On the primitivity and satisfiability of SPARQL

Xiaowang Zhang

Co-work with *Jan Van den Bussche*

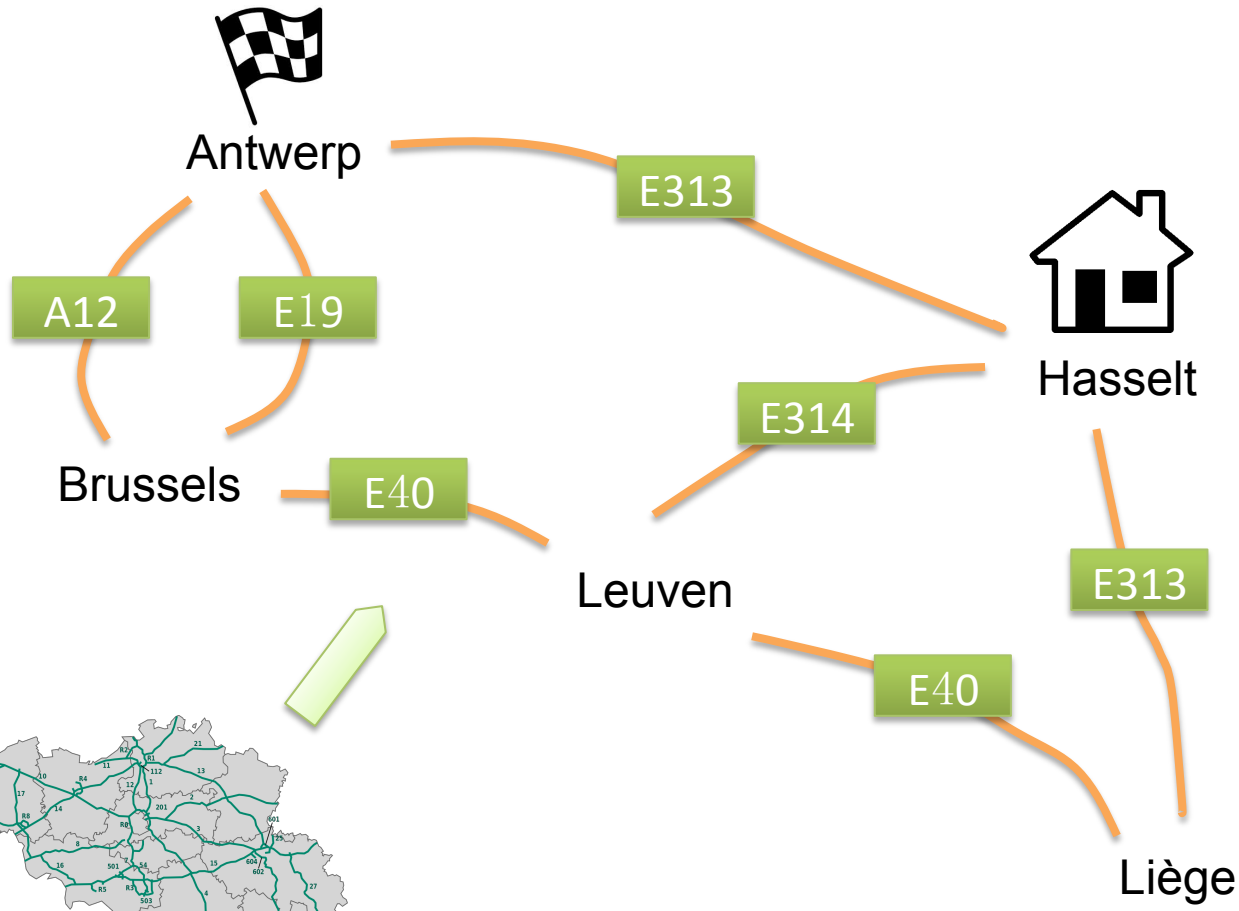
Hasselt University, Belgium

DBDBD 2014

universiteit hasselt

KNOWLEDGE IN ACTION

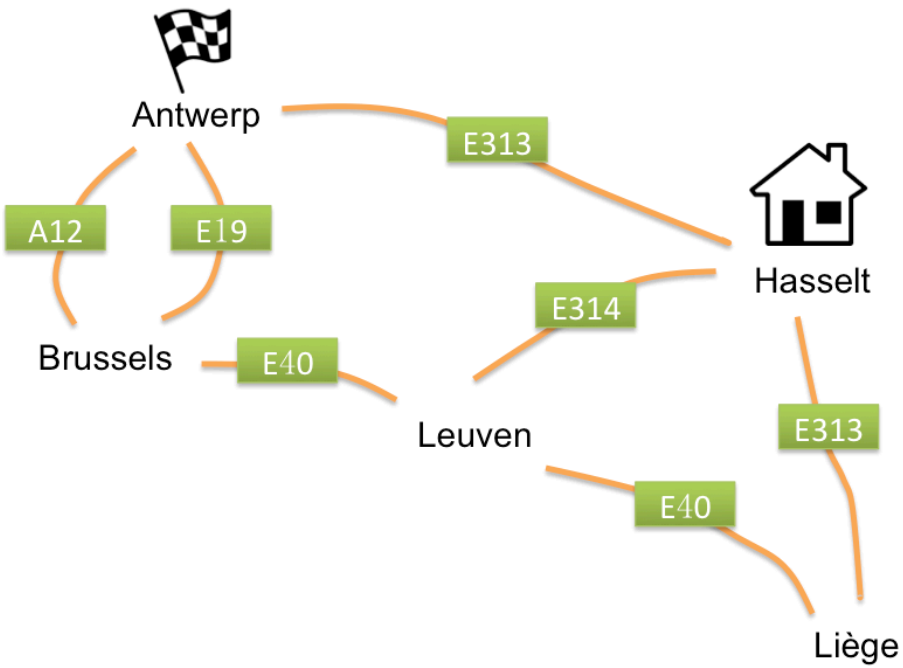
Navigating plan to DBDBD-14



universiteit hasselt
KNOWLEDGE IN ACTION

Modeling in RDF graphs

Map



Triples

(subject, predicate, object)

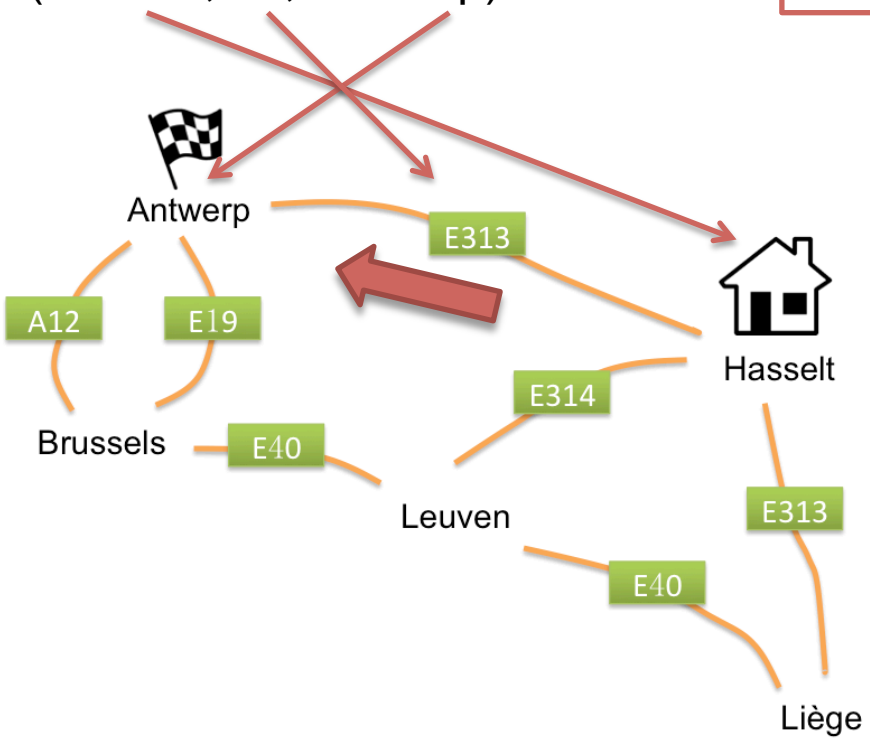
- (Hasselt, E313, Antwerp)
- (Antwerp, E313, Hasselt)
- (Hasselt, E313, Liège)
- (Liège, E313, Hasselt)
- (Hasselt, E314, Leuven)
- (Leuven, E314, Hasselt)
- (Brussels, E19, Antwerp)
- (Antwerp, E19, Brussels)
- (Brussels, A12, Antwerp)
- (Antwerp, A12, Brussels)
- (Leuven, E40, Brussels)
- (Brussels, E40, Leuven)
- (Liège, E40, Leuven)
- (Leuven, E40, Liège)

SPARQL for querying RDF graphs

- Triple patterns: a triple (u, v, w) with variables ?x, ?y, ?z…
- Pattern operators: AND, UNION, OPT, SELECT, FILTER

Which highway connects Hasselt to Antwerp?

