Optimization of Semantic Web Queries

Reinhard Pichler
Vienna University of Technology
Puebla, 19 May, 2013

*joint work with Andrés Letelier, Jorge Pérez, and Sebastian Skritek

Outline

1. Motivation
2. Tools for the Semantic Web
   2.1 RDF
   2.2 SPARQL
3. Role Model: Relational Algebra
4. Query plans and transformation rules for SPARQL
   4.1 Transformation rules for SPARQL
   4.2 Query Plans for SPARQL
5. Static Analysis of SPARQL Queries
6. Conclusion

Acknowledgement

I am very grateful to
- Marcelo Arenas and Jorge Pérez for providing me their slides from tutorials at WWW and PODS, respectively.
- Sebastian Skritek for his great help in creating this slide set.
Optimization of Semantic Web Queries

1. Motivation

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 proposals:
- Resource Description Framework (RDF) and SPARQL

Reinhard Pichler
Puebla, 19 May, 2013

Outline

1. Motivation
2. Tools for the Semantic Web
   2.1 RDF
   2.2 SPARQL
3. Role Model: Relational Algebra
4. Query plans and transformation rules for SPARQL
   4.1 Transformation rules for SPARQL
   4.2 Query Plans for SPARQL
5. Static Analysis of SPARQL Queries
6. Conclusion

RDF in a nutshell

W3C proposal framework for representing information on the web
- Extensible vocabulary based on URIs
  - URI = Uniform Resource Identifier
  - is an atomic piece of data; identifies an abstract resource.
- Abstract syntax based on directed labeled graph.
  - URIs are used as node labels and edge labels
- Formal semantics
- Existential variables as data values
- Support use of XML schema data types
- Schema definition language (RDFS): define new vocabulary
  - Typing, inheritance of classes and properties, ...
- Built-in vocabulary with fixed semantics (RDFS)
- Graph model where nodes may also be edge labels

Reinhard Pichler
Puebla, 19 May, 2013
**RDF formal model**

- **Subject**
- **Predicate**
- **Object**

\[ U \]
\[ B \]
\[ L \]

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

A set of RDF triples is called an **RDF graph**.

---

**RDF: Example**

- `'FC Wacker Innsbruck'`
- `'Bundesliga'`
- `fcw`
- `tivoli`
- `innsbruck`
- `M_1`
- `name`
- `league`
- `stadium`
- `hanappi`
- `wien`
- `museum`
- `home`
- `location`
- `poi`
RDF: Example

```
{fcw name 'FC Wacker Innsbruck', fcw league 'Bundesliga',
 fcw home 'tivoli', tivoli poi 'stadium', tivoli location innsbruck}
```

Reinhard Pichler

Puebla, 19 May, 2013

Querying RDF with triple patterns

- Triple patterns: RDF triple + variables

```
(?X, name, ?Name)
```

Reinhard Pichler

Puebla, 19 May, 2013

Querying RDF with triple patterns

- Triple patterns: RDF triple + variables

```
(?X, name, ?Name)
```

- The base case for RDF queries is a set of triple patterns

```
{t_1, t_2, \ldots, t_k}
```

This is called basic graph pattern (BGP).

Reinhard Pichler

Puebla, 19 May, 2013
Querying RDF with triple patterns

- Triple patterns: RDF triple + variables

\(?X, \text{name}, ?\text{Name}\)

- The base case for RDF queries is a set of triple patterns

\{ \(t_1, t_2, \ldots, t_k\}\).

This is called basic graph pattern (BGP).

Example

\{ (\(?X, \text{name}, ?\text{Name}\), (\(?X, \text{home}, ?\text{Stadium}\)\}\)

Mappings: building block for the semantics

Definition

A mapping is a partial function:

\[\mu : V \rightarrow (U \cup L)\]

Given a mapping \(\mu\) and a basic graph pattern \(P\):

- \(\text{dom}(\mu)\): the domain of \(\mu\).
- \(\mu(P)\): the set of triples obtained from \(P\) by replacing the variables according to \(\mu\).

The evaluation of a graph pattern results in a set of mappings.
The semantics of triple patterns

Given an RDF graph $G$ and a triple pattern $t$.

Definition

The evaluation of $t$ over $G$ is the set of mappings $\mu$ such that:
- $\mu$ has as domain the variables in $t$: $\text{dom}(\mu) = \text{var}(t)$
- $\mu$ makes $t$ to match the graph: $\mu(t) \in G$

Example

triple pattern: $(?X, \text{location}, ?Y)$

<table>
<thead>
<tr>
<th>graph</th>
<th>evaluation</th>
</tr>
</thead>
<tbody>
<tr>
<td>(tivoli, location, innsbruck)</td>
<td>$\mu_1$:</td>
</tr>
<tr>
<td>(tivoli, poi, stadium)</td>
<td>tivoli</td>
</tr>
<tr>
<td>(hanappi, location, wien)</td>
<td>innsbruck</td>
</tr>
</tbody>
</table>

| $\mu_2$:                       |
| hanappi                       |
| wien                           |
Compatible mappings

Definition

Mappings $\mu_1$ and $\mu_2$ are compatible if they agree in their common variables:

If $?X \in \text{dom}(\mu_1) \cap \text{dom}(\mu_2)$, then $\mu_1(?X) = \mu_2(?X)$.

Example

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mu_1$</td>
<td>tivoli</td>
<td>innsbruck</td>
<td>stadium</td>
<td>M₁</td>
</tr>
<tr>
<td>$\mu_2$</td>
<td>tivoli</td>
<td>innsbruck</td>
<td>museum</td>
<td></td>
</tr>
<tr>
<td>$\mu_3$</td>
<td>tivoli</td>
<td>stadium</td>
<td>M₁</td>
<td></td>
</tr>
</tbody>
</table>

$\mu_1 \cup \mu_2$ and $\mu_3$ are not compatible
SPARQL is the W3C recommendation query language for RDF (January 2008).
- SPARQL is a recursive acronym that stands for “SPARQL Protocol and RDF Query Language”

A SPARQL query consists of three parts:
- Pattern matching: optional, union, filtering, ...
- Solution modifiers: projection, distinct, order, limit, offset, ...
- Output part: construction of new triples, boolean queries, ...

