On Expressibility of Non-Monotone Operators in SPARQL

Roman Kontchakov

Department of Computer Science and Inf. Systems, Birkbeck College, London

`http://www.dcs.bbk.ac.uk/~roman`

joint work with **Egor V. Kostylev** (*University of Oxford*)

Basic SPARQL

SPARQL query

```
SELECT ?d WHERE {
  ?d a :Department
}
```

a = rdf:type

← Basic Graph Pattern (BGP)
(a set of triple patterns)

Basic SPARQL

SPARQL query

```
SELECT ?d WHERE {
  ?d a :Department
}
```

a = rdf:type

Basic Graph Pattern (BGP)
(a set of triple patterns)

data instance

(an RDF graph
= a set of triples)

:CS	a	:Department
:Maths	a	:Department
:Adams	a	:Prof
:Brown	a	:Prof
:Clarke	a	:Prof
:Clarke	:worksIn	:Maths
:Brown	:worksIn	:CS
:Davies	:worksIn	:CS

T is the set of **terms**, i.e.,
IRIs and literals (integers, strings, etc.)

Basic SPARQL

SPARQL query

```
SELECT ?d WHERE {
  ?d a :Department
}
```

a = rdf:type

Basic Graph Pattern (BGP)
(a set of triple patterns)

data instance

(an RDF graph
= a set of triples)

T is the set of **terms**, i.e.,
IRIs and literals (integers, strings, etc.)

:CS	a	:Department
:Maths	a	:Department
:Adams	a	:Prof
:Brown	a	:Prof
:Clarke	a	:Prof
:Clarke	:worksIn	:Maths
:Brown	:worksIn	:CS
:Davies	:worksIn	:CS

answer is a set of solution mappings

?d
:CS
:Maths

solution mapping μ is a *partial* map from $\overbrace{\mathbf{V}}^{\text{set of variables}}$ to **T**

dom(μ) is the **domain** of μ

$$[\![P]\!]_G = \{\, \mu\colon \mathrm{var}(P) \to \mathbf{T} \mid \mu(P) \subseteq G \,\} \quad \text{for a BGP } P$$

Basic SPARQL

SPARQL query

```
SELECT ?d WHERE {
  ?d a :Department
}
```

a = rdf:type

Basic Graph Pattern (BGP)
(a set of triple patterns)

data instance

(an RDF graph
= a set of triples)

:CS	a	:Department
:Maths	a	:Department
:Adams	a	:Prof
:Brown	a	:Prof
:Clarke	a	:Prof
:Clarke	:worksIn	:Maths
:Brown	:worksIn	:CS
:Davies	:worksIn	:CS

T is the set of **terms**, i.e.,
IRIs and literals (integers, strings, etc.)

answer is a set of solution mappings

?d
:CS
:Maths

solution mapping μ is a *partial* map from $\overbrace{\mathbf{V}}^{\text{set of variables}}$ to **T**

dom(μ) is the **domain** of μ

$$[\![P]\!]_G = \{\, \mu \colon \mathrm{var}(P) \to \mathbf{T} \mid \mu(P) \subseteq G \,\} \quad \text{for a BGP } P$$

NB: we consider set semantics (SPARQL uses bag semantics, but our negative results hold)

Monotone SPARQL: FILTER

```
SELECT ?p1 ?p2 ?d WHERE {
  ?p1 :worksIn ?d .
  ?p2 :worksIn ?d
  FILTER (?p1 != ?p2)
}
```

:CS	a	:Department
:Maths	a	:Department
:Adams	a	:Prof
:Brown	a	:Prof
:Clarke	a	:Prof
:Clarke	:worksIn	:Maths
:Brown	:worksIn	:CS
:Davies	:worksIn	:CS

Monotone SPARQL: FILTER

```
SELECT ?p1 ?p2 ?d WHERE {
  ?p1 :worksIn ?d .
  ?p2 :worksIn ?d
  FILTER (?p1 != ?p2)
}
```

:CS	a	:Department
:Maths	a	:Department
:Adams	a	:Prof
:Brown	a	:Prof
:Clarke	a	:Prof
:Clarke	:worksIn	:Maths
:Brown	:worksIn	:CS
:Davies	:worksIn	:CS

answer

	?p1	?p2	?d
μ_1	:Davies	:Brown	:CS
μ_2	:Brown	:Davies	:CS

$$[\![\text{FILTER}_F\ P]\!]_G = \{\ \mu \in [\![P]\!]_G \mid F^\mu = \text{true}\ \}$$

filters F are **Boolean combinations** of $?v_1 = ?v_2,\ \ ?v = d,$ etc.

Monotone SPARQL: FILTER

```
SELECT ?p1 ?p2 ?d WHERE {
   ?p1 :worksIn ?d .
   ?p2 :worksIn ?d
   FILTER (?p1 != ?p2)
}
```

:CS	a	:Department
:Maths	a	:Department
:Adams	a	:Prof
:Brown	a	:Prof
:Clarke	a	:Prof
:Clarke	:worksIn	:Maths
:Brown	:worksIn	:CS
:Davies	:worksIn	:CS

answer

	?p1	?p2	?d
μ_1	:Davies	:Brown	:CS
μ_2	:Brown	:Davies	:CS

$$[\![\text{FILTER}_F\ P]\!]_G = \{\ \mu \in [\![P]\!]_G \mid F^\mu = \text{true}\ \}$$

filters F are **Boolean combinations** of $?v_1 = ?v_2,\ \ ?v = d,$ etc.

NB: slight simplification, see Effective Boolean Value in SPARQL Specification

Monotone SPARQL: FILTER

```
SELECT ?p1 ?p2 ?d WHERE {
  ?p1 :worksIn ?d .
  ?p2 :worksIn ?d
  FILTER (?p1 != ?p2)
}
```

:CS	a	:Department
:Maths	a	:Department
:Adams	a	:Prof
:Brown	a	:Prof
:Clarke	a	:Prof
:Clarke	:worksIn	:Maths
:Brown	:worksIn	:CS
:Davies	:worksIn	:CS

answer

	?p1	?p2	?d
μ_1	:Davies	:Brown	:CS
μ_2	:Brown	:Davies	:CS

$$[\![\mathsf{FILTER}_F\ P]\!]_G = \{\ \mu \in [\![P]\!]_G \mid F^\mu = \mathsf{true}\ \}$$

filters F are **Boolean combinations** of $?v_1 = ?v_2,\quad ?v = d,$ etc.

NB: slight simplification, see Effective Boolean Value in SPARQL Specification

NB: SPARQL uses 3-valued logic (like SQL)

Monotone SPARQL: UNION

```
SELECT ?p ?d WHERE {
  { ?p a :Prof .
    ?p :worksIn ?d }
  UNION
  { ?p a :Prof }
}
```

:CS	a	:Department
:Maths	a	:Department
:Adams	a	:Prof
:Brown	a	:Prof
:Clarke	a	:Prof
:Clarke	:worksIn	:Maths
:Brown	:worksIn	:CS
:Davies	:worksIn	:CS

Monotone SPARQL: UNION

```
SELECT ?p ?d WHERE {
  { ?p a :Prof .
    ?p :worksIn ?d }
  UNION
  { ?p a :Prof }
}
```

answer

	?p	?d
μ_1	:Clarke	:Maths
μ_2	:Brown	:CS
μ_3	:Davies	:CS
μ_4	:Adams	
μ_5	:Brown	
μ_6	:Clarke	

:CS	a	:Department
:Maths	a	:Department
:Adams	a	:Prof
:Brown	a	:Prof
:Clarke	a	:Prof
:Clarke	:worksIn	:Maths
:Brown	:worksIn	:CS
:Davies	:worksIn	:CS

$$[\![P_1 \text{ UNION } P_2]\!]_G = [\![P_1]\!]_G \cup [\![P_2]\!]_G$$

Monotone SPARQL: UNION

```
SELECT ?p ?d WHERE {
  { ?p a :Prof .
    ?p :worksIn ?d }
  UNION
  { ?p a :Prof }
}
```

answer

	?p	?d
μ_1	:Clarke	:Maths
μ_2	:Brown	:CS
μ_3	:Davies	:CS
μ_4	:Adams	
μ_5	:Brown	
μ_6	:Clarke	

:CS	a	:Department
:Maths	a	:Department
:Adams	a	:Prof
:Brown	a	:Prof
:Clarke	a	:Prof
:Clarke	:worksIn	:Maths
:Brown	:worksIn	:CS
:Davies	:worksIn	:CS

$$[\![P_1 \text{ UNION } P_2]\!]_G = [\![P_1]\!]_G \cup [\![P_2]\!]_G$$

NB: unlike in SQL, the two arguments do not have to have the same 'schema'

Monotone SPARQL: UNION

```
SELECT ?p ?d WHERE {
  { ?p a :Prof .
    ?p :worksIn ?d }
  UNION
  { ?p a :Prof }
}
```

answer

	?p	?d
μ_1	:Clarke	:Maths
μ_2	:Brown	:CS
μ_3	:Davies	:CS
μ_4	:Adams	
μ_5	:Brown	
μ_6	:Clarke	

:CS	a	:Department
:Maths	a	:Department
:Adams	a	:Prof
:Brown	a	:Prof
:Clarke	a	:Prof
:Clarke	:worksIn	:Maths
:Brown	:worksIn	:CS
:Davies	:worksIn	:CS

$$[\![P_1 \text{ UNION } P_2]\!]_G = [\![P_1]\!]_G \cup [\![P_2]\!]_G$$

NB: unlike in SQL, the two arguments do not have to have the same 'schema'

- the 'missing' values are like **NULL** in SQL with the 3-valued logic

  `(?d = :CS)`$^{\mu_4}$ is $\varepsilon \to$ false and `(?d != :CS)`$^{\mu_4}$ is $\varepsilon \to$ false