Query: (Hasselt, ?n, Antwerp)? *Answer*: ?n = E313.



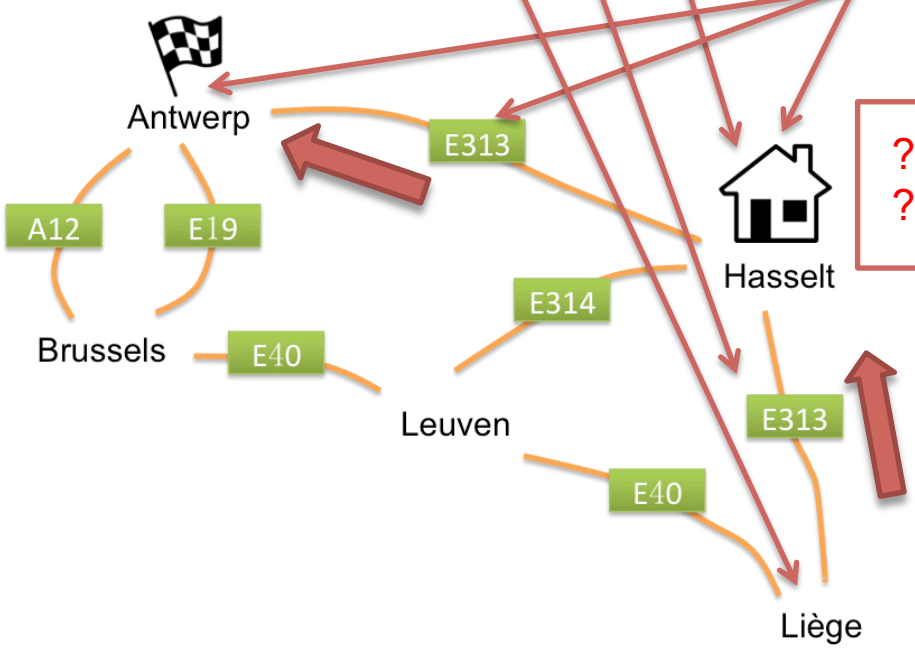
universiteit hasselt KNOWLEDGE IN ACTION

SPARQL for querying RDF graphs

- Triple patterns: a triple (u, v, w) with variables ?x, ?y, ?z…
- Pattern operators: AND, UNION, OPT, SELECT, FILTER

Which city connects to Antwerp through a city?

Query: $\text{SELECT}_{\{?x\}}((?x, ?n, ?y) \text{ AND } (?y, ?m, \text{Antwerp}))$



Answer:

?x =Liège, ?n = E313, ?m =E313, ?y = Hasselt.
?x =Leuven, ?n = E40, ?m =E19, ?y = Brussels.

