RDF and SPARQL: Two basic components of the Semantic Web

Marcelo Arenas

Department of Computer Science
Pontificia Universidad Católica de Chile

Outline

- RDF model
- Querying RDF data
  - Conjunctive queries
  - Entailment of RDF graphs
- Graphs with RDFS vocabulary
  - Inference rules
  - Querying RDFS data: Closure, Core.
- Querying RDF Data in practice: SPARQL
  - Formal semantics for SPARQL
- Complexity of the SPARQL evaluation problem
- A procedural semantics: Well–designed patterns

Semantic Web

"The Semantic Web is an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation."

[Tim Berners-Lee et al. 2001.]

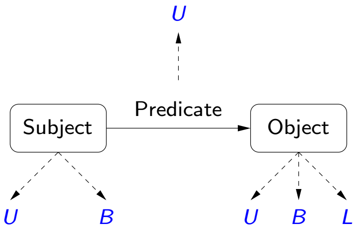
Specific Goals:

- Build a description language with standard semantics.
- Make semantics machine-processable and understandable.
- Incorporate logical infrastructure to reason about resources.
- W3C Proposal: Resource Description Framework (RDF).

RDF in a nutshell

- RDF is the W3C proposal framework for representing information in the Web.
- Abstract syntax based on directed labeled graph.
- Schema definition language (RDFS): Define new vocabulary (typing, inheritance of classes and properties).
- Extensible URI-based vocabulary.
- Support use of XML schema datatypes.
- Formal semantics.

RDF formal model

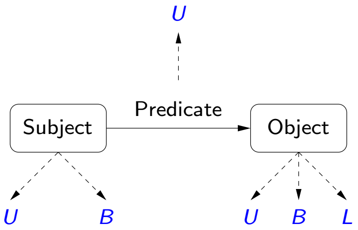


U = set of Uris

B = set of Blank nodes

L = set of Literals

RDF formal model



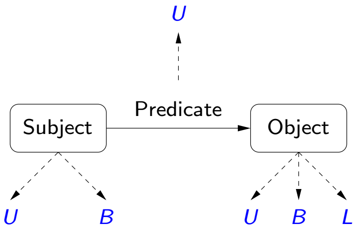
U = set of Uris

B = set of Blank nodes

L = set of Literals

$(s, p, o) \in (U \cup B) \times U \times (U \cup B \cup L)$ is called an RDF triple

RDF formal model



U = set of Uris

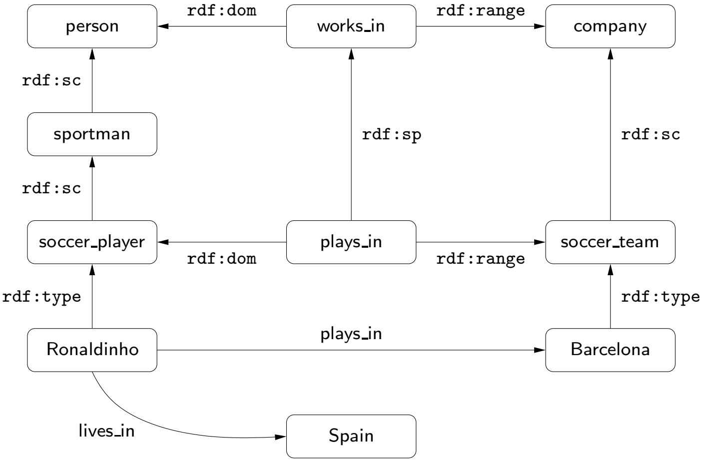
B = set of Blank nodes

L = set of Literals

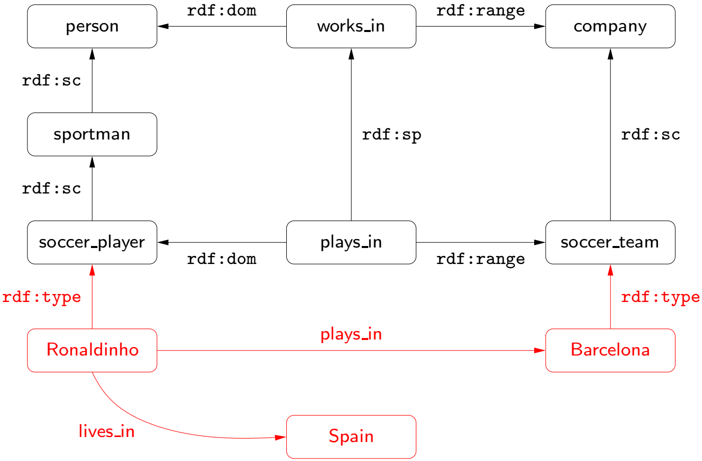
$(s, p, o) \in (U \cup B) \times U \times (U \cup B \cup L)$ is called an RDF triple

A set of RDF triples is called an RDF graph

RDFS: An example



RDFS: An example



RDF model

Some difficulties:

- Existential variables as datavalues
- Built-in vocabulary with fixed semantics (RDFS)
- Graph model where nodes may also be edge labels

RDF data processing can take advantage of database techniques:

- Query processing
- Storing
- Indexing

RDF model

Some difficulties:

- Existential variables as datavalues
- Built-in vocabulary with fixed semantics (RDFS)
- Graph model where nodes may also be edge labels

RDF data processing can take advantage of database techniques:

- Query processing
- Storing
- Indexing

Querying RDF data

Conjunctive query:

$$Q(\overline{X}) \quad = \quad \exists \overline{Y}\, t_1 \wedge t_2 \wedge \cdots \wedge t_k$$

Some examples:

Querying RDF data

Conjunctive query:

$$Q(\overline{X}) \quad = \quad \exists \overline{Y}\, t_1 \wedge t_2 \wedge \cdots \wedge t_k$$

Some examples:

$$(\texttt{Ronaldinho}, \texttt{plays\_in}, \texttt{Barcelona})$$

Querying RDF data

Conjunctive query:

$$Q(\overline{X}) \quad = \quad \exists \overline{Y}\, t_1 \wedge t_2 \wedge \cdots \wedge t_k$$

Some examples:

$$(\texttt{Ronaldinho}, \texttt{plays\_in}, \texttt{Barcelona})$$
$$(\texttt{Ronaldinho}, \texttt{plays\_in}, X)$$

Querying RDF data

Conjunctive query:

$$Q(\overline{X}) \quad = \quad \exists \overline{Y}\, t_1 \wedge t_2 \wedge \cdots \wedge t_k$$

Some examples:

$$\begin{array}{ll} & (\texttt{Ronaldinho}, \texttt{plays\_in}, \texttt{Barcelona}) \\ & (\texttt{Ronaldinho}, \texttt{plays\_in}, X) \\ \exists Y & (X, \texttt{plays\_in}, Y) \wedge (X, \texttt{lives\_in}, \texttt{Spain}) \end{array}$$

Semantics of conjunctive queries

Given an RDF graph G, a conjunctive query $Q(\overline{X})$ and a tuple $\bar{a}$ of values in $U \cup B \cup L$:

Is $\bar{a}$ an answer to $Q(\overline{X})$ in G?

Notation: $G \models Q(\bar{a})$

Notice that $Q(\overline{X})$ and $\bar{a}$ may include blank nodes.

- Blank nodes play a similar role as existential variables.
- $(\texttt{Ronaldinho}, \texttt{plays\_in}, B)$ and
  $\exists X\, (\texttt{Ronaldinho}, \texttt{plays\_in}, X)$ are *equivalent*.

Conjunctive queries and entailment of RDF graphs

$Q(\bar{a})$ can be transformed into an RDF graph G'.

- Notion to define: $G \models G'$

Entailment of RDF graphs:

Conjunctive queries and entailment of RDF graphs

$Q(\bar{a})$ can be transformed into an RDF graph G'.

- Notion to define: $G \models G'$

Entailment of RDF graphs:

- Can be defined in terms of classical notions such model, interpretation, etc

Conjunctive queries and entailment of RDF graphs

$Q(\bar{a})$ can be transformed into an RDF graph G'.

- Notion to define: $G \models G'$

Entailment of RDF graphs:

- Can be defined in terms of classical notions such model, interpretation, etc
  - As for the case of first order logic

Conjunctive queries and entailment of RDF graphs

$Q(\bar{a})$ can be transformed into an RDF graph G'.

- Notion to define: $G \models G'$

Entailment of RDF graphs:

- Can be defined in terms of classical notions such model, interpretation, etc
  - As for the case of first order logic
- Has a graph characterization via homomorphisms.

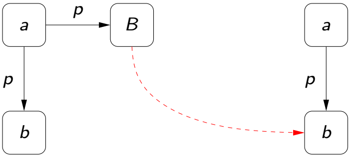
Homomorphism

A function $h : U \cup B \cup L \rightarrow U \cup B \cup L$ is a homomorphism h from G_1 to G_2 if:

- $h(c) = c$ for every $c \in U \cup L$;
- for every $(a, b, c) \in G_1$, $(h(a), h(b), h(c)) \in G_2$

Notation: $G_1 \rightarrow G_2$

Example: $h = \{B \mapsto b\}$



Entailment

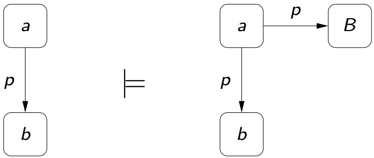
Theorem (CM77)

$G_1 \models G_2$ if and only if there is a homomorphism $G_2 \rightarrow G_1$.

Entailment

Theorem (CM77)

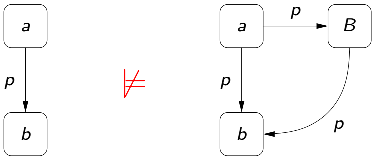
$G_1 \models G_2$ if and only if there is a homomorphism $G_2 \to G_1$.