**SPARQL in a nutshell**

```sparql
SELECT ?Stadium
WHERE
{
  ?Club ex:league 'Bundesliga' .
}
```

A SPARQL query consists of a:
- **Body**: Pattern matching expression
Optimization of Semantic Web Queries

2. Tools for the Semantic Web

SPARQL in a nutshell

```
SELECT ?Stadium
WHERE {
  ?Club ex:league 'Bundesliga' .
}
```

A SPARQL query consists of a:

- **Body**: Pattern matching expression
- **Head**: Processing of the variable assignments

---

Interesting features:

- Grouping
- Optional parts

```
{ { P1
  P2 }

{ P3
  P4 }

{ { P1
  P2
  OPTIONAL { P5 } }

{ P3
  P4
  OPTIONAL { P7 } }

} }
```
Optimization of Semantic Web Queries

2. Tools for the Semantic Web

SPARQL in a nutshell – SPARQL body

Body: Pattern matching expression

Interesting features:
- Grouping
- Optional parts
- Nesting
- Union of patterns
- Filtering
- Querying several sources

```
{ { P1
  P2
  OPTIONAL { P5 } }
{ P3
  P4
  OPTIONAL { P7
    OPTIONAL { P8 } } }
UNION
{ P9
  FILTER ( R ) }
```
Optimization of Semantic Web Queries

2. Tools for the Semantic Web

SPARQL in a nutshell – SPARQL body

**Body**: Pattern matching expression

Interesting features:
- Grouping
- Optional parts
- Nesting
- Union of patterns
- Filtering
- Querying several sources

We focus on the **pattern matching expressions**.

Reinhard Pichler Puebla, 19 May, 2013

A standard algebraic syntax

- **Triple patterns**: triples including variables from a set $V$
  
  ```sparql
  ?X ex:name 'wacker'
  ```

  ```sql
  (?X, name, 'wacker')
  ```

- **Graph patterns**: fully parenthesized algebra
  
  ```sparql
  { P1 P2 }
  ```

  ```sql
  (P1 AND P2)
  ```

  ```sparql
  { P1 OPTIONAL { P2 }}
  ```

  ```sql
  (P1 OPT P2)
  ```

  ```sparql
  { P1 UNION { P2 }
  ```

  ```sql
  (P1 UNION P2)
  ```

  ```sparql
  { P1 FILTER ( R ) }
  ```

  ```sql
  (P1 FILTER R)
  ```

original SPARQL syntax algebraic syntax

Setting considered here

Restrictions

- Consider only the body of SPARQL queries, i.e., SPARQL graph patterns.
- Consider only the operators AND and OPT.
- Further restrictions will be introduced below.

Reinhard Pichler Puebla, 19 May, 2013
Semantics of SPARQL: An example

Example

(tivoli, location, innsbruck)
(tivoli, poi, stadium)
(hanappi, location, wien)

((?X, location, ?Y) OPT (?X, poi, ?E))

<table>
<thead>
<tr>
<th>?X</th>
<th>?Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>tivoli</td>
<td>innsbruck</td>
</tr>
<tr>
<td>hanappi</td>
<td>wien</td>
</tr>
</tbody>
</table>
Semantics of SPARQL: An example

Example

(tivoli, location, innsbruck)
(tivoli, poi, stadium)
(hanappi, location, wien)

((?X, location, ?Y) OPT (?X, poi, ?E))

<table>
<thead>
<tr>
<th>?X</th>
<th>?Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>tivoli</td>
<td>innsbruck</td>
</tr>
<tr>
<td>hanappi</td>
<td>wien</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>?X</th>
<th>?E</th>
</tr>
</thead>
<tbody>
<tr>
<td>tivoli</td>
<td>stadium</td>
</tr>
</tbody>
</table>

Reinhard Pichler
Puebla, 19 May, 2013
**Optimization of Semantic Web Queries**

**2. Tools for the Semantic Web**

**SPARQL in a nutshell**

**Definition**

The EVALUATION problem for SPARQL graph patterns:

**INPUT:** An RDF dataset $G$, a graph pattern $P$ and a mapping $\mu$.

**QUESTION:** Is $\mu \in \llbracket P \rrbracket_G$? ($\llbracket P \rrbracket_G$ = the solutions of $P$ over $G$)

**Theorem (Pérez, Arenas, Gutierrez 2009)**

*EVALUATION* is PSPACE-complete.

**Theorem (Schmidt, Meier, Lausen 2010)**

*EVALUATION* remains PSPACE-complete if $P$ contains operators AND and OPT only.

**Observation**

- High complexity due to unrestricted occurrences of OPT-operator.
- Arbitrary use of OPT-operator is highly unintuitive.

**Example**

**graph:**

(fcw, home, tivoli), (fcw, league, 'Bundesliga'),

(hanappi, location, wien)

**graph pattern:**


**result:**

\{ {?C $\rightarrow$ fcw, $?S$ $\rightarrow$ tivoli} \}

---

**Optimization of Semantic Web Queries**

**2. Tools for the Semantic Web**

**Observation**

- High complexity due to unrestricted occurrences of OPT-operator.
- Arbitrary use of OPT-operator is highly unintuitive.

**Example**

**graph:**

(fcw, home, tivoli), (fcw, league, 'Bundesliga'),

(hanappi, location, wien)

**graph pattern:**


**result:**

\{ {?C $\rightarrow$ fcw, $?S$ $\rightarrow$ tivoli} \}
Observation

- High complexity due to unrestricted occurrences of OPT-operator.
- Arbitrary use of OPT-operator is highly unintuitive.

Example

Graph:
(\text{fcw}, \text{home}, \text{tivoli}), (\text{fcw}, \text{league}, \text{‘Bundesliga’}),
(\text{hanappi}, \text{location}, \text{wien})

Graph pattern:
(\text{?C}, \text{home}, \text{?S}) \text{ OPT } ((\text{?C}, \text{league}, \text{?L}) \text{ OPT } (\text{?S}, \text{location}, \text{?P}))

Result:
\{\{\text{?C} \rightarrow \text{fcw}, \text{?S} \rightarrow \text{tivoli}\}\}

Well-designed graph patterns

Definition
A query in the AND-FILTER-OPT fragment of SPARQL is well-designed if for every OPT in the pattern:

\[
( \cdots \cdots \cdots \cdots \cdots ( P \text{ OPT } Q ) \cdots \cdots \cdots )
\]

if a variable occurs inside Q
Well-designed graph patterns

**Definition**

A query in the AND-FILTER-OPT fragment of SPARQL is well-designed if for every OPT in the pattern:

\[
\begin{array}{c}
\text{( \cdots \cdots ( \ P \ OPT \ Q \ ) \ \cdots \cdots )} \\
\uparrow \quad \uparrow \quad \uparrow \quad \uparrow
\end{array}
\]

if a variable occurs inside \( Q \) and anywhere outside the \( OPT \) operator,

\[
\text{if a variable occurs inside } Q \text{ and anywhere outside the } OPT \text{ operator, then the variable must also occur inside } P.
\]

**Example**

\[
(?C, \text{home}, ?S) \ OPT \left( (?C, \text{league}, ?L) \ OPT \ (?S, \text{location}, ?P) \right)
\]
Well-designed graph patterns

Definition
A query in the AND-FILTER-OPT fragment of SPARQL is well-designed if for every OPT in the pattern:

\[
\begin{array}{c}
( \ldots \ldots \ldots \ldots ( P \text{ OPT } Q ) \ldots \ldots \ldots ) \\
\uparrow \quad \uparrow \quad \uparrow \quad \uparrow
\end{array}
\]

if a variable occurs inside \( Q \) and anywhere outside the OPT operator, then the variable must also occur inside \( P \).

