

2. Unit: SPARQL Formal Semantics

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

- Define a “null-tolerant join” for the relational algebra that acts like the \bowtie of the SPARQL algebra.
- Which SQL construct is similar to the “\” operator in the SPARQL algebra?
- 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.
- Recall the definition of \bowtie in the relational algebra (DB lecture) and define SPARQL’s \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>)

- 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:
 - cartesian product of both relations, immediately evaluating the condition

$$r_1.a = r_2.a \vee (r_1.a \text{ is null }) \vee r_2.a(\text{ is null }) .$$

The result has the format $[R_1.A, R_2.A, R_1.B, R_2.B, C, D]$.

- $R_1.A$ has always the same (non-null) value as $R_2.A$.
 - $R_1.B$ and $R_2.B$ 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.
- apply a projection that removes $R_2.A$.
 - 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):

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

(note that $\text{coalesce}(R_1.B, R_2.B) = \text{coalesce}(R_2.B, R_1.B)$ after evaluating the condition in Step (1)).

- SQL’s “WHERE NOT EXISTS ...” is similar.
Consider R_1 and R_2 as above.
SELECT * FROM R_1 WHERE NOT EXISTS (
 SELECT * FROM R_2
 WHERE $R_1.A = R_2.A$ AND ($R_1.B = R_2.B$ OR $R_1.B$ is null OR $R_2.B$ is null)).
- (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 ($P_2' \text{ OPT } P_2''$), 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 $(?X \text{ q } ?X')$ of some triple pattern $(?X \text{ q } ?Z)$ from P_1 , or any arbitrary pattern that is known to be satisfied in the application)
 and use $((P_1 \text{ OPT } (P_2 \text{ AND } T)) \text{ FILTER } (!\text{bound}(X')))$.
 In practice, instead of $(X' \text{ p } t)$, any pattern can be used where it is sure that it is also bound for each match of P_2 .

Example: all cities in a country which are not its 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)) }
```

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

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

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 $R \bowtie \sqcup S$) 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: $(R \bowtie \sqcup S) \cup (S \bowtie \sqcup R)$. 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:

$$\begin{aligned} & (R \bowtie \sqcup S) \cup ((S \bowtie \sqcup R) \setminus_s (S \bowtie R)) \\ \text{or} & (R \bowtie \sqcup S) \cup ((S \bowtie \sqcup R) \setminus (S \bowtie R)) \end{aligned}$$

(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(?RN)).
```

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 $\text{OPT}(A, B)$ is an optional graph pattern, where A and B are graph patterns, then S is a solution of $\text{OPT}(A, B)$ if

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

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

Query pattern:

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

$$\llbracket P_1 \rrbracket = \{\{X/B_1\}\}.$$

$$\llbracket P_2 \rrbracket = \{\{Y/B_3\}\}.$$

$$\llbracket P_3 \rrbracket = \{\{X/B_2, Z/\text{john@}\}, \{X/B_4, Z/\text{ringo@}\}\}.$$

$$\llbracket P_2 \text{OPT } P_3 \rrbracket = \llbracket P_2 \bowtie P_3 \rrbracket = \{\{X/B_2, Y/B_3, Z/\text{john@}\}, \{X/B_4, Y/B_3, Z/\text{ringo@}\}\}.$$

$$\llbracket P \rrbracket = \llbracket P_1 \rrbracket \bowtie \llbracket P_2 \bowtie P_3 \rrbracket = \{\{X/B_1\}\}.$$

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 $P_2 \text{OPT } P_3$; 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.

Exercise 2.4 (SPARQL: Filter-Safe Expressions) Consider the following definition:

Definition 1 ([PAG06, AG 08]) A SPARQL expression is *filter-safe*, if for every subexpression of the form $(P \text{ FILTER } R)$, $\text{var}(R) \subseteq \text{var}(P)$.

Filters of the forbidden form are rather commonly used, e.g.,

```
{ ?P1 a :Person;   :age ?A1. ?P2 a :Person
  OPTIONAL { ?P2 :age ?A2 . FILTER ( ?A2 > ?A1 ) } }
```

- Sketch an algorithm that rewrites non-filter safe queries into safe ones. First, try it on your own, then maybe look in [AG08].
- Give a SPARQL query for “Give the names of all countries, such that there is some city in that country where more than 1/4 of the population are living in”, and make it filter-safe. Give also its SPARQL algebra expression.
- Give a SPARQL query for “Give the names of all countries, such that there is *no* city in that country where more than 1/4 of the population are living in”, and make it filter-safe. Give also its SPARQL algebra expression.
- Is there a similar thing in SQL, in the relational algebra, and in the relational calculus?

