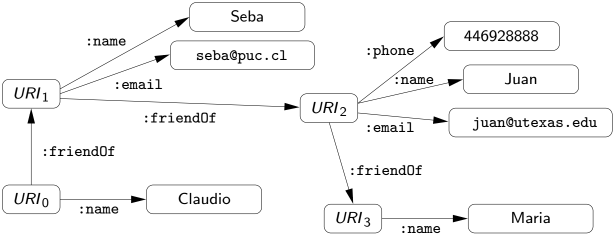
Counting Beyond a Yottabyte, or how SPARQL 1.1 Property Paths will Prevent Adoption of the Standard

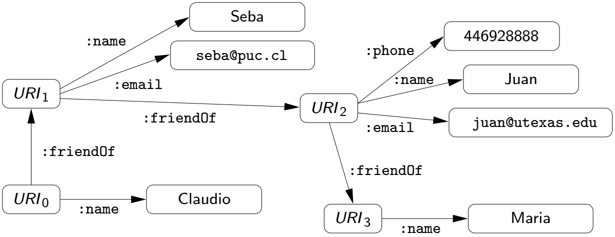
Marcelo Arenas Sebastián Conca Jorge Pérez

PUC-Chile, Universidad de Chile

SPARQL 1.0 provides limited navigational capabilities

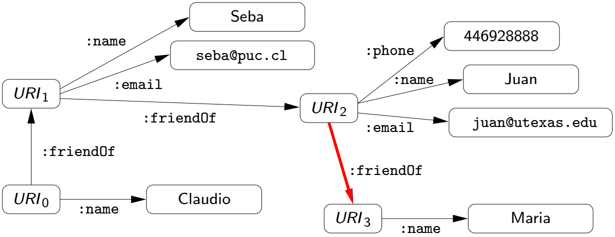


SPARQL 1.0 provides limited navigational capabilities



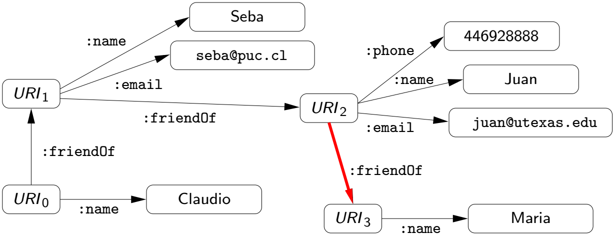

```
SELECT ?X
WHERE
{
   ?X  :friendOf   ?Y .
   ?Y  :name   "Maria" .
}
```

SPARQL 1.0 provides limited navigational capabilities



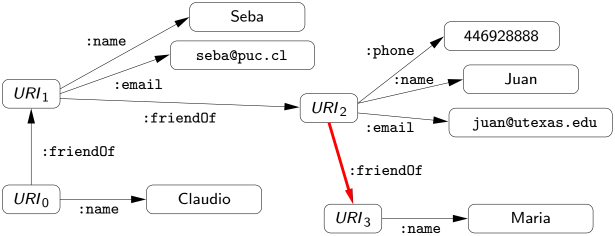

```
SELECT ?X
WHERE
{
   ?X  :friendOf   ?Y .
   ?Y  :name   "Maria" .
}
```

SPARQL 1.0 provides limited navigational capabilities




```
SELECT ?X
WHERE
{
   ?X (:friendOf)* ?Y .
   ?Y  :name   "Maria" .
}
```

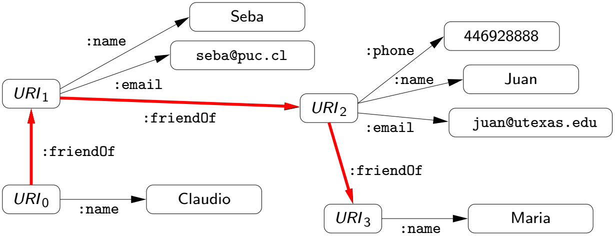
SPARQL 1.0 provides limited navigational capabilities




```
SELECT ?X
WHERE
{
   ?X (:friendOf)* ?Y .
   ?Y  :name   "Maria" .
}
```