universiteit hasselt KNOWLEDGE IN ACTION

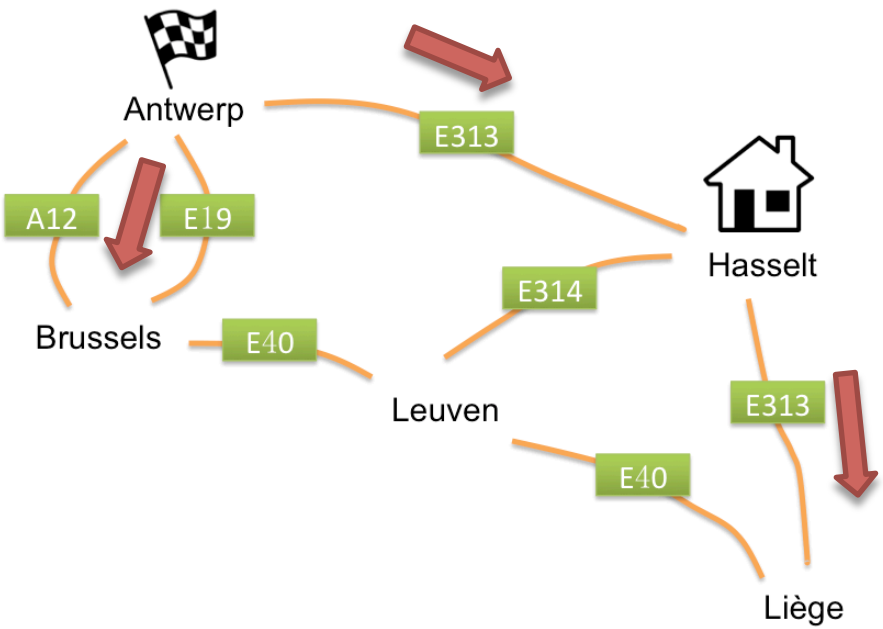
SPARQL for querying RDF graphs

- Triple patterns: a triple (u, v, w) with variables ?x, ?y, ?z…
- Pattern operators: AND, UNION, OPT, SELECT, FILTER

For each highway leaving from Antwerp, give cities reachable by the same highway in two steps. If the highway does not continue, give only the first step.?

Query:

$\text{SELECT}_{\{?x,\ ?y\}}((\text{Antwerp}, ?n, ?x)\ \text{OPT}\ (?x, ?n, ?y))\ \text{FILTER}\ (?y \neq \text{Antwerp})$



Answer:

?x =Brussels, ?n = E19.
?x =Brussels, ?n = A12.
?x =Hasselt, ?n = E313, ?y = Liège.

Outline

- Primitivity of SPARQL operators
- Satisfiability of SPARQL fragments
- Summary

Outline

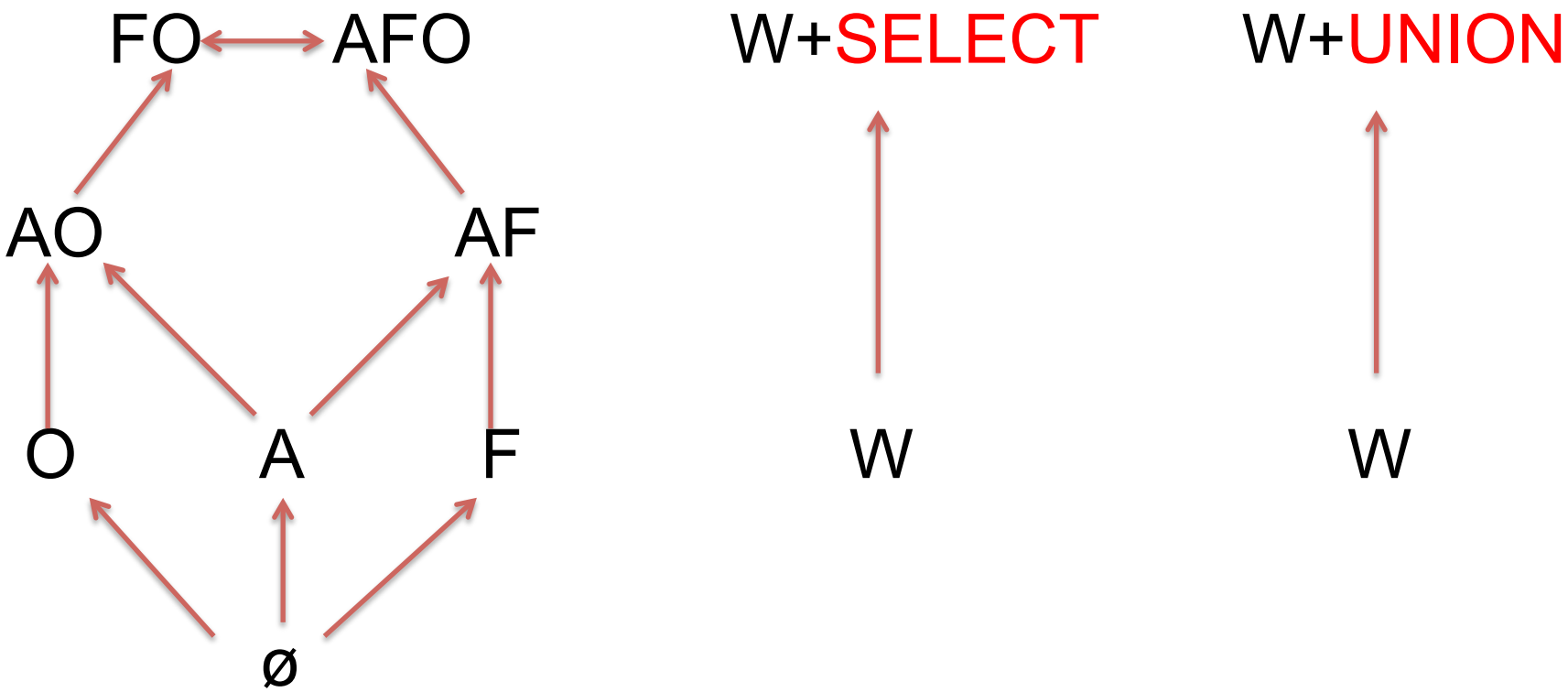
- Primitivity of SPARQL operators
- Satisfiability of SPARQL fragments
- Summary

