Data with Semantics – RDF(S)

Petr Křemen

petr.kremen@fel.cvut.cz

October 10, 2017
Outline

1 RDF(S)
   - Core RDF
   - Metamodeling in RDFS
   - RDF Syntaxes
   - RDF Datasets
   - Semantics of RDF(S)
RDF(S)

- Core RDF
- Metamodeling in RDFS
- RDF Syntaxes
- RDF Datasets
- Semantics of RDF(S)
Core RDF
RDF Basics

- RDF 1.0 – W3C Recommendation in 2004,
RDF Basics

- RDF 1.0 – W3C Recommendation in 2004,
- Intuitively, RDF document is a graph, where each node is either (1) an IRI (ellipse), or (2) a literal (rectangle), or (3) a blank node (blank ellipse)
RDF Basics

- RDF 1.0 – W3C Recommendation in 2004,
- Intuitively, RDF document is a graph, where each node is either (1) an IRI (ellipse), or (2) a literal (rectangle), or (3) a blank node (blank ellipse)
- RDF document is a set of triples (graph edges) in the form (Subject, Predicate, Object)
Main Definitions

**Definitions**

**RDF Graph** is a set of RDF triples,

**RDF Triple** is an ordered triple of the form \((Subject, Predicate, Object)\):

- **Subject**
  - IRI
  - b-node

- **Predicate**
  - IRI

- **Object**
  - IRI
  - b-node
  - literal

**RDF Term** is either an IRI, a *blank node*, or a *literal*

**RDF Source** is a mutable source of RDF graphs (e.g. RDF4J Graph store)
IRIs

- IRI = International Resource Identifier
- denotes a *document*, or a real *thing* 
  
  `<http://myurl.cz/my#Peggy>`
- using hash (`#`) or slash (`/`) for delimiting particular entities in a namespace

**Note**

- Two IRIs are equal iff their string representations are equal.
- No IRI is equal to any blank node, or literal.
Literals

- denote basic data values, like strings, integers, or calendar data.

**Definition**

A literal consists of:

- a lexical form, being a Unicode string,
- a datatype IRI, being an IRI identifying a datatype,
- a language tag, iff the datatype IRI is `http://www.w3.org/1999/02/22-rdf-syntax-ns#langString`.

Two literals are equal iff their 1) lexical forms, 2) datatypes, 3) language tags equal.

**Examples:**

- "dolphin" @en
- "dolphin"^^xsd:string
- 128^^xsd:integer
- "2010-01-19T16:00:00Z"^^xsd:dateTime
Datatypes

- reused from XML Schema (e.g. `xsd:string`) plus `rdf:HTML` and `rdf:XMLLiteral`

**Definition**

A datatype consists of:

- **lexical space**, e.g. a set \{"0", "01", ...\} of strings made of numbers 0-9.
- **value space**, e.g. a set of integers \{0, 1, ..., ∞\},
- **lexical-to-value mapping** $L2V$, e.g.
  
  $$L2V(\text{datatype for } xsd:integer) = \{("01", 1), \ldots\}.$$ 

- most XML Schema built-in datatypes – see [Lanthaler:14:RCA] for complete list:
  - `xsd:string`, `xsd:boolean`, `xsd:integer`, `xsd:decimal`, `xsd:dateTimeStamp`, `xsd:base64Binary`, ...
- `rdf:HTML` – for embedding HTML as literals
- `rdf:XMLLiteral` – for embedding XML as literals
- custom datatypes can be defined on different levels – XML Schema, OWL 2
Namespaces

In many RDF syntaxes, namespaces can be abbreviated by means of prefixes for the sake of readability, e.g. `rdf:type` denotes the IRI `http://www.w3.org/1999/02/22-rdf-syntax-ns#type`. But `rdf:type` is not itself an IRI.

- **rdf**: represents the vocabulary of RDF
  `http://www.w3.org/1999/02/22-rdf-syntax-ns#`. This vocabulary defines basic resources, like `rdf:type`, `rdf:Property`.

- **rdfs**: represents the RDFS vocabulary `http://www.w3.org/2000/01/rdf-schema#` for metamodeling, like `rdfs:Class`, or `rdfs:subPropertyOf`.

- **xsd**: represents the XML Schema vocabulary `http://www.w3.org/2001/XMLSchema#`, for referencing datatypes reused by RDF, like `xsd:integer`, or `xsd:string`.

