5. Exercise Sheet: SPARQL1.1 & nSPARQL & TriAL July 5, 2018

Exercise 1: Aggregations, Subqueries, Explicit Negation

a) The average of Alice's rating to the action movies is 7.5, while Bob's ratings has an average of 6.5.

```
PREFIX foaf:<http://xmlns.com/foaf/0.1/>
PREFIX movies:<http://example.org/movies#>
PREFIX xsd:<http://www.w3.org/2001/XMLSchema#>

SELECT ?user
       ?genre
       (AVG(?rating) as ?mean)
WHERE
{
    ?user movies:hasRated    ?x .
    ?x    movies:hasRating   ?rating ;
          movies:ratedMovie  ?m .
    ?m    movies:hasGenre    ?genre .
}
GROUP BY ?user
         ?genre
```

Listing 1: SPARQL query :: average of ratings in action genre

	action	sci-fi	thriller	drama
Alice	7.5	4.0	8.75	3.0
Bob	6.5	4.0	9.0	-

Table 1: SPARQL query result

b) For this part, Alice does not have a similar taste to Bob.

```
PREFIX foaf:<http://xmlns.com/foaf/0.1/>
PREFIX movies:<http://example.org/movies#>
PREFIX xsd:<http://www.w3.org/2001/XMLSchema#>
SELECT ?user
       (MAX(?diff) as ?max)
WHERE {
    {
    SELECT ?user
           ?genre
           (AVG(?rating) as ?mean)
           (AVG(?bobrating) as ?bobm)
           (ABS(?mean - ?bobm) as ?diff)
    WHERE
```

```
    {
        ?user movies:hasRated   ?x .
        ?x    movies:hasRating  ?rating ;
              movies:ratedMovie ?m .
        ?m    movies:hasGenre   ?genre .
        movies:Bob movies:hasRated   ?y .
        ?y         movies:hasRating  ?bobrating ;
                   movies:ratedMovie ?bobm .
        ?bobm      movies:hasGenre   ?genre .
    }
    GROUP BY ?user
             ?genre
    }
}
GROUP BY ?user
HAVING (MAX(?diff) < 1)
```

Listing 2: Finding people similar to Bob

Exercise 2: nSPARQL

a) $P1 = (?x, (next :: TGV|next :: Seafrance)+, Dover)$
$[\![P1]\!] = \{\mu | dom(\mu) = \{?x\}\, and (\mu(?x), Dover) \in [\![(next :: TGV|next :: Seafrance)+]\!]\}$
–
$[\![(next :: TGV|next :: Seafrance)]\!] = [\![(next :: TGV]\!] \cup [\![(next :: Seafrance]\!]$
$= (\{(x, y)|(x, TGV, y) \in G\} \cup \{(x, y)|(x, Seafrance, y) \in G\})$
$= (\{(Paris, Calais), (Paris, Dijon)\} \cup \{(Calais, Dover)\})$
$= (\{(Paris, Calais), (Paris, Dijon), (Calais, Dover)\})$
–
$[\![P1]\!] = \{\mu | dom(\mu) = \{?x\}\, and (\mu(?x), Dover) \in [\![(next :: TGV|next :: Seafrance)+]\!]\}$
$[\![P1]\!] = \{\{?x \Rightarrow Calais\}, \{?x \Rightarrow Paris\}\}$

b) $P2 = (?x, (next :: TGV|next :: Seafrance)+, Dover) OPT (?x, next :: country, ?y)$
$[\![P2]\!] = [\![P1]\!] OPT [\![T(?x, next :: country, ?y)]\!]$
$[\![P2]\!] = [\![P1]\!] LeftOuterJoin [\![T(?x, next :: country, ?y)]\!]$
–
$[\![(?x, next :: country, ?y)]\!] = \{\mu | dom(\mu) = \{?x, ?y\}\, and (\mu(?x), \mu(?y)) \in [\![(next :: country)]\!]\}$
$[\![(next :: country)]\!] = (\{(x, y)|(x, country, y) \in G\}$
$[\![(next :: country)]\!] = \{(Paris, France)\}$
$[\![(?x, next :: country, ?y)]\!] = \{\mu | dom(\mu) = \{?x, ?y\}\, and (\mu(?x), \mu(?y)) \in \{(Paris, France)\}\}$
$[\![(?x, next :: country, ?y)]\!] = \{\{?x \Rightarrow Paris\}, \{?y \Rightarrow France\}\}$
–
$[\![P2]\!] = [\![P1]\!] LeftOuterJoin [\![T(?x, next :: country, ?y)]\!]$
$[\![P2]\!] = \{\{?x \Rightarrow Calais\}, \{?x \Rightarrow Paris\}\} LeftOuterJoin \{\{?x \Rightarrow Paris\}, \{?y \Rightarrow France\}\}$
$[\![P2]\!] = \{\{?x \Rightarrow Calais\}, \{\{?x \Rightarrow Paris\}, \{?y \Rightarrow France\}\}\}$

c) $P3 = (?x, (next :: Seafrance|next :: NExpress)+/self :: [next :: NExpress = self :: London]/(next :: Seafrance|next :: NExpress)+, ?y)$
–
$(next :: Seafrance|next :: NExpress) + wouldreturn :$
$\{(Calais, Dover), (Dover, Hastings), (Dover, London)\}$
–
After applying $self :: [next :: NExpress = self :: London]$ the result would be:
Dover because London can only be accesed through NExpress from Dover
–
After applying $(next :: Seafrance|next :: NExpress)+$ would return:
$\{(Dover, Hastings), (Dover, London)\}$
$[\![P3]\!] = \{\{\{?x \Rightarrow Dover\}, \{?y \Rightarrow Hastings\}\}, \{\{?x \Rightarrow Dover\}, \{?y \Rightarrow London\}\}\}$

d) $P4 = (?x, (next :: [(next :: sp)/self :: transport])+, ?y)$
–
$next :: [(next :: sp)/self :: transport]$ would return subjects that have predicates that are subclasses of transport. The results would be:
$(Paris, Calais), (Paris, Dijon), (Calais, Dover), (Dover, Hastings), (Dover, London)$
Since we apply the expression above more than once we will end up with the following results:
$(Paris, Calais), (Paris, Dijon), (Paris, Dover), (Paris, Hastings), (Paris, London), (Calais, Dover), (Calais, Hastings), (Calais, London), (Dover, Hastings), (Dover, London)$
Thus the result for P4 is the following: $[\![P4]\!] = \{\{\{?x \Rightarrow Paris\}, \{?y \Rightarrow Calais\}\}, \{\{?x \Rightarrow Paris\}, \{?y \Rightarrow Dijon\}\} ..$
The etc reffers to the order refered above, it will basically have a mapping for every possible route using any possible transportation.

e) $P5 = (?x, ((trans(train)|trans(ferry)) + /self :: [trans(type) = self :: costal_city]), ?y)$

f) trans can be applied here in order to check if the object is a $costal\_city$.

g) trans can be applied in the same way as in f. However, the object here will either be a city or a $costal\_city$.

Exercise 3: TriAL

a) The result of the right Kleene closure is:

St. Andrew	Bus Op 1	London
Edinburgh	Train Op 1	Brussels
Train Op 1	part_of	NatExpress
St. Andrew	Bus Op 1	Brussels

Table 2: Kleene right closure

b) The result of the left Kleene closure is:

Mohammad-Ali A'RÂBI and Youssef EL-HASSANI
Freiburg im Breisgau, 05.07.2018

St. Andrew	part_of	NatExpress
Edinburgh	part_of	EastCoast
London	part_of	EuroStar

Table 3: Kleene left closure