Example

\((?C, \text{home}, ?S) \text{ OPT } ((?C, \text{league}, ?L) \text{ OPT } (?S, \text{location}, ?P))\)

\[
\uparrow \quad \uparrow
\]

How common are well-designed patterns?

What are real SPARQL queries like? [Picalausa and Vansummeren, 2011]
- DBpedia query log: 623,000 queries without UNION operator
- 52% of these queries are well-designed
How common are well-designed patterns?

What are real SPARQL queries like? [Picalausa and Vansummeren, 2011]
- DBpedia query log: 623,000 queries without UNION operator
- 52% of these queries are well-designed

Non-well-designed queries from DBpedia usually look unnatural.
- They often seem to be bugs rather than intended features.

Well-designed graph patterns

Theorem (Pérez, Arenas, Gutierrez 2009)

\textit{EVALUATION} is coNP-complete for well-designed graph pattern expressions constructed by using only AND and OPT operators.

The story so far

- Syntax of RDF
  - RDF triples $(s, p, o)$
  - underlying data model: labelled, directed graph
- Basic querying of RDF
  - RDF triple patterns
  - Mappings
- SPARQL
  - SPARQL graph patterns
  - restriction to operators OPT and AND
  - well-designed SPARQL
  - complexity of SPARQL
What comes next

- **Ultimate goal**: optimization of SPARQL queries
- learn from successful methods from the relational world
- first steps to apply analogous methods to SPARQL

---

**Outline**

1. Motivation
2. Tools for the Semantic Web
   2.1 RDF
   2.2 SPARQL
3. Role Model: Relational Algebra
4. Query plans and transformation rules for SPARQL
   4.1 Transformation rules for SPARQL
   4.2 Query Plans for SPARQL
5. Static Analysis of SPARQL Queries
6. Conclusion

---

**Relational Algebra**

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sigma_\alpha$</td>
<td>Selection</td>
</tr>
<tr>
<td>$\pi_{\overrightarrow{A}}$</td>
<td>Projection</td>
</tr>
<tr>
<td>$\rho_{A \leftarrow B}$</td>
<td>Renaming</td>
</tr>
<tr>
<td>$R_1 \times R_2$</td>
<td>Cross Product</td>
</tr>
<tr>
<td>$R_1 \bowtie R_1$</td>
<td>Join</td>
</tr>
<tr>
<td>$R_1 \setminus R_2$</td>
<td>Set Difference</td>
</tr>
<tr>
<td>$R_1 \cup R_2$</td>
<td>Union</td>
</tr>
<tr>
<td>$R_1 \cap R_2$</td>
<td>Intersection</td>
</tr>
</tbody>
</table>

---

**Example**

**Relations**: location(stadium, city), home(club, stadium), league(club, league).

**SQL Query**:

```
SELECT city
FROM location, home, league
WHERE location.stadium = home.stadium AND
home.club = league.club AND league.league = 'Bundesliga';
```

**Relational Algebra**:

```
\pi_{\text{location.city}}(\sigma_P(\text{location \times home \times league}))
```

with

$P := \text{location.stadium = home.stadium} \land \text{home.club = league.club} \land \text{league.league = 'Bundesliga'}$
Relational Algebra

Example
Query Plan:
\[ \pi_{\text{location.city}} \]
\[ \sigma_{\text{location.stadium}=\text{home.stadium} \land \text{home.club}=\text{league.club} \land \text{league.league}='Bundesliga'} \]
\[ \times \]
\[ \text{location} \]
\[ \times \]
\[ \text{home} \]
\[ \times \]
\[ \text{league} \]
Optimization of Semantic Web Queries

3. Role Model: Relational Algebra

Relational Algebra

Example

Consider the following database instance:

<table>
<thead>
<tr>
<th>location:</th>
<th>home:</th>
</tr>
</thead>
<tbody>
<tr>
<td>tivoli</td>
<td>fcw</td>
</tr>
<tr>
<td>innsbruck</td>
<td>tivoli</td>
</tr>
<tr>
<td>hanappi</td>
<td>scr</td>
</tr>
<tr>
<td>wien</td>
<td>hanappi</td>
</tr>
<tr>
<td>allianz</td>
<td>fcb</td>
</tr>
<tr>
<td>munchen</td>
<td>allianz</td>
</tr>
<tr>
<td>camp nou</td>
<td>barca</td>
</tr>
<tr>
<td>barcelona</td>
<td>barca</td>
</tr>
<tr>
<td>old trafford</td>
<td>manu</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>league:</th>
</tr>
</thead>
<tbody>
<tr>
<td>fcw</td>
</tr>
<tr>
<td>scr</td>
</tr>
<tr>
<td>fcb</td>
</tr>
<tr>
<td>barca</td>
</tr>
<tr>
<td>manu</td>
</tr>
</tbody>
</table>

Recall the Query Plan:

\[ \pi_{\text{location.city}} \left( \sigma_{\text{location.stadium}=\text{home.stadium} \land \text{home.club}=\text{league.club} \land \text{league.league}='\text{Bundesliga}'}(R_1) \right) \]

\[ \times \]

\[ \text{location} \]

\[ \text{home} \]

\[ \text{league} \]
Relational Algebra

Example
Recall the Query Plan:

\[ \pi_{location}.city \]
\[ \sigma_{location.stadium=home.stadium \land home.club=league.club \land league.league='Bundesliga'} \]
\[ \times \]
\[ \text{location: 5, home: 5, league: 5} \]
Relational Algebra

