The 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, James Hendler and Ora Lassila, Scientific American, May 2001

The Web was designed as an information space, with the goal that it should be useful not only for human-human communication, but also that machines would be able to participate and help. One of the major obstacles to this has been the fact that most information on the Web is designed for human consumption, and even if it was derived from a database with well defined meanings (in at least some terms) for its columns, that the structure of the data is not evident to a robot browsing the web. Leaving aside the artificial intelligence problem of training machines to behave like people, the Semantic Web approach instead develops languages for expressing information in a machine-processable form.

Tim Berners-Lee, Semantic Web Roadmap, 1998

More Technically ...

- Vision: Web data will entail semantics in a manner that is understood (and processed, and linked) automatically by computers
- The “Semantic Web” is the technical infrastructure
  - Unambiguous names for resources that may also bind data to real world objects (URI)
  - A common data model to describe/link resources (RDF)
  - Expressive language to access to data (SPARQL)
  - Languages for defining common vocabularies and rules for automated reasoning (e.g., politician is person) (RDFS, OWL)
- Data providers should collaborate: properly publish their data and link it to existing data

RDF and SPARQL Recommendations

- RDF
  - “Resource Description Framework”
  - 1997: first public draft; 1999: recommendation (revised 2004); 2014: RDF1.1
- SPARQL
  - Query language for RDF
  - “SPARQL Protocol and RDF Query Language”
  - Yes, recursive definition
  - 2004: first public draft; 2008: recommendation; 2013: SPARQL1.1

Some Freely Available RDF Repositories

- DBPedia (~1.2b triples)
  - “A crowd-sourced community effort to extract structured information from Wikipedia and make this information available on the Web”
- Freebase (~340m triples)
  - “A community-curated DB of well-known people, places, and things”
- DBLP (~58m triples)
  - Computer science bibliography
- WordNet (~3m triples)
  - English lexical db: synonyms, antonyms, POS, ...
- GeoNames (~94m triples)
  - “Covers all countries, contains over eight million placenames”
- Yago (~120m triples)
  - Information from Wikipedia, Wordnet, GeoNames

There’s More! Linked!

http://lod-cloud.net/
RDF Example from DBPedia

In XML Syntax

Querying (SPARQL)

The sameAs Property

RDF Components

Additional Quotes from Scientific A. 05/01
Ontologies

• Originally, a philosophical discipline
  – Science of Being (Aristotle, Metaphysics, IV, 1)
  – What characterizes being? Eventually, what is being?
    – An ontology is a formal specification of a shared conceptualization (T. R. Gruber)
• Practically speaking, in the world of Semantic Web:
  – A public collection of entity and relationship names
  – An agreed-upon meaning (usually based on textual descriptions and context, e.g., HTML table or furniture table)
  – Identified by a known namespace URI
  – Rules on behavior (e.g., FOL)

Public Ontologies

• Best practice is to define data using existing ontologies
• Famous examples:
  – Dublin Core (media, Web resources)
  – FOAF: Friend of a Friend (people/social)
  – The Music Ontology
  – Good Relations (e-commerce)
    • Used by Best Buy for their RDF catalog

Main Lecture Resources

• Lecture notes by Marcelo Arenas
  – RDF y SPARQL: Dos componentes básicos para la Web de datos
• Lecture notes by Yaron Kanza
  – The Semantic Web

Outline

• Introduction
  ▶ Uniform Resource Identifiers
  • RDF
  • SPARQL
    ▶ Formal Definition
  • RDF Schema

URI

• Uniform Resource Identifier
• Syntax has a specification in W3C
  – RFC 2396
• A URL is a URI that points to a location on the network (e.g., server on the network)
  – e.g.: http://www.admin.technion.ac.il/labbooks/emptycode.asp
  – A general URI needs not be an actual URL
    • e.g., urn:isbn:0-486-27557-4
• But, RDF requires URI to be dereferenceable
  – That is, they should point to a physical description of the entity they represent

Relevant Quotes from S. A. 05/2001

• Because RDF uses URIs to encode this information in a document, the URIs ensure that concepts are not just words in a document but are tied to a unique definition that everyone can find on the Web.
• For example, imagine that we have access to a variety of databases with information about people, including their addresses. If we want to find people living in a specific zip code, we need to know which fields in each database represent names and which represent zip codes.
• RDF can specify that “(field 5 in database A) (is a field of type) (zip code),” using URIs rather than phrases for each term.
Example Revisited

Terminology

Namespaces in Other Formats

Outline
An RDF graph is a set of triples

- Example:
  - (dbr:technion, dbp:president, dbr:Lavie)
  - (dbr:technion, dbp:students, 13253)