**Note**

We will slightly abuse syntax and use shortened versions of IRIs (e.g. `rdf:type`) in place of IRIs (`http://www.w3.org/1999/02/22-rdf-syntax-ns#type`). Often, a shortened IRI with empty prefix (e.g. `:x`) is used in examples. In such cases, the namespace is fixed, but unimportant for the example, if not stated otherwise.
Vocabularies

Various predefinied vocabularies can be reused in your data, e.g.:

- schema.org – http://schema.org/docs/schemas.html
- Dublin Core – http://dublincore.org/documents/dc-rdf/
- FOAF – http://www.foaf-project.org/
- VOID – http://www.w3.org/TR/void/
- ... and many others
Blank Nodes (b-nodes)

- denote existentially quantified variables,

**Definition**

**Ground RDF Graph** is an RDF Graph containing no b-nodes.

**Instance** of RDF Graph $G_1$ is an RDF Graph in which some b-nodes are be replaced by an arbitrary RDF Term.

**Lean RDF Graph** $G_1$ has no instance $G_2$ which is a proper subgraph of $G_1$. 
Blank Nodes (b-nodes)

- denote existentially quantified variables,
- have local scope to the RDF document and cannot be reused,

**Definition**

**Ground RDF Graph** is an RDF Graph containing no b-nodes.

**Instance** of RDF Graph $G_1$ is an RDF Graph in which some b-nodes are be replaced by an arbitrary RDF Term.

**Lean RDF Graph** $G_1$ has no instance $G_2$ which is a proper subgraph of $G_1$. 

![Diagram of Original and Lean RDF Graphs]
Blank Nodes (b-nodes)

- denote existentially quantified variables,
- have local scope to the RDF document and cannot be reused,
- in Turtle/N-TRIPLES/SPARQL have `_:` prefix, e.g. `_:x`,

Definition

**Ground RDF Graph** is an RDF Graph containing no b-nodes.

**Instance** of RDF Graph $G_1$ is an RDF Graph in which some b-nodes are be replaced by an arbitrary RDF Term.

**Lean RDF Graph** $G_1$ has no instance $G_2$ which is a proper subgraph of $G_1$. 

![Diagram of Original RDF Graph and Lean RDF Graph](image)
Blank Nodes Usage

- describing higher-order statements (reification)
Blank Nodes Usage

- describing higher-order statements (reification)

- expressing complex values
Blank Nodes Usage

- describing higher-order statements (reification)

- expressing complex values

- container description – multisets, sequences, alternatives
Blank Nodes Usage

- describing higher-order statements (reification)

  ![Diagram showing RDF structure with blank nodes](powered_by_yfiles)

- expressing complex values

  ![Diagram showing container description](powered_by_yfiles)

- container description – multisets, sequences, alternatives
- modeling n-ary relations (e.g. birth)
Blank Node Skolemization

- replacing the blank nodes with fresh IRIs (Skolem IRI) to allow stronger identification of those resources
- the meaning of the RDF graph remains the same as before skolemization
- skolemized IRIs http://.../.well-known/genid/xxx, where xxx is a placeholder for a generated identifier.
Different syntaxes

Turtle syntax:

```turtle
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix my: <http://www.myurl.cz/my#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
my:George my:loves my:Peggy .
my:Peggy my:hasHusband my:John .
my:John rdf:type <http://www.otherurl.org/other#Person> ;
  my:hates my:George ;
  my:hasAge "27"^^xsd:integer.
```

RDF/XML syntax:

```xml
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:base="http://myurl.cz/my"
  xmlns:my="http://myurl.cz/my">
  <rdf:Description rdf:ID="George">
    <my:loves rdf:about="http://myurl.cz/my#Peggy"/>
  </rdf:Description>
  <rdf:Description rdf:ID="Peggy">
    <my:hasHusband rdf:about="http://myurl.cz/my#John"/>
  </rdf:Description>
  <rdf:Description rdf:ID="John">
    <rdf:type rdf:about="http://otherurl.org/other#Person"/>
    <my:hates rdf:about="http://myurl.cz/my#George"/>
    <my:hasAge rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">27</my:hasAge>
  </rdf:Description>
</rdf:RDF>
```
RDF containers

- **rdf:Bag** denotes an unordered sets of possibly repeating elements (multiset),
RDF containers

- `rdf:Bag` denotes an unordered sets of possibly repeating elements (multiset),
- `rdf:Seq` denotes an ordered sequence,
RDF containers

- `rdf:Bag` denotes an unordered sets of possibly repeating elements (multiset),
- `rdf:Seq` denotes an ordered sequence,
- `rdf:Alt` denotes an alternative choice from given resources/literals,
RDF containers