Monotone SPARQL: UNION

```
SELECT ?p ?d WHERE {
  { ?p a :Prof .
    ?p :worksIn ?d }
  UNION
  { ?p a :Prof }
}
```

answer

	?p	?d
μ_1	:Clarke	:Maths
μ_2	:Brown	:CS
μ_3	:Davies	:CS
μ_4	:Adams	
μ_5	:Brown	
μ_6	:Clarke	

:CS	a	:Department
:Maths	a	:Department
:Adams	a	:Prof
:Brown	a	:Prof
:Clarke	a	:Prof
:Clarke	:worksIn	:Maths
:Brown	:worksIn	:CS
:Davies	:worksIn	:CS

$$[\![P_1 \text{ UNION } P_2]\!]_G = [\![P_1]\!]_G \cup [\![P_2]\!]_G$$

NB: unlike in SQL, the two arguments do not have to have the same 'schema'

- the 'missing' values are like **NULL** in SQL with the 3-valued logic

  $(\texttt{?d = :CS})^{\mu_4}$ is $\varepsilon \to$ false and $(\texttt{?d != :CS})^{\mu_4}$ is $\varepsilon \to$ false

  $(\mathrm{bound}(?v))^{\mu}$ is true $\Leftrightarrow$ $?v \in \mathrm{dom}(\mu)$ (similar to **IS NOT NULL** in SQL)

Monotone SPARQL: UNION

```
SELECT ?p ?d WHERE {
  { ?p a :Prof .
    ?p :worksIn ?d }
  UNION
  { ?p a :Prof }
}
```

answer

	?p	?d
μ_1	:Clarke	:Maths
μ_2	:Brown	:CS
μ_3	:Davies	:CS
μ_4	:Adams	
μ_5	:Brown	
μ_6	:Clarke	

:CS	a	:Department
:Maths	a	:Department
:Adams	a	:Prof
:Brown	a	:Prof
:Clarke	a	:Prof
:Clarke	:worksIn	:Maths
:Brown	:worksIn	:CS
:Davies	:worksIn	:CS

$$[\![P_1 \text{ UNION } P_2]\!]_G = [\![P_1]\!]_G \cup [\![P_2]\!]_G$$

NB: unlike in SQL, the two arguments do not have to have the same 'schema'

- the 'missing' values are like **NULL** in SQL with the 3-valued logic

$(\texttt{?d = :CS})^{\mu_4}$ is $\varepsilon \to$ false and $(\texttt{?d != :CS})^{\mu_4}$ is $\varepsilon \to$ false

$(\text{bound}(?v))^{\mu}$ is true $\Leftrightarrow$ $?v \in \text{dom}(\mu)$ (similar to **IS NOT NULL** in SQL)

NB: the 3-valued logic it is not essential — see **Zhang & Van den Bussche (2014)**

Monotone SPARQL: JOIN

μ_1 and μ_2 are **compatible** $\mu_1 \sim \mu_2$ if

$$\mu_1(?v) = \mu_2(?v), \quad \text{for all } ?v \in \text{dom}(\mu_1) \cap \text{dom}(\mu_2)$$

Monotone SPARQL: JOIN

μ_1 and μ_2 are **compatible** $\boldsymbol{\mu_1 \sim \mu_2}$ if

$$\mu_1(?v) = \mu_2(?v), \quad \text{for all } ?v \in \text{dom}(\mu_1) \cap \text{dom}(\mu_2)$$

$$[\![P_1 \text{ JOIN } P_2]\!]_G = \{\, \mu_1 \oplus \mu_2 \mid \mu_1 \in [\![P_1]\!]_G \text{ and } \mu_2 \in [\![P_2]\!]_G \text{ with } \mu_1 \sim \mu_2 \,\}$$

?p	?d	?t
:Adams	:Maths	
:Clarke		8506

JOIN

?p	?d
:Adams	
:Clarke	:Maths
:Clarke	:CS
:Davies	:CS

=

Monotone SPARQL: JOIN

μ_1 and μ_2 are **compatible** $\boldsymbol{\mu_1 \sim \mu_2}$ if

$\mu_1(?v) = \mu_2(?v)$, for all $?v \in \mathrm{dom}(\mu_1) \cap \mathrm{dom}(\mu_2)$

$$[\![P_1 \text{ JOIN } P_2]\!]_G = \{\, \mu_1 \oplus \mu_2 \mid \mu_1 \in [\![P_1]\!]_G \text{ and } \mu_2 \in [\![P_2]\!]_G \text{ with } \mu_1 \sim \mu_2 \,\}$$

?p	?d	?t
:Adams	:Maths	
:Clarke		8506

JOIN

?p	?d
:Adams	
:Clarke	:Maths
:Clarke	:CS
:Davies	:CS

=

?p	?d	?t
:Adams	:Maths	
:Clarke	:Maths	8506
:Clarke	:CS	8506

Monotone SPARQL: JOIN

μ_1 and μ_2 are **compatible** $\mu_1 \sim \mu_2$ if

$$\mu_1(?v) = \mu_2(?v), \quad \text{for all } ?v \in \text{dom}(\mu_1) \cap \text{dom}(\mu_2)$$

$$[\![P_1 \text{ JOIN } P_2]\!]_G = \{\, \mu_1 \oplus \mu_2 \mid \mu_1 \in [\![P_1]\!]_G \text{ and } \mu_2 \in [\![P_2]\!]_G \text{ with } \mu_1 \sim \mu_2 \,\}$$

?p	?d	?t
:Adams	:Maths	
:Clarke		8506

JOIN

?p	?d
:Adams	
:Clarke	:Maths
:Clarke	:CS
:Davies	:CS

=

?p	?d	?t
:Adams	:Maths	
:Clarke	:Maths	8506
:Clarke	:CS	8506

compatibility in SQL is quite different!

?p	?d	?t
:Adams	:Maths	NULL
:Clarke	NULL	8506

JOIN^{DB}

?p	?d
:Adams	NULL
:Clarke	:Maths
:Clarke	:CS
:Davies	:CS

=

Monotone SPARQL: JOIN

μ_1 and μ_2 are **compatible** $\boldsymbol{\mu_1 \sim \mu_2}$ if

$$\mu_1(?v) = \mu_2(?v), \quad \text{for all } ?v \in \mathrm{dom}(\mu_1) \cap \mathrm{dom}(\mu_2)$$

$$[\![P_1 \text{ JOIN } P_2]\!]_G = \{\, \mu_1 \oplus \mu_2 \mid \mu_1 \in [\![P_1]\!]_G \text{ and } \mu_2 \in [\![P_2]\!]_G \text{ with } \mu_1 \sim \mu_2 \,\}$$

?p	?d	?t
:Adams	:Maths	
:Clarke		8506

JOIN

?p	?d
:Adams	
:Clarke	:Maths
:Clarke	:CS
:Davies	:CS

=

?p	?d	?t
:Adams	:Maths	
:Clarke	:Maths	8506
:Clarke	:CS	8506

compatibility in SQL is quite different!

?p	?d	?t
:Adams	:Maths	NULL
:Clarke	NULL	8506

JOIN$^{\text{DB}}$

?p	?d
:Adams	NULL
:Clarke	:Maths
:Clarke	:CS
:Davies	:CS

=

?p	?d	?t

NB: careful use of `COALESCE` (or `IF`) is required, see **Prud'hommeaux & Bertails (2008)**

Monotone SPARQL: Algebraic View

unique $\boldsymbol{\mu_\emptyset}$ with $\mathrm{dom}(\mu_\emptyset) = \emptyset$ is compatible with **any** solution mapping

empty BGP $\{\}$ $[\![\{\}]\!]_G = \{\mu_\emptyset\}$, for any G

Monotone SPARQL: Algebraic View

unique $\mu_\emptyset$ with $\mathrm{dom}(\mu_\emptyset) = \emptyset$ is compatible with **any** solution mapping

empty BGP {} $[\![\{\}]\!]_G = \{\mu_\emptyset\}$, for any G

Pérez et al. (2006), Schmidt et al. (2010), Geerts et al. (2013)

1. Bags of solution mappings form a **commutative semiring** with operations UNION and JOIN ($\emptyset$ is the identity for UNION and $\{\mu_\emptyset\}$ is the identity for JOIN)

Monotone SPARQL: Algebraic View

unique $\mu_\emptyset$ with $\mathrm{dom}(\mu_\emptyset) = \emptyset$ is compatible with **any** solution mapping

empty BGP {} $\quad [\![\{\}]\!]_G = \{\mu_\emptyset\}$, for any G

Pérez et al. (2006), Schmidt et al. (2010), Geerts et al. (2013)

1. Bags of solution mappings form a **commutative semiring** with operations UNION and JOIN ($\emptyset$ is the identity for UNION and $\{\mu_\emptyset\}$ is the identity for JOIN)

S_1 UNION $S_2 = S_2$ UNION S_1 $\quad$ S_1 UNION (S_2 JOIN S_3) = (S_1 UNION S_2) JOIN S_3

S UNION $\emptyset = S$

S_1 JOIN $S_2 = S_2$ JOIN S_1 $\quad$ S_1 JOIN (S_2 JOIN S_3) = (S_1 JOIN S_2) JOIN S_3

S JOIN $\{\mu_\emptyset\} = S$

S JOIN $\emptyset = \emptyset$ $\quad$ S_1 JOIN (S_2 UNION S_3) = (S_1 JOIN S_2) JOIN (S_1 JOIN S_3)

Monotone SPARQL: Algebraic View

unique $\mu_\emptyset$ with $\text{dom}(\mu_\emptyset) = \emptyset$ is compatible with **any** solution mapping

empty BGP {} $\llbracket\{\}\rrbracket_G = \{\mu_\emptyset\}$, for any G

Pérez et al. (2006), Schmidt et al. (2010), Geerts et al. (2013)

1. Bags of solution mappings form a **commutative semiring** with operations UNION and JOIN ($\emptyset$ is the identity for UNION and $\{\mu_\emptyset\}$ is the identity for JOIN)