← *SPARQL 1.1 property path*

SPARQL 1.0 provides limited navigational capabilities

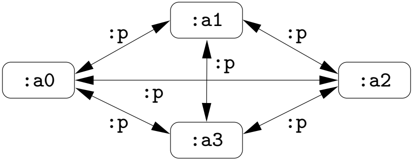



```
SELECT ?X
WHERE
{
   ?X (:friendOf)* ?Y .      ←  SPARQL 1.1 property path
   ?Y  :name    "Maria" .
}
```

SPARQL 1.1 implementations had a poor performance

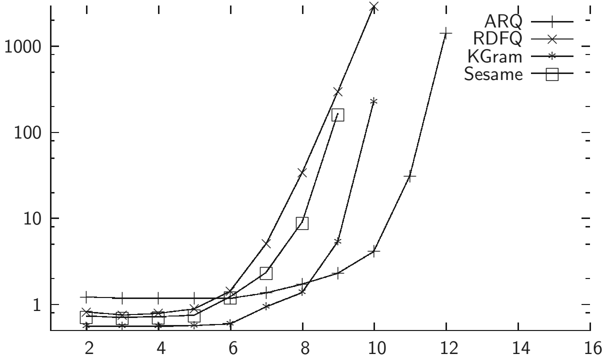
Data:

- *cliques* (complete graphs) of different size
- from 2 nodes (87 bytes) to 13 nodes (970 bytes)



RDF clique with 4 nodes (127 bytes)

SPARQL 1.1 implementations had a poor performance

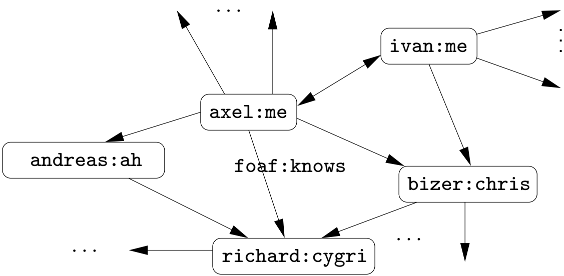


SELECT * WHERE { :a0 (:p)* :a1 }

Poor performance with real Web data of small size

Data:

- Social Network data given by `foaf:knows` links
- Crawled from Axel Polleres' foaf document (3 steps)
- Different documents, deleting some nodes



Poor performance with real Web data of small size

```
SELECT * WHERE { axel:me (foaf:knows)* ?x }
```

Poor performance with real Web data of small size

```
SELECT * WHERE { axel:me (foaf:knows)* ?x }
```

Input	ARQ	RDFQ	Kgram	Sesame
9.2KB	5.13	75.70	313.37	–
10.9KB	8.20	325.83	–	–
11.4KB	65.87	–	–	–
13.2KB	292.43	–	–	–
14.8KB	–	–	–	–
17.2KB	–	–	–	–
20.5KB	–	–	–	–
25.8KB	–	–	–	–

(time in seconds, timeout = 1hr)

Poor performance with real Web data of small size

```
SELECT * WHERE { axel:me (foaf:knows)* ?x }
```

Input	ARQ	RDFQ	Kgram	Sesame
9.2KB	5.13	75.70	313.37	–
10.9KB	8.20	325.83	–	–
11.4KB	65.87	–	–	–
13.2KB	292.43	–	–	–
14.8KB	–	–	–	–
17.2KB	–	–	–	–
20.5KB	–	–	–	–
25.8KB	–	–	–	–

(time in seconds, timeout = 1hr)

Is this a problem of these particular implementations?

We show that this is a problem of the specification

Any implementation that follows SPARQL 1.1 standard
is doomed to show the same behavior

We show that this is a problem of the specification

Any implementation that follows SPARQL 1.1 standard
is doomed to show the same behavior

Technical contributions:

- Experimental study of property paths
- Complete study of the complexity of path evaluation
- Identification of the main sources of complexity *(counting!)*
- Proposal for a semantics with efficient evaluation

We show that this is a problem of the specification

Any implementation that follows SPARQL 1.1 standard
is doomed to show the same behavior

Technical contributions:

- Experimental study of property paths
- Complete study of the complexity of path evaluation
- Identification of the main sources of complexity *(counting!)*
- Proposal for a semantics with efficient evaluation

Impact on W3C standard:

- Normative semantics of SPARQL 1.1 property paths
  will be changed to overcome the issues raised in our paper

Counting Beyond a Yottabyte, or how SPARQL 1.1 Property Paths will Prevent Adoption of the Standard

Marcelo Arenas Sebastián Conca Jorge Pérez

PUC-Chile, Universidad de Chile

Counting Beyond a Yottabyte, or how SPARQL 1.1 Property Paths ~~will Prevent~~ Adoption of the Standard
would have Prevented?