Entailment

Theorem (CM77)

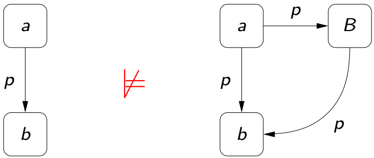
$G_1 \models G_2$ if and only if there is a homomorphism $G_2 \to G_1$.



Entailment

Theorem (CM77)

$G_1 \models G_2$ if and only if there is a homomorphism $G_2 \to G_1$.



Complexity

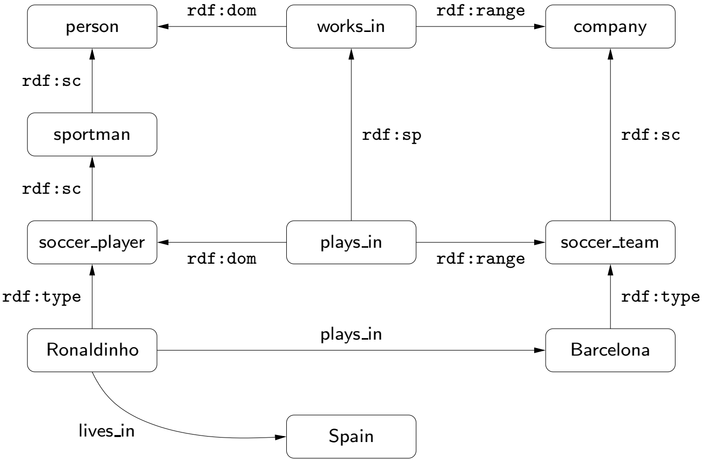
Entailment for RDF is NP-complete

Graphs with RDFS vocabulary

Previous characterization of entailment is not enough to deal with RDFS vocabulary:

Graphs with RDFS vocabulary

Previous characterization of entailment is not enough to deal with RDFS vocabulary: $(\texttt{Ronaldinho}, \texttt{rdf}:\texttt{type}, \texttt{person})$



Graphs with RDFS vocabulary

Built-in predicates have pre-defined semantics:

Graphs with RDFS vocabulary

Built-in predicates have pre-defined semantics:

`rdf:sc`: transitive

Graphs with RDFS vocabulary

Built-in predicates have pre-defined semantics:

- `rdf:sc`: transitive
- `rdf:sp`: transitive

Graphs with RDFS vocabulary

Built-in predicates have pre-defined semantics:

- `rdf:sc`: transitive
- `rdf:sp`: transitive
- More complicated interactions: $\dfrac{(p, \texttt{rdf:dom}, c) \quad (a, p, b)}{(a, \texttt{rdf:type}, c)}$

Graphs with RDFS vocabulary

Built-in predicates have pre-defined semantics:

`rdf:sc`: transitive

`rdf:sp`: transitive

More complicated interactions: $\dfrac{(p, \texttt{rdf:dom}, c) \quad (a, p, b)}{(a, \texttt{rdf:type}, c)}$

RDFS-entailment can be characterized by a set of rules

- An Existential rule
- Subproperty rules
- Subclass rules
- Typing rules
- Implicit typing

Graphs with RDFS vocabulary: Inference rules

Inference system in [MPG07] has 14 rules:

Existential rule :

Subproperty rules :

Subclass rules :

Typing rules :

Implicit typing :

Graphs with RDFS vocabulary: Inference rules

Inference system in [MPG07] has 14 rules:

Existential rule : $\frac{G_1}{G_2}$ if $G_2 \rightarrow G_1$

Subproperty rules :

Subclass rules :

Typing rules :

Implicit typing :

Graphs with RDFS vocabulary: Inference rules

Inference system in [MPG07] has 14 rules:

Existential rule : $\frac{G_1}{G_2}$ if $G_2 \rightarrow G_1$

Subproperty rules : $\frac{(p, \texttt{rdf:sp}, q) \quad (a, p, b)}{(a, q, b)}$

Subclass rules :

Typing rules :

Implicit typing :

Graphs with RDFS vocabulary: Inference rules

Inference system in [MPG07] has 14 rules:

Existential rule : $\frac{G_1}{G_2}$ if $G_2 \rightarrow G_1$

Subproperty rules : $\frac{(p, \texttt{rdf:sp}, q) \quad (a, p, b)}{(a, q, b)}$

Subclass rules : $\frac{(a, \texttt{rdf:sc}, b) \quad (b, \texttt{rdf:sc}, c)}{(a, \texttt{rdf:sc}, c)}$

Typing rules :

Implicit typing :

Graphs with RDFS vocabulary: Inference rules

Inference system in [MPG07] has 14 rules:

Existential rule : $\frac{G_1}{G_2}$ if $G_2 \rightarrow G_1$

Subproperty rules : $\frac{(p, \texttt{rdf:sp}, q) \quad (a, p, b)}{(a, q, b)}$

Subclass rules : $\frac{(a, \texttt{rdf:sc}, b) \quad (b, \texttt{rdf:sc}, c)}{(a, \texttt{rdf:sc}, c)}$

Typing rules : $\frac{(p, \texttt{rdf:dom}, c) \quad (a, p, b)}{(a, \texttt{rdf:type}, c)}$

Implicit typing :

Graphs with RDFS vocabulary: Inference rules

Inference system in [MPG07] has 14 rules:

Existential rule : $\frac{G_1}{G_2}$ if $G_2 \rightarrow G_1$

Subproperty rules : $\frac{(p, \texttt{rdf:sp}, q) \quad (a, p, b)}{(a, q, b)}$

Subclass rules : $\frac{(a, \texttt{rdf:sc}, b) \quad (b, \texttt{rdf:sc}, c)}{(a, \texttt{rdf:sc}, c)}$

Typing rules : $\frac{(p, \texttt{rdf:dom}, c) \quad (a, p, b)}{(a, \texttt{rdf:type}, c)}$

Implicit typing : $\frac{(q, \texttt{rdf:dom}, a) \quad (p, \texttt{rdf:sp}, q) \quad (b, p, c)}{(b, \texttt{rdf:type}, a)}$

Graphs with RDFS vocabulary: Inference rules

Inference system in [MPG07] has 14 rules:

Existential rule :

Subproperty rules : $\dfrac{(p, \texttt{rdf:sp}, q) \quad (a, p, b)}{(a, q, b)}$

Subclass rules :

Typing rules : $\dfrac{(p, \texttt{rdf:dom}, c) \quad (a, p, b)}{(a, \texttt{rdf:type}, c)}$

Implicit typing : $\dfrac{(q, \texttt{rdf:dom}, a) \quad (p, \texttt{rdf:sp}, q) \quad (b, p, c)}{(b, \texttt{rdf:type}, a)}$

Graphs with RDFS vocabulary: Inference rules

Inference system in [MPG07] has 14 rules:

Existential rule :

Subproperty rules : $\dfrac{(p, \texttt{rdf:sp}, q) \quad (a, p, b)}{(a, q, b)}$

Subclass rules :

Typing rules : $\dfrac{(p, \texttt{rdf:dom}, c) \quad (a, p, b)}{(a, \texttt{rdf:type}, c)}$

Implicit typing : $\dfrac{(B, \texttt{rdf:dom}, a) \quad (p, \texttt{rdf:sp}, B) \quad (b, p, c)}{(b, \texttt{rdf:type}, a)}$

RDFS Entailment

Theorem (H03,GHM04,MPG07)

$G_1 \models G_2$ iff there is a proof of G_2 from G_1 using the system of 14 inference rules.

Complexity

RDFS-entailment is NP-complete.

Proof idea

Membership in NP: If $G_1 \models G_2$, then there exists a polynomial-size proof of this fact.

Querying RDFS data

System of inference rules can be used as a mechanism for evaluating queries.

- It is difficult to implement.

Is there any practical mechanism for evaluating queries?

Querying RDFS data

System of inference rules can be used as a mechanism for evaluating queries.

- It is difficult to implement.

Is there any practical mechanism for evaluating queries?

- Making explicit the implicit information.

Closure of an RDF Graph

Notation:

ground(G)	:	Graph obtained by replacing every blank B in G by a constant c_B.
ground^{-1}(G)	:	Graph obtained by replacing every constant c_B in G by B.

Closure of an RDF graph G (denoted by closure(G)):

Closure of an RDF Graph

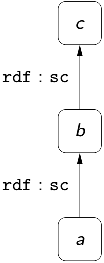
Notation:

ground(G)	:	Graph obtained by replacing every blank B in G by a constant c_B.
ground^{-1}(G)	:	Graph obtained by replacing every constant c_B in G by B.

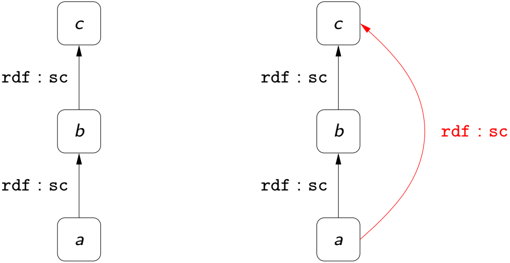
Closure of an RDF graph G (denoted by closure(G)):

$$G \cup \{t \in (U \cup B) \times U \times (U \cup B \cup L) \mid \text{there exists a ground tuple } t' \text{ such that } \text{ground}(G) \models t' \text{ and } t = \text{ground}^{-1}(t')\}$$

Closure of an RDF Graph: Example



Closure of an RDF Graph: Example



Querying RDFS data: Using the closure of a graph

Proposition (H03,GHM04,MPG07)

$G_1 \models G_2$ *iff* $G_2 \rightarrow \mathit{closure}(G_1)$

Complexity

The closure of G can be computed in time $O(|G|^4 \cdot \log |G|)$.

