SPARQL: Un Lenguaje de Consulta para la Web Semántica

Marcelo Arenas

Pontificia Universidad Católica de Chile
y
Centro de Investigación de la Web
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
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.
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

\[(s, p, o) \in (U \cup B) \times U \times (U \cup B \cup L)\] is called an RDF triple.
RDF formal model

\[
\begin{align*}
U & = \text{set of } U\text{ris} \\
B & = \text{set of Blank nodes} \\
L & = \text{set of Literals}
\end{align*}
\]

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

A set of RDF triples is called an RDF graph
RDFS: An example

Person rdf:dom sportman

Sportman rdf:sc soccer_player

Soccer_player rdf:dom Ronaldinho

Ronaldinho rdf:type soccer_player

Soccer_player rdf:sp Barcelona

Barcelona rdf:range plays_in

Plays_in rdf:range soccer_team

Soccer_team rdf:sc Spain

Spain rdf:dom lives_in

Lives_in rdf:range person

Person rdf:range company

Company rdf:sc M. Arenas – SPARQL: Un Lenguaje de Consulta para la Web Semántica
RDFS: An example

```
RDFS: An example
```

![RDFS diagram](image)

```
RDFS: An example
```

```
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:

$$Q(\overline{X}) = \exists \overline{Y} \; t_1 \land t_2 \land \cdots \land t_k$$

Some examples:
Querying RDF data

Conjunctive query:

$$Q(\bar{X}) = \exists Y t_1 \land t_2 \land \cdots \land t_k$$

Some examples:

$$(\text{Ronaldinho, plays_in, Barcelona})$$
Querying RDF data

Conjunctive query:

\[ Q(\overline{X}) = \exists \overline{Y} \ t_1 \land t_2 \land \cdots \land t_k \]

Some examples:

(Ronaldinho, plays_in, Barcelona)
(Ronaldinho, plays_in, X)
Querying RDF data

 Conjunctive query:

\[
Q(X) = \exists Y \, t_1 \land t_2 \land \cdots \land t_k
\]

Some examples:

(Ronaldinho, plays_in, Barcelona)
(Ronaldinho, plays_in, X)
\[\exists Y \, (X, \text{plays_in}, Y) \land (X, \text{lives_in}, \text{Spain})\]
Semantics of conjunctive queries

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

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

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

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

- Blank nodes play a similar role as existential variables.
- $(\text{Ronaldinho}, \text{plays}_\text{in}, B)$ and $\exists X (\text{Ronaldinho}, \text{plays}_\text{in}, X)$ 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

\( Q(\tilde{a}) \) 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 as model, interpretation, etc.
$Q(\bar{a})$ 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 as 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.
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 $G_2 \rightarrow G_1$. 
Theorem (CM77)

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

Theorem (CM77)

\[ G_1 \models G_2 \text{ if and only if there is a homomorphism } G_2 \to G_1. \]
Entailment

**Theorem (CM77)**

\[ G_1 \models G_2 \text{ if and only if there is a homomorphism } G_2 \rightarrow G_1. \]

**Complexity**

*Entailment for RDF is NP-complete*
Graphs with RDFS vocabulary

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

Built-in predicates have pre-defined semantics:
Graphs with RDFS vocabulary

Built-in predicates have pre-defined semantics:

\texttt{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:

\[
\text{rdf:sc: transitive}
\]

\[
\text{rdf:sp: transitive}
\]

More complicated interactions:

\[
(p, \text{rdf:dom}, c) \ (a, p, b)
\]

\[
(a, \text{rdf:type}, c)
\]
Graphs with RDFS vocabulary

Built-in predicates have pre-defined semantics:

\( \text{rdf:sc} \): transitive

\( \text{rdf:sp} \): transitive

More complicated interactions:

\[
(p, \text{rdf:dom}, c) \quad (a, p, b) \\
(a, \text{rdf:type}, c)
\]

RDFS-entailment can be characterized by a set of rules

- An Existential rule
- Subproperty rules
- Subclass rules
- Typing rules
- Implicit typing
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 :
Inference system in [MPG07] has 14 rules:

Existential rule : \[ \frac{G_1}{G_2} \text{ if } G_2 \rightarrow G_1 \]

Subproperty rules : \[ (p, \text{rdf:sp}, q) \rightarrow (a, p, b) \rightarrow (a, q, b) \]

Subclass rules :

Typing rules :

Implicit typing :
Graphs with RDFS vocabulary: Inference rules

Inference system in [MPG07] has 14 rules:

Existential rule: \[ \frac{G_2}{G_1} \text{ if } G_2 \rightarrow G_1 \]

Subproperty rules: \[ \frac{(p, \text{rdf:sp}, q)}{(a, p, b)} \frac{(a, p, b)}{(a, q, b)} \]

Subclass rules: \[ \frac{(a, \text{rdf:sc}, b)}{(b, \text{rdf:sc}, c)} \frac{(b, \text{rdf:sc}, c)}{(a, \text{rdf:sc}, c)} \]

Typing rules:

Implicit typing:
Graphs with RDFS vocabulary: Inference rules