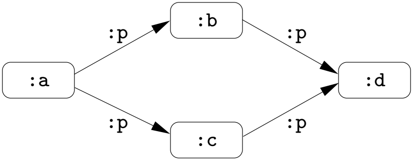
Marcelo Arenas Sebastián Conca Jorge Pérez

PUC-Chile, Universidad de Chile

Outline

Property paths match regular expressions, but also *count*!

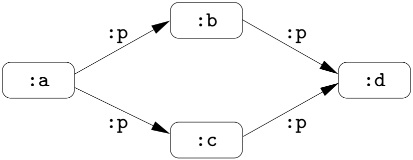
Property paths: regular expressions (/, |, ∗)




```
SELECT ?X
WHERE { :a  (:p)*  ?X }
```

Property paths match regular expressions, but also *count*!

Property paths: regular expressions (/, |, $*$)

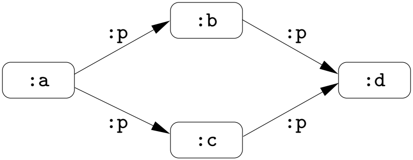



```
SELECT ?X
WHERE { :a  (:p)*  ?X }
```

?*X*
:a
:b
:c
:d
:d

Property paths match regular expressions, but also *count*!

Property paths: regular expressions (/, |, $*$)

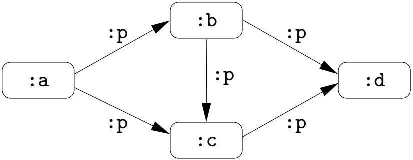



```
SELECT ?X
WHERE { :a  (:p)*  ?X }
```

?*X*
:a
:b
:c
:d
:d

Property paths match regular expressions, but also *count*!

Property paths: regular expressions (/, |, ∗)

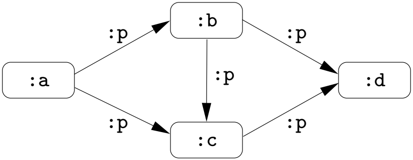



```
SELECT ?X
WHERE { :a  (:p)*  ?X }
```

?X
:a
:b
:c
:d
:d

Property paths match regular expressions, but also *count*!

Property paths: regular expressions (/, |, ∗)

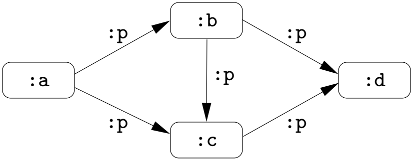



```
SELECT ?X
WHERE { :a  (:p)*  ?X }
```

?*X*
:a
:b
:c
:d
:d
:c
:d

Property paths match regular expressions, but also *count*!

Property paths: regular expressions (/, |, ∗)




```
SELECT ?X
WHERE { :a  (:p)*  ?X }
```

?X
:a
:b
:c
:d
:d
:c
:d

But what if we have cycles?

SPARQL 1.1 document provides a special procedure to handle cycles (and make the count)

Evaluation of *path*$*$

"the algorithm extends the multiset of results by one application of path. If a node has been visited for path, it is not a candidate for another step. A node can be visited multiple times if different paths visit it."

SPARQL 1.1 Last Call (Jan 2012)

SPARQL 1.1 document provides a special procedure to handle cycles (and make the count)

Evaluation of *path*∗

"the algorithm extends the multiset of results by one application of path. If a node has been visited for path, it is not a candidate for another step. A node can be visited multiple times if different paths visit it."

SPARQL 1.1 Last Call (Jan 2012)

- W3C document provides a procedure (`ArbitraryLengthPath`)
- We formalize this procedure in the paper

Counting the number of solutions...

Data: Clique of size n

```
{ :a0 (:p)* :a1 }
```

every solution is a copy of the *empty mapping* (| | in ARQ)

Counting the number of solutions...