Example
Recall the Query Plan:

\[
\begin{align*}
\pi_{\text{location.city}} & \quad | \\
\sigma_{\text{league.league} = \text{Bundesliga}} & \quad | \\
\bowtie R_1.\text{club} = \text{league.club} & \quad | \\
\bowtie \text{location.stadium} = \text{home.stadium} & \quad | \\
\text{location: 5} & \quad \text{home: 5} & \quad \text{league: 5}
\end{align*}
\]
Relational Algebra

Example

Query Plan:

\[ \pi_{\text{location.city}} \]

\[ \bowtie_{R_1.\text{club}=\text{league.club}} \]

\[ \sigma_{\text{league.\text{league}='Bundesliga'}}, \text{ \text{fcw}}, \text{ \text{'Bundesliga'}}, \text{ \text{'Bundesliga'}} \]

\[ \bowtie_{\text{location.stadium}=\text{home.stadium}} \]

\[ \text{tivoli}, \text{innsbruck}, \text{fcb}, \text{hanappi}, \text{wien}, \text{scr}, \text{allianz}, \text{münchen}, \text{fcb}, \text{camp nou}, \text{barcelona}, \text{barca}, \text{old trafford}, \text{manchester}, \text{manu}, \text{camp nou}, \text{barcelona}, \text{barca}, \text{old trafford}, \text{manchester}, \text{manu} \]

\[ \text{location}, \text{home}, \text{league} \]

Lessons learned from the relational world

We need . . .

- an appropriate data structure for representing queries: simply use expression trees
- a semantics to directly use these trees for query evaluation: consider the trees as Query Execution Plans
- equivalence preserving transformations

Conclusion: We also need an appropriate algebra for SPARQL!
Lessons learned from the relational world

We need . . .

- an appropriate data structure for representing queries: simply use expression trees
- a semantics to directly use these trees for query evaluation: consider the trees as Query Execution Plans
- equivalence preserving transformations

Conclusion: We also need an appropriate algebra for SPARQL!

Normal form of well-designed graph patterns

OPT normal form [Pérez, Arenas, Gutierrez 2009]

- a basic graph pattern is in OPT-normal form
- \((P_1 \text{ OPT } P_2)\) is in OPT normal form if \(P_1\) and \(P_2\) are.

Intuition: no OPT-operator is in the scope of an AND-operator.

Rewrite rules to obtain OPT normal form

- \((P1 \text{ AND } (P2 \text{ OPT } P3)) \rightarrow ((P1 \text{ AND } P2) \text{ OPT } P3)\)
- \(((P1 \text{ OPT } P2) \text{ AND } P3) \rightarrow ((P1 \text{ AND } P3) \text{ OPT } P2)\)
Normal form of well-designed graph patterns

OPT normal form [Pérez, Arenas, Gutierrez 2009]

- a basic graph pattern is in OPT-normal form
- \((P_1 \text{ OPT } P_2)\) is in OPT normal form if \(P_1\) and \(P_2\) are.

Intuition: no OPT-operator is in the scope of an AND-operator.

Rewrite rules to obtain OPT normal form

- \((P_1 \text{ AND } (P_2 \text{ OPT } P_3))\) \(\rightarrow\) \(((P_1 \text{ AND } P_2) \text{ OPT } P_3)\)
- \(((P_1 \text{ OPT } P_2) \text{ AND } P_3)\) \(\rightarrow\) \(((P_1 \text{ AND } P_3) \text{ OPT } P_2)\)

Property [Pérez, Arenas, Gutierrez 2009]

For well-designed SPARQL patterns:
\(((P_1 \text{ OPT } P_2) \text{ OPT } P_3)\) \(\equiv\) \(((P_1 \text{ OPT } P_3) \text{ OPT } P_2)\)

Optimization of Semantic Web Queries 4. Query plans and transformation rules for SPARQL

Nesting of OPTs

\(((P_1 \text{ OPT } P_2) \text{ OPT } P_3)\)

- First try to extend solutions of \(P_1\) to solutions of \(P_2\).
- Try to extend the resulting mappings to solutions of \(P_3\).
- As seen above, we could change this order.
- \(\Rightarrow\) \(P_2\) and \(P_3\) are on the “same level”.

\((P_1 \text{ OPT } (P_2 \text{ OPT } P_3))\)

- First try to extend solutions of \(P_2\) to solutions of \(P_3\).
- Then try to extend the solutions of \(P_1\) to these mappings.
- Solutions of \(P_1\) are extended to solutions of \(P_3\) only if we first can extend the solutions of \(P_1\) to \(P_2\).
- \(\Rightarrow\) \(P_2\) is “one level above” \(P_3\).
Pattern trees

Consider a pattern in OPT-normal form

\[((?S, poi, stadium) \text{AND} (?C, home, ?S) \text{AND} (?C, league, 'Bundesliga'))\]

\[((?S, location, ?L) \text{AND} (?U, category, university) \text{AND} (?U, location, ?L))\]

\[((?A, location, ?L) \text{AND} (?S, location, ?L) \text{AND} (?A, poi, ?I))\]

\[((?S, offers, visitor_center) \text{AND} (?S, ticketprice, ?P))\]

\[((?S, phone, ?N))\]

Pattern trees as query plans

\{{(?S, poi, stadium), (?C, home, ?S), (?C, league, 'Bundesliga')}}

\{{(?S, location, ?L), (?U, category, university), (?U, location, ?L)}\}

\{{(?A, location, ?L), (?S, location, ?L), (?A, poi, ?I)}\}

\{{(?S, offers, visitor_center), (?S, ticketprice, ?P)}\}

\{{(?S, phone, ?N)}\}

Pattern trees as query plans – Bottom Up Evaluation

Example RDF graph

(louvre, location, paris), (louvre, poi, museum), (eiffelt., location, paris), (eiffelt., poi, steel),
(g.cathedral, location, galway), (g.cathedral, poi, church), (santaluciahill, location, santiago),
(santaluciahill, poi, hill), (albertina, location, wien), (albertina, poi, museum),
(st.jakob, location, innsbruck), (st.jakob, poi, church), (UPEC, category, university),
(UPE, category, university), (NUIG, category, university), (NUIG, location, galway),
(PUC, category, university), (PUC, location, santiago), (TUW, category, university),
(TUW, location, wien), (UIBK, category, university),
(tivoli, poi, stadium), (tivoli, location, innsbruck), (fcw, home, tivoli), (fcw, league, 'Bundesliga'),
(hanappi, poi, stadium), (hannapi, location, wien), (scr, home, hanappi),
(scr, league, 'Bundesliga'), (pappel, poi, stadium), (pappel, location, mattersburg),
(svm, home, pappel), (svm, league, 'Bundesliga'),
(hanappi, offers, visitor_center), (eiffelt., offers, visitor_center),
(louvre, ticketprice, '39'),
(albertina, ticketprice, '11'), (hannapi, ticketprice, '6'), (tivoli, ticketprice, '20'),
(PUC, phone, nr. 1), (TUW, phone, nr. 2), (albertina, phone, nr. 3),
(hannapi, phone, nr. 4), (hannapi, phone, nr. 5), (pappel, phone, nr. 6),
Pattern trees as query plans – Bottom Up Evaluation