Inference system in [MPG07] has 14 rules:

Existential rule : \[ \frac{G_1}{G_2} \text{ if } G_2 \rightarrow G_1 \]

Subproperty rules : \[ (p, \text{rdf:sp}, q) \rightarrow (a, p, b) \]
\[ (a, q, b) \]

Subclass rules : \[ (a, \text{rdf:sc}, b) \rightarrow (b, \text{rdf:sc}, c) \]
\[ (a, \text{rdf:sc}, c) \]

Typing rules : \[ (p, \text{rdf:dom}, c) \rightarrow (a, p, b) \]
\[ (a, \text{rdf:type}, c) \]

Implicit typing : 

Inference system in [MPG07] has 14 rules:

**Existential rule**
\[
\frac{G_1}{G_2} \text{ if } G_2 \rightarrow G_1
\]

**Subproperty rules**
\[
(p, \text{rdf:sp}, q) \quad (a, p, b) \quad \frac{(a, q, b)}{}
\]

**Subclass rules**
\[
(a, \text{rdf:sc}, b) \quad (b, \text{rdf:sc}, c) \quad \frac{(a, \text{rdf:sc}, c)}{}
\]

**Typing rules**
\[
(p, \text{rdf:dom}, c) \quad (a, p, b) \quad \frac{(a, \text{rdf:type}, c)}{}
\]

**Implicit typing**
\[
(q, \text{rdf:dom}, a) \quad (p, \text{rdf:sp}, q) \quad (b, p, c) \quad \frac{(b, \text{rdf:type}, a)}{}
\]
Graphs with RDFS vocabulary: Inference rules

Inference system in [MPG07] has 14 rules:

Existential rule :

Subproperty rules : \[
\frac{(p, \text{rdf:sp}, q) \quad (a, p, b)}{(a, q, b)}
\]

Subclass rules :

Typing rules : \[
\frac{(p, \text{rdf:dom}, c) \quad (a, p, b)}{(a, \text{rdf:type}, c)}
\]

Implicit typing : \[
\frac{(q, \text{rdf:dom}, a) \quad (p, \text{rdf:sp}, q) \quad (b, p, c)}{(b, \text{rdf:type}, a)}
\]
Inference system in [MPG07] has 14 rules:

- **Existential rule**: 
  
- **Subproperty rules**: 
  \[
  (p, \text{rdf:sp}, q) \quad (a, p, b) \\
  \overbrace{(a, q, b)}
  \]

- **Subclass rules**: 

- **Typing rules**: 
  \[
  (p, \text{rdf:dom}, c) \quad (a, p, b) \\
  \overbrace{(a, \text{rdf:type}, c)}
  \]

- **Implicit typing**: 
  \[
  (B, \text{rdf:dom}, a) \quad (p, \text{rdf:sp}, B) \quad (b, p, c) \\
  \overbrace{(b, \text{rdf:type}, a)}
  \]
RDFS Entailment

Theorem (H03, GHM04, MPG07)

\( G_1 \models G_2 \iff \text{there is a proof of } G_2 \text{ from } G_1 \text{ using the system of } 14 \text{ inference rules.} \)

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.
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:

\[ \text{ground}(G) : \text{Graph obtained by replacing every blank } B \text{ in } G \text{ by a constant } c_B. \]

\[ \text{ground}^{-1}(G) : \text{Graph obtained by replacing every constant } c_B \text{ in } G \text{ by } B. \]

Closure of an RDF graph \( G \) (denoted by \( \text{closure}(G) \)):
Closure of an RDF Graph

Notation:

\[ \text{ground}(G) : \text{Graph obtained by replacing every blank } B \text{ in } G \text{ by a constant } c_B. \]

\[ \text{ground}^{-1}(G) : \text{Graph obtained by replacing every constant } c_B \text{ in } G \text{ by } B. \]

Closure of an RDF graph \( G \) (denoted by \( \text{closure}(G) \)):

\[ G \cup \{ t \in (U \cup B) \times U \times (U \cup B \cup L) \mid \text{there exists a ground tuple } t' \text{ such that } \text{ground}(G) \models t' \text{ and } t = \text{ground}^{-1}(t') \} \]
Closure of an RDF Graph: Example

c
\text{rdf:sc}\n
b
\text{rdf:sc}\n
a
Closure of an RDF Graph: Example
Querying RDFS data: Using the closure of a graph

Proposition (H03, GHM04, MPG07)

\[ G_1 \models G_2 \text{ iff } G_2 \rightarrow \text{closure}(G_1) \]

Complexity

*The closure of \( G \) can be computed in time \( O(\|G\|^4 \cdot \log \|G\|) \).*
Proposition (H03,GHM04,MPG07)

\[ G_1 \models G_2 \text{ iff } G_2 \rightarrow \text{closure}(G_1) \]

 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)

\[ G_1 \models G_2 \iff G_2 \rightarrow closure(G_1) \]

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)

\[ G_1 \models G_2 \text{ iff } G_2 \rightarrow \text{closure}(G_1) \]

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?
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 $\text{core}(G)$).
- Deciding if $G$ is a core is coNP-complete.
- Deciding if $G = \text{core}(G')$ is DP-complete.
For RDF graphs with RDFS vocabulary, the core of $G$ may contain redundant information:
For RDF graphs with RDFS vocabulary, the core of $G$ may contain redundant information:
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 \( \text{nf}(G_1) \cong \text{nf}(G_2) \).
- \( G_1 \models G_2 \) iff \( G_2 \rightarrow \text{nf}(G_1) \)
A normal form for RDF graphs

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

\[ \text{nf}(G) = \text{core(closure}(G)) \]

**Theorem (GHM04)**

- \(G_1\) is equivalent to \(G_2\) iff \(\text{nf}(G_1) \equiv \text{nf}(G_2)\).
- \(G_1 \models G_2\) iff \(G_2 \rightarrow \text{nf}(G_1)\)

**Complexity**

The problem of deciding if \(G_1 = \text{nf}(G_2)\) is DP-complete.
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