Data: Clique of size n

```
{ :a0 (:p)* :a1 }
```

n	# Sol.
9	13,700
10	109,601
11	986,410
12	9,864,101
13	–

every solution is a copy of the *empty mapping* (| | in ARQ)

Counting the number of solutions...

Data: Clique of size n

`{ :a0 (:p)* :a1 }`   `{ :a0 ((:p)*)* :a1 }`

n	# Sol.
9	13,700
10	109,601
11	986,410
12	9,864,101
13	–

every solution is a copy of the *empty mapping* (| | in ARQ)

Counting the number of solutions...

Data: Clique of size n

`{ :a0 (:p)* :a1 }`

n	# Sol.
9	13,700
10	109,601
11	986,410
12	9,864,101
13	–

`{ :a0 ((:p)*)* :a1 }`

n	# Sol
2	1
3	6
4	305
5	418,576
6	–

every solution is a copy of the *empty mapping* (| | in ARQ)

Counting the number of solutions...

Data: Clique of size n

`{ :a0 (:p)* :a1 }` `{ :a0 ((:p)*)* :a1 }` `{:a0 (((:p)*)*)* :a1}`

n	# Sol.
9	13,700
10	109,601
11	986,410
12	9,864,101
13	–

n	# Sol
2	1
3	6
4	305
5	418,576
6	–

every solution is a copy of the *empty mapping* (| | in ARQ)

Counting the number of solutions...

Data: Clique of size n

`{ :a0 (:p)* :a1 }`

n	# Sol.
9	13,700
10	109,601
11	986,410
12	9,864,101
13	–

`{ :a0 ((:p)*)* :a1 }`

n	# Sol
2	1
3	6
4	305
5	418,576
6	–

`{:a0 (((:p)*)*)* :a1}`

n	# Sol.
2	1
3	42
4	–

every solution is a copy of the *empty mapping* (| | in ARQ)

More on counting the number of solutions...

Data: foaf links crawled from the Web

```
{ axel:me (foaf:knows)* ?x }
```

More on counting the number of solutions...

Data: foaf links crawled from the Web

`{ axel:me (foaf:knows)* ?x }`

File	# URIs	# Sol.	Output Size
9.2KB	38	29,817	2MB
10.9KB	43	122,631	8.4MB
11.4KB	47	1,739,331	120MB
13.2KB	52	8,511,943	587MB
14.8KB	54	–	–

More on counting the number of solutions...

Data: foaf links crawled from the Web

```
{ axel:me (foaf:knows)* ?x }
```

File	# URIs	# Sol.	Output Size
9.2KB	38	29,817	2MB
10.9KB	43	122,631	8.4MB
11.4KB	47	1,739,331	120MB
13.2KB	52	8,511,943	587MB
14.8KB	54	–	–

What is really happening here?

More on counting the number of solutions...

Data: foaf links crawled from the Web

```
{ axel:me (foaf:knows)* ?x }
```

File	# URIs	# Sol.	Output Size
9.2KB	38	29,817	2MB
10.9KB	43	122,631	8.4MB
11.4KB	47	1,739,331	120MB
13.2KB	52	8,511,943	587MB
14.8KB	54	–	–

What is really happening here?

Theory can help!

Outline

A bit on complexity classes...

We measure the complexity by using *counting-complexity classes*

NP	**#P**
SAT: is a propositional formula satisfiable?	COUNTSAT: how many assignments satisfy a propositional formula?

A bit on complexity classes...

We measure the complexity by using *counting-complexity classes*

NP

SAT: is a propositional formula satisfiable?

#P

COUNTSAT: how many assignments satisfy a propositional formula?

Formally

A function $f(\cdot)$ on strings is in **#P** if there exists a polynomial-time non-deterministic TM M such that

$$f(x) = \text{number of accepting computations of } M \text{ with input } x$$

A bit on complexity classes...

We measure the complexity by using *counting-complexity classes*

NP	**#P**
SAT: is a propositional formula satisfiable?	COUNTSAT: how many assignments satisfy a propositional formula?