\[
\begin{align*}
\{(S, \text{poi}, \text{stadium}), (C, \text{home}, S), (C, \text{league}, 'Bundesliga')\} \\
\{(S, \text{location}, L), (U, \text{category}, \text{university}), (U, \text{location}, L)\} \\
\{(A, \text{location}, L), (S, \text{location}, L), \{(S, \text{offers}, \text{visitor_center})\} \\
\{(S, \text{ticketprice}, P)\} \\
\{(S, \text{phone}, N)\}
\end{align*}
\]

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>PUC</td>
<td>nr. 1</td>
</tr>
<tr>
<td>TUW</td>
<td>nr. 2</td>
</tr>
<tr>
<td>albertina</td>
<td>nr. 3</td>
</tr>
<tr>
<td>louvre</td>
<td>nr. 4</td>
</tr>
<tr>
<td>hannapi</td>
<td>nr. 5</td>
</tr>
<tr>
<td>pappel</td>
<td>nr. 6</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
\{(S, \text{ticketprice}, P)\} \\
\{(S, \text{phone}, N)\}
\end{align*}
\]
Pattern trees as query plans – Bottom Up Evaluation

\(?S\) \(?P\) \{(?S, ticketprice, ?P)\}


\{(?S, location, ?L), (?U, category, university), (?U, location, ?L)\}

\{(?A, location, ?L), (?S, location, ?L), (?A, poi, ?I)\}

\{(?S, offers, visitor_center)\}

\{(?S, ticketprice, ?P), (?S, phone, ?N)\}
Pattern trees as query plans – Bottom Up Evaluation

\[((?S, offers, visitor\_center), (hanappi, offers, visitor\_center), (eiffelt, offers, visitor\_center), \ldots)\]

```
?S
louvre
hanappi
eiffelt.
```

\{(?S, offers, visitor\_center)\}

---


\{(?S, location, ?L), (?U, category, university), (?U, location, ?L)\}

\{(?A, location, ?L), (?S, location, ?L), (?A, poi, ?I)\}

\{(?S, offers, visitor\_center)\}

\{(?S, ticketprice, ?P)\}

\{(?S, phone, ?N)\}

---

\{(?S, ticketprice, ?P)\}

\{(?S, phone, ?N)\}

---

\{(?S, offers, visitor\_center)\}

\{(?S, ticketprice, ?P)\}

\{(?S, phone, ?N)\}
Pattern trees as query plans – Bottom Up Evaluation


\{(?S, location, ?L), (?U, category, university), (?U, location, ?L)\}

\{(?A, location, ?L), (?S, location, ?L), (?A, poi, ?I)\}

\{(?S, offers, visitor_center)\}

\{(?S, ticketprice, ?P)\}

\{(?S, phone, ?N)\}
Pattern trees as query plans – Bottom Up Evaluation

{(louvre, location, paris), (louvre, poi, museum), (eiffelt., location, paris), (eiffelt., poi, steel), . . . .
. . (st.jakob, location, innsbruck), (st.jakob, poi, church), . . . .
(tivoli, location, innsbruck), . . . . . .

{louvre, location, paris), (louvre, poi, museum), (eiffelt., location, paris), (eiffelt., poi, steel), . . . .
. . (st.jakob, location, innsbruck), (st.jakob, poi, church), . . . .
(tivoli, location, innsbruck), . . . . . .

{(louvre, location, paris), (eiffelt., location, paris), (eiffelt., poi, steel), . . . .
. . (st.jakob, location, innsbruck), (st.jakob, poi, church), . . . .
(tivoli, location, innsbruck), . . . . . .

{(louvre, location, paris), (eiffelt., location, paris), (eiffelt., poi, steel), . . . .
. . (st.jakob, location, innsbruck), (st.jakob, poi, church), . . . .
(tivoli, location, innsbruck), . . . . . .
Pattern trees as query plans – Bottom Up Evaluation

(louvre, location, paris), . . . (g.cathedral, location, galway), . . . .
(UPEC, category, university), (UPEC, location, paris), (NUIG, category, university),
(NUIG, location, galway), . . . (UIBK, category, university), (UIBK, location, innsbruck), .
tivoli, location, innsbruck

{| ?S | ?L | ?U |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>louvre</td>
<td>paris</td>
<td>UPEC</td>
</tr>
<tr>
<td>UPEC</td>
<td>paris</td>
<td>UPEC</td>
</tr>
<tr>
<td>eiffelt.</td>
<td>paris</td>
<td>UPEC</td>
</tr>
<tr>
<td>g.cathedral</td>
<td>galway</td>
<td>NUIG</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>tivoli</td>
<td>innsbruck</td>
<td>UIBK</td>
</tr>
</tbody>
</table>

{{ ?S, location, ?L },
{ ?U, category, university },
{ ?U, location, ?L } }

{{ ?S, poi, stadium },
{ ?C, home, ?S },
{ ?C, league, ‘Bundesliga’ } }

{{ ?A, location, ?L },
{ ?S, location, ?L },
{ ?A, poi, ?I } }

{{ ?S, offers, visitor_center } }

{{ ?S, ticketprice, ?P } }

{{ ?S, phone, ?N } }

{| ?S | ?L | ?U |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>louvre</td>
<td>louvre</td>
<td>paris</td>
</tr>
<tr>
<td>UPEC</td>
<td>UPEC</td>
<td>UPEC</td>
</tr>
<tr>
<td>eiffelt.</td>
<td>eiffelt.</td>
<td>paris</td>
</tr>
<tr>
<td>g.cathedral</td>
<td>g.cathedral</td>
<td>NUIG</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>st.jakob</td>
<td>tivoli</td>
<td>innsbruck</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>louvre</td>
<td>paris</td>
<td>UPEC</td>
<td>louvre</td>
<td>museum</td>
</tr>
<tr>
<td>UPEC</td>
<td>paris</td>
<td>UPEC</td>
<td>eiffelt.</td>
<td>paris</td>
</tr>
<tr>
<td>eiffelt.</td>
<td>eiffelt.</td>
<td>paris</td>
<td>steel</td>
<td></td>
</tr>
<tr>
<td>g.cathedral</td>
<td>galway</td>
<td>NUIG</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
| ... | ... | ... | ... | ...
| st.jakob | tivoli | innsbruck | church |
Pattern trees as query plans – Bottom Up Evaluation

\[
\{(S, poi, stadium), (C, home, S), (C, league, ‘Bundesliga’)\}
\]

\[
\{(S, location, L), (U, category, university), (U, location, L)\}
\]

\[
\{(A, location, L), (S, location, L), (A, poi, I)\}
\]

\[
\{(S, offers, visitor_center)\}
\]

\[
\{(S, ticketprice, P)\} \quad \{(S, phone, N)\}
\]
Pattern trees as query plans – Bottom Up Evaluation