```sparql
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

```sql
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

```sql
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
- ...

```sparql
{ { 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 } } } } UN

UNION

{ P9 }
But things can become more complex ...

Interesting features of pattern matching on graphs

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

```sparql
{ 
  { 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
- ...

```sparql
{ { 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 → 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 ← 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*

  \[
  \text{?X :name "john"} \\
  \text{(?X, name, john)}
  \]

- **Graph patterns**: full parenthesized algebra

  \[
  \{ \text{P1 P2 } \} \\
  \{ \text{P1 OPTIONAL } \{ \text{P2 } \}\} \\
  \{ \text{P1 } \text{UNION } \{ \text{P2 } \}\} \\
  \{ \text{P1 FILTER ( R ) } \}
  \]

  \[
  ( \text{P1 AND P2 } ) \\
  ( \text{P1 OPT P2 } ) \\
  ( \text{P1 UNION P2 } ) \\
  ( \text{P1 FILTER R } )
  \]

**original SPARQL syntax**

**algebraic syntax**
A standard algebraic syntax

- Explicit precedence/association

Example

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

(((( t1 AND t2 ) OPT t3 ) OPT t4 ) AND t5 )
```
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 $t$

<table>
<thead>
<tr>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>The <strong>evaluation</strong> of $t$ is the set of mappings that</td>
</tr>
</tbody>
</table>
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**

<table>
<thead>
<tr>
<th>graph</th>
<th>triple</th>
<th>evaluation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(R_1, \text{name, john})$</td>
<td>$(?X, \text{name, ?Y})$</td>
<td>$\mu_1: R_1 \rightarrow \text{john}$</td>
</tr>
<tr>
<td>$(R_1, \text{email, <a href="mailto:J@ed.ex">J@ed.ex</a>})$</td>
<td></td>
<td>$\mu_2: R_2 \rightarrow \text{paul}$</td>
</tr>
<tr>
<td>$(R_2, \text{name, paul})$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$\implies$
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**