Formally, we assume two sets:
- \( U \) is a set of URIs,
- \( L \) is a set of literals

An RDF triple is a member of \( U \times U \times (U \cup L) \)

An RDF graph is a finite set of RDF triples

---

Our Model is a Simplification

- We do not represent blank nodes
  - These are nodes without URIs
- We do not represent datatypes
  - e.g., integer, float, string, date, ...
- Further reading for interest: RDF books, online tutorials, W3C

---

Running Example: RDF Graph

<table>
<thead>
<tr>
<th>URI</th>
<th>Literal</th>
</tr>
</thead>
<tbody>
<tr>
<td>dbr:Nate_Huffman</td>
<td>dbp:team</td>
</tr>
<tr>
<td>dbr:Nate_Huffman</td>
<td>dbr:Maccabi_TelAviv</td>
</tr>
<tr>
<td>dbr:Nate_Huffman</td>
<td>dbr:Toronto_Raptors</td>
</tr>
<tr>
<td>dbr:Gal_Mekel</td>
<td>dbp:team</td>
</tr>
<tr>
<td>dbr:Gal_Mekel</td>
<td>dbr:Dallas_Mavericks</td>
</tr>
<tr>
<td>dbr:Lior_Eliyahu</td>
<td>dbp:team</td>
</tr>
<tr>
<td>dbr:Lior_Eliyahu</td>
<td>dbr:Maccabi_TelAviv</td>
</tr>
</tbody>
</table>

---

Representing RDF Graphs

- There are several formats for representing RDF triples
  - We have seen examples of RDF/XML
    - Namespaces, Description, ...
  - JSON-LD
    - LD stands for Linked Data
  - N-Triple
    - Straightforward triples
  - Turtle
    - More compact than N-Triple

---

Outline

- Introduction
- Uniform Resource Identifiers
- RDF
  - SPARQL
    - Formal Definition
- RDF Schema
Query Language for RDF?

- To query RDF, we can use existing languages
  - SQL (over the triple relation)
  - XQuery (over RDF/XML)
- But:
  - We want QLs that are simpler and intuitive on graphs (triples)
    - e.g., graph patterns should be easily phrased
  - The open-world nature of the semantic Web makes incompleteness a first-class citizen
    - We should easily allow for optional information when exists (not NULL reasoning)

SPARQL Query Types

- SELECT
  - Returns mappings
- ASK
  - Boolean – exists at least one mapping?
- DESCRIBE
  - Includes a detailed description of found URIs
- CONSTRUCT
  - Construct a new RDF via the query
- We restrict to SELECT!
  - But the main idea is very similar on all

SPARQL Examples

```
SELECT 
?x ?n 
{ 
?x dbp:team dbr:Maccabi_TelAviv. 
?x dbp:na9onality ?n.  
FILTER(?n>20). }
Maccabi players + nationalities

SELECT 
?x ?n 
{ 
?x dbp:team dbr:Maccabi_TelAviv. 
?x dbp:na9onality ?n.  
OPTIONAL { 
?x dbo:height ?h.  
FILTER(?h>2) }  }
Restrict to players >2m

SELECT 
?x ?n ?h ?t 
{ 
?x dbp:team dbr:Maccabi_TelAviv. 
?x dbp:na9onality ?n.  
OPTIONAL { 
?x dbp:team ?t.  
?t rdf:type yago:Na9onalBasketballAssocia9onTeams }  
OPTIONAL { 
?x dbo:height ?h.  
}  
}
Add both height and NBA memberships optionally
```

Try it on http://dbpedia.org/sparql

```
SELECT distinct ?x ?n ?h 
{ 
?x dbp:team <http://dbpedia.org/resource/Maccabi_Tel_Aviv_B.C.>.  
?x dbp:team <http://dbpedia.org/resource/Maccabi_Tel_Aviv_B.C.>.  
OPTIONAL { 
?x dbo:height ?h.  
}  }
```

Outline

- Introduction
- Uniform Resource Identifiers
- RDF
- SPARQL
  - Formal Definition
- RDF Schema

RDF Query Model

- Next, we will see a formal definition of the SPARQL syntax and semantics
- Our definition covers the core of SPARQL, but omits many details
  - Does not cover blank nodes
  - Does not list the plethora of functions and Boolean predicates used for filtering
- As usual, more details in books, online tutorials, W3C, etc.
### Basic SPARQL Query