Formally

A function $f(\cdot)$ on strings is in **#P** if there exists a polynomial-time non-deterministic TM M such that

$$f(x) = \text{number of accepting computations of } M \text{ with input } x$$

- COUNTSAT is **#P**-complete

Counting problem for property paths

CountW3C

Input: RDF graph G
Property path triple $\{$ `:a` *path* `:b` $\}$

Output: Count the number of solutions of $\{$ `:a` *path* `:b` $\}$ over G
(according to the semantics proposed by W3C)

The complexity of property paths is *intractable*

Theorem

COUNTW3C *is outside* **#P**

The complexity of property paths is *intractable*

Theorem

COUNTW3C *is outside* **#P**

COUNTW3C is hard to solve even if **P** = **NP**

The complexity of property paths is *intractable*

Theorem

$\textsc{CountW3C}$ *is outside* **#P**

$\textsc{CountW3C}$ is hard to solve even if $\mathbf{P} = \mathbf{NP}$

Proof idea

We provide a *doubly exponential* lower bound for counting

A *doubly exponential* lower bound for counting

- Let $path_s$ be a property path of the form

$$(\cdots((\texttt{:p})*)*)\cdots)*$$

with s nested stars

A *doubly exponential* lower bound for counting

- Let $path_s$ be a property path of the form

$$(\cdots((\texttt{:p})*)*)\cdots)*$$

  with s nested stars
- Let K_n be a clique with n nodes

A *doubly exponential* lower bound for counting

- Let $path_s$ be a property path of the form

$$\texttt{(}\cdots\texttt{((:p)*)*)}\cdots\texttt{)*}$$

  with s nested stars

- Let K_n be a clique with n nodes
- Let $CountClique(s, n)$ be the number of solutions of $\{$ `:a0` $path_s$ `:a1` $\}$ over K_n

A *doubly exponential* lower bound for counting

- Let $path_s$ be a property path of the form

$$\texttt{(}\cdots\texttt{((:p)*)*)}\cdots\texttt{)*}$$

  with s nested stars

- Let K_n be a clique with n nodes
- Let $CountClique(s, n)$ be the number of solutions of $\{\ \texttt{:a0}\ \ path_s\ \ \texttt{:a1}\ \}$ over K_n

Lemma

$$CountClique(s, n) \quad \geq \quad (n-2)!^{(n-1)^{s-1}}$$

A *doubly exponential* lower bound for counting

- Let $path_s$ be a property path of the form

$$\texttt{(}\cdots\texttt{((:p)*)*)}\cdots\texttt{)*}$$

  with s nested stars

- Let K_n be a clique with n nodes
- Let $CountClique(s, n)$ be the number of solutions of $\{$ `:a0` $path_s$ `:a1` $\}$ over K_n

Lemma

$$CountClique(s, n) \;\geq\; (n-2)!^{(n-1)^{s-1}}$$

In the paper:

Recursive formula for calculating $CountClique(s, n)$

We can now explain our experimental results

CountClique(s, n) also allows us to *fill in the blanks*

$$\{\texttt{:a0 ((:p)*)* :a1}\}$$

n	# Sol.
2	1
3	6
4	305
5	418,576
6	–
7	–
8	–

We can now explain our experimental results

CountClique(s, n) also allows us to *fill in the blanks*

{ `:a0 ((:p)*)* :a1` }

n	# Sol.	
2	1	✓
3	6	
4	305	
5	418,576	
6	–	
7	–	
8	–	

We can now explain our experimental results

CountClique(s, n) also allows us to *fill in the blanks*

`{ :a0 ((:p)*)* :a1 }`

n	# Sol.	
2	1	✓
3	6	✓
4	305	
5	418,576	
6	–	
7	–	
8	–	

We can now explain our experimental results

CountClique(s, n) also allows us to *fill in the blanks*

```
{ :a0 ((:p)*)* :a1 }
```

n	# Sol.	
2	1	✓
3	6	✓
4	305	✓
5	418,576	
6	–	
7	–	
8	–	

We can now explain our experimental results

CountClique(s, n) also allows us to *fill in the blanks*

{ `:a0 ((:p)*)* :a1` }

n	# Sol.	
2	1	✓
3	6	✓
4	305	✓
5	418,576	✓
6	–	
7	–	
8	–	

We can now explain our experimental results

$CountClique(s, n)$ also allows us to *fill in the blanks*

$$\{\ \texttt{:a0 ((:p)*)* :a1}\ \}$$

n	# Sol.		
2	1	✓	
3	6	✓	
4	305	✓	
5	418,576	✓	
6	–	←	28×10^9
7	–		
8	–		

We can now explain our experimental results

CountClique(s, n) also allows us to *fill in the blanks*

`{ :a0 ((:p)*)* :a1 }`

n	# Sol.		
2	1	✓	
3	6	✓	
4	305	✓	
5	418,576	✓	
6	–	$\leftarrow$	28×10^9
7	–	$\leftarrow$	144×10^{15}
8	–		