<table>
<thead>
<tr>
<th>graph</th>
<th>triple</th>
<th>evaluation</th>
</tr>
</thead>
</table>
| \((R_1, \text{name, john})\)             | \((?X, \text{name, } ?Y)\) | \(\mu_1: \)
| \((R_1, \text{email, J@ed.ex})\)         |                         | \(R_1\)    |
| \((R_2, \text{name, paul})\)             |                         | \(\mu_2: \)
|                                            |                         | \(R_2\)    |

\(\mu_1: R_1 \mapsto \text{john}\)
\(\mu_2: R_2 \mapsto \text{paul}\)
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**

<table>
<thead>
<tr>
<th>graph</th>
<th>triple</th>
<th>evaluation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(R_1, \text{name}, \text{john})$</td>
<td>$(\text{?X}, \text{name}, \text{?Y})$</td>
<td>$\mu_1: R_1 \rightarrow \text{john}$, $\mu_2: R_2 \rightarrow \text{paul}$</td>
</tr>
<tr>
<td>$(R_1, \text{email}, <a href="mailto:J@ed.ex">J@ed.ex</a>)$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$(R_2, \text{name}, \text{paul})$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Compatible mappings

**Definition**

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

**Example**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mu_1$ :</td>
<td>$R_1$</td>
<td>john</td>
<td><a href="mailto:J@edu.ex">J@edu.ex</a></td>
<td></td>
</tr>
<tr>
<td>$\mu_2$ :</td>
<td>$R_1$</td>
<td></td>
<td><a href="mailto:P@edu.ex">P@edu.ex</a></td>
<td>$R_2$</td>
</tr>
<tr>
<td>$\mu_3$ :</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Compatible mappings

**Definition**

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

**Example**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mu_1$ :</td>
<td>$R_1$</td>
<td>john</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\mu_2$ :</td>
<td>$R_1$</td>
<td></td>
<td><a href="mailto:J@edu.ex">J@edu.ex</a></td>
<td></td>
</tr>
<tr>
<td>$\mu_3$ :</td>
<td></td>
<td><a href="mailto:P@edu.ex">P@edu.ex</a></td>
<td></td>
<td>$R_2$</td>
</tr>
</tbody>
</table>
**Definition**

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

**Example**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mu_1$ :</td>
<td>$R_1$</td>
<td>john</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\mu_2$ :</td>
<td>$R_1$</td>
<td></td>
<td><a href="mailto:J@edu.ex">J@edu.ex</a></td>
<td></td>
</tr>
<tr>
<td>$\mu_3$ :</td>
<td></td>
<td><a href="mailto:P@edu.ex">P@edu.ex</a></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\mu_1 \cup \mu_2$ :</td>
<td>$R_1$</td>
<td>john</td>
<td></td>
<td><a href="mailto:J@edu.ex">J@edu.ex</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$R_2$</td>
<td></td>
</tr>
</tbody>
</table>
Compatible mappings

**Definition**

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

**Example**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mu_1$ :</td>
<td>$R_1$</td>
<td>john</td>
<td><a href="mailto:J@edu.ex">J@edu.ex</a></td>
<td></td>
</tr>
<tr>
<td>$\mu_2$ :</td>
<td>$R_1$</td>
<td></td>
<td><a href="mailto:P@edu.ex">P@edu.ex</a></td>
<td>$R_2$</td>
</tr>
<tr>
<td>$\mu_3$ :</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\mu_1 \cup \mu_2$ :</td>
<td>$R_1$</td>
<td>john</td>
<td><a href="mailto:J@edu.ex">J@edu.ex</a></td>
<td></td>
</tr>
</tbody>
</table>
Definition

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

Example

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>μ₁</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>μ₂</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>μ₃</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>μ₁ ∪ μ₂</td>
<td>∪</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>μ₁ ∪ μ₃</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>μ₁</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>μ₂</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>μ₃</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>μ₁ ∪ μ₂</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>μ₁ ∪ μ₃</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\( R_1 \)