Primitivity of SPARQL operators

A fragment W1 is *expressible* in a fragment W2 if for any pattern P in W1 there exists some pattern Q in W2 such that P is equivalent to Q.

W1 → W2

Result 1



universiteit hasselt KNOWLEDGE IN ACTION

Primitivity of SPARQL operators

AFO → FO

P AND Q = (P OPT Q) MINUS (P MINUS Q)

P MINUS Q = (P OPT (Q OPT (?x, ?y, ?z)) FILTER ¬bound(?x),

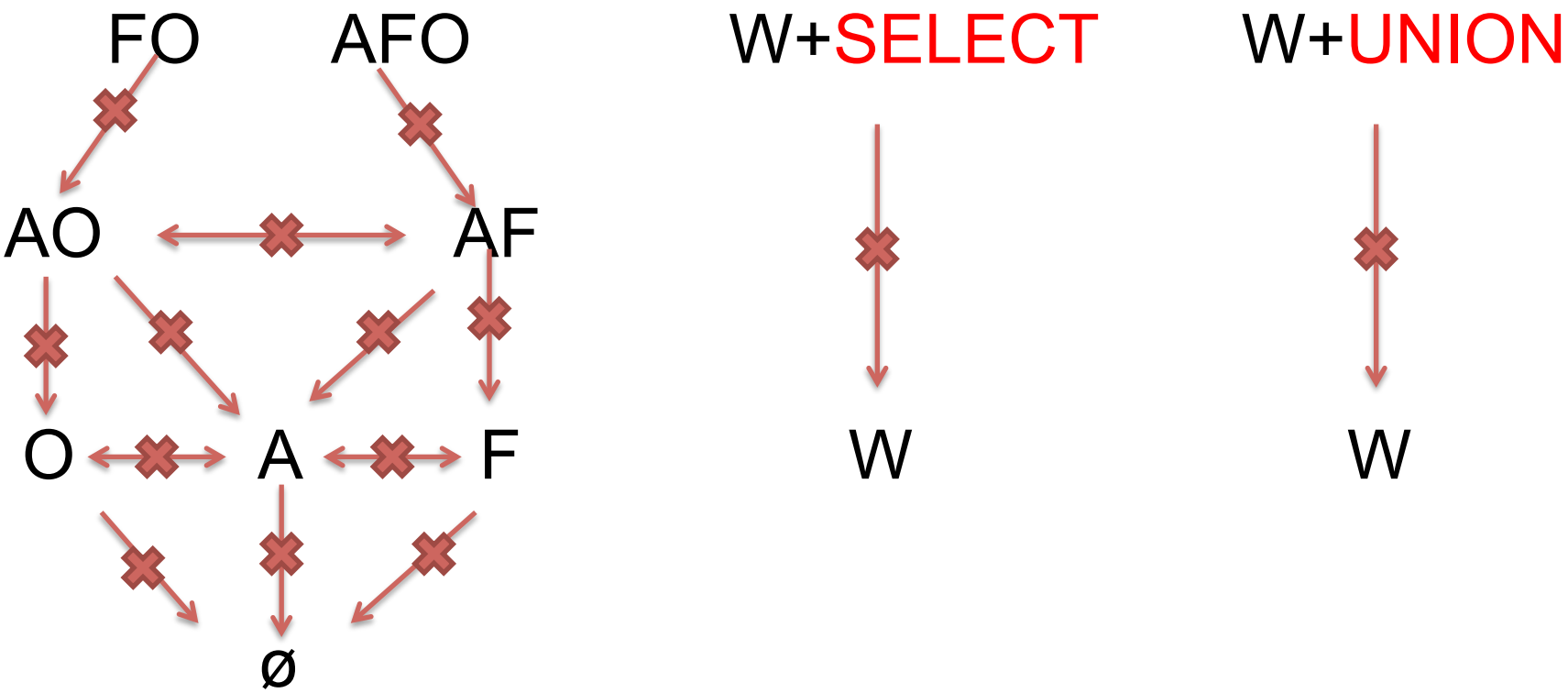
where ?x, ?y, ?z are fresh variables.

