2. Unit: SPARQL Formal Semantics

Exercise 2.1 (SPARQL Formal Semantics) Consider the SPARQL Formal Semantics.

a) Define a "null-tolerant join" for the relational algebra that acts like the $\bowtie$ of the SPARQL algebra.

b) Which SQL construct is similar to the "\" operator in the SPARQL algebra?

c) In the SPARQL algebra, OPT is expressed via left outer join, which is defined via "\" (while a corresponding MINUS does not exist in the SPARQL syntax).
Such a MINUS (cf. part (b) of this exercise) provides a more intuitive idea of negation than "! bound(x)". Give a general pattern how to express (P_1 MINUS P_2) in SPARQL 1.0 syntax.

d) Recall the definition of $\sqsupset\!\bowtie$ in the relational algebra (DB lecture) and define SPARQL's $\sqsupset\!\bowtie$ in a similar way.

(Parts of the solution are taken from [PAG06]: Jorge Pérez, Marcelo Arenas, Claudio Gutierrez: Semantics and Complexity of SPARQL. International Semantic Web Conference 2006: 30-43, and from [AG08]: Renzo Angles and Claudio Gutierrez: The Expressive Power of SPARQL. International Semantic Web Conference 2008: 114-129; use `http://www.dblp.org`)

a) Consider $R(A, B, C)$ and $S(A, B, D)$ where A is non-null, and B can contain nulls. Then, the null-tolerant join $\bowtie_{null}$ can be defined by the following steps:

1) cartesian product of both relations, immediately evaluating the condition

$$R.A = S.A \wedge (R.B = S.B \vee (R.B \text{ is null}) \vee (S.B \text{ is null}) \ .$$

The result has the format $[R.A, S.A, R.B, S.B, C, D]$.

   – $R.A$ has always the same (non-null) value as $S.A$.
   – $R.B$ and $S.B$ can contain the same non-null-value, but also any of them can contain a null value, while the other is also null, or contains a non-null value.

2) apply a projection that removes $S.A$.

3) For handling B, a new basic operator has to be defined (similar to SQL's binary "coalesce" function: if the first argument is null, take the second one):

$$\text{coalesce} : ANY \times ANY, \qquad \begin{array}{ll} (v_1, v_2) \mapsto v_1 & \text{if } v_1 \text{ is not null,} \\ (null, v) \mapsto v & \end{array}$$

(note that coalesce$(R.B, S.B)$ = coalesce$(S.B, R.B)$ after evaluating the condition in Step (1)).

The algebra expression is then

$$\pi[R.A, B \leftarrow \text{coalesce}(R.B, S.B), C, D]$$
$$|$$
$$\sigma[R.A = S.A \wedge (R.B = S.B \vee (R.B \text{ is null}) \vee (S.B \text{ is null}))]$$
$$|$$
$$\times$$
$$/ \quad \backslash$$
$$R \qquad S$$

b) SQL's "`WHERE NOT EXISTS ...`" is similar.
Consider $R \setminus S$ with R and S as above.

```
SELECT * FROM R WHERE NOT EXISTS (
      SELECT * FROM S
      WHERE R.A = S.A AND (R.B = S.B OR R.B is null OR S.B is null)).
```

c) (taken from [AG08], Section 3)
The basic idea is to replace (P_1 MINUS P_2) by

$$((P_1 \text{ OPT } P_2) \text{ FILTER } (!(\text{bound}(?Y))))$$

where Y is a variable that occurs in P_2, but not in P_1.
Two more aspects have to be considered:

- If P_2 is of the form (P_2' OPT P_2''), then Y must be a variable from P_2' – i.e., a *non-optional variable* (otherwise there are solutions to P_2 that do not bind it).
- If there is no such variable (i.e. all non-optional variables of P_2 occur also in P_1), one must introduce one: take any non-optional triple pattern T that
  i) contains at least one new variable X' and
  ii) is sure to be satisfied whenever (P_1 and) P_2 is satisfied (i.e., it can be a renamed copy (?X q ?X') of some triple pattern (?X q ?Z) from P_1, or any arbitrary pattern that is known to be satisfied in the application)

  and use $((P_1 \text{ OPT } (P_2 \text{ AND } T)) \text{ FILTER } (!(\text{bound}(X'))))$.

Example: Names of countries with their cities, where the city is not the capital:

```
# cities-not-capital.sparql
prefix mon: <http://www.semwebtech.org/mondial/10/meta#>
prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
select ?X ?C
from <file:mondial.n3>
where { ?X a mon:Country; mon:hasCity ?C
        OPTIONAL {?X mon:capital ?C . ?X mon:capital ?C2}
        FILTER (!(bound(?C2))) }
```

Variables X and C occur in P_1 and in P_2, so a (useless) triple pattern is added to bind C_2.

d) $\Omega_1 ⟕ \Omega_2 = (\Omega_1 \bowtie \Omega_2) \cup (\Omega_1 \setminus_s (\Omega_1 \ltimes \Omega_2))$ where the semijoin is defined as usual as $\Omega_1 \ltimes \Omega_2 = \pi[\text{var}(\Omega_1)](\Omega_1 \bowtie \Omega_2)$, and $\setminus_s$ denotes the classical set difference from the relational algebra. Note that it is not necessary to extend the second part of the union with null values (which must be done in the relational algebra to have the same format on both sides of the union).