Querying RDFS data: Using the closure of a graph

Proposition (H03,GHM04,MPG07)

$G_1 \models G_2$ *iff* $G_2 \rightarrow \mathit{closure}(G_1)$

Complexity

The closure of G can be computed in time $O(|G|^4 \cdot \log |G|)$.

Can the closure be used in practice?

Querying RDFS data: Using the closure of a graph

Proposition (H03,GHM04,MPG07)

$G_1 \models G_2$ *iff* $G_2 \rightarrow \mathit{closure}(G_1)$

Complexity

The closure of G can be computed in time $O(|G|^4 \cdot \log |G|)$.

Can the closure be used in practice?

- Can we use an alternative materialization?

Querying RDFS data: Using the closure of a graph

Proposition (H03,GHM04,MPG07)

$G_1 \models G_2$ *iff* $G_2 \rightarrow closure(G_1)$

Complexity

The closure of G can be computed in time $O(|G|^4 \cdot \log |G|)$.

Can the closure be used in practice?

- Can we use an alternative materialization?
- Can we materialize a small part of the closure?

Core of an RDF Graph

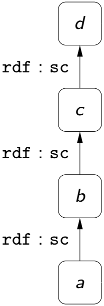
An RDF Graph G is a *core* if there is no homomorphism from G to a proper subgraph of it.

Theorem (HN92,FKP03,GHM04)

- *Each RDF graph G has a unique core (denoted by $core(G)$).*
- *Deciding if G is a core is coNP-complete.*
- *Deciding if $G = core(G')$ is DP-complete.*

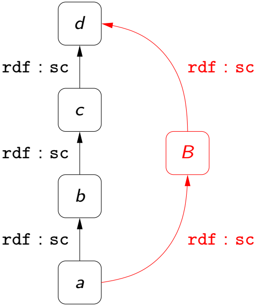
Core and RDFS

For RDF graphs with RDFS vocabulary, the core of G may contain redundant information:



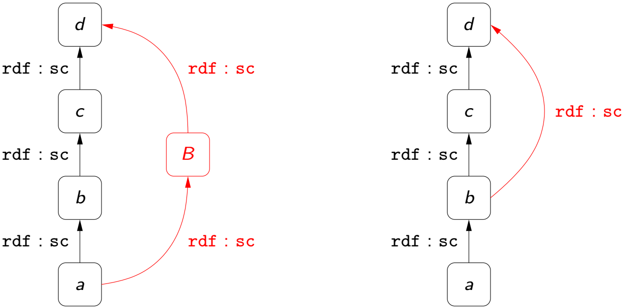
Core and RDFS

For RDF graphs with RDFS vocabulary, the core of G may contain redundant information:



Core and RDFS

For RDF graphs with RDFS vocabulary, the core of G may contain redundant information:



A normal form for RDF graphs

To reduce the size of the materialization, we can combine both core and closure.

A normal form for RDF graphs

To reduce the size of the materialization, we can combine both core and closure.

- $\text{nf}(G) = \text{core}(\text{closure}(G))$

A normal form for RDF graphs

To reduce the size of the materialization, we can combine both core and closure.

- $\text{nf}(G) = \text{core}(\text{closure}(G))$

Theorem (GHM04)

- *G_1 is equivalent to G_2 iff $nf(G_1) \cong nf(G_2)$.*
- $G_1 \models G_2$ *iff* $G_2 \rightarrow nf(G_1)$

A normal form for RDF graphs

To reduce the size of the materialization, we can combine both core and closure.

- $\text{nf}(G) = \text{core}(\text{closure}(G))$

Theorem (GHM04)

- *G_1 is equivalent to G_2 iff $nf(G_1) \cong nf(G_2)$.*
- *$G_1 \models G_2$ iff $G_2 \rightarrow nf(G_1)$*

Complexity

The problem of deciding if $G_1 = nf(G_2)$ is DP-complete.

Querying RDF Data in practice

- SPARQL is the W3C candidate recommendation query language for RDF.
- SPARQL is a graph-matching query language.
- A SPARQL query consists of three parts:
  - Pattern matching: optional, union, nesting, filtering.
  - Solution modifiers: projection, distinct, order, limit, offset.
  - Output part: construction of new triples, ...

A simple RDF query language

```
SELECT ?Name ?Email
WHERE
{
  ?X :name ?Name
  ?X :email ?Email
}
```

A simple RDF query language

```
SELECT ?Name ?Email
WHERE
{
  ?X :name ?Name
  ?X :email ?Email
}
```

A simple RDF query language

```
SELECT ?Name ?Email
WHERE
{
  ?X :name ?Name
  ?X :email ?Email
}
```

A simple RDF query language

```
SELECT ?Name ?Email
WHERE
{
  ?X :name ?Name
  ?X :email ?Email
}
```

A simple RDF query language

```
SELECT ?Name ?Email
WHERE
{
  ?X :name ?Name
  ?X :email ?Email
}
```

In general, in a query we have:

$$H \leftarrow$$

- Head: processing of some variables.

A simple RDF query language

```
SELECT ?Name ?Email
WHERE
{
  ?X :name ?Name
  ?X :email ?Email
}
```

In general, in a query we have:

$$H \leftarrow P$$

- Head: processing of some variables.
- Body: pattern matching expression.

A simple RDF query language

```
SELECT ?Name ?Email
WHERE
{
  ?X :name ?Name
  ?X :email ?Email
}
```

In general, in a query we have:

$$H \leftarrow P$$

- Head: processing of some variables.
- Body: pattern matching expression.

We focus on P.

But things can become more complex ...

Interesting features of pattern matching on graphs

- Grouping
- Optional parts
- Nesting
- Union of patterns
- Filtering
- ...

```
{ P1
  P2 }
```

But things can become more complex ...

Interesting features of pattern matching on graphs

- Grouping
- Optional parts
- Nesting
- Union of patterns
- Filtering
- ...

```
{ { P1
    P2 }


  { P3
    P4 }


}
```

But things can become more complex ...

Interesting features of pattern matching on graphs

- Grouping
- Optional parts
- Nesting
- Union of patterns
- Filtering
- ...

```
{ { P1
    P2
    OPTIONAL { P5 }  }

  { P3
    P4
    OPTIONAL { P7 }  }

}
```

But things can become more complex ...

Interesting features of pattern matching on graphs

- Grouping
- Optional parts
- Nesting
- Union of patterns
- Filtering
- ...

```
{ { P1
    P2
    OPTIONAL { P5 }  }

  { P3
    P4
    OPTIONAL { P7
      OPTIONAL { P8 }  }  }
}
```

But things can become more complex ...

Interesting features of pattern matching on graphs

- Grouping
- Optional parts
- Nesting
- Union of patterns
- Filtering
- ...

```
{ { P1
    P2
    OPTIONAL { P5 }  }

  { P3
    P4
    OPTIONAL { P7
      OPTIONAL { P8 }  }  }
}
UNION
{ P9 }
```

But things can become more complex ...

Interesting features of pattern matching on graphs

- Grouping
- Optional parts
- Nesting
- Union of patterns
- Filtering
- ...

```
{ { P1
    P2
    OPTIONAL { P5 }  }

  { P3
    P4
    OPTIONAL { P7
      OPTIONAL { P8 }  }  }
}
UNION
{ P9
  FILTER ( R ) }
```

But things can become more complex ...

Interesting features of pattern matching on graphs

- Grouping
- Optional parts
- Nesting
- Union of patterns
- Filtering
- ...

```
{ { P1
    P2
    OPTIONAL { P5 }  }

  { P3
    P4
    OPTIONAL { P7
      OPTIONAL { P8 }  }  }
}
UNION
{ P9
  FILTER ( R ) }
```

A formal semantics for SPARQL is needed.

A formal approach would be beneficial for:

- Clarifying corner cases
- Helping in the implementation process
- Providing sound foundations

A formal semantics for SPARQL is needed.

A formal approach would be beneficial for:

- Clarifying corner cases
- Helping in the implementation process
- Providing sound foundations $\leftarrow$ Our primary interest

A formal semantics for SPARQL is needed.

A formal approach would be beneficial for:

- Clarifying corner cases
- Helping in the implementation process
- Providing sound foundations $\leftarrow$ Our primary interest

In our work:

- A formal compositional semantics (for simple RDF)
- Complexity bounds
- Optimization procedures

A standard algebraic syntax

- Triple patterns: just triples + variables, without blanks

original SPARQL syntax	algebraic syntax
`?X :name "john"`	$(?X, \text{name}, \text{john})$

- Graph patterns: full parenthesized algebra

original SPARQL syntax	algebraic syntax
`{ P1 P2 }`	$(P_1 \text{ AND } P_2)$
`{ P1 OPTIONAL { P2 }}`	$(P_1 \text{ OPT } P_2)$
`{ P1 } UNION { P2 }`	$(P_1 \text{ UNION } P_2)$
`{ P1 FILTER ( R ) }`	$(P_1 \text{ FILTER } R)$

original SPARQL syntax algebraic syntax

A standard algebraic syntax

- Explicit precedence/association

Example

```
{ t1
  t2
  OPTIONAL { t3 }
  OPTIONAL { t4 }
  t5
}
```

$$((((\, t_1 \text{ AND } t_2 \,) \text{ OPT } t_3 \,) \text{ OPT } t_4 \,) \text{ AND } t_5 \,)$$

Mappings: building block for the semantics

Definition

A mapping is a partial function from variables to RDF terms.

The evaluation of a pattern results in a set of mappings.