{(tivoli, poi, stadium), (fcw, home, tivoli), (fcw, league, ‘Bundesliga’), (hanappi, poi, stadium), (scr, home, hanappi), (scr, league, ‘Bundesliga’), (pappel, poi, stadium), (svm, home, pappel), (svm, league, ‘Bundesliga’)}

{(tivoli, poi, stadium), (fcw, home, tivoli), (fcw, league, ‘Bundesliga’), (hanappi, poi, stadium), (scr, home, hanappi), (scr, league, ‘Bundesliga’), (pappel, poi, stadium), (svm, home, pappel), (svm, league, ‘Bundesliga’)}

{(tivoli, poi, stadium), (fcw, home, tivoli), (fcw, league, ‘Bundesliga’), (hanappi, poi, stadium), (scr, home, hanappi), (scr, league, ‘Bundesliga’), (pappel, poi, stadium), (svm, home, pappel), (svm, league, ‘Bundesliga’)}

{(tivoli, poi, stadium), (fcw, home, tivoli), (fcw, league, ‘Bundesliga’), (hanappi, poi, stadium), (scr, home, hanappi), (scr, league, ‘Bundesliga’), (pappel, poi, stadium), (svm, home, pappel), (svm, league, ‘Bundesliga’)}

{(tivoli, poi, stadium), (fcw, home, tivoli), (fcw, league, ‘Bundesliga’), (hanappi, poi, stadium), (scr, home, hanappi), (scr, league, ‘Bundesliga’), (pappel, poi, stadium), (svm, home, pappel), (svm, league, ‘Bundesliga’)}
Pattern trees as query plans – Bottom Up Evaluation

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>tivoli</td>
<td>fcw</td>
<td>innsbruck</td>
<td>UIBK</td>
<td>tivoli</td>
<td>stadium</td>
<td></td>
<td></td>
</tr>
<tr>
<td>hanappi</td>
<td>scr</td>
<td>wien</td>
<td>TUW</td>
<td>hanappi</td>
<td>stadium ‘6’ nr. 5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pappel</td>
<td>svm</td>
<td>mattersburg</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Result of Bottom-up evaluation

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>tivoli</td>
<td>fcw</td>
<td>innsbruck</td>
<td>UIBK</td>
<td>tivoli</td>
<td>stadium</td>
<td></td>
<td></td>
</tr>
<tr>
<td>hanappi</td>
<td>scr</td>
<td>wien</td>
<td>TUW</td>
<td>hanappi</td>
<td>stadium ‘6’ nr. 5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pappel</td>
<td>svm</td>
<td>mattersburg</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Alternative Evaluation: Top-Down Evaluation

Observation

- Bottom-up evaluation starts at the leaves.
- Mappings computed for some node can possibly not be extended upwards.
- The natural direction for extending graph patterns is downwards.
- Idea: choose a top-down evaluation instead.
Pattern trees as query plans – Top Down Evaluation

\{(\textcolor{red}{?S}, \textcolor{blue}{\text{pois}}, \textcolor{black}{\text{stadium}}), (\textcolor{blue}{\text{C}}, \textcolor{red}{\text{home}}, \textcolor{red}{?S}), (\textcolor{blue}{\text{C}}, \textcolor{black}{\text{league}}, \textcolor{black}{\text{\textquote{\textquoteright\textbf{Bundesliga}}}}))\}

\{(\textcolor{red}{?S}, \textcolor{blue}{\text{location}}, \textcolor{blue}{?L}), (\textcolor{blue}{\text{U}}, \textcolor{black}{\text{category}}, \textcolor{black}{\text{university}}), (\textcolor{blue}{\text{U}}, \textcolor{black}{\text{location}}, \textcolor{blue}{\text{?L}})\}

\{(\textcolor{red}{?A}, \textcolor{blue}{\text{location}}, \textcolor{blue}{?L}), (\textcolor{red}{?S}, \textcolor{blue}{\text{location}}, \textcolor{blue}{?L}), (\textcolor{red}{?A}, \textcolor{blue}{\text{pois}}, \textcolor{blue}{?I}))\}

\{(\textcolor{red}{?S}, \textcolor{blue}{\text{offers}}, \textcolor{blue}{\text{visitor}}\textcolor{black}{\text{\_center}})\}

\{(\textcolor{red}{?S}, \textcolor{blue}{\text{ticketprice}}, \textcolor{blue}{?P})\}

\{(\textcolor{red}{?S}, \textcolor{blue}{\text{phone}}, \textcolor{blue}{?N})\}

Optimization of Semantic Web Queries

4. Query plans and transformation rules for SPARQL

Pattern trees as query plans – Top Down Evaluation

\{(\textcolor{red}{?S}, \textcolor{blue}{\text{pois}}, \textcolor{black}{\text{stadium}}), (\textcolor{blue}{\text{C}}, \textcolor{red}{\text{home}}, \textcolor{red}{?S}), (\textcolor{blue}{\text{C}}, \textcolor{black}{\text{league}}, \textcolor{black}{\text{\textquote{\textquoteright\textbf{Bundesliga}}}}))\}

\{(\textcolor{red}{?S}, \textcolor{blue}{\text{location}}, \textcolor{blue}{?L}), (\textcolor{blue}{\text{U}}, \textcolor{black}{\text{category}}, \textcolor{black}{\text{university}}), (\textcolor{blue}{\text{U}}, \textcolor{black}{\text{location}}, \textcolor{blue}{\text{?L}})\}

\{(\textcolor{red}{?A}, \textcolor{blue}{\text{location}}, \textcolor{blue}{?L}), (\textcolor{red}{?S}, \textcolor{blue}{\text{location}}, \textcolor{blue}{?L}), (\textcolor{red}{?A}, \textcolor{blue}{\text{pois}}, \textcolor{blue}{?I}))\}