Exercise 2.2 (Outer Join) Recall that SPARQL's OPTIONAL corresponds to a left outer join.

a) Give a general pattern how to express a *full* outer join (i.e., "outer" to both sides) in the SPARQL algebra (consider as input two mappings R and S and give an expression for $R ⟗ S$) and in SPARQL.

b) Give all cities (name as ?XN) that are the capital of a country (:capital) or that are located at a river (:locatedAt) or both (return the names ?CN of the country and/or the river (?RN)).

a) Replace the full outer join by a two left outer joins: $(R ⟕ S) \cup (S ⟕ R)$. Note that the intersection of both subterms is the inner join. With set semantics, these duplicates are automatically removed. Otherwise apply a DISTINCT.
Alternatively, the inner join can be removed from the second term:

$$(R ⟕ S) \cup ((S ⟕ R) \setminus_s (S \bowtie R))$$
$$\text{or} \quad (R ⟕ S) \cup ((S ⟕ R) \setminus (S \ltimes R))$$

(recall that $\setminus$ denotes the not-exists-like operator from the SPARQL algebra, and $\setminus_s$ denotes the classical set difference).
For SPARQL, the query is of the form

```
    DISTINCT ... WHERE { { P_R(X) OPTIONAL P_S(Y) }
                         UNION
                         { P_S(Y) OPTIONAL P_R(X) } }
  or
    ... WHERE { { P_R(X) OPTIONAL P_S(Y) }
                UNION
                { { P_S(Y) OPTIONAL P_R(X) } FILTER (!(bound(X))) } }
```

b) There is an intuitive solution that replaces the outer join by two optionals: take a city, and if it is a capital, list the country, and if it is located at a river, list the river:

```
# capitals-at-rivers-1.sparql
prefix mon: <http://www.semwebtech.org/mondial/10/meta#>
prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
select ?XN ?CN ?RN
from <file:mondial.n3>
where { ?X a mon:City ; mon:name ?XN.
        OPTIONAL { ?C a mon:Country; mon:name ?CN; mon:capital ?X }
        OPTIONAL { ?X mon:locatedAt ?R . ?R a mon:River; mon:name ?RN }
        ## FILTER (bound(?C) || bound(?R)) }
```

The query yields one line for *each* city, including those that are neither capitals, nor located at a river. These can be removed by adding
`FILTER (bound(?C) || bound(?R))`.

The second solution applies the solution of (a):

```
# capitals-at-rivers-2.sparql
prefix mon: <http://www.semwebtech.org/mondial/10/meta#>
prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
select ?XN ?CN ?RN
from <file:mondial.n3>
where { { ?X a mon:City ; mon:name ?XN .
          ?C a mon:Country; mon:name ?CN; mon:capital ?X
          OPTIONAL { ?X mon:locatedAt ?R . ?R a mon:River; mon:name ?RN } }
        UNION
        { ?X a mon:City ; mon:name ?XN .
          ?X mon:locatedAt ?R . ?R a mon:River; mon:name ?RN
          OPTIONAL { ?C a mon:Country; mon:name ?CN; mon:capital ?X }
          FILTER (!bound(?C)) }
      }
```

Exercise 2.3 (SPARQL Formal Semantics: OPTIONAL) Consider the SPARQL Formal Semantics.

Prove or show a counterexample:

The statement (from W3C SPARQL Working Draft 20061004)

If OPT(A, B) is an optional graph pattern, where A and B are graph patterns, then S is a solution of OPT(A,B) if

- S is a pattern solution of A and of B, or
- S is a solution to A, but not to A and B.

describes the same semantics as above.

The given characterization is the one from the W3C SPARQL Recommendation from 20061004. The counterexample is taken from [PAG06], Examples 1 and 3:

Consider the RDF database D:

$D = \{$	$(B_1$	name	paul),	$(B_1$	phone	777-3426),	
	$(B_2$	name	john),	$(B_2$	email	john@acd.edu),	
	$(B_3$	name	george),	$(B_3$	webPage	www.george.edu),	
	$(B_4$	name	ringo),	$(B_4$	email	ringo@acd.edu),	
	$(B_4$	webPage	www.starr.edu),	$(B_4$	phone	888-4537)	$\}$

Query pattern:

$$P = ((?X, \text{name, paul})\ \text{OPT}\ ((?Y, \text{name, george})\ \text{OPT}\ (?X, \text{email}, ?Z))) =: (P_1\ \text{OPT}\ (P_2\ \text{OPT}\ P_3))\,.$$

$[\![P_1]\!] = \{\{X/B_1\}\}$.
$[\![P_2]\!] = \{\{Y/B_3\}\}$.
$[\![P_3]\!] = \{\{X/B_2, Z/\text{john@}\},\ \{X/B_4, Z/\text{ringo@}\}\}$.
$[\![P_2\ \text{OPT}\ P_3]\!] = [\![P_2 \sqsupset\!\!\bowtie P_3]\!] = \{\{X/B_2, Y/B_3, Z/\text{john@}\},\ \{X/B_4, Y/B_3, Z/\text{ringo@}\}\}$.
$[\![P]\!] = [\![P_1]\!] \sqsupset\!\!\bowtie [\![P_2 \sqsupset\!\!\bowtie P_3]\!] = \{\{X/B_1\}\}$.

On the other hand according to the textual W3C characterization, $S := \{\{X/B_1, Y/B_3\}\}$ is a solution to P: $S := \{\{X/B_1, Y/B_3\}\}$ is a solution to P_1 and to P_2 OPT P_3; the latter holds since it is a solution to P_2, although not to P_3.

The counterexample exploits the fact that it is not *well-designed* (i.e., X occurs inside the inner optional, and in the outermost pattern, but not directly outside the inner optional).

Note that the "declarative", but non-algebraic W3C characterization is also problematic from the operational aspect since the solution must first be guessed before being tested. An algebraic (and thus compositional) semantics allows a bottom-up computation from inside-out.