S_1 UNION $S_2 = S_2$ UNION S_1

S UNION $\emptyset = S$

S_1 UNION (S_2 JOIN S_3) = (S_1 UNION S_2) JOIN S_3

S_1 JOIN $S_2 = S_2$ JOIN S_1

S JOIN $\{\mu_\emptyset\} = S$

S_1 JOIN (S_2 JOIN S_3) = (S_1 JOIN S_2) JOIN S_3

S JOIN $\emptyset = \emptyset$

S_1 JOIN (S_2 UNION S_3) = (S_1 JOIN S_2) JOIN (S_1 JOIN S_3)

under the set semantics: S UNION $S = S$

Monotone SPARQL: Algebraic View

unique $\mu_\emptyset$ with $\text{dom}(\mu_\emptyset) = \emptyset$ is compatible with **any** solution mapping

empty BGP $\{\}$ $\quad \llbracket\{\}\rrbracket_G = \{\mu_\emptyset\}$, for any G

Pérez et al. (2006), Schmidt et al. (2010), Geerts et al. (2013)

1. Bags of solution mappings form a **commutative semiring** with operations UNION and JOIN ($\emptyset$ is the identity for UNION and $\{\mu_\emptyset\}$ is the identity for JOIN)

S_1 UNION $S_2 = S_2$ UNION S_1 $\qquad$ S_1 UNION $(S_2$ JOIN $S_3) = (S_1$ UNION $S_2)$ JOIN S_3

S UNION $\emptyset = S$

S_1 JOIN $S_2 = S_2$ JOIN S_1 $\qquad$ S_1 JOIN $(S_2$ JOIN $S_3) = (S_1$ JOIN $S_2)$ JOIN S_3

S JOIN $\{\mu_\emptyset\} = S$

S JOIN $\emptyset = \emptyset$ $\qquad$ S_1 JOIN $(S_2$ UNION $S_3) = (S_1$ JOIN $S_2)$ JOIN $(S_1$ JOIN $S_3)$

under the set semantics: S UNION $S = S$ $\qquad$ S JOIN $S = S$

Monotone SPARQL: Algebraic View

unique $\mu_\emptyset$ with $\mathrm{dom}(\mu_\emptyset) = \emptyset$ is compatible with **any** solution mapping

empty BGP {} $\quad [\![\{\}]\!]_G = \{\mu_\emptyset\}$, for any G

Pérez et al. (2006), Schmidt et al. (2010), Geerts et al. (2013)

1. Bags of solution mappings form a **commutative semiring** with operations UNION and JOIN ($\emptyset$ is the identity for UNION and $\{\mu_\emptyset\}$ is the identity for JOIN)

S_1 UNION S_2 = S_2 UNION S_1 $\qquad$ S_1 UNION (S_2 JOIN S_3) = (S_1 UNION S_2) JOIN S_3

S UNION $\emptyset$ = S

S_1 JOIN S_2 = S_2 JOIN S_1 $\qquad$ S_1 JOIN (S_2 JOIN S_3) = (S_1 JOIN S_2) JOIN S_3

S JOIN $\{\mu_\emptyset\}$ = S

S JOIN $\emptyset$ = $\emptyset$ $\qquad$ S_1 JOIN (S_2 UNION S_3) = (S_1 JOIN S_2) JOIN (S_1 JOIN S_3)

under the set semantics: S UNION S = S $\qquad$ ~~S JOIN S = S~~ $\qquad$ **only** $\supseteq$

Monotone SPARQL: Algebraic View

unique $\mu_\emptyset$ with $\mathrm{dom}(\mu_\emptyset) = \emptyset$ is compatible with **any** solution mapping

empty BGP {} $\quad [\![\{\}]\!]_G = \{\mu_\emptyset\}$, for any G

Pérez et al. (2006), Schmidt et al. (2010), Geerts et al. (2013)

1. Bags of solution mappings form a **commutative semiring** with operations UNION and JOIN ($\emptyset$ is the identity for UNION and $\{\mu_\emptyset\}$ is the identity for JOIN)

S_1 UNION $S_2 = S_2$ UNION S_1 $\qquad$ S_1 UNION (S_2 JOIN S_3) = (S_1 UNION S_2) JOIN S_3

S UNION $\emptyset = S$

S_1 JOIN $S_2 = S_2$ JOIN S_1 $\qquad$ S_1 JOIN (S_2 JOIN S_3) = (S_1 JOIN S_2) JOIN S_3

S JOIN $\{\mu_\emptyset\} = S$

S JOIN $\emptyset = \emptyset$ $\qquad$ S_1 JOIN (S_2 UNION S_3) = (S_1 JOIN S_2) JOIN (S_1 JOIN S_3)

under the set semantics: S UNION $S = S$ $\qquad$ ~~S JOIN $S = S$~~ $\qquad$ **only** $\supseteq$

2. FILTER distributes over UNION

FILTER$_F$(S_1 UNION S_2) = FILTER$_F$ S_1 UNION FILTER$_F$ S_2

~~FILTER$_F$(S_1 JOIN S_2) = FILTER$_F$ S_1 JOIN FILTER$_F$ S_2~~

Non-monotone SPARQL: OPTIONAL

```
SELECT ?p ?d WHERE {
  ?p a :Prof
  OPTIONAL { ?p :worksIn ?d
         FILTER (?d != :CS) }
}
```

:Adams	a	:Prof
:Brown	a	:Prof
:Clarke	a	:Prof
:Brown	:worksIn	:CS
:Clarke	:worksIn	:Maths

Non-monotone SPARQL: OPTIONAL

```
SELECT ?p ?d WHERE {
  ?p a :Prof
  OPTIONAL { ?p :worksIn ?d
         FILTER (?d != :CS) }
}
```

:Adams	a	:Prof
:Brown	a	:Prof
:Clarke	a	:Prof
:Brown	:worksIn	:CS
:Clarke	:worksIn	:Maths

$$P_1 \ \mathrm{OPT}_F \ P_2 \quad = \quad \underbrace{\mathrm{FILTER}_F(P_1 \ \mathrm{JOIN} \ P_2)}_{\text{`}P_1\text{ that have a compatible }P_2\text{ with }F\text{'}} \quad \mathrm{UNION} \quad \underbrace{P_1 \ \mathrm{DIFF}_F \ P_2}_{\text{`}P_1\text{ that have no compatible }P_2\text{ with }F\text{'}}$$

Non-monotone SPARQL: OPTIONAL

```
SELECT ?p ?d WHERE {
  ?p a :Prof
  OPTIONAL { ?p :worksIn ?d
         FILTER (?d != :CS) }
}
```

:Adams	a	:Prof
:Brown	a	:Prof
:Clarke	a	:Prof
:Brown	:worksIn	:CS
:Clarke	:worksIn	:Maths

$$[\![P_1 \text{ DIFF}_F\, P_2]\!]_G \;=\; \{\, \mu_1 \in [\![P_1]\!]_G \mid \text{ there is no } \mu_2 \in [\![P_2]\!]_G \text{ with } \mu_1 \sim \mu_2 \;\text{ and }\; F^{\mu_1 \oplus \mu_2} = \text{true} \,\}$$

$$P_1 \text{ OPT}_F\, P_2 \;=\; \underbrace{\text{FILTER}_F(P_1 \text{ JOIN } P_2)}_{\text{`}P_1\text{ that have a compatible }P_2\text{ with }F\text{'}} \;\text{ UNION }\; \underbrace{P_1 \text{ DIFF}_F\, P_2}_{\text{`}P_1\text{ that have no compatible }P_2\text{ with }F\text{'}}$$

Non-monotone SPARQL: OPTIONAL

```
SELECT ?p ?d WHERE {
  ?p a :Prof
  OPTIONAL { ?p :worksIn ?d
         FILTER (?d != :CS) }
}
```

:Adams	a	:Prof
:Brown	a	:Prof
:Clarke	a	:Prof
:Brown	:worksIn	:CS
:Clarke	:worksIn	:Maths

$$[\![P_1 \text{ DIFF}_F\ P_2]\!]_G = \{\mu_1 \in [\![P_1]\!]_G \mid \text{there is no } \mu_2 \in [\![P_2]\!]_G \text{ with } \mu_1 \sim \mu_2 \text{ and } F^{\mu_1 \oplus \mu_2} = \text{true}\}$$

$$P_1 \text{ OPT}_F\ P_2 = \underbrace{\text{FILTER}_F(P_1 \text{ JOIN } P_2)}_{\text{`}P_1\text{ that have a compatible }P_2\text{ with }F\text{'}} \text{ UNION } \underbrace{P_1 \text{ DIFF}_F\ P_2}_{\text{`}P_1\text{ that have no compatible }P_2\text{ with }F\text{'}}$$

answer

	?p	?d
μ_1	:Adams	
μ_2	:Clarke	:Maths
μ_3	:Brown	

Non-monotone SPARQL: OPTIONAL

```
SELECT ?p ?d WHERE {
  ?p a :Prof
  OPTIONAL { ?p :worksIn ?d
             FILTER (?d != :CS) }
}
```