**Example 1**

- SELECT ?x
- ?x
- ?x

#### Terminology

- Two mappings \( \mu_1 \) and \( \mu_2 \) are **compatible** if they agree on their common values:
  \[ \forall x ( \text{vars}(\mu_1) \cap \text{vars}(\mu_2) \Rightarrow \mu_1(x) = \mu_2(x)) \]

- The **join** \( \mu_1 \cup \mu_2 \) of two compatible mappings \( \mu_1 \) and \( \mu_2 \) is the mapping \( \mu \) with:
  - \( \text{vars}(\mu) = \text{vars}(\mu_1) \cup \text{vars}(\mu_2) \)
  - \( \mu(x) = \mu_1(x) \) whenever \( x \in \text{vars}(\mu_1) \)
  - \( \mu(x) = \mu_2(x) \) whenever \( x \in \text{vars}(\mu_2) \)

- The **projection** \( \pi_\lambda(\mu) \) of a mapping \( \mu \) to a set \( \lambda \) of variables is the mapping \( \lambda \) with:
  - \( \text{vars}(\lambda) = \text{vars}(\mu) \cap \lambda \)
  - \( \lambda(x) = \mu(x) \) for all \( x \) in \( \text{vars}(\lambda) \)

### Triple Patterns

- The simplest (atomic) pattern is a **triple pattern**
- Semantics of a triple pattern \( \langle s, p, o \rangle \):
  - \( \{ \mu : \text{vars}(\mu) \rightarrow U \cup L \mid \mu \in G \} \)
  - That is, \( \tau(G) \) contains all of the mappings \( \mu \) such that:
    - \( \text{vars}(\mu) \) consists of the variables \( v \) in \( G \)
    - When replacing each variable \( v \) with the constant \( \mu(v) \), the triple \( \langle s, p, o \rangle \) becomes a triple in \( G \)
  - In the definition we denote the new triple as \( \tau(G) \)

### Example 2

**Example: Compatibility and Join**

#### Terminology

- Example: Compatibility and Join

- Basic SPARQL Query

- Triple Patterns
Example: Projection

\[ \pi_{\{x,t\}}(\mu) \]

Example: Equivalent Presentations of Mapping Sets

Equivalent Presentations of Mapping Sets

\[ \pi_{\{x,t\}}(\mu) \]

Example: Algebraic Operations

- Let \( \Omega_1 \) and \( \Omega_2 \) be two sets of matches
- Join: \( \Omega_1 \bowtie \Omega_2 = \{ \mu_1 \bowtie \mu_2 \mid \mu_1 \in \Omega_1 \wedge \text{compatible}(\mu_1, \mu_2) \} \)
- Difference: \( \Omega_1 - \Omega_2 = \{ \mu_1 \in \Omega_1 \mid \text{no } \mu_2 \in \Omega_2 \text{ is compatible with } \mu_1 \} \)
- Left outer join: \( \Omega_1 \bowtie \Omega_2 = (\Omega_1 \bowtie \Omega_2) \cup (\Omega_1 - \Omega_2) \)
- Projection: \( \pi_{\{x,t\}}(\Omega_1) = \{ \pi_{\{x,t\}}(\mu) \mid \mu \in \Omega_1 \} \)
- Filter (selection) for a Boolean condition \( F \) over matches \( \sigma_F(\Omega_1) = \{ \mu \in \Omega_1 \mid \mu \text{ satisfies } F \} \)

Example: Join

Example: 2xJoin

Example: Difference (1)
### Example: Difference (2)

<table>
<thead>
<tr>
<th>Nate Huffman</th>
<th>Lior Eliyahu</th>
<th>Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maccabi Tel-Aviv</td>
<td>Toronto Raptors</td>
<td>2.05 - 2.05</td>
</tr>
</tbody>
</table>

### Example: Difference (3)

<table>
<thead>
<tr>
<th>Nate Huffman</th>
<th>Lior Eliyahu</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maccabi Tel-Aviv</td>
<td>Toronto Raptors</td>
</tr>
</tbody>
</table>

### Example: Left Outer Join (1)

<table>
<thead>
<tr>
<th>Nate Huffman</th>
<th>Lior Eliyahu</th>
<th>Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maccabi Tel-Aviv</td>
<td>Toronto Raptors</td>
<td>2.05 - 2.05</td>
</tr>
</tbody>
</table>

### Example: Left Outer Join (2)

<table>
<thead>
<tr>
<th>Nate Huffman</th>
<th>Lior Eliyahu</th>
<th>Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maccabi Tel-Aviv</td>
<td>Toronto Raptors</td>
<td>2.05 - 2.05</td>
</tr>
</tbody>
</table>