\{(\textcolor{red}{?S}, \textcolor{blue}{\text{offers}}, \textcolor{blue}{\text{visitor}}\textcolor{black}{\text{\_center}})\}

\{(\textcolor{red}{?S}, \textcolor{blue}{\text{ticketprice}}, \textcolor{blue}{?P})\}

\{(\textcolor{red}{?S}, \textcolor{blue}{\text{phone}}, \textcolor{blue}{?N})\}
Pattern trees as query plans – Top Down Evaluation

\{(\texttt{?A, location, ?L}), (\texttt{?S, location, ?L}), (\texttt{?A, poi, ?I})\}

\[
\begin{array}{c|c|c|c|c}
\hline
\texttt{tivoli} & \texttt{innsbruck} & \texttt{UIBK} & \texttt{fcw} \\
\texttt{hanappi} & \texttt{wien} & \texttt{TUW} & \texttt{scr} \\
\texttt{pappel} & & & \texttt{svm} \\
\end{array}
\]

\texttt{Reinhard Pichler}

Puebla, 19 May, 2013
Pattern trees as query plans – Top Down Evaluation


\{(?S, location, ?L), (?U, category, university), (?U, location, ?L)\}

\{(?A, location, ?L), (?S, location, ?L), (?A, poi, ?I)\}

\{(?S, ticketprice, ?P)\} \{(?S, phone, ?N)\}

\{(?S, offers, visitor_center)\}
Pattern trees as query plans – Top Down Evaluation

\{(??S, ticketprice, ?P)\}

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>tivoli</td>
<td>stadium</td>
<td>tivoli</td>
<td>innsbruck</td>
<td>TUW</td>
<td>scr</td>
</tr>
<tr>
<td>st.jakob</td>
<td>church</td>
<td>tivoli</td>
<td>innsbruck</td>
<td>TUW</td>
<td>fcw</td>
</tr>
<tr>
<td>hanappi</td>
<td>stadium</td>
<td>hanappi</td>
<td>wien</td>
<td>TUW</td>
<td>scr</td>
</tr>
<tr>
<td>albertina</td>
<td>museum</td>
<td>hanappi</td>
<td>wien</td>
<td>TUW</td>
<td>scr</td>
</tr>
<tr>
<td>pappel</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>svm</td>
</tr>
</tbody>
</table>

\{(??S, ticketprice, ?P)\}

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>tivoli</td>
<td>stadium</td>
<td>tivoli</td>
<td>innsbruck</td>
<td>TUW</td>
<td>scr</td>
</tr>
<tr>
<td>st.jakob</td>
<td>church</td>
<td>tivoli</td>
<td>innsbruck</td>
<td>TUW</td>
<td>fcw</td>
</tr>
<tr>
<td>hanappi</td>
<td>stadium</td>
<td>hanappi</td>
<td>wien</td>
<td>TUW</td>
<td>scr</td>
</tr>
<tr>
<td>albertina</td>
<td>museum</td>
<td>hanappi</td>
<td>wien</td>
<td>TUW</td>
<td>scr</td>
</tr>
<tr>
<td>pappel</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>svm</td>
</tr>
</tbody>
</table>

\{(??S, phone, ?P)\}

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>tivoli</td>
<td>stadium</td>
<td>tivoli</td>
<td>innsbruck</td>
<td>TUW</td>
<td>scr</td>
</tr>
<tr>
<td>st.jakob</td>
<td>church</td>
<td>tivoli</td>
<td>innsbruck</td>
<td>TUW</td>
<td>fcw</td>
</tr>
<tr>
<td>hanappi</td>
<td>stadium</td>
<td>hanappi</td>
<td>wien</td>
<td>TUW</td>
<td>scr</td>
</tr>
<tr>
<td>albertina</td>
<td>museum</td>
<td>hanappi</td>
<td>wien</td>
<td>TUW</td>
<td>scr</td>
</tr>
<tr>
<td>pappel</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>svm</td>
</tr>
</tbody>
</table>

\{(??S, phone, ?P)\}

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>tivoli</td>
<td>stadium</td>
<td>tivoli</td>
<td>innsbruck</td>
<td>TUW</td>
<td>scr</td>
</tr>
<tr>
<td>st.jakob</td>
<td>church</td>
<td>tivoli</td>
<td>innsbruck</td>
<td>TUW</td>
<td>fcw</td>
</tr>
<tr>
<td>hanappi</td>
<td>stadium</td>
<td>hanappi</td>
<td>wien</td>
<td>TUW</td>
<td>scr</td>
</tr>
<tr>
<td>albertina</td>
<td>museum</td>
<td>hanappi</td>
<td>wien</td>
<td>TUW</td>
<td>scr</td>
</tr>
<tr>
<td>pappel</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>svm</td>
</tr>
</tbody>
</table>

\{(??S, phone, ?P)\}

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>tivoli</td>
<td>stadium</td>
<td>tivoli</td>
<td>innsbruck</td>
<td>TUW</td>
<td>scr</td>
</tr>
<tr>
<td>st.jakob</td>
<td>church</td>
<td>tivoli</td>
<td>innsbruck</td>
<td>TUW</td>
<td>fcw</td>
</tr>
<tr>
<td>hanappi</td>
<td>stadium</td>
<td>hanappi</td>
<td>wien</td>
<td>TUW</td>
<td>scr</td>
</tr>
<tr>
<td>albertina</td>
<td>museum</td>
<td>hanappi</td>
<td>wien</td>
<td>TUW</td>
<td>scr</td>
</tr>
<tr>
<td>pappel</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>svm</td>
</tr>
</tbody>
</table>
The story so far

- RDF
- SPARQL
- restriction to well-designed SPARQL
- OPT normal form
- Pattern trees
  - representation of well-designed SPARQL queries
  - useful as query execution plans
  - "top-down" instead of "bottom-up" evaluation