- \texttt{rdf:Bag} denotes an unordered sets of possibly repeating elements (multiset),
- \texttt{rdf:Seq} denotes an ordered sequence,
- \texttt{rdf:Alt} denotes an alternative choice from given resources/literals,
- Container elements can be addressed by means of the \texttt{rdf:_x} property, where ‘x’ is a positive number,
RDF containers

- `rdf:Bag` denotes an unordered sets of possibly repeating elements (multiset),
- `rdf:Seq` denotes an ordered sequence,
- `rdf:Alt` denotes an alternative choice from given resources/literals,
- Container elements can be addressed by means of the `rdf:_x` property, where 'x' is a positive number,
- Containers are not closed – someone else can assert statements assigning elements to our container,
RDF containers

- `rdf:Bag` denotes an unordered sets of possibly repeating elements (multiset),
- `rdf:Seq` denotes an ordered sequence,
- `rdf:Alt` denotes an alternative choice from given resources/literals,
- Container elements can be addressed by means of the `rdf:_x` property, where 'x' is a positive number,
- Containers are not closed – someone else can assert statements assigning elements to our container,
- Containers can be modeled by means of blank nodes.
RDF collections

- represent "closable" containers, similarly as LISP/Prolog lists
RDF collections

- Represent "closable" containers, similarly as LISP/Prolog lists.
- \texttt{rdf:List} represents a list; the list head is available through \texttt{rdf:first} and the property is available through \texttt{rdf:rest}. The list can be closed by means of an empty list \texttt{rdf:nil}. 

\begin{tikzpicture}
  \node [shape=circle,draw] (http://myurl.cz/my#Year) {
    \texttt{http://myurl.cz/my#Year}
  };
  \node [shape=circle,draw] (http://myurl.cz/hasSeasons) {
    \texttt{http://myurl.cz/hasSeasons}
  };
  \node [shape=ellipse,draw] (Spring) {
    \texttt{Spring}
  };
  \node [shape=ellipse,draw] (Summer) {
    \texttt{Summer}
  };
  \node [shape=ellipse,draw] (Autumn) {
    \texttt{Autumn}
  };
  \node [shape=ellipse,draw] (Winter) {
    \texttt{Winter}
  };

  \draw [->] (http://myurl.cz/my#Year) -- (http://myurl.cz/hasSeasons);
  \draw [->] (Spring) -- (http://myurl.cz/my#Year);
  \draw [->] (Summer) -- (Spring);
  \draw [->] (Autumn) -- (Summer);
  \draw [->] (Winter) -- (Autumn);
  \draw [->] (Winter) -- (http://myurl.cz-nil);

\end{tikzpicture}
RDF Model – Axiomatic Triples

Figure: Visualization of axiomatic triples of RDF. Precise definition can be found in [Patel-Schneider:14:RS]
RDF 1.1

RDF 1.1

- a set of W3C Recommendations in February 2014
RDF 1.1

- a set of W3C Recommendations in February 2014
- main differences to RDF 1.0:
RDF 1.1

- a set of W3C Recommendations in February 2014
- main differences to RDF 1.0:
  - identification of resources by IRIs
RDF 1.1

- a set of W3C Recommendations in February 2014
- main differences to RDF 1.0:
  - identification of resources by IRIs
  - all literals are *typed*, new datatypes introduced:

```plaintext
rdf:langString
rdf:HTML
rdf:XMLLiteral
```

The last two are non-normative in RDF 1.1
RDF 1.1

- a set of W3C Recommendations in February 2014
- main differences to RDF 1.0:
  - identification of resources by IRIs
  - all literals are *typed*, new datatypes introduced:
    - `rdf:langString`
    - `rdf:HTML`
    - `rdf:XMLLiteral`

    The last two are non-normative in RDF 1.1
  - additional XSD datatypes
    - `xsd:duration,`
    - `xsd:dayTimeDuration,`
    - `xsd:yearMonthDuration,`
    - `xsd:dateTimeStamp`
RDF 1.1

- a set of W3C Recommendations in February 2014
- main differences to RDF 1.0:
  - identification of resources by IRIs
  - all literals are typed, new datatypes introduced:
    - `rdf:langString`
    - `rdf:HTML`
    - `rdf:XMLLiteral`
    - The last two are non-normative in RDF 1.1
- additional XSD datatypes
  - `xsd:duration`,
  - `xsd:dayTimeDuration`,
  - `xsd:yearMonthDuration`,
  - `xsd:dateTimeStamp`
- additional serialization – JSON-LD, Turtle, TriG, N-Quads
Metamodelling in RDFS
RDFS Basics