\µ_1: R_1, John

\µ_2: R_1, John

\µ_3: P, edu.ex

\µ_1 ∪ \µ_2: R_1, John, edu.ex

\µ_1 ∪ \µ_3: R_1, John, P, edu.ex

\µ_1 ∪ \µ_2: R_2

\µ_1 ∪ \µ_3: R_2
Compatible mappings

Definition

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

Example

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>( \mu_1 ) :</td>
<td>( R_1 )</td>
<td>john</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \mu_2 ) :</td>
<td>( R_1 )</td>
<td></td>
<td><a href="mailto:J@edu.ex">J@edu.ex</a></td>
<td></td>
</tr>
<tr>
<td>( \mu_3 ) :</td>
<td></td>
<td><a href="mailto:P@edu.ex">P@edu.ex</a></td>
<td></td>
<td>( R_2 )</td>
</tr>
<tr>
<td>( \mu_1 \cup \mu_2 ) :</td>
<td>( R_1 )</td>
<td>john</td>
<td><a href="mailto:J@edu.ex">J@edu.ex</a></td>
<td></td>
</tr>
<tr>
<td>( \mu_1 \cup \mu_3 ) :</td>
<td>( R_1 )</td>
<td>john</td>
<td><a href="mailto:P@edu.ex">P@edu.ex</a></td>
<td>( R_2 )</td>
</tr>
</tbody>
</table>

\( \mu_2 \) and \( \mu_3 \) 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:** $M_1 \Join M_2$

- 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:** $M_1 \Join M_2$
- extending mappings in $M_1$ with compatible mappings in $M_2$

**Difference:** $M_1 \setminus M_2$
- 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:** $M_1 \Join M_2$
- extending mappings in $M_1$ with compatible mappings in $M_2$

**Difference:** $M_1 \setminus M_2$
- 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**: $M_1 \Join M_2$
  - extending mappings in $M_1$ with compatible mappings in $M_2$
- **Difference**: $M_1 \setminus M_2$
  - 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**: $M_1 \Joinleft M_2 = (M_1 \Join M_2) \cup (M_1 \setminus M_2)$
Semantics of SPARQL operators

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

**Definition**

The evaluation of:

- $(P_1 \text{ AND } P_2)$ →
- $(P_1 \text{ UNION } P_2)$ →
- $(P_1 \text{ OPT } P_2)$ →
Semantics of SPARQL operators

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

**Definition**

The evaluation of:

- $(P_1 \text{ AND } P_2) \rightarrow M_1 \bowtie M_2$
- $(P_1 \text{ UNION } P_2) \rightarrow$
- $(P_1 \text{ OPT } P_2) \rightarrow$
Semantics of SPARQL operators

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

**Definition**

The evaluation of:

- $(P_1 \ \text{AND} \ P_2) \rightarrow M_1 \ \bowtie \ M_2$
- $(P_1 \ \text{UNION} \ P_2) \rightarrow M_1 \cup M_2$
- $(P_1 \ \text{OPT} \ P_2) \rightarrow$
Semantics of SPARQL operators

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

**Definition**

The evaluation of:

\[
\begin{align*}
(P_1 \text{ AND } P_2) & \rightarrow M_1 \bowtie M_2 \\
(P_1 \text{ UNION } P_2) & \rightarrow M_1 \cup M_2 \\
(P_1 \text{ OPT } P_2) & \rightarrow M_1 \nabla M_2
\end{align*}
\]
Simple example

Example

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

\[(X, \text{name, Y}) \text{OPT} (X, \text{email, E})\]
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

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

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

<table>
<thead>
<tr>
<th>?X</th>
<th>?Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>(R_1)</td>
<td>john</td>
</tr>
<tr>
<td>(R_2)</td>
<td>paul</td>
</tr>
</tbody>
</table>
### Example

$(R_1, \text{name}, \text{john})$

$(R_1, \text{email}, \text{J@ed.ex})$

$(R_2, \text{name}, \text{paul})$

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

<table>
<thead>
<tr>
<th>?X</th>
<th>?Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R_1$</td>
<td>john</td>
</tr>
<tr>
<td>$R_2$</td>
<td>paul</td>
</tr>
</tbody>
</table>
Simple example

Example

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

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

<table>
<thead>
<tr>
<th>?X</th>
<th>?Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_1</td>
<td>john</td>
</tr>
<tr>
<td>R_2</td>
<td>paul</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>?X</th>
<th>?E</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_1</td>
<td><a href="mailto:J@ed.ex">J@ed.ex</a></td>
</tr>
</tbody>
</table>
Simple example