Next step: transformations of pattern trees

---

Example

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>tivoli</td>
<td>stadium</td>
<td>tivoli</td>
<td>innsbruck</td>
<td>UIBK</td>
<td>fcw</td>
<td></td>
<td></td>
</tr>
<tr>
<td>st.jakob</td>
<td>church</td>
<td>tivoli</td>
<td>innsbruck</td>
<td>UIBK</td>
<td>fcw</td>
<td></td>
<td></td>
</tr>
<tr>
<td>hanappi</td>
<td>stadium</td>
<td>hanappi</td>
<td>wien</td>
<td>TUW</td>
<td>scr '6'</td>
<td></td>
<td></td>
</tr>
<tr>
<td>albertina</td>
<td>museum</td>
<td>hanappi</td>
<td>wien</td>
<td>TUW</td>
<td>scr '6'</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>pappel</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>svm</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>tivoli</td>
<td>stadium</td>
<td>tivoli</td>
<td>innsbruck</td>
<td>UIBK</td>
<td>fcw</td>
<td></td>
<td></td>
</tr>
<tr>
<td>st.jakob</td>
<td>church</td>
<td>tivoli</td>
<td>innsbruck</td>
<td>UIBK</td>
<td>fcw</td>
<td></td>
<td></td>
</tr>
<tr>
<td>hanappi</td>
<td>stadium</td>
<td>hanappi</td>
<td>wien</td>
<td>TUW</td>
<td>scr '6'</td>
<td></td>
<td></td>
</tr>
<tr>
<td>albertina</td>
<td>museum</td>
<td>hanappi</td>
<td>wien</td>
<td>TUW</td>
<td>scr '6'</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>pappel</td>
<td></td>
<td></td>
<td></td>
<td>svm</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Transformation Rules for Pattern Trees

\[
\{(S, \text{location}, L), (C, \text{category}, \text{university}), (U, \text{location}, L)\}
\]

\[
\{(C, \text{home}, S), (C, \text{league}, \text{‘Bundesliga’})\}
\]

\[
\{(S, \text{ticketprice}, P), (S, \text{phone}, N)\}
\]

\[
\{(A, \text{location}, L), (S, \text{location}, L), (A, \text{poi}, I)\}
\]

\[
\{(S, \text{offers}, \text{visitor center})\}
\]

Reinhard Pichler
Puebla, 19 May, 2013
Transformation Rules for Pattern Trees


{(?S, location, ?L), (?U, category, university), (?U, location, ?L),
 (?A, location, ?L), (?A, poi, ?I)}

{(?S, offers, visitor_center)}

{(?S, ticketprice, ?P)}

{(?S, phone, ?N)}

R1 If a triple pattern \( t \) belongs to node \( n \) and to a
descendant \( n' \) of \( n \), then delete \( t \) from \( n' \).

R2 If node \( n \) does not introduce any new variable
w.r.t. its ancestors, then push copies of \( n \) into its
children.

R3 If there is a homomorphism from node \( n \) to the
branch from the root to the parent of \( n \), then merge
node \( n \) with its parent.
SPARQL: Subsumption

Definition
Let $P_1$ and $P_2$ be two SPARQL graph patterns. We say that $P_1$ is subsumed by $P_2$ ($P_1 \subseteq P_2$) if, for every RDF graph $G$, every solution of $P_1$ over $G$ can be extended to a solution of $P_2$ over $G$.

In case of SPARQL, subsumption is the more natural measure.

Example
- $P_1 = (\exists C, n, ?N)$
- $P_2 = (\exists C, n, ?N) \text{OPT}(\exists C, h, ?H)$
- $[P_1]_G = \{ \mu \}$ with $\mu = \{ ?C \rightarrow \text{fcw}, \exists N \rightarrow \text{wacker} \}$
- $[P_2]_G = \{ \mu' \}$ with $\mu' = \{ ?C \rightarrow \text{fcw}, \exists N \rightarrow \text{wacker}, \exists H \rightarrow \text{tivoli} \}$
- $G = \{ (\text{fcw}, n, \text{wacker}), (\text{fcw}, h, \text{tivoli}) \}$
- $P_1 \not\subseteq P_2$
Relationship: Subsumption and Equivalence

**Theorem**

Let $P_1$ and $P_2$ be two SPARQL graph patterns.
Then $P_1 \equiv P_2$ if and only if $P_1 \sqsubseteq P_2$ and $P_2 \sqsubseteq P_1$.

**Proof idea.**

$\Rightarrow$) Trivial

$\Leftarrow$) Assume $\mu \in \llbracket P_1 \rrbracket_G$ with $\mu \notin \llbracket P_2 \rrbracket_G$.

- By $P_1 \sqsubseteq P_2$, there is an extension $\mu_2$ of $\mu$, s.t. $\mu_2 \in \llbracket P_2 \rrbracket_G$.
- Since $\mu \notin \llbracket P_2 \rrbracket_G$, $\mu_2$ is a proper extension of $\mu$.
- By $P_2 \sqsubseteq P_1$, there is an extension $\mu_1$ of $\mu$, s.t. $\mu_1 \in \llbracket P_1 \rrbracket_G$.
- Hence, $\mu_1$ is a proper extension of $\mu$, s.t. $\mu_1 \in \llbracket P_1 \rrbracket_G$.
- Contradiction to $\mu \in \llbracket P_1 \rrbracket_G$.

Complexity of Subsumption

**Definition**

Let SUBSUMPTION be the following problem:

INPUT: Two SPARQL graph patterns $P_1$ and $P_2$.

QUESTION: Does $P_1 \sqsubseteq P_2$ hold?

**Theorem**

For well-designed SPARQL graph patterns, SUBSUMPTION is $\Pi_2^P$-complete.

Complexity of Equivalence

**Definition**

Let EQUIVALENCE be the following problem:

INPUT: SPARQL graph patterns $P_1$ and $P_2$.

QUESTION: Does $P_1 \equiv P_2$ hold?

**Theorem**

For well-designed SPARQL graph patterns, EQUIVALENCE is NP-complete.
Complexity of Equivalence

Proof idea.

- pattern trees in normal form w.r.t. rules R1, R2, and R3.
- necessary conditions for equivalence
  - the two pattern trees must have the same set of variables
  - they must have identical roots
- define an appropriate notion of homomorphism
- give an equivalence criterion based on the existence of certain homomorphisms (between nodes with common new variables)

Conclusion

Evaluation of SPARQL graph patterns
- pattern trees: an abstract representation formalism for well-designed SPARQL graph patterns
- top-down evaluation of pattern trees
- optimization and transformation rules on pattern trees

Fundamental computational problems
- SUBSUMPTION is $\Pi_2^P$-complete for well-designed SPARQL graph patterns.
- EQUIVALENCE is NP-complete for well-designed SPARQL graph patterns.

Future Work

- Extend work on well-designed SPARQL
  - Identify further transformation rules
  - Verify usefulness of transformation rules for optimization
- Consider extensions of well-designed SPARQL
  - Extend representation formalism and transformation rules to more operators, especially UNION and FILTER
  - Consider SPARQL 1.1 features, e.g., entailment regimes
  - Consider bag semantics