We can now explain our experimental results

CountClique(s, n) also allows us to *fill in the blanks*

`{ :a0 ((:p)*)* :a1 }`

n	# Sol.		
2	1	✓	
3	6	✓	
4	305	✓	
5	418,576	✓	
6	–	←	28×10^{9}
7	–	←	144×10^{15}
8	–	←	79×10^{24}

We can now explain our experimental results

CountClique(s, n) also allows us to *fill in the blanks*

$$\{\texttt{:a0 ((:p)*)* :a1}\}$$

n	# Sol.		
2	1	✓	
3	6	✓	
4	305	✓	
5	418,576	✓	
6	–	$\leftarrow$	28×10^{9}
7	–	$\leftarrow$	144×10^{15}
8	–	$\leftarrow$	79×10^{24}

79 Yottabytes for the answer over a file of 379 bytes

We can now explain our experimental results

CountClique(s, n) also allows us to *fill in the blanks*

{ `:a0 ((:p)*)* :a1` }

n	# Sol.		
2	1	✓	
3	6	✓	
4	305	✓	
5	418,576	✓	
6	–	$\leftarrow$	28×10^9
7	–	$\leftarrow$	144×10^{15}
8	–	$\leftarrow$	79×10^{24}

79 Yottabytes for the answer over a file of 379 bytes

1 Yottabyte $>$ the estimated capacity of all digital storage in the world

Data complexity of property path is still intractable

Common assumption in Databases:

- queries are much smaller than data sources

Data complexity of property path is still intractable

Common assumption in Databases:

- queries are much smaller than data sources

Data complexity

- measure the complexity considering the query fixed

Data complexity of property path is still intractable

Common assumption in Databases:

- queries are much smaller than data sources

Data complexity

- measure the complexity considering the query fixed
- Data complexity of SQL, XPath, SPARQL 1.0, etc. are all polynomial

Data complexity of property path is still intractable

Common assumption in Databases:

- queries are much smaller than data sources

Data complexity

- measure the complexity considering the query fixed
- Data complexity of SQL, XPath, SPARQL 1.0, etc. are all polynomial

Theorem

Data complexity of $\textsc{CountW3C}$ is **#P**-complete

Data complexity of property path is still intractable

Common assumption in Databases:

- queries are much smaller than data sources

Data complexity

- measure the complexity considering the query fixed
- Data complexity of SQL, XPath, SPARQL 1.0, etc. are all polynomial

Theorem

Data complexity of $\textsc{CountW3C}$ is **#P**-complete

Corollary

SPARQL 1.1 query evaluation is intractable in Data Complexity

Data complexity of property path is still intractable

Common assumption in Databases:

- queries are much smaller than data sources

Data complexity

- measure the complexity considering the query fixed
- Data complexity of SQL, XPath, SPARQL 1.0, etc. are all polynomial

Theorem

Data complexity of $\textsc{CountW3C}$ is **#P**-complete

Corollary

SPARQL 1.1 query evaluation is intractable in Data Complexity

In the paper: several other complexity results...

Outline

An existential semantics to the rescue!

Possible solution

Do not count

Just check whether *there exists* a path satisfying the property path expression

An existential semantics to the rescue!

Possible solution

Do not count

Just check whether *there exists* a path satisfying the property path expression

Years of experiences (theory and practice) in:

- Graph Databases
- XML
- SPARQL 1.0 (PSPARQL, Gleen)

+ equivalent regular expressions giving equivalent results

Existential semantics: decision problems

Input: RDF graph G
Property path triple $\{$ `:a` *path* `:b` $\}$

EXISTSPATH

Question: Is there a path from `:a` to `:b` in G satisfying the regular expression *path*?

EXISTSW3C

Question: Is the number of solutions of $\{$ `:a` *path* `:b` $\}$ over G greater than 0 (according to W3C semantics)?