Example

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

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

<table>
<thead>
<tr>
<th>?X</th>
<th>?Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>(R_1)</td>
<td>john</td>
</tr>
<tr>
<td>(R_2)</td>
<td>paul</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>?X</th>
<th>?E</th>
</tr>
</thead>
<tbody>
<tr>
<td>(R_1)</td>
<td>\text{<a href="mailto:J@ed.ex">J@ed.ex</a>}</td>
</tr>
</tbody>
</table>
Simple example

Example

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

\[((?X, \text{name, ?Y}) \text{OPT} (?X, \text{email, ?E}))\]
Simple example

Example

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

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

\[
\begin{array}{c|c}
?X & ?Y \\
\hline
R_1 & \text{john} \\
R_2 & \text{paul} \\
\end{array}
\]

\[
\begin{array}{c|c|c}
?X & ?Y & ?E \\
\hline
R_1 & \text{john} & \text{J@ed.ex} \\
R_2 & \text{paul} & \\
\end{array}
\]

\[
\begin{array}{c|c}
?X & ?E \\
\hline
R_1 & \text{J@ed.ex} \\
\end{array}
\]

from the Join
Simple example

Example

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

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

\begin{array}{|c|c|}
\hline
?X & ?Y \\
\hline
R_1 & john \\
R_2 & paul \\
\hline
\end{array}

\begin{array}{|c|c|}
\hline
?X & ?Y & ?E \\
\hline
R_1 & john & J@ed.ex \\
R_2 & paul & ?E \\
\hline
\end{array}

\begin{array}{|c|c|}
\hline
?X & ?E \\
\hline
R_1 & J@ed.ex \\
\hline
\end{array}

from the Difference
Example

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

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

<table>
<thead>
<tr>
<th>?X</th>
<th>?Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>(R_1)</td>
<td>john</td>
</tr>
<tr>
<td>(R_2)</td>
<td>paul</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>?X</th>
<th>?Y</th>
<th>?E</th>
</tr>
</thead>
<tbody>
<tr>
<td>(R_1)</td>
<td>john</td>
<td><a href="mailto:J@ed.ex">J@ed.ex</a></td>
</tr>
<tr>
<td>(R_2)</td>
<td>paul</td>
<td><a href="mailto:J@ed.ex">J@ed.ex</a></td>
</tr>
</tbody>
</table>

▶ from the \text{Union}
In filter expressions we consider

- equality $=$ among variables and RDF terms
- unary predicate **bound**
- boolean combinations ($\land$, $\lor$, $\lnot$)
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 \( ?Y \)
▶ \( \text{bound}(?X) \) if it is defined for \( ?X \)

Definition

Evaluation of \((P \ \text{FILTER} \ R)\): Set of mappings in the evaluation of \( P \) that satisfy \( R \).
Natural algebraic properties: A simple normal form

- 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 \textit{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:

\[ O(\text{size of the pattern} \times \text{size of the graph}). \]
Evaluation of simple patterns is polynomial.

**Theorem (PAG06)**

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

\[ O(\text{size of the pattern} \times \text{size of the graph}) \].

**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.
- \( \neg \text{bound} \) 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 OPTs 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 OPTs are used to encode quantifier alternation.

(This time, we do not need \( \neg \) bound.)
Assume $\varphi = \forall x_1 \exists y_1 \psi$, where $\psi = (x_1 \lor \neg y_1) \land (\neg x_1 \lor y_1)$.

We generate $G$, $P_\varphi$ and $\mu_0$ such that $\mu_0$ belongs to the answer of $P_\varphi$ over $G$ iff $\varphi$ is valid:

$G$ : 

$P_\psi$ : 

$P_\varphi$ :

$\mu_0$ :
Assume $\varphi = \forall x_1 \exists y_1 \psi$, where $\psi = (x_1 \lor \neg y_1) \land (\neg x_1 \lor y_1)$.

We generate $G$, $P_\varphi$ and $\mu_0$ such that $\mu_0$ belongs to the answer of $P_\varphi$ over $G$ iff $\varphi$ is valid:

\begin{align*}
G & : \{(a, tv, 0), (a, tv, 1), (a, false, 0), (a, true, 1)\} \\
P_\psi & : \\
P_\varphi & : \\
\mu_0 & : 
\end{align*}
Assume $\varphi = \forall x_1 \exists y_1 \psi$, where $\psi = (x_1 \lor \neg y_1) \land (\neg x_1 \lor y_1)$.