:Adams	a	:Prof
:Brown	a	:Prof
:Clarke	a	:Prof
:Brown	:worksIn	:CS
:Clarke	:worksIn	:Maths

$$[\![P_1 \text{ DIFF}_F\ P_2]\!]_G = \{\, \mu_1 \in [\![P_1]\!]_G \mid \text{there is no } \mu_2 \in [\![P_2]\!]_G \text{ with } \mu_1 \sim \mu_2 \text{ and } F^{\mu_1 \oplus \mu_2} = \text{true} \,\}$$

$$P_1 \text{ OPT}_F\ P_2 = \underbrace{\text{FILTER}_F(P_1 \text{ JOIN } P_2)}_{\text{`}P_1\text{ that have a compatible }P_2\text{ with }F\text{'}} \text{ UNION } \underbrace{P_1 \text{ DIFF}_F\ P_2}_{\text{`}P_1\text{ that have no compatible }P_2\text{ with }F\text{'}}$$

answer

	?p	?d
μ_1	:Adams	
μ_2	:Clarke	:Maths
μ_3	:Brown	

NB: SPARQL 1.1 specification **incorrectly** says `Written in full that is:

$$[\![P_1 \text{ OPT}_F\ P_2]\!]_G = \{\, \mu_1 \oplus \mu_2 \mid \mu_1 \in [\![P_1]\!]_G, \mu_2 \in [\![P_2]\!]_G \text{ and } F^{\mu_1 \oplus \mu_2} = \text{true} \,\}$$
$$\cup\ \{\, \mu_1 \in [\![P_1]\!]_G \mid \mu_1 \not\sim \mu_2, \text{ for all } \mu_2 \in [\![P_2]\!]_G, \text{ or } [\![P_2]\!]_G = \emptyset \,\}$$
$$\cup\ \{\, \mu_1 \in [\![P_1]\!]_G \mid \text{there is } \mu_2 \in [\![P_2]\!]_G \text{ with } \mu_1 \sim \mu_2 \text{ and } F^{\mu_1 \oplus \mu_2} = \text{false} \,\}$$

Non-monotone SPARQL: OPTIONAL

```
SELECT ?p ?d WHERE {
  ?p a :Prof
  OPTIONAL { ?p :worksIn ?d
         FILTER (?d != :CS) }
}
```

:Adams	a	:Prof
:Brown	a	:Prof
:Clarke	a	:Prof
:Brown	:worksIn	:CS
:Clarke	:worksIn	:Maths

$$[\![P_1 \ \textsc{Diff}_F \ P_2]\!]_G = \{ \mu_1 \in [\![P_1]\!]_G \mid \text{there is no } \mu_2 \in [\![P_2]\!]_G \text{ with } \mu_1 \sim \mu_2 \text{ and } F^{\mu_1 \oplus \mu_2} = \text{true} \}$$

$$P_1 \ \textsc{Opt}_F \ P_2 = \underbrace{\textsc{Filter}_F(P_1 \ \textsc{Join} \ P_2)}_{\text{`}P_1\text{ that have a compatible }P_2\text{ with }F\text{'}} \ \textsc{Union} \ \underbrace{P_1 \ \textsc{Diff}_F \ P_2}_{\text{`}P_1\text{ that have no compatible }P_2\text{ with }F\text{'}}$$

answer

	?p	?d
μ_1	:Adams	
μ_2	:Clarke	:Maths
μ_3	:Brown	

NB: SPARQL 1.1 specification **incorrectly** says 'Written in full that is:

$$[\![P_1 \ \textsc{Opt}_F \ P_2]\!]_G = \{ \mu_1 \oplus \mu_2 \mid \mu_1 \in [\![P_1]\!]_G, \mu_2 \in [\![P_2]\!]_G \text{ and } F^{\mu_1 \oplus \mu_2} = \text{true} \}$$
$$\cup \ \{ \mu_1 \in [\![P_1]\!]_G \mid \mu_1 \not\sim \mu_2, \text{ for all } \mu_2 \in [\![P_2]\!]_G, \text{ or } [\![P_2]\!]_G = \emptyset \}$$
$$\cup \ \{ \mu_1 \in [\![P_1]\!]_G \mid \text{there is } \mu_2 \in [\![P_2]\!]_G \text{ with } \mu_1 \sim \mu_2 \text{ and } F^{\mu_1 \oplus \mu_2} = \text{false} \}$$

Non-monotone SPARQL: OPTIONAL

```
SELECT ?p ?d WHERE {
  ?p a :Prof
  OPTIONAL { ?p :worksIn ?d
        FILTER (?d != :CS) }
}
```

:Adams	a	:Prof
:Brown	a	:Prof
:Clarke	a	:Prof
:Brown	:worksIn	:CS
:Clarke	:worksIn	:Maths

$$[\![P_1 \text{ DIFF}_F\ P_2]\!]_G = \{\ \mu_1 \in [\![P_1]\!]_G \mid \text{there is no } \mu_2 \in [\![P_2]\!]_G \text{ with } \mu_1 \sim \mu_2 \text{ and } F^{\mu_1 \oplus \mu_2} = \text{true}\ \}$$

$$P_1 \text{ OPT}_F\ P_2 = \underbrace{\text{FILTER}_F(P_1 \text{ JOIN } P_2)}_{\text{`}P_1 \text{ that have a compatible } P_2 \text{ with } F\text{'}} \text{ UNION } \underbrace{P_1 \text{ DIFF}_F\ P_2}_{\text{`}P_1 \text{ that have no compatible } P_2 \text{ with } F\text{'}}$$

answer

	?p	?d
μ_1	:Adams	
μ_2	:Clarke	:Maths
μ_3	:Brown	

NB: SPARQL 1.1 specification **incorrectly** says 'Written in full that is:

$$[\![P_1 \text{ OPT}_F\ P_2]\!]_G = \{\ \mu_1 \oplus \mu_2 \mid \mu_1 \in [\![P_1]\!]_G, \mu_2 \in [\![P_2]\!]_G \text{ and } F^{\mu_1 \oplus \mu_2} = \text{true}\ \}$$

$$\cup\ \{\ \mu_1 \in [\![P_1]\!]_G \mid \mu_1 \not\sim \mu_2, \text{ for all } \mu_2 \in [\![P_2]\!]_G, \text{ or } [\![P_2]\!]_G = \emptyset\ \}$$

$$\cup\ \{\ \mu_1 \in [\![P_1]\!]_G \mid \text{there is } \mu_2 \in [\![P_2]\!]_G \text{ with } \mu_1 \sim \mu_2 \text{ and } F^{\mu_1 \oplus \mu_2} = \text{false}\ \}$$

On Diff and Opt (1)

equivalent patterns $P_1 \equiv P_2 \iff [\![P_1]\!]_G = [\![P_2]\!]_G$, for all G

On DIFF and OPT (1)

equivalent patterns $P_1 \equiv P_2 \iff [\![P_1]\!]_G = [\![P_2]\!]_G$, for all G

Angles & Gutierrez (2008)

$$P_1 \text{ DIFF}_\top \; P_2 \quad \equiv \quad \text{FILTER}_{\neg\text{bound}(?u)}(P_1 \text{ OPT}_\top \; (P_2 \text{ JOIN } \{?u \; ?v \; ?w\}))$$

On DIFF and OPT (1)

equivalent patterns $P_1 \equiv P_2 \iff [\![P_1]\!]_G = [\![P_2]\!]_G$, for all G

Angles & Gutierrez (2008)

$$P_1 \ \mathrm{DIFF}_\top \ P_2 \quad \equiv \quad \mathrm{FILTER}_{\neg \mathrm{bound}(?u)}(P_1 \ \mathrm{OPT}_\top \ (P_2 \ \mathrm{JOIN} \ \{?u \ ?v \ ?w\}))$$

'universal' triple pattern
'always' gives a binding for $?u$

On DIFF and OPT (1)

equivalent patterns $P_1 \equiv P_2 \iff [\![P_1]\!]_G = [\![P_2]\!]_G$, for all G

Angles & Gutierrez (2008)

$$P_1 \text{ DIFF}_\top P_2 \equiv \text{FILTER}_{\neg\text{bound}(?u)}(P_1 \text{ OPT}_\top (P_2 \text{ JOIN } \{?u\ ?v\ ?w\}))$$

'universal' triple pattern
'always' gives a binding for $?u$

is **not** quite correct: if $P_1 = P_2 = \{\}$ and $G = \emptyset$, then $[\![P_i]\!]_G = \{\mu_\emptyset\}$

On DIFF and OPT (1)

equivalent patterns $P_1 \equiv P_2 \iff [\![P_1]\!]_G = [\![P_2]\!]_G$, for all G

Angles & Gutierrez (2008)

$$P_1 \ \text{DIFF}_\top \ P_2 \quad \equiv \quad \text{FILTER}_{\neg\text{bound}(?u)}(P_1 \ \text{OPT}_\top \ (P_2 \ \text{JOIN} \ \{?u \ ?v \ ?w\}))$$

'universal' triple pattern
'always' gives a binding for $?u$

is **not** quite correct: if $P_1 = P_2 = \{\}$ and $G = \emptyset$, then $[\![P_i]\!]_G = \{\mu_\emptyset\}$

so, $[\![P_1 \ \text{DIFF}_\top \ P_2]\!]_G = \emptyset$ (as $\mu_\emptyset$ is compatible with $\mu_\emptyset$)

but $[\![\{?u \ ?v \ ?w\}]\!]_G = \emptyset$ and so, $[\![P_1 \ \text{OPT}_F \ldots]\!]_G = \{\mu_\emptyset\}$

On DIFF and OPT (1)

equivalent patterns $P_1 \equiv P_2 \iff [\![P_1]\!]_G = [\![P_2]\!]_G$, for all G

Angles & Gutierrez (2008)

~~P_1 DIFF$_\top$ P_2 $\equiv$ FILTER$_{\neg\text{bound}(?u)}(P_1$ OPT$_\top$ $(P_2$ JOIN $\{?u\ ?v\ ?w\}))$~~

'universal' triple pattern
'always' gives a binding for $?u$

is **not** quite correct: if $P_1 = P_2 = \{\}$ and $G = \emptyset$, then $[\![P_i]\!]_G = \{\mu_\emptyset\}$

so, $[\![P_1$ DIFF$_\top$ $P_2]\!]_G = \emptyset$ (as $\mu_\emptyset$ is compatible with $\mu_\emptyset$)

but $[\![\{?u\ ?v\ ?w\}]\!]_G = \emptyset$ and so, $[\![P_1$ OPT$_F \ldots]\!]_G = \{\mu_\emptyset\}$

Polleres (2009): a fix that avoids the problem
by effectively making the dataset non-empty (GRAPH operation)