R. Angles & C. Gutierrez, 2008

Primitivity of SPARQL operators

A fragment W1 is not *expressible* in a fragment W2 if there exists some pattern P in W1 such that for any pattern Q in W2, P is not equivalent to Q.

Non-trivial results



universiteit hasselt KNOWLEDGE IN ACTION

Primitivity of SPARQL operators

A $\not\longrightarrow$ FSU

Consider the pattern P = (?x, ?y, ?z) AND (?u, ?v, ?w): each mapping contains 6 variables. However, for any pattern Q in FSU, each mapping contains at most 3 variables.

A $\not\longrightarrow$ OSU

Consider the pattern P= (?x, p, ?y) AND (?x, q, ?y): not satisfied in any singleton graph. However, for any satisfiable pattern Q in OSU, there must be a singleton graph satisfying it.

Both OPT and FILTER are necessary to express AND.

Primitivity of SPARQL operators

An operator X is *primitive* if for any X-free fragment W, W+X is not expressible in W.

Result 2

FILTER is primitive.

$$F \not\longrightarrow AOSU$$

Consider the pattern P = (?x, ?y, ?z) FILTER ?x = ?y and the graph G = {a, b} × {a, b} × {a, b}: each mapping satisfies ?x = ?y. However, for any pattern Q in AOSU, it always contains all possible mappings from variables in Q to {a, b}.

Primitivity of SPARQL operators

Result 3

SELECT is primitive.

$$S \not\longrightarrow AOFU$$

Consider the pattern P = $SELECT_{\{?x\}}$ (?x, ?y, ?z). If there exists some pattern Q in AOFU such that P is expressible as Q. Then all variables in all triple patterns of Q are ?x. Consider the graph G = {(a, p, b)}, where a, p, b are fresh constants. P returns a mapping ?x= a, however, Q returns nothing.

Primitivity of SPARQL operators

Result 4

UNION is primitive.

U $\not\longrightarrow$ AOFS

Consider the pattern P= (?x, r, ?y) UNION (?y, r, ?x) and G = {(a, r, b)}, where there are two mappings be to returned.
However, for any pattern Q in AOFS, at most one mapping will be returned.

Primitivity of SPARQL operators

Result 5

Only **AND** is **not** primitive.

(AND, FILTER, OPT, SELECT, UNION)

Primitivity of SPARQL operators

Well-designed patterns

defined by *Perez, Arenas and Gutierrez, 2008*.

$$P_1 \text{ UNION } P_2 \text{ UNION } \ldots \text{ UNION } P_m$$

where each P_i is

- UNION-free
- Safe
- For every sub-pattern Q of the form Q_1 OPT Q_2, all variables occurring in both Q_2 and outside of Q must occur in Q_1.

P AND Q = (P OPT Q) MINUS (P MINUS Q)

P MINUS Q = (P OPT (Q OPT (?x, ?y, ?z)) FILTER ¬bound(?x).

Non-well-designed

Result 6

AND is primitive for well-designed patterns.

Outline

- Primitivity of SPARQL operators
- Satisfiability of SPARQL fragments
- Summary

Satisfiability of SPARQL fragments

A pattern P is satisfiable if there exists some graph G where P returns at least one solution.

The satisfiability problem of the full fragment (AFOSU) is undecidable because the full fragment has the same expressiveness as relational algebra whose satisfiability problem is well known to be undecidable.

Fragments via Constraints	
SPARQL(bound)	bound(?x)
SPARQL(¬bound)	¬bound(?x)
SPARQL($=_c$)	?x = c
SPARQL($\neq_c$)	?x ≠ c
SPARQL(=)	?x = ?y
SPARQL(≠)	?x ≠ ?y

Satisfiability of SPARQL fragments

Result 7

The satisfiability problem of : AFOSU(bound, =, $\neq_c$)
AFOSU(bound, $\neq$, $\neq_c$)
is decidable.