We generate $G$, $P_\varphi$ and $\mu_0$ such that $\mu_0$ belongs to the answer of $P_\varphi$ over $G$ iff $\varphi$ is valid:

\[
G : \{(a, tv, 0), (a, tv, 1), (a, false, 0), (a, true, 1)\}
\]

\[
P_\psi : \left((a, true, ?X_1) \cup (a, false, ?Y_1)\right) \land \left((a, false, ?X_1) \cup (a, true, ?Y_1)\right)
\]

\[
P_\varphi :
\]

\[
\mu_0 :
\]
Assume $\varphi = \forall x_1 \exists y_1 \psi$, where $\psi = (x_1 \lor \neg y_1) \land (\neg x_1 \lor y_1)$.

We generate $G$, $P_\varphi$ and $\mu_0$ such that $\mu_0$ belongs to the answer of $P_\varphi$ over $G$ iff $\varphi$ is valid:

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

$P_\psi : ((a, \text{true}, ?X_1) \cup (a, \text{false}, ?Y_1)) \land ((a, \text{false}, ?X_1) \cup (a, \text{true}, ?Y_1))$

$P_\varphi : (a, \text{true}, ?B_0) \text{ OPT } (P_1 \text{ OPT } (Q_1 \text{ AND } P_\psi))$

$\mu_0 : \ $
Assume $\varphi = \forall x_1 \exists y_1 \psi$, where $\psi = (x_1 \lor \neg y_1) \land (\neg x_1 \lor y_1)$.

We generate $G$, $P_\varphi$ and $\mu_0$ such that $\mu_0$ belongs to the answer of $P_\varphi$ over $G$ iff $\varphi$ is valid:

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

$P_\psi : ((a, \text{true}, ?X_1) \text{ UNION } (a, \text{false}, ?Y_1)) \text{ AND } ((a, \text{false}, ?X_1) \text{ UNION } (a, \text{true}, ?Y_1))$

$P_\varphi : (a, \text{true}, ?B_0) \text{ OPT } (P_1 \text{ OPT } (Q_1 \text{ AND } P_\psi))$

$\mu_0 : \{ ?B_0 \mapsto 1 \}$
PSPACE-hardness: A closer look

\[ P_\varphi : (a, \text{true}, ?B_0) \text{ OPT } (P_1 \text{ OPT } (Q_1 \text{ AND } P_\psi)) \]
\[ P_1 : (a, \text{tv}, ?X_1) \]
\[ Q_1 : (a, \text{tv}, ?X_1) \text{ AND } (a, \text{tv}, ?Y_1) \text{ AND } (a, \text{false}, ?B_0) \]
PSPACE-hardness: A closer look

\[ P_\varphi : (a, \text{true}, ?B_0) \text{ OPT } (P_1 \text{ OPT } (Q_1 \text{ AND } P_\psi)) \]
\[ P_1 : (a, \text{tv}, ?X_1) \]
\[ Q_1 : (a, \text{tv}, ?X_1) \text{ AND } (a, \text{tv}, ?Y_1) \text{ AND } (a, \text{false}, ?B_0) \]

?B_0 \mapsto 1
PSPACE-hardness: A closer look

\[ P_{\varphi} : (a, \text{true}, ?B_0) \text{ OPT } (P_1 \text{ OPT } (Q_1 \text{ AND } P_\psi)) \]

\[ P_1 : (a, \text{tv}, ?X_1) \]

\[ Q_1 : (a, \text{tv}, ?X_1) \text{ AND } (a, \text{tv}, ?Y_1) \text{ AND } (a, \text{false}, ?B_0) \]
PSPACE-hardness: A closer look

\[ P_\varphi : (a, \text{true}, ?B_0) \text{ OPT } (P_1 \text{ OPT } (Q_1 \text{ AND } P_\psi)) \]
\[ P_1 : (a, \text{tv}, ?X_1) \]
\[ Q_1 : (a, \text{tv}, ?X_1) \text{ AND } (a, \text{tv}, ?Y_1) \text{ AND } (a, \text{false}, ?B_0) \]

\[ ?X_1 \leftrightarrow 0 \quad \rightarrow \quad ?X_1 \leftrightarrow 0 \quad ?Y_1 \leftrightarrow i \quad ?B_0 \leftrightarrow 0 \]
\[ ?B_0 \leftrightarrow 1 \]
PSPACE-hardness: A closer look