On DIFF and OPT (2)

$\mathcal{S}$ is a set of SPARQL operators

e.g., $\mathcal{S} = \{\,$ FILTER, UNION, JOIN $\}$

operator O is **$\mathcal{S}$-expressible** if,

for any pattern over $\mathcal{S} \cup \{O\}$, there is an equivalent pattern over $\mathcal{S}$

On Diff and Opt (2)

$\mathcal{S}$ is a set of SPARQL operators

e.g., $\mathcal{S} = \{\,\text{Filter},\ \text{Union},\ \text{Join}\,\}$

operator O is **$\mathcal{S}$-expressible** if,

for any pattern over $\mathcal{S} \cup \{O\}$, there is an equivalent pattern over $\mathcal{S}$

Zhang & Van den Bussche (2014) Join is $\{\text{Filter},\ \text{Opt}_\top\}$-expressible;
all other operators in the set $\{\,\text{Join},\ \text{Union},\ \text{Opt}_\top,\ \text{Filter},\ \text{Proj}\,\}$
are not expressible via the rest.

proof idea: $$P_1 \text{ Join } P_2 \equiv (P_1 \text{ Opt}_\top\ P_2) \text{ Diff}_\top\ (P_1 \text{ Diff}_\top\ P_2)$$

and then $\text{Diff}_\top$ **carefully** via Filter and $\text{Opt}_\top$

On DIFF and OPT (2)

$\mathcal{S}$ is a set of SPARQL operators

e.g., $\mathcal{S} = \{\,\text{FILTER},\ \text{UNION},\ \text{JOIN}\,\}$

operator O is **$\mathcal{S}$-expressible** if,

for any pattern over $\mathcal{S} \cup \{O\}$, there is an equivalent pattern over $\mathcal{S}$

Zhang & Van den Bussche (2014) JOIN is $\{\text{FILTER},\ \text{OPT}_\top\}$-expressible;

all other operators in the set $\{\,\text{JOIN},\ \text{UNION},\ \text{OPT}_\top,\ \text{FILTER},\ \text{PROJ}\,\}$

are not expressible via the rest.

proof idea:

$$P_1\ \text{JOIN}\ P_2 \equiv (P_1\ \text{OPT}_\top\ P_2)\ \text{DIFF}_\top\ (P_1\ \text{DIFF}_\top\ P_2)$$

and then $\text{DIFF}_\top$ **carefully** via FILTER and $\text{OPT}_\top$

Theorem $\text{DIFF}_\top$ is **not** $\mathcal{S} \cup \{\text{OPT}_F\}$-expressible

proof idea: P over $\mathcal{S} \cup \{\text{OPT}_F\} \implies$ if $\mu_\emptyset \in [\![P]\!]_G$ then $\mu_\emptyset \in [\![P]\!]_\emptyset$

On DIFF and OPT (2)

$\mathcal{S}$ is a set of SPARQL operators

e.g., $\mathcal{S} = \{\,\text{FILTER},\ \text{UNION},\ \text{JOIN}\,\}$

operator O is **$\mathcal{S}$-expressible** if,

for any pattern over $\mathcal{S} \cup \{O\}$, there is an equivalent pattern over $\mathcal{S}$

Zhang & Van den Bussche (2014) JOIN is $\{\text{FILTER},\ \text{OPT}_\top\}$-expressible;

all other operators in the set $\{\,\text{JOIN},\ \text{UNION},\ \text{OPT}_\top,\ \text{FILTER},\ \text{PROJ}\,\}$

are not expressible via the rest.

proof idea: $P_1 \text{ JOIN } P_2 \equiv (P_1 \text{ OPT}_\top\ P_2) \text{ DIFF}_\top\ (P_1 \text{ DIFF}_\top\ P_2)$

and then $\text{DIFF}_\top$ **carefully** via FILTER and $\text{OPT}_\top$

Theorem $\text{DIFF}_\top$ is **not** $\mathcal{S} \cup \{\text{OPT}_F\}$-expressible

proof idea: P over $\mathcal{S} \cup \{\text{OPT}_F\} \implies$ if $\mu_\emptyset \in [\![P]\!]_G$ then $\mu_\emptyset \in [\![P]\!]_\emptyset$

$$P = \{\}\ \text{DIFF}_\top\ \text{FILTER}_{\neg\text{bound}(?u)}(\{\}\ \text{OPT}_\top\ \{?u\ ?v\ ?w\})$$

$[\![P]\!]_\emptyset = \emptyset$ but $[\![P]\!]_G = \{\mu_\emptyset\}$, for any $G \neq \emptyset$

Projection in SPARQL. On DIFF and OPT (3)

```
SELECT ?p WHERE {
  ?p :worksIn ?d
}
```

← `?d` is projected away

:Adams	a	:Prof
:Brown	a	:Prof
:Clarke	a	:Prof
:Brown	:worksIn	:CS
:Clarke	:worksIn	:Maths
:Davies	:worksIn	:CS

Projection in SPARQL. On DIFF and OPT (3)

```
SELECT ?p WHERE {
  ?p :worksIn ?d
}
```

← ?d is projected away

:Adams	a	:Prof
:Brown	a	:Prof
:Clarke	a	:Prof
:Brown	:worksIn	:CS
:Clarke	:worksIn	:Maths
:Davies	:worksIn	:CS

answer

?p
:Brown
:Clarke
:Davies

$$[\![\text{PROJ}_{\boldsymbol{V}}\, \boldsymbol{P}]\!]_G = \{\, \boldsymbol{\mu}|_{\boldsymbol{V}} \mid \boldsymbol{\mu} \in [\![\boldsymbol{P}]\!]_G \,\}$$

where $\boldsymbol{\mu}|_{\boldsymbol{V}}$ is the restriction of $\boldsymbol{\mu}$ to $\boldsymbol{V}$

Projection in SPARQL. On DIFF and OPT (3)

```
SELECT ?p WHERE {
  ?p :worksIn ?d
}
```

`?d` is projected away

:Adams	a	:Prof
:Brown	a	:Prof
:Clarke	a	:Prof
:Brown	:worksIn	:CS
:Clarke	:worksIn	:Maths
:Davies	:worksIn	:CS

answer

?p
:Brown
:Clarke
:Davies

$$[\![\text{PROJ}_V\ P]\!]_G = \{\ \mu|_V \mid \mu \in [\![P]\!]_G\ \}$$

where $\mu|_V$ is the restriction of μ to V

NB: projection in SPARQL is only at the top level

however, PROJ can always be pushed up (by careful variable renaming)

Projection in SPARQL. On DIFF and OPT (3)

```
SELECT ?p WHERE {
   ?p :worksIn ?d
}
```

← ?d is projected away

:Adams	a	:Prof
:Brown	a	:Prof
:Clarke	a	:Prof
:Brown	:worksIn	:CS
:Clarke	:worksIn	:Maths
:Davies	:worksIn	:CS

answer

?p
:Brown
:Clarke
:Davies

$$[\![\text{PROJ}_V\ P]\!]_G = \{\ \mu|_V \mid \mu \in [\![P]\!]_G\ \}$$

where $\mu|_V$ is the restriction of μ to V

NB: projection in SPARQL is only at the top level

however, PROJ can always be pushed up (by careful variable renaming)

Theorem DIFF_F is $\{\text{FILTER}, \text{UNION}, \text{PROJ}, \text{OPT}_F\}$-expressible

Projection in SPARQL. On DIFF and OPT (3)

```
SELECT ?p WHERE {
  ?p :worksIn ?d
}
```

← `?d` is projected away

:Adams	a	:Prof
:Brown	a	:Prof
:Clarke	a	:Prof
:Brown	:worksIn	:CS
:Clarke	:worksIn	:Maths
:Davies	:worksIn	:CS

answer

?p
:Brown
:Clarke
:Davies

$$[\![\text{PROJ}_V\ P]\!]_G = \{\ \mu|_V \mid \mu \in [\![P]\!]_G\ \}$$

where $\mu|_V$ is the restriction of μ to V

NB: projection in SPARQL is only at the top level
however, PROJ can always be pushed up (by careful variable renaming)

Theorem DIFF_F is $\{\text{FILTER}, \text{UNION}, \text{PROJ}, \text{OPT}_F\}$-expressible

$$P_1\ \text{DIFF}_F\ P_2 \equiv \text{ON\_EMPTY}_{P_1 \text{DIFF}_F P_2}\ \ \text{UNION}$$
$$\text{PROJ}_{\text{var}(P_1)}\ \text{FILTER}_{\neg\text{bound}(?u_2)}\big((P_1\ \text{JOIN}\ \{?u_1\ ?v_1\ ?w_1\})\ \ \text{OPT}_F$$
$$(P_2\ \text{JOIN}\ \{?u_2\ ?v_2\ ?w_2\})\big)$$

Projection in SPARQL. On DIFF and OPT (3)

```
SELECT ?p WHERE {
  ?p :worksIn ?d
}
```

?d is projected away

:Adams	a	:Prof
:Brown	a	:Prof
:Clarke	a	:Prof
:Brown	:worksIn	:CS
:Clarke	:worksIn	:Maths
:Davies	:worksIn	:CS

answer

?p
:Brown
:Clarke
:Davies

$$[\![\text{PROJ}_V\ P]\!]_G = \{\ \mu|_V \mid \mu \in [\![P]\!]_G\ \}$$

where $\mu|_V$ is the restriction of μ to V

NB: projection in SPARQL is only at the top level

however, PROJ can always be pushed up (by careful variable renaming)

Theorem DIFF_F is $\{\text{FILTER}, \text{UNION}, \text{PROJ}, \text{OPT}_F\}$-expressible