- RDFS = RDF Schema
- simple metamodeling language
- rdfs being shortcut for http://www.w3.org/2000/01/rdf-schema#
- rdf being shortcut for http://www.w3.org/1999/02/22-rdf-syntax-ns#
- RDF Schema 1.0 – W3C Recommendation in 2004 [Brickley:04:RVD]
- basic metamodeling vocabulary:

  rdf:type,
rdfs:Class,
rdfs:subClassOf,
rdf:Property,
rdfs:subPropertyOf,
rdfs:domain,
rdfs:range
Classes

- define instances:
  
  ```
  ex:John rdf:type ex:Person .
  ```
Classes

- define instances:
  ```
  ex:John rdf:type ex:Person .
  ```

- define classes (class rdfs:Class):
  ```
  ex:Person rdf:type rdfs:Class .
  ```
Classes

- define instances:
  
  \[
  \text{ex:John} \ rdfs\text{:type} \ \text{ex:Person}.
  \]

- define classes (class rdfs\text{:Class}):
  
  \[
  \text{ex:Person} \ rdfs\text{:type} \ rdfs\text{:Class}.
  \]

- create class hierarchies (property rdfs\text{:subClassOf}):
  
  \[
  \text{ex:Woman} \ rdfs\text{:subClassOf} \ \text{ex:Person}.
  \]
Classes

- **define instances:**
  
  \[\text{ex:John rdf:type ex:Person} .\]

- **define classes (class rdfs:Class):**
  
  \[\text{ex:Person rdf:type rdfs:Class} .\]

- **create class hierarchies (property rdfs:subClassOf):**
  
  \[\text{ex:Woman rdfs:subClassOf ex:Person} .\]

- **multiple inheritance:**
  
  \[\text{ex:Woman rdfs:subClassOf ex:Person} . \]
  
  \[\text{ex:Woman rdfs:subClassOf ex:Female}.\]
Properties

- **property definitions (resource \texttt{rdf:Property}):**
  \begin{verbatim}
  ex:hasParent rdf:type rdf:Property .
  \end{verbatim}

- **creation of property hierarchies (property \texttt{rdfs:subPropertyOf}):**
  \begin{verbatim}
  ex:hasMother rdfs:subPropertyOf ex:hasParent .
  \end{verbatim}

- **multiple inheritance**

- **domain and range definition:**
  \begin{verbatim}
  ex:hasMother rdfs:domain ex:Person .
  ex:hasMother rdfs:range ex:Woman
  \end{verbatim}

- **domains/ranges considered as conjunction:**
  \begin{verbatim}
  ex:hasMother rdfs:range ex:Person .
  ex:hasMother rdfs:range ex:Female .
  \end{verbatim}
RDFS Model – Axiomatic Triples

Figure: Visualization of axiomatic triples of RDFS. Precise definition can be found in [Patel-Schneider:14:RS]
RDF Syntaxes
Syntaxes

**RDF/XML**, a frame-based syntax

**N-TRIPLES**, simple triples, for batch processing

**TURTLE**, well-readable, compact

**TriG**, extension of TURTLE for multiple graphs (RDF datasets)

**N-QUADS**, extension of N-TRIPLES for multiple graphs (RDF datasets)

**JSON-LD**, JSON syntax for RDF 1.1

**RDF-A**, syntax for embedding RDF 1.1 into HTML
RDF/XML
readable, expressive, plenty of syntactic sugar

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:base="http://myurl.cz/my#"
    xmlns:my="http://myurl.cz/my#"
    xmlns:other="http://otherurl.org/other#">
  <rdf:Description rdf:ID="George">
    <my:loves rdf:about="http://myurl.cz/my#Peggy"/>
  </rdf:Description>
  <rdf:Description rdf:ID="Peggy">
    <my:hasHusband rdf:about="http://myurl.cz/my#John"/>
  </rdf:Description>
  <other:Person rdf:ID="John">
    <my:hasAge rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">
      27
    </my:hasAge>
  </other:Person>
</rdf:RDF>
```
N-TRIPLES
suitable for loading large data volumes

TURTLE

extension of N-TRIPLES, allowing shortcuts