\[ P_{\varphi} : (a, \text{true}, ?B_0) \text{ OPT } (P_1 \text{ OPT } (Q_1 \text{ AND } P_{\psi})) \]
\[ P_1 : (a, \text{tv}, ?X_1) \]
\[ Q_1 : (a, \text{tv}, ?X_1) \text{ AND } (a, \text{tv}, ?Y_1) \text{ AND } (a, \text{false}, ?B_0) \]
PSPACE-hardness: A closer look

\[ P_\varphi \quad : \quad (a, \text{true}, ?B_0) \ \text{OPT} \ (P_1 \ \text{OPT} \ (Q_1 \ \text{AND} \ P_\psi)) \]
\[ P_1 \quad : \quad (a, \text{tv}, ?X_1) \]
\[ Q_1 \quad : \quad (a, \text{tv}, ?X_1) \ \text{AND} \ (a, \text{tv}, ?Y_1) \ \text{AND} \ (a, \text{false}, ?B_0) \]
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 $A \text{OPT}(B \text{OPT} C)$:

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 $C$.

Depth-first traversal of queries parse trees.
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 $C$.

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 \textit{bottom–up} evaluation.
- Alternative semantics: \textit{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
### Definition

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

\[
\left( \ldots \ldots \ldots \left( \ A \ \text{OPT} \ B \right) \ldots \ldots \ldots \right)
\]

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

( ⋮ ⋮ ( A OPT B ) ⋮ ⋮ )

↑

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

\[ ( \ldots \quad ( \text{A} \quad \text{OPT} \quad \text{B} ) \quad \ldots \ldots \ldots ) \]

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

\[(\ldots \ (A \ OPT \ B) \ \ldots \ )\]

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

\[
\begin{array}{c}
\text{-----------} \\
\downarrow \\
\text{A} \\
\text{OPT} \\
\text{B} \\
\text{-----------}
\end{array}
\]

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

Example

\[
\begin{array}{c}
\left( \left( (?Y, \text{name, paul}) \text{OPT} (?X, \text{email, ?Z}) \right) \text{AND} \left( (?X, \text{name, john}) \right) \right)
\end{array}
\]
Well–designed patterns

Definition

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

\[(\cdots (A \text{ OPT } B) \cdots)\]

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

Example

\[( ((?Y, \text{name, paul}) \text{ OPT } (?X, \text{email, } ?Z)) \text{ AND } (?X, \text{name, john}) ) \]
Definition
A graph pattern is well–designed iff for every OPT in the pattern

\[
( \cdots \cdots \cdot ( A \text{ OPT } B ) \cdots \cdots )
\]

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

Example
\[
( ( (?Y, \text{name, paul}) \text{ OPT } (?X, \text{email, ?Z}) ) \text{ AND } (?X, \text{name, john}) )
\]
Well–designed patterns

**Definition**

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

\[
\left( \begin{array}{c}
\uparrow \\
A \text{ OPT } B \\
\uparrow \\
\uparrow \\
\uparrow \\
\uparrow
\end{array} \right)
\]

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

**Example**

\[
\left( \begin{array}{c}
\times \\
(?Y, \text{name, paul}) \text{ OPT } (?X, \text{email, } ?Z) \\
\uparrow \\
\uparrow
\end{array} \right) \text{ AND } (?X, \text{name, john})
\]
In the PSPACE-hardness reduction we use this formula:

\[
P_{\varphi} : (a, \text{true}, ?B_0) \text{ OPT } (P_1 \text{ OPT } (Q_1 \text{ AND } P_{\psi}))
\]

\[
P_1 : (a, \text{tv}, ?X_1)
\]

\[
Q_1 : (a, \text{tv}, ?X_1) \text{ AND } (a, \text{tv}, ?Y_1) \text{ AND } (a, \text{false}, ?B_0)
\]
In the PSPACE-hardness reduction we use this formula:

\[
P_\varphi : (a, \text{true}, ?B_0) \text{ OPT } (P_1 \text{ OPT } (Q_1 \text{ AND } P_\psi))
\]

\[
P_1 : (a, \text{tv}, ?X_1)
\]

\[
Q_1 : (a, \text{tv}, ?X_1) \text{ AND } (a, \text{tv}, ?Y_1) \text{ AND } (a, \text{false}, ?B_0)
\]

It is not well-designed: \( B_0 \)
Well–designed patterns

Theorem (PAG06)

For well–designed graph patterns:

\[ \text{depth–first traversal evaluation} = \text{compositional semantics} \]
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 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 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.
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
  - ...