- An intuitive solution is to duplicate (relevant parts of) the pattern outside the OPTIONAL for binding of the variable to inside:

```
{ ?P1 a :Person;   :age ?A1. ?P2 a :Person
  OPTIONAL { ?P1 a :Person; :age ?A1 . ?P2 a :Person . ?P2 :age ?A2 . FILTER ( ?A2 > ?A1 ) } }
```

The variables occurring in the inner pattern and in the outer pattern act as join variables when computing the outer join.

The complete algorithm can be found in [AG08]. There, the fourth if-case (lines 7-10) covers the above case. The fifth if-case (lines 12-24) just throws an error in case a filter is still unsafe. Note (DB Theory lecture): consider the similarity with the transformation of general formulas into RANF when moving conjuncts into negated subformulas to make them *self-contained*.

b) The solution is of the same form as above:

```
# nobigcities2.sparql
prefix mon: <http://www.semwebtech.org/mondial/10/meta#>
prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT distinct ?N
FROM <file:mondial.n3>
WHERE {?X rdf:type mon:Country; mon:name ?N; mon:population ?XP .
       OPTIONAL { ?X mon:hasCity ?C . ?C mon:population ?CP . FILTER (?CP > 0.25 * ?XP) } .
       FILTER (!BOUND(?C)) }
```

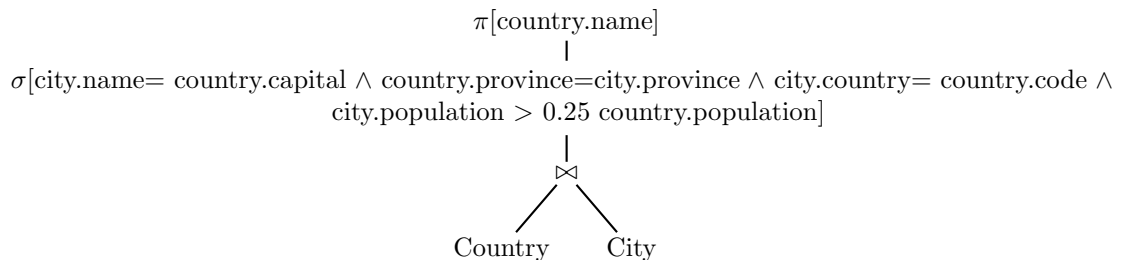
and its filter-safe variant is

```
# nobigcities3.sparql
prefix mon: <http://www.semwebtech.org/mondial/10/meta#>
prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT distinct ?N
FROM <file:mondial.n3>
WHERE {?X rdf:type mon:Country; mon:name ?N; mon:population ?XP .
       OPTIONAL { ?X mon:hasCity ?C; mon:population ?XP .
                  ?C mon:population ?CP . FILTER (?CP > 0.25 * ?XP) } .
       FILTER (!BOUND(?C)) }
```

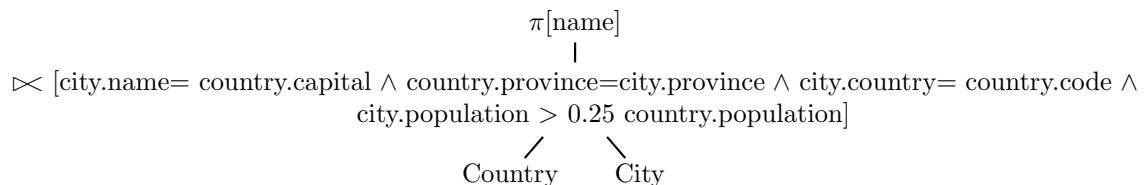
c) Yes: correlated subqueries. For instance: “all countries where more than 25% of the population live in the capital”:

```
SELECT name FROM country
WHERE population < 4 *
  (SELECT population FROM city
   WHERE city.name= country.capital AND country.province=city.province
   AND city.country= country.code)
```

In the relational algebra, subqueries (=subtrees) are not allowed inside a selection condition. Here, the corresponding algebra expression combines the outer query and the subquery by using a broad join:



The internal evaluation is done with iterators as a semijoin: iterate over all countries, look up the capital and its population, and output if more than 1/4:



The query from (b) is translated into an NOT EXISTS in SQL:

```
SELECT name
FROM country
WHERE NOT EXISTS
  (SELECT *
   FROM city
   WHERE city.country= country.code
        AND city.population > 0.25 * country.population)
```

Internal evaluation is again iterator-based using an *antijoin* (the counterpart to a semijoin - tuples survive if they do not match).

In the relational calculus ...