### Example: Join, Difference, Outer Join (1)

<table>
<thead>
<tr>
<th>Nate Huffman</th>
<th>Lior Eliyahu</th>
<th>Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maccabi Tel-Aviv</td>
<td>Toronto Raptors</td>
<td>2.05 - 2.05</td>
</tr>
</tbody>
</table>

### Example: Join, Difference, Outer Join (2)

<table>
<thead>
<tr>
<th>Nate Huffman</th>
<th>Lior Eliyahu</th>
<th>Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maccabi Tel-Aviv</td>
<td>Toronto Raptors</td>
<td>2.05 - 2.05</td>
</tr>
</tbody>
</table>
**SPARQL Basic Syntax**

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Semantics: P(G)</th>
</tr>
</thead>
<tbody>
<tr>
<td>triple pattern ?</td>
<td>( \pi ) (G)</td>
</tr>
<tr>
<td>( P_1 \cup P_2 )</td>
<td>( P_1(G) \cup P_2(G) )</td>
</tr>
<tr>
<td>( P_1 \setminus P_2 )</td>
<td>( P_1(G) \setminus P_2(G) )</td>
</tr>
<tr>
<td>( P_1 \text{ OPTIONAL } (P_2) )</td>
<td>( P_1(G) \cup P_2(G) )</td>
</tr>
<tr>
<td>( P_1 \text{ UNION } (P_2) )</td>
<td>( P_1(G) \cup P_2(G) )</td>
</tr>
<tr>
<td>( P_1 \text{ FILTER } (\varphi) )</td>
<td>( \varphi \land P_1(G) )</td>
</tr>
<tr>
<td>( \sigma \varphi )</td>
<td>( \pi ) (G)</td>
</tr>
</tbody>
</table>

**Projection Example**

\[
\pi_{x,y} R =
\]

\[
\begin{array}{c|c}
\text{John} & \text{Mark} \\
\hline
\text{Technion} & \text{Technion} \\
\text{Technion} & \text{Technion} \\
\text{Technion} & \text{Technion} \\
\text{Technion} & \text{Technion} \\
\end{array}
\]

**Filter Example**

\[
\sigma_{x=y} R =
\]

\[
\begin{array}{c|c}
\text{John} & \text{Mark} \\
\hline
\text{Technion} & \text{Technion} \\
\text{Technion} & \text{Technion} \\
\text{Technion} & \text{Technion} \\
\text{Technion} & \text{Technion} \\
\end{array}
\]

**SPARQL Filter Language**

- Atomic binary predicates (+scalar functions)
  - \( \text{bound}(x) \)
  - \( x \neq y \) \( \tau \) Technology
  - \( \mu \) if involved variables are bound and equality holds
  - \( \text{regex}(\text{str(mbox)}, "@work.example") \)
- Composition via Boolean operators
  - \( \text{and, or, not} \)

**Examples Revisited**

\[
\text{SELECT } ?x \text{ WHERE } \{ \}
\]

\[
\text{SELECT } ?x \text{ WHERE } \{ \}
\]

\[
\text{SELECT } ?x \text{ WHERE } \{ \}
\]
An important ability in SPARQL is the ability to query multiple graphs (from multiple sources)

This is formalized as follows

- An RDF dataset \( D \) is a pair \((G, Ext)\), where:
  - \( G \) is the default RDF graph
  - \( Ext \) is a finite mapping of URIs to RDF graphs
- These are the other graphs that can be referenced by means of URIs:

Write SPARQL queries that:
1. Find the presidents who were in one or more of Lincoln’s parties
2. Find the presidents who were in none of Lincoln’s parties
3. For each president in (1) and (2), add his spouse if exists