Consistency of constraints

For any set S of constraints of forms:

- bound(?x), ?x =?y, ?x ≠ c)
- bound(?x), ?x ≠ ?y, ?x ≠ c)

there always exists some mapping that satisfies each constraints of S.

Satisfiability of SPARQL fragments

Result 8

The satisfiability problem of :

AFOSU(¬bound)
AFOSU(=, ≠)
AFOSU($=_c$)

is undecidable.

The set difference operator can be expressed in

- AFOSU(¬bound)
- AFOSU(=, ≠)
- AFOSU($=_c$)

The satisfiability problem of the downward algebra (DA: union, composition, and difference) is undecidable. *T. Tan, J. Van den Bussche, X. Zhang 2014*

Satisfiability of SPARQL fragments

Result 5

The satisfiability problem of well-designed patterns is decidable.

- Every UNION-free well-designed pattern can be reduced to a pattern in AF with preserving satisfiability.
- The satisfiabilily problem of AF is decidable.

Outline

- Primitivity of SPARQL operators
- Satisfiability of SPARQL fragments
- Summary

Summary

- Presenting the Hasse diagram of expressivity of all fragments with basic SPARQL operators.
- Investigating the satisfiability problem of of all fragments with basic SPARQL operators.
- Future work: adding SPARQL 1.1 new operators, such as BIND, VALUE, MINUS etc.

Q&A

Thanks

xiaowang.zhang@uhasselt.be

References

- Steve Harris and Andy Seaborne, *SPARQL 1.1 Query Language*, W3C Recommendation, 21 March 2013.
- Frank Manola and Andy Seaborne, *RDF Primer*, W3C Recommendation, 10 February 2004.
- Xiaowang Zhang and Jan Van den Bussche, *On the primitivity of operators in SPARQL*, Information Processing Letters, 114(9):480-485, 2014.
- Xiaowang Zhang and Jan Van den Bussche, *On the satisfiability problem for SPARQL patterns*, arXiv: 1406.1404, 2014.
- Tony Tan, Jan Van den Bussche, and Xiaowang Zhang *Undecidability of satisfiability in the algebra of finite binary relations with union, composition, and difference*, arXiv: 1406.0349, 2014.

SPARQL for querying RDF graphs

- Triple patterns: a triple (u, v, w) with variables ?x, ?y, ?z…
- Pattern operators: AND, UNION, OPT, SELECT, FILTER

Constraints and its three-valued semantics (*true*, *false*, *error*)

- bound(?x)
- ¬ bound(?x)
- ?x = c
- ?x ≠ c
- ?x = ?y
- ?x ≠ ?y
- … ...
- disjunction: ∨
- conjunction: ∧

- μ(bound(?x)) = *true* if ?x occurs in μ; *false* otherwise
- μ(¬ bound(?x)) = *true* if ?x occurs in μ; *false* otherwise
- μ(?x = c) = *error* if ?x does not occur in μ;
- μ(?x = c) = *false* else if μ(?x) = c;
- μ(?x = c) = *true* otherwise.
- μ(?x ≠ c) = *error* if ?x does not occur in μ;
- μ(?x ≠ c) = *true* else if μ(?x) ≠ c;
- μ(?x ≠ c) = *false* otherwise.
- … …

Difference is expressible in three fragments.

Difference ($e_1 - e_2$) is expressible in SPARQL(¬bound)

$$(P_{e_1} \text{ OPT } (P_{e_2} \text{ AND } (?u, r, ?w))) \text{ FILTER } \neg\text{bound}(?u).$$

Difference ($e_1 - e_2$) is expressible in SPARQL (=, ≠)

$$\Big((P_{e_1} \text{ OPT } ((P_{e_2} \text{ AND } adom_{?u} \text{ AND } adom_{?u'}) \text{ FILTER } ?u \neq ?u')) \text{ AND } adom_{?u} \text{ AND } adom_{?u'}\Big) \text{ FILTER } ?u = ?u'.$$

Difference ($e_1 - e_2$) is expressible in SPARQL (≠, $\neq_c$)

$$\Big((P_{e_1} \text{ OPT } ((P_{e_2} \text{ AND } adom_{?u}) \text{ FILTER } ?u = a)) \text{ AND } adom_{?u}\Big) \text{ FILTER } ?u = b.$$