Evaluating existential paths is tractable

Theorem (well-known result)

EXISTSPATH can be solved in $O(|G| \times |path|)$

Evaluating existential paths is tractable

Theorem (well-known result)

ExistsPath can be solved in $O(|G| \times |path|)$

Theorem

ExistsPath and ExistsW3C are *equivalent* decision problems

Evaluating existential paths is tractable

Theorem (well-known result)

$\textsc{ExistsPath}$ can be solved in $O(|G| \times |path|)$

Theorem

$\textsc{ExistsPath}$ and $\textsc{ExistsW3C}$ are *equivalent* decision problems

Corollary

$\textsc{ExistsW3C}$ can be solved in $O(|G| \times |path|)$

So we have possibilities for optimization

So we have possibilities for optimization

Corollary

Property path queries with `SELECT DISTINCT`
can be efficiently evaluated

So we have possibilities for optimization

Corollary

Property path queries with `SELECT DISTINCT`
can be efficiently evaluated

And we can also use `DISTINCT` over general queries

Theorem

`SELECT DISTINCT` SPARQL 1.1 queries are tractable in Data Complexity

SPARQL 1.1 implementations do not take advantage of `SELECT DISTINCT`

```
SELECT DISTINCT * WHERE { axel:me (foaf:knows)* ?x }
```

Input	ARQ	RDFQ	Kgram	Sesame	Psparql	Gleen

SPARQL 1.1 implementations do not take advantage of `SELECT DISTINCT`

```
SELECT DISTINCT * WHERE { axel:me (foaf:knows)* ?x }
```

Input	ARQ	RDFQ	Kgram	Sesame	Psparql	Gleen

SPARQL 1.1 implementations do not take advantage of SELECT DISTINCT

```
SELECT DISTINCT * WHERE { axel:me (foaf:knows)* ?x }
```

Input	ARQ	RDFQ	Kgram	Sesame	Psparql	Gleen
9.2KB	2.24	47.31	2.37	–	0.29	1.39
10.9KB	2.60	204.95	6.43	–	0.30	1.32
11.4KB	6.88	3222.47	80.73	–	0.30	1.34
13.2KB	24.42	–	394.61	–	0.31	1.38
14.8KB	–	–	–	–	0.33	1.38
17.2KB	–	–	–	–	0.35	1.42
20.5KB	–	–	–	–	0.44	1.50
25.8KB	–	–	–	–	0.45	1.52

SPARQL 1.1 implementations do not take advantage of SELECT DISTINCT

```
SELECT DISTINCT * WHERE { axel:me (foaf:knows)* ?x }
```

Input	ARQ	RDFQ	Kgram	Sesame	Psparql	Gleen
9.2KB	2.24	47.31	2.37	–	0.29	1.39
10.9KB	2.60	204.95	6.43	–	0.30	1.32
11.4KB	6.88	3222.47	80.73	–	0.30	1.34
13.2KB	24.42	–	394.61	–	0.31	1.38
14.8KB	–	–	–	–	0.33	1.38
17.2KB	–	–	–	–	0.35	1.42
20.5KB	–	–	–	–	0.44	1.50
25.8KB	–	–	–	–	0.45	1.52

Optimization possibilities can remain hidden
in a complicated semantics

Concluding remarks and a proposal

Counting in general is not feasible, but
W3C has use cases to count (when there are no cycles)

Concluding remarks and a proposal

Counting in general is not feasible, but
W3C has use cases to count (when there are no cycles)

We propose to:

- Have an existential semantics as *default*
- Provide users with a keyword to count paths

Concluding remarks and a proposal

Counting in general is not feasible, but
W3C has use cases to count (when there are no cycles)

We propose to:

- Have an existential semantics as *default*
- Provide users with a keyword to count paths
  even we (authors) haven't reached a consensus on this

"How cool would it be to (reach a consensus and) have a design that meets both use cases!"

TBL yesterday

(about W3C standardization)

Counting Beyond a Yottabyte, or how SPARQL 1.1 Property Paths ~~will Prevent~~ Adoption of the Standard
would have Prevented?

Marcelo Arenas Sebastián Conca Jorge Pérez

PUC-Chile, Universidad de Chile

Outline