$P_1\ \text{DIFF}_F\ P_2$ on the empty graph

$$P_1\ \text{DIFF}_F\ P_2 \equiv \text{ON\_EMPTY}_{P_1 \text{DIFF}_F P_2}\ \text{UNION}$$
$$\text{PROJ}_{\text{var}(P_1)}\ \text{FILTER}_{\neg\text{bound}(?u_2)}\big((P_1\ \text{JOIN}\ \{?u_1\ ?v_1\ ?w_1\})\ \ \text{OPT}_F$$
$$(P_2\ \text{JOIN}\ \{?u_2\ ?v_2\ ?w_2\})\big)$$

Ternary OPTIONAL of SPARQL

Angles and Gutierrez (2008), **(Pérez et al., 2009)**, . . .

$$P_1 \text{ OPT}_F\ P_2 \quad \equiv \quad P_1 \text{ OPT}_{\top} \text{ FILTER}_F(P_1 \text{ JOIN } P_2)$$

Ternary OPTIONAL of SPARQL

Angles and Gutierrez (2008), **(Pérez et al., 2009)**, . . .

$$P_1 \ \text{OPT}_F \ P_2 \quad \equiv \quad P_1 \ \text{OPT}_\top \ \text{FILTER}_F(P_1 \ \text{JOIN} \ P_2)$$

$[\![P_1]\!]_G$

?u	?v
:a	:b
:a	

$[\![P_2]\!]_G$

?u	?w
:a	:c

$$F = \text{bound}(?v)$$

Ternary OPTIONAL of SPARQL

Angles and Gutierrez (2008), **(Pérez et al., 2009)**, ...

$$P_1 \ \mathrm{OPT}_F \ P_2 \quad \equiv \quad P_1 \ \mathrm{OPT}_\top \ \mathrm{FILTER}_F(P_1 \ \mathrm{JOIN} \ P_2)$$

$[\![P_1]\!]_G$

?u	?v
:a	:b
:a	

$[\![P_2]\!]_G$

?u	?w
:a	:c

$[\![P_1 \ \mathrm{OPT}_F \ P_2]\!]_G$

?u	?v	?w
:a	:b	:c
:a		

$$F = \mathrm{bound}(?v)$$

Ternary OPTIONAL of SPARQL

Angles and Gutierrez (2008), (Pérez et al., 2009), . . .

$$P_1 \text{ OPT}_F \; P_2 \quad \equiv \quad P_1 \text{ OPT}_\top \text{ FILTER}_F(P_1 \text{ JOIN } P_2)$$

$[\![P_1]\!]_G$

?u	?v
:a	:b
:a	

$[\![P_2]\!]_G$

?u	?w
:a	:c

$[\![P_1 \text{ OPT}_F \; P_2]\!]_G$

?u	?v	?w
:a	:b	:c
:a		

$[\![P_1 \text{ JOIN } P_2]\!]_G$

?u	?v	?w
:a	:b	:c
:a		:c

$$F = \text{bound}(?v)$$

Ternary OPTIONAL of SPARQL

Angles and Gutierrez (2008), **(Pérez et al., 2009)**, . . .

~~$P_1 \text{ OPT}_F\ P_2 \;=\; P_1 \text{ OPT}_\top \text{ FILTER}_F(P_1 \text{ JOIN } P_2)$~~

$[\![P_1]\!]_G$

?u	?v
:a	:b
:a	

$[\![P_2]\!]_G$

?u	?w
:a	:c

$F = \text{bound}(?v)$

$[\![P_1 \text{ OPT}_F\ P_2]\!]_G$

?u	?v	?w
:a	:b	:c
:a		

$\neq$

$[\![P_1 \text{ OPT}_\top \ldots]\!]_G$

?u	?v	?w
:a	:b	:c

$[\![P_1 \text{ JOIN } P_2]\!]_G$

?u	?v	?w
:a	:b	:c
:a		:c

Ternary OPTIONAL of SPARQL

Angles and Gutierrez (2008), (Pérez et al., 2009), . . .

~~$P_1 \text{ OPT}_F\ P_2 \quad = \quad P_1 \text{ OPT}_\top \text{ FILTER}_F(P_1 \text{ JOIN } P_2)$~~

$[\![P_1]\!]_G$

?u	?v
:a	:b
:a	

$[\![P_2]\!]_G$

?u	?w
:a	:c

$[\![P_1 \text{ OPT}_F\ P_2]\!]_G$

?u	?v	?w
:a	:b	:c
:a		

$\neq$

$[\![P_1 \text{ OPT}_\top \ldots]\!]_G$

?u	?v	?w
:a	:b	:c

$[\![P_1 \text{ JOIN } P_2]\!]_G$

?u	?v	?w
:a	:b	:c
:a		:c

$F = \text{bound}(?v)$

Theorem OPT_F is $\{\text{FILTER}, \text{ UNION}, \text{ OPT}_\top\}$-expressible

Ternary OPTIONAL of SPARQL

Angles and Gutierrez (2008), **(Pérez et al., 2009)**, ...

~~$P_1 \ \mathrm{OPT}_F \ P_2 \ \equiv \ P_1 \ \mathrm{OPT}_\top \ \mathrm{FILTER}_F(P_1 \ \mathrm{JOIN} \ P_2)$~~

$[\![P_1]\!]_G$

?u	?v
:a	:b
:a	

$[\![P_2]\!]_G$

?u	?w
:a	:c

$[\![P_1 \ \mathrm{OPT}_F \ P_2]\!]_G$

?u	?v	?w
:a	:b	:c
:a		

$\neq$

$[\![P_1 \ \mathrm{OPT}_\top \ \ldots]\!]_G$

?u	?v	?w
:a	:b	:c

$[\![P_1 \ \mathrm{JOIN} \ P_2]\!]_G$

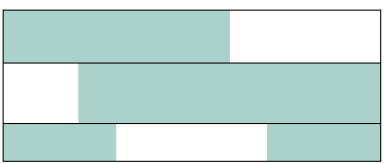
?u	?v	?w
:a	:b	:c
:a		:c

$F = \mathrm{bound}(?v)$

Theorem OPT_F is $\{\mathrm{FILTER}, \ \mathrm{UNION}, \ \mathrm{OPT}_\top\}$-expressible

$$P_1 \ \mathrm{OPT}_F \ P_2 \ \equiv \bigcup_{V \subseteq \mathrm{var}(P_1) \cap \mathrm{var}(P_2)} \Big[(\mathrm{FILTER}_{F_V} \ P_1) \ \mathrm{OPT}_\top \ \mathrm{FILTER}_F((\mathrm{FILTER}_{F_V} \ P_1) \ \mathrm{JOIN} \ P_2)\Big]$$

F_V selects the **V-uniform slice** of P_1: $F_V = \bigwedge_{?v \in V} \mathrm{bound}(?v) \ \wedge \bigwedge_{?v \in (\mathrm{var}(P_1) \cap \mathrm{var}(P_2)) \setminus V} \neg \mathrm{bound}(?v)$



horizontal decomposition in DBs

Ternary OPTIONAL of SPARQL

Angles and Gutierrez (2008), **(Pérez et al., 2009)**, . . .

~~$P_1 \text{ OPT}_F\ P_2 \;\equiv\; P_1 \text{ OPT}_\top \text{ FILTER}_F(P_1 \text{ JOIN } P_2)$~~

$[\![P_1]\!]_G$

?u	?v
:a	:b
:a	

$[\![P_2]\!]_G$

?u	?w
:a	:c

$[\![P_1 \text{ OPT}_F\ P_2]\!]_G$

?u	?v	?w
:a	:b	:c
:a		

$\neq$

$[\![P_1 \text{ OPT}_\top \ldots]\!]_G$

?u	?v	?w
:a	:b	:c

$[\![P_1 \text{ JOIN } P_2]\!]_G$

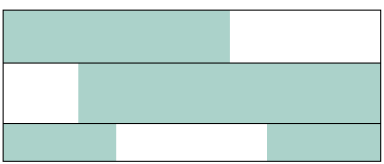
?u	?v	?w
:a	:b	:c
:a		:c

$F = \text{bound}(?v)$

Theorem OPT_F is $\{\text{FILTER}, \text{UNION}, \text{OPT}_\top\}$-expressible

$$P_1 \text{ OPT}_F\ P_2 \;\equiv \bigcup_{V \subseteq \text{var}(P_1) \cap \text{var}(P_2)}^{\text{UNION}} \big[(\text{FILTER}_{F_V}\ P_1) \text{ OPT}_\top \text{ FILTER}_F((\text{FILTER}_{F_V}\ P_1) \text{ JOIN } P_2)\big]$$

F_V selects the **V-uniform slice** of P_1: $F_V = \bigwedge_{?v \in V} \text{bound}(?v) \;\wedge \bigwedge_{?v \in (\text{var}(P_1) \cap \text{var}(P_2)) \setminus V} \neg\text{bound}(?v)$



horizontal decomposition in DBs

the UNION is exponential. . . is it unavoidable?