Mappings: building block for the semantics

Definition

A mapping is a partial function from variables to RDF terms.

The evaluation of a pattern results in a set of mappings.

The semantics of triple patterns

Given an RDF graph and a triple pattern t

Definition

The evaluation of t is the set of mappings that

The semantics of triple patterns

Given an RDF graph and a triple pattern t

Definition

The evaluation of t is the set of mappings that

- make t to match the graph

The semantics of triple patterns

Given an RDF graph and a triple pattern t

Definition

The evaluation of t is the set of mappings that

- make t to match the graph
- have as domain the variables in t.

The semantics of triple patterns

Given an RDF graph and a triple pattern t

Definition

The evaluation of t is the set of mappings that

- make t to match the graph
- have as domain the variables in t.

Example

graph	triple	evaluation
$(R_1$, name, john$)$ $(R_1$, email, J@ed.ex$)$ $(R_2$, name, paul$)$	$(?X$, name, $?Y)$	see table below

	$?X$	$?Y$
μ_1:	R_1	john
μ_2:	R_2	paul

The semantics of triple patterns

Given an RDF graph and a triple pattern t

Definition

The evaluation of t is the set of mappings that

- make t to match the graph
- have as domain the variables in t.

Example

graph	triple	evaluation
(R_1, name, john)		
(R_1, email, J@ed.ex)	(?X, name, ?Y)	
(R_2, name, paul)		

	?X	?Y
μ_1:	R_1	john
μ_2:	R_2	paul

The semantics of triple patterns

Given an RDF graph and a triple pattern t

Definition

The evaluation of t is the set of mappings that

- make t to match the graph
- have as domain the variables in t.

Example

graph	triple	evaluation
$(R_1$, name, john$)$		
$(R_1$, email, J@ed.ex$)$	$(?X$, name, $?Y)$	
$(R_2$, name, paul$)$		

	$?X$	$?Y$
μ_1:	R_1	john
μ_2:	R_2	paul

Compatible mappings

Definition

Two mappings are compatible if they agree in their shared variables.

Example

	?X	?Y	?Z	?V
μ_1 :	R_1	john		
μ_2 :	R_1		J@edu.ex	
μ_3 :			P@edu.ex	R_2

Compatible mappings

Definition

Two mappings are compatible if they agree in their shared variables.

Example

	$?X$	$?Y$	$?Z$	$?V$
μ_1 :	R_1	john		
μ_2 :	R_1		J@edu.ex	
μ_3 :			P@edu.ex	R_2

Compatible mappings

Definition

Two mappings are compatible if they agree in their shared variables.

Example

	?X	?Y	?Z	?V
μ_1 :	R_1	john		
μ_2 :	R_1		J@edu.ex	
μ_3 :			P@edu.ex	R_2
$\mu_1 \cup \mu_2$:	R_1	john	J@edu.ex	

Compatible mappings

Definition

Two mappings are compatible if they agree in their shared variables.

Example

	$?X$	$?Y$	$?Z$	$?V$
μ_1 :	R_1	john		
μ_2 :	R_1		J@edu.ex	
μ_3 :			P@edu.ex	R_2
$\mu_1 \cup \mu_2$:	R_1	john	J@edu.ex	

Compatible mappings

Definition

Two mappings are compatible if they agree in their shared variables.

Example

	$?X$	$?Y$	$?Z$	$?V$
μ_1 :	R_1	john		
μ_2 :	R_1		J@edu.ex	
μ_3 :			P@edu.ex	R_2
$\mu_1 \cup \mu_2$:	R_1	john	J@edu.ex	
$\mu_1 \cup \mu_3$:	R_1	john	P@edu.ex	R_2

Compatible mappings

Definition

Two mappings are compatible if they agree in their shared variables.

Example

	$?X$	$?Y$	$?Z$	$?V$
μ_1 :	R_1	john		
μ_2 :	R_1		J@edu.ex	
μ_3 :			P@edu.ex	R_2
$\mu_1 \cup \mu_2$:	R_1	john	J@edu.ex	
$\mu_1 \cup \mu_3$:	R_1	john	P@edu.ex	R_2

- μ_2 and μ_3 are not compatible

Sets of mappings and operations

Let M_1 and M_2 be sets of mappings:

Definition

Sets of mappings and operations

Let M_1 and M_2 be sets of mappings:

Definition

Join: $M_1 \bowtie M_2$

- extending mappings in M_1 with compatible mappings in M_2

Sets of mappings and operations

Let M_1 and M_2 be sets of mappings:

Definition

Join: $M_1 \bowtie M_2$

- extending mappings in M_1 with compatible mappings in M_2

Difference: $M_1 \smallsetminus M_2$

- mappings in M_1 that cannot be extended with mappings in M_2

Sets of mappings and operations

Let M_1 and M_2 be sets of mappings:

Definition

Join: $M_1 \bowtie M_2$

- extending mappings in M_1 with compatible mappings in M_2

Difference: $M_1 \smallsetminus M_2$

- mappings in M_1 that cannot be extended with mappings in M_2

Union: $M_1 \cup M_2$

- mappings in M_1 plus mappings in M_2 (set theoretical union)

Sets of mappings and operations

Let M_1 and M_2 be sets of mappings:

Definition

Join: $M_1 \bowtie M_2$

- extending mappings in M_1 with compatible mappings in M_2

Difference: $M_1 \smallsetminus M_2$

- mappings in M_1 that cannot be extended with mappings in M_2

Union: $M_1 \cup M_2$

- mappings in M_1 plus mappings in M_2 (set theoretical union)

Definition

Left Outer Join: $M_1 ⟕ M_2 = (M_1 \bowtie M_2) \cup (M_1 \smallsetminus M_2)$

Semantics of SPARQL operators

Let M_1 and M_2 be the result of evaluating P_1 and P_2.

Definition

The evaluation of:

$$\begin{array}{rcl}
(P_1 \text{ AND } P_2) & \longrightarrow & \\
(P_1 \text{ UNION } P_2) & \longrightarrow & \\
(P_1 \text{ OPT } P_2) & \longrightarrow &
\end{array}$$

Semantics of SPARQL operators

Let M_1 and M_2 be the result of evaluating P_1 and P_2.

Definition

The evaluation of:

$$\begin{array}{rcl}
(P_1 \text{ AND } P_2) & \longrightarrow & M_1 \bowtie M_2 \\
(P_1 \text{ UNION } P_2) & \longrightarrow & \\
(P_1 \text{ OPT } P_2) & \longrightarrow &
\end{array}$$

Semantics of SPARQL operators

Let M_1 and M_2 be the result of evaluating P_1 and P_2.

Definition

The evaluation of:

$$
\begin{array}{ccc}
(P_1 \text{ AND } P_2) & \longrightarrow & M_1 \bowtie M_2 \\
(P_1 \text{ UNION } P_2) & \longrightarrow & M_1 \cup M_2 \\
(P_1 \text{ OPT } P_2) & \longrightarrow &
\end{array}
$$

Semantics of SPARQL operators

Let M_1 and M_2 be the result of evaluating P_1 and P_2.

Definition

The evaluation of:

$$\begin{array}{ccc} (P_1 \text{ AND } P_2) & \longrightarrow & M_1 \bowtie M_2 \\ (P_1 \text{ UNION } P_2) & \longrightarrow & M_1 \cup M_2 \\ (P_1 \text{ OPT } P_2) & \longrightarrow & M_1 \;⟕\; M_2 \end{array}$$

Simple example

Example

$$(R_1, \text{name}, \text{john})$$
$$(R_1, \text{email}, \text{J@ed.ex})$$
$$(R_2, \text{name}, \text{paul})$$

$$((?X, \text{ name}, ?Y) \text{ OPT } (?X, \text{ email}, ?E))$$

Simple example

Example

$$(R_1, \text{name}, \text{john})$$
$$(R_1, \text{email}, \text{J@ed.ex})$$
$$(R_2, \text{name}, \text{paul})$$

$$(\,(?X,\ \text{name},\ ?Y)\ \text{OPT}\ (?X,\ \text{email},\ ?E)\,)$$

Simple example

Example

$$(R_1,\ \text{name},\ \text{john})$$
$$(R_1,\ \text{email},\ \text{J@ed.ex})$$
$$(R_2,\ \text{name},\ \text{paul})$$

$$(\,(?X,\ \text{name},\ ?Y)\ \text{OPT}\ (?X,\ \text{email},\ ?E)\,)$$

$?X$	$?Y$
R_1	john
R_2	paul

Simple example

Example

$$(R_1,\ \text{name},\ \text{john})$$
$$(R_1,\ \text{email},\ \text{J@ed.ex})$$
$$(R_2,\ \text{name},\ \text{paul})$$

$$(\,(?X,\ \text{name},\ ?Y)\ \text{OPT}\ (?X,\ \text{email},\ ?E)\,)$$

$?X$	$?Y$
R_1	john
R_2	paul

Simple example

Example

$(R_1,\ \text{name},\ \text{john})$
$(R_1,\ \text{email},\ \text{J@ed.ex})$
$(R_2,\ \text{name},\ \text{paul})$

$$((?X,\ \text{name},\ ?Y)\ \text{OPT}\ (?X,\ \text{email},\ ?E))$$

$?X$	$?Y$
R_1	john
R_2	paul

$?X$	$?E$
R_1	J@ed.ex

Simple example

Example

$(R_1, \text{name}, \text{john})$
$(R_1, \text{email}, \text{J@ed.ex})$
$(R_2, \text{name}, \text{paul})$

$$((?X,\ \text{name},\ ?Y)\ \textcolor{red}{\text{OPT}}\ (?X,\ \text{email},\ ?E))$$