You can send your answers for Ex. 1-4 to bennyk@cs to get feedback

To get feedback
Adding Graph to Basic Syntax

**Pattern P** | **Semantics: P(G)**
---|---
triple pattern $t$ | $t(G)$

$P_1 \cdot P_2$ | $P_1(G) \bowtie P_2(G)$

$P_1 \bigtriangleup P_2$ | $P_1(G) \cup P_2(G)$

$P_1 \cdot$ OPTIONAL $(P_2)$ | $P_1(G) \bowtie P_2(G)$

$P_1 \cdot$ UNION $(P_2)$ | $P_1(G) \cup P_2(G)$

$P_1 \cdot$ FILTER $(f)$ | $\sigma_f(P_1(G))$

**SELECT** $\pi_{1,..,n}(P_1)$ | $\pi_{1,..,n}(P_1(G))$

**GRAPH** $h_1(P_1)$ | $h_1(P_1(G))$

**GRAPH** $h_2(P_1)$ | $h_2(P_1(G))$

Try at http://lod2.openlinksw.com/sparql

**Example**

```
PREFIX dbp: <http://dbpedia.org/ontology/>
PREFIX dbor: <http://dbpedia.org/resource/>
PREFIX dbpext: <http://dbpedia-ext.org/>
PREFIX dbx: <http://dbpedia-ext.org/>
PREFIX dbpxsd: <http://dbpedia-ext.org/sparql>
PREFIX dbpext: <http://dbpedia-ext.org/>
PREFIX dbp: <http://dbpedia.org/ontology/>
PREFIX dbor: <http://dbpedia.org/resource/>
PREFIX dbpext: <http://dbpedia-ext.org/>
PREFIX dbx: <http://dbpedia-ext.org/>
PREFIX dbpxsd: <http://dbpedia-ext.org/sparql>
```

Additional Features (SPARQL 1.1)

**Aggregate Queries**

```
GROUP BY ?y1 ?y2
ORDER BY DESC(cosum?)
```

**Property Path**

```
?y1 a dbp:doctoralAdvisor ?y2
```

Outline

- Introduction
- Uniform Resource Identifiers
- RDF
- SPARQL
  - Formal Definition
    - RDF Schema
RDF Schema

- RDF Schema (RDFS) enriches RDF with semantics
  - Types/subtypes, classes/subclasses
  - Properties and sub-properties
  - Typing of properties
- A simple extension to describe ontologies
  - Very limited compared to other concepts
  - Monotone – does not say what “cannot” happen
  - Very different notion from SQL schema, DTD, ...
- Not outside RDF, extends RDF
- Why is it useful?
  - General reasoning/integration power
  - Allows to automatically infer implicit information

Example (Wikipedia)

We Discuss Four RDFS Properties

- p rdfs:domain C
  - Example: ex:teaches rdfs:domain ex:professor
- p rdfs:range C
  - Example: ex:teaches rdfs:range ex:course
- p rdfs:subPropertyOf q
  - Example: ex:motherOf rdfs:subPropertyOf ex:parentOf
- C rdfs:subClassOf D
  - Example: ex:animal rdfs:subClassOf ex:Food

Inference System for RDFS

Rules for inferring additional triples from existing RDF/RDFS triples:

<table>
<thead>
<tr>
<th>Given</th>
<th>Infer</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ p rdfs:domain, } \cup { s, o }</td>
<td>{ ___ rdfs:domain, } \cup { s, o }</td>
</tr>
<tr>
<td>{ p rdfs:range, } \cup { s, o }</td>
<td>{ ___ rdfs:range, } \cup { s, o }</td>
</tr>
<tr>
<td>{ p rdfs:subPropertyOf, } \cup { s, o }</td>
<td>{ ___ rdfs:subPropertyOf, } \cup { s, o }</td>
</tr>
<tr>
<td>{ C rdfs:subClassOf, } \cup { s, o }</td>
<td>{ ___ rdfs:subClassOf, } \cup { s, o }</td>
</tr>
</tbody>
</table>
RDFS Closure

• The RDFS closure of an RDF graph $G$ is the graph obtained from $G$ by iteratively adding triples inferred from the rules until impossible (fixpoint)
  
  – We denote this graph by $cl(G)$

• A SPARQL engine that supports RDFS gives the result of $cl(G)$
  
  $$P_{\text{RDFS}}(G) = P(cl(G))$$

RDFS Expressiveness Limitations

• No negation
  
  – Cannot state disjointness (e.g., animals / people)

• Limited containment relationship
  
  – For example, every mammal is either an animal or a person

• No cardinality/inclusion constraints
  
  – Every mammal has exactly two biological parents
  
  – Every course has at least one lecturer

• No logical constraints on relations
  
  – For example, inverse (parent/child), transitivity

Beyond RDFS: OWL

• OWL (Web Ontology Language) is designed to provide rich expressiveness for ontology specification and reasoning
  
  – “OWL is now the most used KR language in the history of AI...” (Jim Hendler)

• Several classes (different expressiveness/complexity tradeoffs)
  
  – OWL Full, OWL DL, OWL Lite
  
  – In all versions, specs are in RDF

Examples

• foaf:made owl:inverseOf foaf:maker

• ex:friend rdf:type owl:SymmetricProperty

• geo:located rdf:type owl:TransitiveProperty

• Additional properties:
  
  – owl:FunctionalProperty, owl:ComplementOf, owl:IntersectionOf