~.

t(a, a) v q(a, a) :- r(a), s(a).

t(a, b) v q(a, b) :- r(a), s(b).

t(b, a) v q(b, a) :- r(b), s(a).

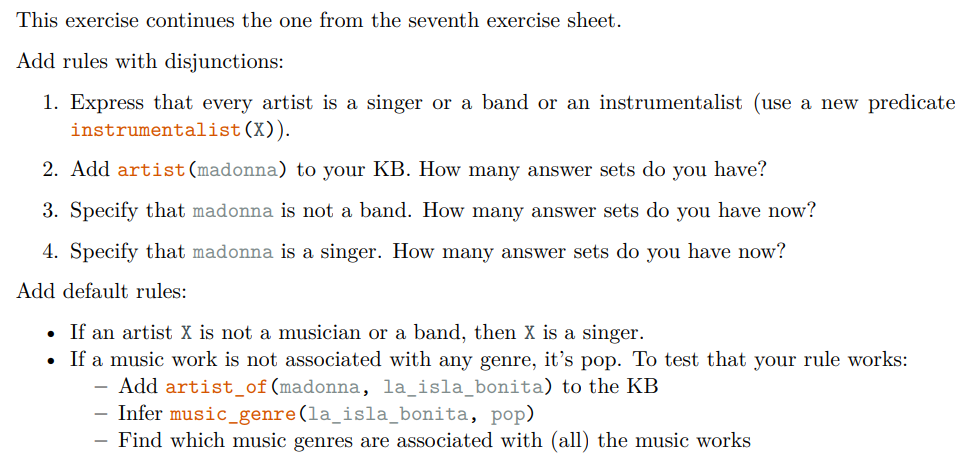
t(b, b) v q(b, b) :- r(b), s(b).

r(a).

s(b).

Is a model

## !!! E.S. 8.2 - Music Knowledge Base



## !!! E.S. 8.3 - Happy Parent and Capitals