$?X$	$?Y$
R_1	john
R_2	paul

$?X$	$?E$
R_1	J@ed.ex

Simple example

Example

$$(R_1,\ \text{name},\ \text{john})$$
$$(R_1,\ \text{email},\ \text{J@ed.ex})$$
$$(R_2,\ \text{name},\ \text{paul})$$

$$(\,(?X,\ \text{name},\ ?Y)\ \text{OPT}\ (?X,\ \text{email},\ ?E)\,)$$

$?X$	$?Y$
R_1	john
R_2	paul

$?X$	$?Y$	$?E$
R_1	john	J@ed.ex
R_2	paul	

$?X$	$?E$
R_1	J@ed.ex

Simple example

Example

$(R_1, \text{name}, \text{john})$
$(R_1, \text{email}, \text{J@ed.ex})$
$(R_2, \text{name}, \text{paul})$

$$((?X,\ \text{name},\ ?Y)\ \text{OPT}\ (?X,\ \text{email},\ ?E))$$

$?X$	$?Y$
R_1	john
R_2	paul

$?X$	$?Y$	$?E$
R_1	john	J@ed.ex
R_2	paul	

$?X$	$?E$
R_1	J@ed.ex

- from the Join

Simple example

Example

$(R_1, \text{name}, \text{john})$
$(R_1, \text{email}, \text{J@ed.ex})$
$(R_2, \text{name}, \text{paul})$

$$((?X,\ \text{name},\ ?Y)\ \text{OPT}\ (?X,\ \text{email},\ ?E))$$

$?X$	$?Y$
R_1	john
R_2	paul

$?X$	$?Y$	$?E$
R_1	john	J@ed.ex
R_2	paul	

$?X$	$?E$
R_1	J@ed.ex

- from the Difference

Simple example

Example

$(R_1, \text{name}, \text{john})$
$(R_1, \text{email}, \text{J@ed.ex})$
$(R_2, \text{name}, \text{paul})$

$$((?X,\ \text{name},\ ?Y)\ \text{OPT}\ (?X,\ \text{email},\ ?E))$$

$?X$	$?Y$
R_1	john
R_2	paul

$?X$	$?Y$	$?E$
R_1	john	J@ed.ex
R_2	paul	

$?X$	$?E$
R_1	J@ed.ex

- from the Union

Boolean filter expressions (value constraints)

In filter expressions we consider

- equality $=$ among variables and RDF terms
- unary predicate bound
- boolean combinations ($\wedge$, $\vee$, $\neg$)

Satisfaction of value constraints

A mapping satisfies

- $?X = c$ if it gives the value c to variable $?X$
- $?X = ?Y$ if it gives the same value to $?X$ and $?Y$
- bound($?X$) if it is defined for $?X$

Definition

Evaluation of (P FILTER R): Set of mappings in the evaluation of P that satisfy R.

Natural algebraic properties: A simple normal from

- AND and UNION are commutative and associative.
- AND, OPT, and FILTER distribute over UNION.

Theorem (UNION Normal Form)

Every graph pattern is equivalent to one of the form

$$P_1 \ \text{UNION} \ P_2 \ \text{UNION} \ \cdots \ \text{UNION} \ P_n$$

where each P_i is UNION–free.

The evaluation problem

Input:

A mapping, a graph pattern, and an RDF graph.

Question:

Is the mapping in the evaluation of the pattern against the graph?

Evaluation of simple patterns is polynomial.

Theorem (PAG06)

For patterns using only AND *and* FILTER *operators, the evaluation problem is polynomial:*

$$O(\textit{size of the pattern} \times \textit{size of the graph}).$$

Evaluation of simple patterns is polynomial.

Theorem (PAG06)

For patterns using only AND *and* FILTER *operators, the evaluation problem is polynomial:*

$$O(\textit{size of the pattern} \times \textit{size of the graph}).$$

Proof idea

- *Check that the mapping makes every triple to match.*
- *Then check that the mapping satisfies the* FILTERs*.*

Evaluation including UNION is NP-complete.

Theorem (PAG06)

For patterns using only AND, FILTER *and* UNION *operators, the evaluation problem is NP-complete.*

Evaluation including UNION is NP-complete.

Theorem (PAG06)

For patterns using only AND, FILTER *and* UNION *operators, the evaluation problem is NP-complete.*

Proof idea

- *Reduction from 3SAT.*
- *A pattern encodes the propositional formula.*
- $\neg$ bound *is used to encode negation.*

Evaluation including UNION is NP-complete.

Theorem (PAG06)

For patterns using only AND, FILTER *and* UNION *operators, the evaluation problem is NP-complete.*

Proof idea

- *Reduction from 3SAT.*
- *A pattern encodes the propositional formula.*
- $\neg$ bound *is used to encode negation.*

In general: Evaluation problem is PSPACE-complete.

Theorem (PAG06)

For general patterns that include OPT *operator, the evaluation problem is PSPACE-complete.*

In general: Evaluation problem is PSPACE-complete.

Theorem (PAG06)

For general patterns that include OPT *operator, the evaluation problem is PSPACE-complete.*

Proof idea

- *Reduction from QBF*
- *A pattern encodes a quantified propositional formula:*

$$\forall x_1 \exists y_1 \forall x_2 \exists y_2 \cdots \psi.$$

- *nested* OPT*s are used to encode quantifier alternation.*

  (This time, we do not need $\neg$ *bound.)*

In general: Evaluation problem is PSPACE-complete.

Theorem (PAG06)

For general patterns that include OPT *operator, the evaluation problem is PSPACE-complete.*

Proof idea

- *Reduction from QBF*
- *A pattern encodes a quantified propositional formula:*

$$\forall x_1 \exists y_1 \forall x_2 \exists y_2 \cdots \psi.$$

- *nested* OPT*s are used to encode quantifier alternation.*

  (This time, we do not need $\neg$ bound.*)*

PSPACE-hardness: A closer look

Assume $\varphi = \forall x_1 \exists y_1\, \psi$, where $\psi = (x_1 \vee \neg y_1) \wedge (\neg x_1 \vee y_1)$.

We generate G, P_φ and μ_0 such that μ_0 belongs to the answer of P_φ over G iff φ is valid:

G :

P_ψ :

P_φ :

μ_0 :

PSPACE-hardness: A closer look

Assume $\varphi = \forall x_1 \exists y_1 \, \psi$, where $\psi = (x_1 \vee \neg y_1) \wedge (\neg x_1 \vee y_1)$.

We generate G, P_φ and μ_0 such that μ_0 belongs to the answer of P_φ over G iff φ is valid:

G : $\{(a, \texttt{tv}, 0),\ (a, \texttt{tv}, 1),\ (a, \texttt{false}, 0),\ (a, \texttt{true}, 1)\}$

P_ψ :

P_φ :

μ_0 :

PSPACE-hardness: A closer look

Assume $\varphi = \forall x_1 \exists y_1\, \psi$, where $\psi = (x_1 \vee \neg y_1) \wedge (\neg x_1 \vee y_1)$.

We generate G, P_φ and μ_0 such that μ_0 belongs to the answer of P_φ over G iff φ is valid:

G : $\{(a, \texttt{tv}, 0),\ (a, \texttt{tv}, 1),\ (a, \texttt{false}, 0),\ (a, \texttt{true}, 1)\}$

P_ψ : $((a, \texttt{true}, ?X_1)$ UNION $(a, \texttt{false}, ?Y_1))$ AND $((a, \texttt{false}, ?X_1)$ UNION $(a, \texttt{true}, ?Y_1))$

P_φ :

μ_0 :

PSPACE-hardness: A closer look

Assume $\varphi = \forall x_1 \exists y_1\, \psi$, where $\psi = (x_1 \vee \neg y_1) \wedge (\neg x_1 \vee y_1)$.

We generate G, P_φ and μ_0 such that μ_0 belongs to the answer of P_φ over G iff φ is valid:

$$
\begin{array}{lcl}
G & : & \{(a, \texttt{tv}, 0),\ (a, \texttt{tv}, 1),\ (a, \texttt{false}, 0),\ (a, \texttt{true}, 1)\} \\
 & & \\
P_\psi & : & ((a, \texttt{true}, ?X_1)\ \text{UNION}\ (a, \texttt{false}, ?Y_1))\ \text{AND} \\
 & & ((a, \texttt{false}, ?X_1)\ \text{UNION}\ (a, \texttt{true}, ?Y_1)) \\
P_\varphi & : & (a, \texttt{true}, ?B_0)\ \text{OPT}\ (P_1\ \text{OPT}\ (Q_1\ \text{AND}\ P_\psi)) \\
 & & \\
\mu_0 & : &
\end{array}
$$

PSPACE-hardness: A closer look

Assume $\varphi = \forall x_1 \exists y_1\, \psi$, where $\psi = (x_1 \vee \neg y_1) \wedge (\neg x_1 \vee y_1)$.