Polynomial Expressibility

operator O is **polynomially $\mathcal{S}$-expressible** if there is a polynomial f such that,
for any $P = O(P_1, \ldots, P_n)$ with the P_i over $\mathcal{S}$,
there is an equivalent pattern P' over $\mathcal{S}$ with $|P'| = f(|P|)$

Polynomial Expressibility

operator $\boldsymbol{O}$ is **polynomially $\mathcal{S}$-expressible** if there is a polynomial $\boldsymbol{f}$ such that,
for any $\boldsymbol{P} = \boldsymbol{O}(\boldsymbol{P}_1, \ldots, \boldsymbol{P}_n)$ with the $\boldsymbol{P}_i$ over $\mathcal{S}$,
there is an equivalent pattern $\boldsymbol{P}'$ over $\mathcal{S}$ with $|\boldsymbol{P}'| = \boldsymbol{f}(|\boldsymbol{P}|)$

So far:

- $\text{DIFF}_{\top}$ is **not** $\mathcal{S} \cup \{\text{OPT}_{\boldsymbol{F}}\}$-expressible
- $\text{DIFF}_{\boldsymbol{F}}$ is polynomially $\{\text{FILTER}, \text{UNION}, \text{PROJ}, \text{OPT}_{\boldsymbol{F}}\}$-expressible
- $\text{OPT}_{\boldsymbol{F}}$ is $\{\text{FILTER}, \text{UNION}, \text{OPT}_{\top}\}$-expressible

Polynomial Expressibility

operator O is **polynomially $\mathcal{S}$-expressible** if there is a polynomial f such that,
for any $P = O(P_1, \ldots, P_n)$ with the P_i over $\mathcal{S}$,
there is an equivalent pattern P' over $\mathcal{S}$ with $|P'| = f(|P|)$

So far:

- $\text{DIFF}_{\top}$ is **not** $\mathcal{S} \cup \{\text{OPT}_F\}$-expressible
- DIFF_F is polynomially $\{\text{FILTER}, \text{UNION}, \text{PROJ}, \text{OPT}_F\}$-expressible
- OPT_F is $\{\text{FILTER}, \text{UNION}, \text{OPT}_{\top}\}$-expressible

but **not polynomially** (under the standard complexity-theoretic assumptions)

Ternary OPT is NOT Polynomially Expressible via Binary OPT

singular graph $G_a = \{(\texttt{:a :a :a})\}$

L1 '$[\![P]\!]_{G_a} \neq \emptyset$' for patterns P over $\mathcal{S} \cup \{\text{OPT}_F\}$ of $\underbrace{\text{o-rank}}_{\text{nesting depth of OPT}_F} \leq n$ is **Σ^p_{n+1}-hard**

Ternary OPT is NOT Polynomially Expressible via Binary OPT

singular graph $G_a = \{(\texttt{:a :a :a})\}$

L1 '$[\![P]\!]_{G_a} \neq \emptyset$' for patterns P over $\mathcal{S} \cup \{\text{OPT}_F\}$ of $\underbrace{\text{o-rank}}_{\text{nesting depth of OPT}_F}$ $\leq n$ is Σ^p_{n+1}**-hard**

Proof by encoding QBF $\exists \vec{x}_1 \forall \vec{x}_2 \ldots Q \vec{x}_{n+1}\, \psi$

if n is odd and $Q = \forall$, then $\phi_{n+1} = \neg\psi$ and $\phi_k = \forall \vec{x}_{k+1} \neg \phi_{k+1}$, for $k \leq n$

Ternary OPT is NOT Polynomially Expressible via Binary OPT

singular graph $G_a = \{(\texttt{:a :a :a})\}$

L1 '$[\![P]\!]_{G_a} \neq \emptyset$' for patterns P over $\mathcal{S} \cup \{\text{OPT}_F\}$ of $\underbrace{\text{o-rank}}_{\text{nesting depth of OPT}_F} \leq n$ is Σ^p_{n+1}**-hard**

Proof by encoding QBF $\exists \vec{x}_1 \forall \vec{x}_2 \ldots Q \vec{x}_{n+1}\, \psi$

if n is odd and $Q = \forall$, then $\phi_{n+1} = \neg\psi$ and $\phi_k = \forall \vec{x}_{k+1}\, \neg\phi_{k+1}$, for $k \leq n$

$$\phi_k \quad \approx \quad P_k \; = \; \text{FILTER}_{\neg\text{bound}(?v_{k+1})}(B_k \text{ OPT}_{F_k}\; P_{k+1})$$

Ternary OPT is NOT Polynomially Expressible via Binary OPT

singular graph $G_a = \{(\texttt{:a :a :a})\}$

L1 '$[\![P]\!]_{G_a} \neq \emptyset$' for patterns P over $\mathcal{S} \cup \{\text{OPT}_F\}$ of $\underbrace{\text{o-rank}}_{\text{nesting depth of OPT}_F}$ $\leq n$ is Σ^p_{n+1}**-hard**

Proof by encoding QBF $\exists \vec{x}_1 \forall \vec{x}_2 \ldots Q \vec{x}_{n+1}\, \psi$

if n is odd and $Q = \forall$, then $\phi_{n+1} = \neg\psi$ and $\phi_k = \forall \vec{x}_{k+1}\, \neg\phi_{k+1}$, for $k \leq n$

$$\phi_k \approx P_k = \text{FILTER}_{\neg\text{bound}(?v_{k+1})}(B_k\ \text{OPT}_{F_k}\ P_{k+1})$$

L2 '$[\![P]\!]_{G_a} \neq \emptyset$' for patterns P over $\mathcal{S} \cup \{\text{PROJ}, \text{OPT}_\top\}$ is **in** Δ^p_2

polynomial deterministic algorithm with $|P| + 1$ calls to an NP-oracle (P^{NP})

Ternary OPT is NOT Polynomially Expressible via Binary OPT

singular graph $G_a = \{(\texttt{:a :a :a})\}$

L1 '$[\![P]\!]_{G_a} \neq \emptyset$' for patterns P over $\mathcal{S} \cup \{\text{OPT}_F\}$ of $\underbrace{\text{o-rank}}_{\text{nesting depth of OPT}_F} \leq n$ is Σ^p_{n+1}**-hard**

Proof by encoding QBF $\exists \vec{x}_1 \forall \vec{x}_2 \dots Q \vec{x}_{n+1}\, \psi$

if n is odd and $Q = \forall$, then $\phi_{n+1} = \neg\psi$ and $\phi_k = \forall \vec{x}_{k+1}\, \neg\phi_{k+1}$, for $k \leq n$

$$\phi_k \approx P_k = \text{FILTER}_{\neg\text{bound}(?v_{k+1})}(B_k\ \text{OPT}_{F_k}\ P_{k+1})$$

L2 '$[\![P]\!]_{G_a} \neq \emptyset$' for patterns P over $\mathcal{S} \cup \{\text{PROJ}, \text{OPT}_\top\}$ is **in Δ^p_2**

polynomial deterministic algorithm with $|P| + 1$ calls to an NP-oracle (P^{NP})

Proof

$$[\![P_1\ \text{OPT}_\top\ P_2]\!]_{G_a} = \begin{cases} [\![P_1\ \text{JOIN}\ P_2]\!]_{G_a}, & \text{if } [\![P_2]\!]_{G_a} \neq \emptyset \\ [\![P_1]\!]_{G_a}, & \text{if } [\![P_2]\!]_{G_a} = \emptyset \end{cases}$$

Ternary OPT is NOT Polynomially Expressible via Binary OPT

singular graph $G_a = \{(\texttt{:a :a :a})\}$

L1 '$[\![P]\!]_{G_a} \neq \emptyset$' for patterns P over $\mathcal{S} \cup \{\text{OPT}_F\}$ of $\underbrace{\text{o-rank}}_{\text{nesting depth of OPT}_F} \leq n$ is Σ^p_{n+1}**-hard**

Proof by encoding QBF $\exists \vec{x}_1 \forall \vec{x}_2 \ldots Q \vec{x}_{n+1}\, \psi$

if n is odd and $Q = \forall$, then $\phi_{n+1} = \neg\psi$ and $\phi_k = \forall \vec{x}_{k+1}\, \neg\phi_{k+1}$, for $k \leq n$

$$\phi_k \quad \approx \quad P_k = \text{FILTER}_{\neg \text{bound}(?v_{k+1})}(B_k \text{ OPT}_{F_k}\ P_{k+1})$$

L2 '$[\![P]\!]_{G_a} \neq \emptyset$' for patterns P over $\mathcal{S} \cup \{\text{PROJ}, \text{OPT}_\top\}$ is **in Δ^p_2**

polynomial deterministic algorithm with $|P| + 1$ calls to an NP-oracle (P^{NP})

Proof

$$[\![P_1 \text{ OPT}_\top P_2]\!]_{G_a} = \begin{cases} [\![P_1 \text{ JOIN } P_2]\!]_{G_a}, & \text{if } [\![P_2]\!]_{G_a} \neq \emptyset \\ [\![P_1]\!]_{G_a}, & \text{if } [\![P_2]\!]_{G_a} = \emptyset \end{cases}$$

checking '$[\![P_2]\!]_{G_a} = \emptyset$' for a pattern P_2 over $\mathcal{S} \cup \{\text{PROJ}\}$ is **NP-complete**

Ternary OPT is NOT Polynomially Expressible via Binary OPT

singular graph $G_a = \{(\texttt{:a :a :a})\}$

L1 '$[\![P]\!]_{G_a} \neq \emptyset$' for patterns P over $\mathcal{S} \cup \{\text{OPT}_F\}$ of $\underbrace{\text{o-rank}}_{\text{nesting depth of OPT}_F} \leq n$ is **Σ^p_{n+1}-hard**

Proof by encoding QBF $\exists \vec{x}_1 \forall \vec{x}_2 \dots Q \vec{x}_{n+1}\, \psi$

if n is odd and $Q = \forall$, then $\phi_{n+1} = \neg\psi$ and $\phi_k = \forall \vec{x}_{k+1}\, \neg\phi_{k+1}$, for $k \leq n$

$$\phi_k \approx P_k = \text{FILTER}_{\neg\text{bound}(?v_{k+1})}(B_k \text{ OPT}_{F_k}\, P_{k+1})$$

L2 '$[\![P]\!]_{G_a} \neq \emptyset$' for patterns P over $\mathcal{S} \cup \{\text{PROJ}, \text{OPT}_\top\}$ is **in Δ^p_2**

polynomial deterministic algorithm with $|P| + 1$ calls to an NP-oracle (P^{NP})

Proof

$$[\![P_1 \text{ OPT}_\top\, P_2]\!]_{G_a} = \begin{cases} [\![P_1 \text{ JOIN } P_2]\!]_{G_a}, & \text{if } [\![P_2]\!]_{G_a} \neq \emptyset \\ [\![P_1]\!]_{G_a}, & \text{if } [\![P_2]\!]_{G_a} = \emptyset \end{cases}$$

checking '$[\![P_2]\!]_{G_a} = \emptyset$' for a pattern P_2 over $\mathcal{S} \cup \{\text{PROJ}\}$ is **NP-complete**