We generate G, P_φ and μ_0 such that μ_0 belongs to the answer of P_φ over G iff φ is valid:

$$
\begin{array}{lcl}
G & : & \{(a, \texttt{tv}, 0),\ (a, \texttt{tv}, 1),\ (a, \texttt{false}, 0),\ (a, \texttt{true}, 1)\} \\[2ex]
P_\psi & : & ((a, \texttt{true}, ?X_1)\ \text{UNION}\ (a, \texttt{false}, ?Y_1))\ \text{AND} \\
 & & ((a, \texttt{false}, ?X_1)\ \text{UNION}\ (a, \texttt{true}, ?Y_1)) \\
P_\varphi & : & (a, \texttt{true}, ?B_0)\ \text{OPT}\ (P_1\ \text{OPT}\ (Q_1\ \text{AND}\ P_\psi)) \\[2ex]
\mu_0 & : & \{?B_0 \mapsto 1\}
\end{array}
$$

PSPACE-hardness: A closer look

$$
\begin{array}{lcl}
P_\varphi & : & (a, \texttt{true}, ?B_0) \text{ OPT } (P_1 \text{ OPT } (Q_1 \text{ AND } P_\psi)) \\
P_1 & : & (a, \texttt{tv}, ?X_1) \\
Q_1 & : & (a, \texttt{tv}, ?X_1) \text{ AND } (a, \texttt{tv}, ?Y_1) \text{ AND } (a, \texttt{false}, ?B_0)
\end{array}
$$

PSPACE-hardness: A closer look

$$
\begin{array}{lcl}
P_\varphi & : & (a, \texttt{true}, ?B_0) \text{ OPT } (P_1 \text{ OPT } (Q_1 \text{ AND } P_\psi)) \\
P_1 & : & (a, \texttt{tv}, ?X_1) \\
Q_1 & : & (a, \texttt{tv}, ?X_1) \text{ AND } (a, \texttt{tv}, ?Y_1) \text{ AND } (a, \texttt{false}, ?B_0)
\end{array}
$$

$?B_0 \mapsto 1$

PSPACE-hardness: A closer look

$$
\begin{array}{lcl}
P_\varphi & : & (a, \texttt{true}, ?B_0) \ \text{OPT} \ (P_1 \ \text{OPT} \ (Q_1 \ \text{AND} \ P_\psi)) \\
P_1 & : & (a, \texttt{tv}, ?X_1) \\
Q_1 & : & (a, \texttt{tv}, ?X_1) \ \text{AND} \ (a, \texttt{tv}, ?Y_1) \ \text{AND} \ (a, \texttt{false}, ?B_0)
\end{array}
$$

P_1

$?X_1 \mapsto 0$

$?B_0 \mapsto 1$

PSPACE-hardness: A closer look

$$
\begin{array}{lcl}
P_\varphi & : & (a, \texttt{true}, ?B_0) \text{ OPT } (P_1 \text{ OPT } (Q_1 \text{ AND } P_\psi)) \\
P_1 & : & (a, \texttt{tv}, ?X_1) \\
Q_1 & : & (a, \texttt{tv}, ?X_1) \text{ AND } (a, \texttt{tv}, ?Y_1) \text{ AND } (a, \texttt{false}, ?B_0)
\end{array}
$$

P_1 $\qquad\qquad$ Q_1

$?X_1 \mapsto 0 \longrightarrow ?X_1 \mapsto 0 \quad ?Y_1 \mapsto i \quad ?B_0 \mapsto 0$

$?B_0 \mapsto 1 \nearrow$

PSPACE-hardness: A closer look

$$
\begin{array}{lcl}
P_\varphi & : & (a, \texttt{true}, ?B_0) \text{ OPT } (P_1 \text{ OPT } (Q_1 \text{ AND } P_\psi)) \\
P_1 & : & (a, \texttt{tv}, ?X_1) \\
Q_1 & : & (a, \texttt{tv}, ?X_1) \text{ AND } (a, \texttt{tv}, ?Y_1) \text{ AND } (a, \texttt{false}, ?B_0)
\end{array}
$$

$$
\begin{tikzcd}
 & P_1 & Q_1 \\
 & ?X_1 \mapsto 0 \arrow[r] & ?X_1 \mapsto 0 \quad ?Y_1 \mapsto i \quad ?B_0 \mapsto 0 \\
?B_0 \mapsto 1 \arrow[ur] \arrow[dr] & & \\
 & \color{red}{?X_1 \mapsto 1} &
\end{tikzcd}
$$

PSPACE-hardness: A closer look

$$
\begin{array}{lcl}
P_\varphi & : & (a, \texttt{true}, ?B_0) \text{ OPT } (P_1 \text{ OPT } (Q_1 \text{ AND } P_\psi)) \\
P_1 & : & (a, \texttt{tv}, ?X_1) \\
Q_1 & : & (a, \texttt{tv}, ?X_1) \text{ AND } (a, \texttt{tv}, ?Y_1) \text{ AND } (a, \texttt{false}, ?B_0)
\end{array}
$$

$$
\begin{tikzcd}
 & P_1 & Q_1 \\
 & ?X_1 \mapsto 0 \arrow[r] & ?X_1 \mapsto 0 \quad ?Y_1 \mapsto i \quad ?B_0 \mapsto 0 \\
?B_0 \mapsto 1 \arrow[ur] \arrow[dr] & & \\
 & \color{red}{?X_1 \mapsto 1} \arrow[r] & \color{red}{?X_1 \mapsto 1 \quad ?Y_1 \mapsto j \quad ?B_0 \mapsto 0}
\end{tikzcd}
$$

Data–complexity is polynomial

Theorem (PAG06)

When patterns are consider to be fixed (data complexity), the evaluation problem is in LOGSPACE.

Data–complexity is polynomial

Theorem (PAG06)

When patterns are consider to be fixed (data complexity), the evaluation problem is in LOGSPACE.

Proof idea

From data–complexity of first–order logic.

A procedural semantics

Suggestion of the W3C to evaluate query $A \,\text{OPT}\,(B \,\text{OPT}\, C)$:

> First compute the mappings that match A, then check which of these mappings match B, and for those who match B check whether they also match C.

A procedural semantics

Suggestion of the W3C to evaluate query $A\ \mathrm{OPT}(B\ \mathrm{OPT}\ C)$:

> First compute the mappings that match A, then check which of these mappings match B, and for those who match B check whether they also match C.

Depth–first traversal of queries parse trees.

A procedural semantics

Suggestion of the W3C to evaluate query $A \, \mathrm{OPT} (B \, \mathrm{OPT} \, C)$:

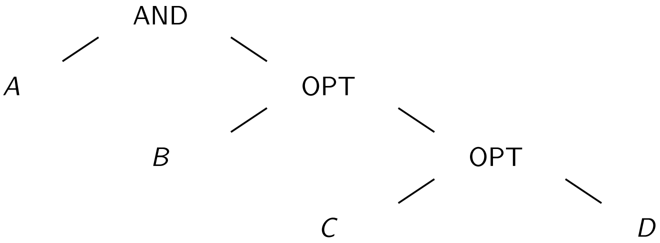
> First compute the mappings that match A, then check which of these mappings match B, and for those who match B check whether they also match C.

Depth–first traversal of queries parse trees.

- As opposed to the bottom-up evaluation induced by the compositional semantics.

A procedural semantics

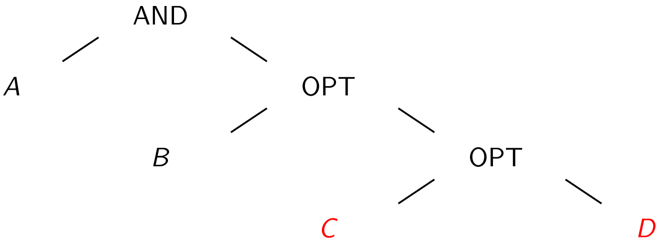
Consider: $(A \text{ AND } (B \text{ OPT } (C \text{ OPT } D)))$



- Algebraic semantics: induces the usual bottom–up evaluation.
- Alternative semantics: depth–first traversal of the parse tree.
  - Similar to the procedural semantics of Jena/ARQ
  - Navigational semantics of nested OPTs in official SPARQL (April 2006)
- These two evaluation algorithms do not always coincide.

A procedural semantics

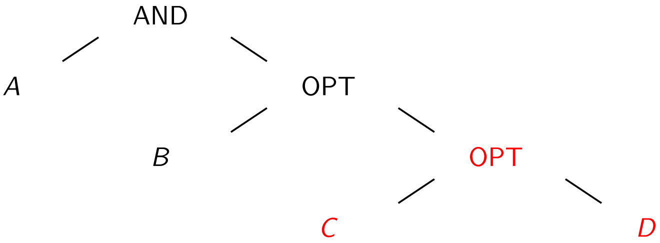
Consider: $(A \text{ AND } (B \text{ OPT } (C \text{ OPT } D)))$



- Algebraic semantics: induces the usual bottom–up evaluation.
- Alternative semantics: depth–first traversal of the parse tree.
  - Similar to the procedural semantics of Jena/ARQ
  - Navigational semantics of nested OPTs in official SPARQL (April 2006)
- These two evaluation algorithms do not always coincide.

A procedural semantics

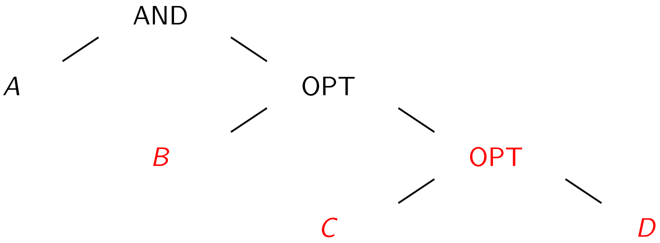
Consider: $(A \text{ AND } (B \text{ OPT } (C \text{ OPT } D)))$



- Algebraic semantics: induces the usual bottom–up evaluation.
- Alternative semantics: depth–first traversal of the parse tree.
  - Similar to the procedural semantics of Jena/ARQ
  - Navigational semantics of nested OPTs in official SPARQL (April 2006)
- These two evaluation algorithms do not always coincide.