L1 + **L2** for $P_1 \text{ OPT}_F\, P_2$ ⟶ **not poly-expressible** (unless $\Delta^p_2 = \Sigma^p_2$)

Expressing Ternary OPT via Binary OPT

E1 $P_1 \text{ DIFF}_F P_2 \quad \equiv \quad P_1 \text{ SETMINUS}$ [pattern that selects $\mu_1 \in [\![P_1]\!]_G$ that have a compatible $\mu_2 \in [\![P_2]\!]_G$ with $F^{\mu_1 \oplus \mu_2} = \text{true}$]

where $[\![P_1 \text{ SETMINUS } P_2]\!]_G = [\![P_1]\!]_G \setminus [\![P_2]\!]_G$

not the MINUS of SPARQL

Expressing Ternary OPT via Binary OPT

E1 P_1 DIFF$_F$ P_2 $\equiv$ P_1 SETMINUS
| pattern that selects $\mu_1 \in [\![P_1]\!]_G$ that have a compatible $\mu_2 \in [\![P_2]\!]_G$ with $F^{\mu_1 \oplus \mu_2} = \text{true}$ |

where $[\![P_1 \text{ SETMINUS } P_2]\!]_G = [\![P_1]\!]_G \setminus [\![P_2]\!]_G$

not the MINUS of SPARQL

E2 P_1 SETMINUS P_2 $\equiv$

ON_EMPTY$_{P_1 \text{SETMINUS} P_2}$ UNION

(P_1 MONOMINUS P_2) JOIN ONE UNION

| pattern that uses two distinct elements as indicators for 'not bound' | JOIN TWO

Expressing Ternary OPT via Binary OPT

E1 P_1 DIFF$_F$ P_2 $\equiv$ P_1 SETMINUS

pattern that selects $\mu_1 \in [\![P_1]\!]_G$ that have a compatible $\mu_2 \in [\![P_2]\!]_G$ with $F^{\mu_1 \oplus \mu_2}$ = true

where $[\![P_1 \text{ SETMINUS } P_2]\!]_G = [\![P_1]\!]_G \setminus [\![P_2]\!]_G$

not the MINUS of SPARQL

E2 P_1 SETMINUS P_2 $\equiv$

ON_EMPTY$_{P_1 \text{SETMINUS} P_2}$ UNION

(P_1 MONOMINUS P_2) JOIN ONE UNION

pattern that uses two distinct elements as indicators for 'not bound'

JOIN TWO

polynomial

Expressing Ternary OPT via Binary OPT

E1 P_1 DIFF$_F$ P_2 $\equiv$ P_1 SETMINUS

pattern that selects $\mu_1 \in [\![P_1]\!]_G$ that have a compatible $\mu_2 \in [\![P_2]\!]_G$ with $F^{\mu_1 \oplus \mu_2}$ = true

where $[\![P_1 \text{ SETMINUS } P_2]\!]_G = [\![P_1]\!]_G \setminus [\![P_2]\!]_G$

not the MINUS of SPARQL

E2 P_1 SETMINUS P_2 $\equiv$

ON_EMPTY$_{P_1 \text{SETMINUS} P_2}$ UNION ← polynomial

(P_1 MONOMINUS P_2) JOIN ONE UNION

pattern that uses two distinct elements as indicators for 'not bound'

JOIN TWO ← polynomial

Expressing Ternary OPT via Binary OPT

E1 P_1 DIFF$_F$ P_2 $\equiv$ P_1 SETMINUS [pattern that selects $\mu_1 \in [\![P_1]\!]_G$ that have a compatible $\mu_2 \in [\![P_2]\!]_G$ with $F^{\mu_1 \oplus \mu_2} =$ true]

where $[\![P_1 \text{ SETMINUS } P_2]\!]_G = [\![P_1]\!]_G \setminus [\![P_2]\!]_G$ — not the MINUS of SPARQL

E2 P_1 SETMINUS P_2 $\equiv$

ON_EMPTY$_{P_1 \text{SETMINUS} P_2}$ UNION ← **polynomial**

(P_1 MONOMINUS P_2) JOIN ONE UNION

[pattern that uses two distinct elements as indicators for 'not bound'] JOIN TWO ← **polynomial**

E3 if NP = coNP then,

for every pattern P_1 MONOMINUS P_2, with the P_i over $\mathcal{S} \cup \{\text{PROJ}\}$, there is a **polynomial** pattern over $\mathcal{S} \cup \{\text{PROJ}\}$ that gives the same answers on singular graphs

MINUS of SPARQL 1.1

not to be confused with

- MINUS of (Angles & Gutierrez, 2008)
- set-theoretic complement SETMINUS, or $\setminus$

$$[\![P_1 \text{ MINUS } P_2]\!]_G = \{\, \mu_1 \in [\![P_1]\!]_G \mid \text{there is no } \mu_2 \in [\![P_2]\!]_G \text{ with } \mu_1 \sim \mu_2 \text{ and } \mathrm{dom}(\mu_1) \cap \mathrm{dom}(\mu_2) \neq \emptyset \,\}$$

$$[\![P_1 \text{ DIFF}_F\ P_2]\!]_G = \{\, \mu_1 \in [\![P_1]\!]_G \mid \text{there is no } \mu_2 \in [\![P_2]\!]_G \text{ with } \mu_1 \sim \mu_2 \text{ and } F^{\mu_1 \oplus \mu_2} = \text{true} \,\}$$

MINUS of SPARQL 1.1

not to be confused with

- MINUS of (Angles & Gutierrez, 2008)
- set-theoretic complement SETMINUS, or \

$$[\![P_1 \text{ MINUS } P_2]\!]_G = \{\, \mu_1 \in [\![P_1]\!]_G \mid \text{there is no } \mu_2 \in [\![P_2]\!]_G \text{ with } \mu_1 \sim \mu_2 \text{ and } \mathrm{dom}(\mu_1) \cap \mathrm{dom}(\mu_2) \neq \emptyset \,\}$$

$$[\![P_1 \text{ DIFF}_F\ P_2]\!]_G = \{\, \mu_1 \in [\![P_1]\!]_G \mid \text{there is no } \mu_2 \in [\![P_2]\!]_G \text{ with } \mu_1 \sim \mu_2 \text{ and } F^{\mu_1 \oplus \mu_2} = \text{true} \,\}$$

Theorem MINUS is polynomially $\{\text{DIFF}_F\}$- and $\{\text{OPT}_F, \text{FILTER}\}$-expressible

$\text{DIFF}_\top$ and $\text{OPT}_\top$ are not $\mathcal{S} \cup \{\text{PROJ}, \text{MINUS}\}$-expressible

$\mathcal{S} \cup \{O'\}$- and $\mathcal{S}_\pi \cup \{O'\}$-expressibility of O

$\mathcal{S} = \{$ FILTER, UNION, JOIN $\}$

$O' \backslash O$	DIFF$_F$	OPT$_F$	DIFF$_\top$	OPT$_\top$	MINUS
DIFF$_F$		$+$	$+$	$+$	$+$
OPT$_F$	$-$		$-$	$+$	$+$
DIFF$_\top$	$\pm$	$\pm$		$+$	$+_?$
OPT$_\top$	$-$	$\pm$	$-$		$+_?$
MINUS	$-$	$-$	$-$	$-$	

$\mathcal{S}_\pi = \mathcal{S} \cup \{$ PROJ $\}$

$O' \backslash O$	DIFF$_F$	OPT$_F$	DIFF$_\top$	OPT$_\top$	MINUS
DIFF$_F$		$+$	$+$	$+$	$+$
OPT$_F$	$+$		$+$	$+$	$+$
DIFF$_\top$	$\pm^\dagger$	$\pm^\dagger$		$+$	$+_?^\dagger$
OPT$_\top$	$\pm^\dagger$	$\pm^\dagger$	$+$		$+_?^\dagger$
MINUS	$-$	$-$	$-$	$-$	

- $-$ not expressible
- $+$ polynomially expressible
- $\pm$ expressible, but not polynomially if $\Delta_2^p \neq \Sigma_2^p$
- $+_?$ expressible, but not known if polynomially

the results with † become $+$ if NP = coNP

Summary and Open Problems

- the **ternary** OPTIONAL in SPARQL is more complex than commonly assumed
- some widely-known SPARQL equivalences are **false**

  or use assumptions different from SPARQL specification

Summary and Open Problems

- the **ternary** OPTIONAL in SPARQL is more complex than commonly assumed
- some widely-known SPARQL equivalences are **false**

  or use assumptions different from SPARQL specification
- stronger notion of polynomial expressibility: every pattern over $\mathcal{S} \cup \{O\}$ has an equivalent polynomially-sized pattern over $\mathcal{S}$

$$P_1 \text{ OPT}_F\ P_2 \equiv \text{FILTER}_F(P_1 \text{ JOIN } P_2) \text{ UNION } (P_1 \text{ DIFF}_F\ P_2)$$

- expressive power of `NOT EXISTS`
- expressiveness over non-empty RDF graphs

Summary and Open Problems

- the **ternary** OPTIONAL in SPARQL is more complex than commonly assumed
- some widely-known SPARQL equivalences are **false**

  or use assumptions different from SPARQL specification
- stronger notion of polynomial expressibility: every pattern over $\mathcal{S} \cup \{O\}$ has an equivalent polynomially-sized pattern over $\mathcal{S}$

$$P_1 \text{ OPT}_F\ P_2 \equiv \text{FILTER}_F(P_1 \text{ JOIN } P_2) \text{ UNION } (P_1 \text{ DIFF}_F\ P_2)$$

- expressive power of `NOT EXISTS`
- expressiveness over non-empty RDF graphs

Is SPARQL intuitive?

or is it just confusing names, e.g., OPTIONAL v LEFTJOIN?
MINUS v \