A procedural semantics

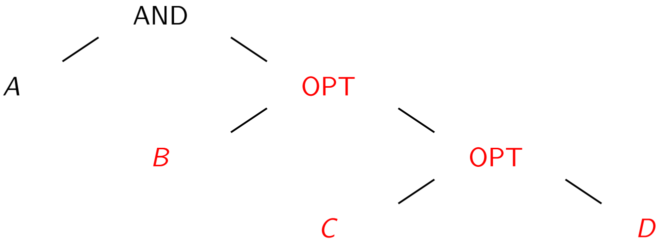
Consider: $(A \text{ AND } (B \text{ OPT } (C \text{ OPT } D)))$



- Algebraic semantics: induces the usual bottom–up evaluation.
- Alternative semantics: depth–first traversal of the parse tree.
  - Similar to the procedural semantics of Jena/ARQ
  - Navigational semantics of nested OPTs in official SPARQL (April 2006)
- These two evaluation algorithms do not always coincide.

A procedural semantics

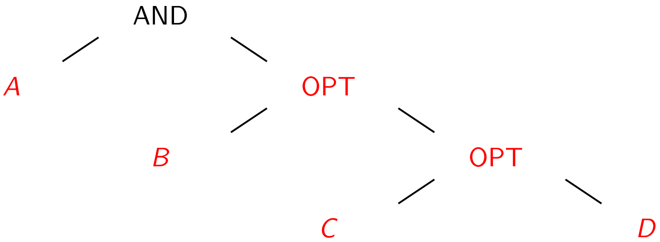
Consider: $(A \text{ AND } (B \text{ OPT } (C \text{ OPT } D)))$



- Algebraic semantics: induces the usual bottom–up evaluation.
- Alternative semantics: depth–first traversal of the parse tree.
  - Similar to the procedural semantics of Jena/ARQ
  - Navigational semantics of nested OPTs in official SPARQL (April 2006)
- These two evaluation algorithms do not always coincide.

A procedural semantics

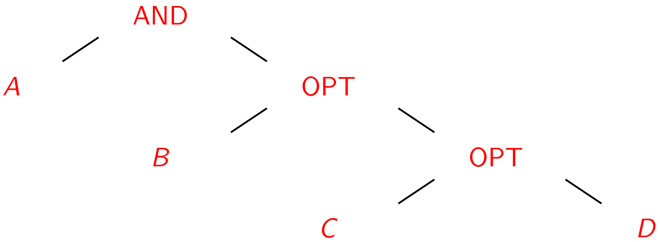
Consider: $(A \text{ AND } (B \text{ OPT } (C \text{ OPT } D)))$



- Algebraic semantics: induces the usual bottom–up evaluation.
- Alternative semantics: depth–first traversal of the parse tree.
  - Similar to the procedural semantics of Jena/ARQ
  - Navigational semantics of nested OPTs in official SPARQL (April 2006)
- These two evaluation algorithms do not always coincide.

A procedural semantics

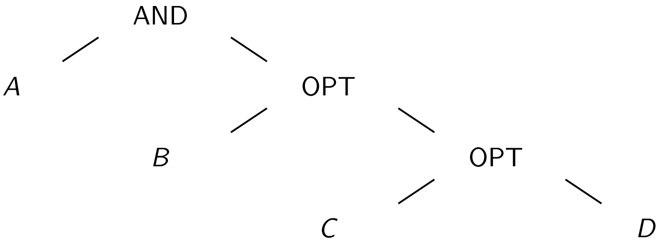
Consider: $(A \text{ AND } (B \text{ OPT } (C \text{ OPT } D)))$



- Algebraic semantics: induces the usual bottom–up evaluation.
- Alternative semantics: depth–first traversal of the parse tree.
  - Similar to the procedural semantics of Jena/ARQ
  - Navigational semantics of nested OPTs in official SPARQL (April 2006)
- These two evaluation algorithms do not always coincide.

A procedural semantics

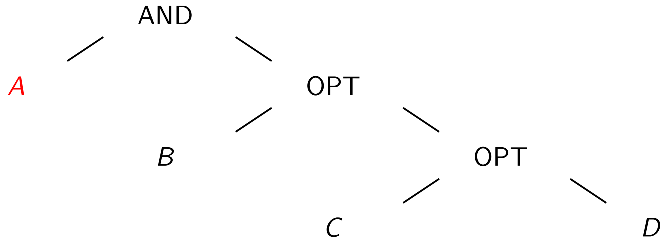
Consider: $(A \text{ AND } (B \text{ OPT } (C \text{ OPT } D)))$



- Algebraic semantics: induces the usual bottom–up evaluation.
- Alternative semantics: depth–first traversal of the parse tree.
  - Similar to the procedural semantics of Jena/ARQ
  - Navigational semantics of nested OPTs in official SPARQL (April 2006)
- These two evaluation algorithms do not always coincide.

A procedural semantics

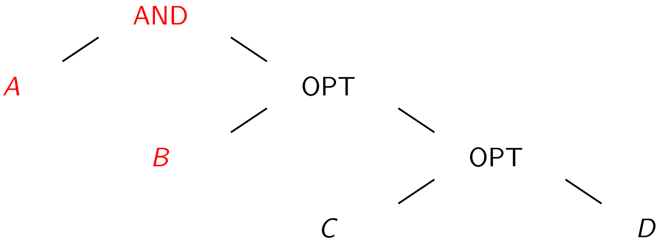
Consider: $(A \text{ AND } (B \text{ OPT } (C \text{ OPT } D)))$



- Algebraic semantics: induces the usual bottom–up evaluation.
- Alternative semantics: depth–first traversal of the parse tree.
  - Similar to the procedural semantics of Jena/ARQ
  - Navigational semantics of nested OPTs in official SPARQL (April 2006)
- These two evaluation algorithms do not always coincide.

A procedural semantics

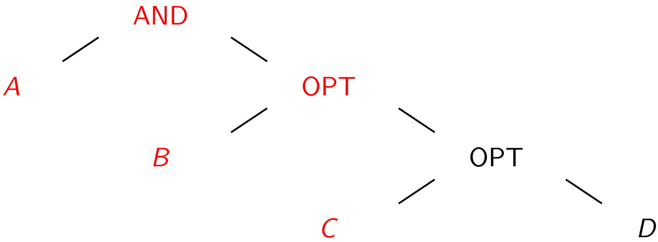
Consider: $(A \text{ AND } (B \text{ OPT } (C \text{ OPT } D)))$



- Algebraic semantics: induces the usual bottom–up evaluation.
- Alternative semantics: depth–first traversal of the parse tree.
  - Similar to the procedural semantics of Jena/ARQ
  - Navigational semantics of nested OPTs in official SPARQL (April 2006)
- These two evaluation algorithms do not always coincide.

A procedural semantics

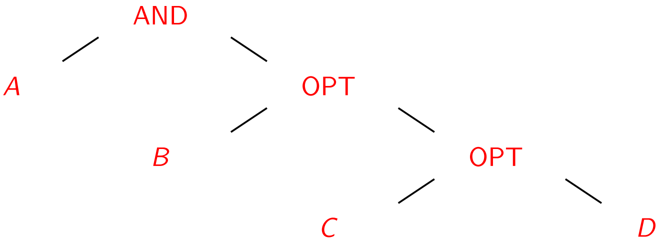
Consider: $(A \text{ AND } (B \text{ OPT } (C \text{ OPT } D)))$



- Algebraic semantics: induces the usual bottom–up evaluation.
- Alternative semantics: depth–first traversal of the parse tree.
  - Similar to the procedural semantics of Jena/ARQ
  - Navigational semantics of nested OPTs in official SPARQL (April 2006)
- These two evaluation algorithms do not always coincide.

A procedural semantics

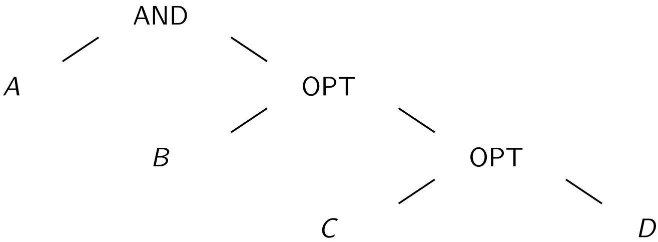
Consider: $(A \text{ AND } (B \text{ OPT } (C \text{ OPT } D)))$



- Algebraic semantics: induces the usual bottom–up evaluation.
- Alternative semantics: depth–first traversal of the parse tree.
  - Similar to the procedural semantics of Jena/ARQ
  - Navigational semantics of nested OPTs in official SPARQL (April 2006)
- These two evaluation algorithms do not always coincide.

A procedural semantics

Consider: $(A \text{ AND } (B \text{ OPT } (C \text{ OPT } D)))$



- Algebraic semantics: induces the usual bottom–up evaluation.
- Alternative semantics: depth–first traversal of the parse tree.
  - Similar to the procedural semantics of Jena/ARQ
  - Navigational semantics of nested OPTs in official SPARQL (April 2006)
- These two evaluation algorithms do not always coincide.

A procedural semantics

Depth–first traversal evaluation:

- Efficient (greedy): uses intermediate results to avoid some computations.

A procedural semantics

Depth–first traversal evaluation:

- Efficient (greedy): uses intermediate results to avoid some computations.
- non-compositional
- AND of patterns is non-commutative

Well–designed patterns

Definition

A graph pattern is well–designed iff for every OPT in the pattern

$$(\cdots\cdots\cdots\cdots \quad (\quad A \quad \text{OPT} \quad B \quad) \quad \cdots\cdots\cdots\cdots)$$

if a variable occurs inside B and anywhere outside the OPT, then the variable must also occur inside A.

Well–designed patterns

Definition

A graph pattern is well–designed iff for every OPT in the pattern

$$(\cdots\cdots\cdots\cdots \quad (\quad A \quad \mathrm{OPT} \quad \underset{\uparrow}{B} \quad) \quad \cdots\cdots\cdots\cdots)$$

if a variable occurs inside B and anywhere outside the OPT, then the variable must also occur inside A.

Well–designed patterns

Definition

A graph pattern is well–designed iff for every OPT in the pattern

$$(\cdots\cdots\cdots\cdots \quad (\quad A \quad \text{OPT} \quad B \quad) \quad \cdots\cdots\cdots\cdots)$$

$$\uparrow \qquad\qquad\qquad\qquad \uparrow \qquad\qquad\qquad \uparrow$$

if a variable occurs inside B and anywhere outside the OPT, then the variable must also occur inside A.

Well–designed patterns

Definition

A graph pattern is well–designed iff for every OPT in the pattern

$$(\cdots\cdots\cdots\cdots \quad (\quad A \quad \mathrm{OPT} \quad B \quad) \quad \cdots\cdots\cdots\cdots)$$
$$\uparrow \qquad\qquad \uparrow \qquad\qquad \uparrow \qquad\qquad \uparrow$$

if a variable occurs inside B and anywhere outside the OPT, then the variable must also occur inside A.

Well–designed patterns

Definition

A graph pattern is well–designed iff for every OPT in the pattern

$$(\cdots\cdots\cdots\cdots \quad (\quad A \quad \mathrm{OPT} \quad B \quad) \quad \cdots\cdots\cdots\cdots)$$

$$\uparrow \qquad\qquad \uparrow \qquad\qquad \uparrow \qquad\qquad \uparrow$$

if a variable occurs inside B and anywhere outside the OPT, then the variable must also occur inside A.

Example

$(\,(\,(?Y, \text{name}, \text{paul})\ \mathrm{OPT}\ (?X, \text{email}, ?Z)\,)\ \ \mathrm{AND}\ \ (?X, \text{name}, \text{john})\,)$

Well–designed patterns

Definition

A graph pattern is well–designed iff for every OPT in the pattern

$$(\cdots\cdots\cdots\cdots \quad (\quad A \quad \mathrm{OPT} \quad B \quad) \quad \cdots\cdots\cdots\cdots)$$
$$\uparrow \qquad\qquad \uparrow \qquad\qquad \uparrow \qquad\qquad \uparrow$$

if a variable occurs inside B and anywhere outside the OPT, then the variable must also occur inside A.

Example

$$(\ (\ (?Y, \text{name}, \text{paul})\ \mathrm{OPT}\ (\underset{\uparrow}{?X}, \text{email}, ?Z)\ )\quad \mathrm{AND}\quad (?X, \text{name}, \text{john})\ )$$

Well–designed patterns

Definition

A graph pattern is well–designed iff for every OPT in the pattern

$$(\cdots\cdots\cdots\cdots \quad (\quad A \quad \text{OPT} \quad B \quad) \quad \cdots\cdots\cdots\cdots)$$

if a variable occurs inside B and anywhere outside the OPT, then the variable must also occur inside A.

Example

$$(\; (\; (?Y, \text{name}, \text{paul}) \text{ OPT } (?X, \text{email}, ?Z) \;) \quad \text{AND} \quad (?X, \text{name}, \text{john}) \;)$$

Well–designed patterns

Definition

A graph pattern is well–designed iff for every OPT in the pattern

$$(\cdots\cdots\cdots\cdots \quad (\quad A \quad \mathrm{OPT} \quad B \quad) \quad \cdots\cdots\cdots\cdots)$$

if a variable occurs inside B and anywhere outside the OPT, then the variable must also occur inside A.

Example

$$(\; (\; (?Y, \text{name}, \text{paul}) \; \mathrm{OPT} \; (?X, \text{email}, ?Z) \;) \quad \mathrm{AND} \quad (?X, \text{name}, \text{john}) \;)$$

Well–designed patterns and PSPACE-hardness

In the PSPACE-hardness reduction we use this formula:

$$\begin{array}{lcl} P_\varphi & : & (a, \texttt{true}, ?B_0) \ \text{OPT} \ (P_1 \ \text{OPT} \ (Q_1 \ \text{AND} \ P_\psi)) \\ P_1 & : & (a, \texttt{tv}, ?X_1) \\ Q_1 & : & (a, \texttt{tv}, ?X_1) \ \text{AND} \ (a, \texttt{tv}, ?Y_1) \ \text{AND} \ (a, \texttt{false}, ?B_0) \end{array}$$

Well–designed patterns and PSPACE-hardness

In the PSPACE-hardness reduction we use this formula:

$$
\begin{array}{lcl}
P_\varphi & : & (a, \texttt{true}, ?B_0) \ \text{OPT} \ (P_1 \ \text{OPT} \ (Q_1 \ \text{AND} \ P_\psi)) \\
P_1 & : & (a, \texttt{tv}, ?X_1) \\
Q_1 & : & (a, \texttt{tv}, ?X_1) \ \text{AND} \ (a, \texttt{tv}, ?Y_1) \ \text{AND} \ (a, \texttt{false}, ?B_0)
\end{array}
$$

It is not well-designed: B_0

Well–designed patterns

Theorem (PAG06)

For well–designed graph patterns:

$$\textit{depth–first traversal evaluation} \;=\; \textit{compositional semantics}$$

Classical optimization is not directly applicable.

- Classical optimization assumes null–rejection.
  - null–rejection: the join/outer–join condition must fail in the presence of null.
- SPARQL operations are not null–rejecting.
  - by definition of compatible mappings.

Classical optimization is not directly applicable.

- Classical optimization assumes null–rejection.
  - null–rejection: the join/outer–join condition must fail in the presence of null.
- SPARQL operations are not null–rejecting.
  - by definition of compatible mappings.

Classical optimization is not directly applicable.

- Classical optimization assumes null–rejection.
  - null–rejection: the join/outer–join condition must fail in the presence of null.
- SPARQL operations are not null–rejecting.
  - by definition of compatible mappings.

Well–designed graph patterns and optimization

Well–designed patterns are suitable for reordering–optimization:

Theorem (OPT Normal Form)

Every well–designed pattern is equivalent to one of the form

$$(\cdots(t_1 \text{ AND } \cdots \text{ AND } t_k) \text{ OPT } O_1) \cdots) \text{ OPT } O_n)$$

where each t_i is a triple pattern, and each O_j is a pattern of the same form.

Well–designed graph patterns and optimization

Well–designed patterns are suitable for reordering–optimization:

Theorem (OPT Normal Form)

Every well–designed pattern is equivalent to one of the form

$$(\cdots(\, t_1 \text{ AND } \cdots \text{ AND } t_k\,) \text{ OPT } O_1\,)\cdots) \text{ OPT } O_n\,)$$

where each t_i is a triple pattern, and each O_j is a pattern of the same form.

Final remarks

- RDFS can be considered a new data model.
  - It is the W3C's recommendation for describing Web metadata.
- RDFS can definitely benefit from database technology.

Final remarks

- RDFS can be considered a new data model.
  - It is the W3C's recommendation for describing Web metadata.
- RDFS can definitely benefit from database technology.
  - RDFS:

Final remarks

- RDFS can be considered a new data model.
  - It is the W3C's recommendation for describing Web metadata.
- RDFS can definitely benefit from database technology.
  - RDFS: Formal semantics,

Final remarks

- RDFS can be considered a new data model.
  - It is the W3C's recommendation for describing Web metadata.
- RDFS can definitely benefit from database technology.
  - RDFS: Formal semantics, entailment of RDFS graphs,

Final remarks

- RDFS can be considered a new data model.
  - It is the W3C's recommendation for describing Web metadata.
- RDFS can definitely benefit from database technology.
  - RDFS: Formal semantics, entailment of RDFS graphs, normal forms for RDFS graphs (closure and core).

Final remarks

- RDFS can be considered a new data model.
  - It is the W3C's recommendation for describing Web metadata.
- RDFS can definitely benefit from database technology.
  - RDFS: Formal semantics, entailment of RDFS graphs, normal forms for RDFS graphs (closure and core).
  - SPARQL:

Final remarks

- RDFS can be considered a new data model.
  - It is the W3C's recommendation for describing Web metadata.
- RDFS can definitely benefit from database technology.
  - RDFS: Formal semantics, entailment of RDFS graphs, normal forms for RDFS graphs (closure and core).
  - SPARQL: Formal semantics,

Final remarks

- RDFS can be considered a new data model.
  - It is the W3C's recommendation for describing Web metadata.

- RDFS can definitely benefit from database technology.
  - RDFS: Formal semantics, entailment of RDFS graphs, normal forms for RDFS graphs (closure and core).
  - SPARQL: Formal semantics, complexity of query evaluation,

Final remarks

- RDFS can be considered a new data model.
  - It is the W3C's recommendation for describing Web metadata.
- RDFS can definitely benefit from database technology.
  - RDFS: Formal semantics, entailment of RDFS graphs, normal forms for RDFS graphs (closure and core).
  - SPARQL: Formal semantics, complexity of query evaluation, query optimization.

Final remarks

- RDFS can be considered a new data model.
  - It is the W3C's recommendation for describing Web metadata.
- RDFS can definitely benefit from database technology.
  - RDFS: Formal semantics, entailment of RDFS graphs, normal forms for RDFS graphs (closure and core).
  - SPARQL: Formal semantics, complexity of query evaluation, query optimization.
  - Updating

Final remarks

- RDFS can be considered a new data model.
  - It is the W3C's recommendation for describing Web metadata.
- RDFS can definitely benefit from database technology.
  - RDFS: Formal semantics, entailment of RDFS graphs, normal forms for RDFS graphs (closure and core).
  - SPARQL: Formal semantics, complexity of query evaluation, query optimization.
  - Updating
  - ...