```
:a :p1 :o1 ;
 :p2 :o2 .
:a :p1 [ 
 :p2 :o2 ;
 :p3 :o3 .
 ]
:a :p1 _:x .
 _:x :p2 :o2 .
 _:x :p3 :o3 .
```

@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix my: <http://www.myurl.cz/my#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
my:George my:loves my:Peggy .
my:Peggy my:hasHusband my:John .
my:John rdf:type <http://www.otherurl.org/other#Person> ;
 my:hates my:George ;
 my:hasAge "27"^^xsd:integer.
RDF Datasets
**Definition**

**RDF dataset** is a collection of RDF graphs:

\[ DS = \{ DG, (i_1, G_1), \ldots, (i_n, G_n) \} \]

consisting of a **default (unnamed) RDF graph** \( DG \) and zero or more **named RDF graphs** \( G_k \) identified by their IRI/blank node \( i_k \).

- Default graphs might be independent on named graphs (in Sesame they are not – default graph contains union of all named graphs).
- Blank nodes can be reused between different graphs in a single RDF dataset.
- For SPARQL 1.1, RDF dataset cannot use blank nodes as graph names.
RDF Merge

- **Merge** of RDF graphs $G_1$ and $G_2$ is an RDF graph created as follows:
  - rename b-nodes in $G_1$, so that no b-node label occur in both $G_1$ and $G_2$.
  - union $G_1$ and $G_2$.

- **Example:**
  - $G_1$:
    ```rdfs
    @prefix : <http://www.myurl.cz/my#> .
    :a :p _:b .
    :a :q _:c .
    ```
  - $G_2$:
    ```rdfs
    @prefix : <http://www.myurl.cz/my#> .
    :a :s _:c .
    :a :t _:d .
    ```
  - merge of $G_1$ and $G_2$:
    ```rdfs
    @prefix : <http://www.myurl.cz/my#> .
    :a :p _:b .
    :a :q _:c .
    :a :s _:e .
    :a :t _:d .
    ```
Semantics of RDF(S)
Entailment Regimes and Semantic Extension

Precise definition of RDF semantics can be found in [Patel-Schneider:14:RS]

Definition

Semantic Extension is a set of semantic constraints on an RDF graph.
Entailment Regime is a set of entailments defined by the corresponding semantic extension.

- Four entailment regimes are predefined in RDF specs:
  - Simple entailment provides only structural matching of graphs with possible b-node renaming
  - RDF entailment interprets RDF vocabulary
  - RDFS entailment interprets RDF and RDFS vocabularies
  - D entailment additionally interprets datatypes

- All entailment regimes must be monotonic extensions of simple entailment
Simple Interpretation

Definition

A finite interpretation \( I = (IR, IP, IEXT, IS, IL) \) w.r.t. vocabulary \( N = (N_{IR}, N_{lit}) \) is defined as follows:

- \( IR \) is a set of resources
- \( IP \) is a set of properties (often \( IP \subseteq IR \))
- \( IEXT \) is a mapping \( IEXT : IP \to IR \times IR \)
- \( IS \) is a mapping \( IS : N_{IR} \to IR \cup IP \)
- \( IL \) is a partial mapping \( IL : N_{lit} \to IR \)

For example

- \( IR = \{John, Mary, 2\} \) (real resources)
- \( IP = \{loves, childcount\} \) (real properties)
- \( IEXT = \{(loves, \langle John, Mary\rangle), (childcount, \langle John, 2\rangle)\} \)
- \( IS = \{\langle http://www.myurl.cz/my#John, John\rangle, \langle http://www.myurl.cz/my#Mary, Mary\rangle, \langle http://www.myurl.cz/my#loves, loves\rangle, \langle http://www.myurl.cz/my#childcount, childcount\rangle\} \)
- \( IL = \{\langle"2"^\hat{\text{^\textcopyright}\text{^\textregistered}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\texttrademark}\text{^\textrad...\}} \}

@prefix : <http://www.myurl.cz/my#> .
:John :loves :Mary .
:John :childcount 2 .

Petr Křemen (petr.kremen@fel.cvut.cz) Data with Semantics – RDF(S) October 10, 2017 34 / 42
Simple Entailment

Simple entailment is just a “structural matching with b-node rewriting.”

Semantic Conditions on Simple Entailment

- If $E$ is a literal, then $I(E) = IL(E)$
- If $E$ is an IRI, then $I(E) = IS(E)$
- If $E$ is a ground triple $(s, p, o)$, then $I(E) = true$ iff $I(p) \in IP$ and $\langle I(s), I(o) \rangle \in IEXT(I(p))$
- If $E$ is a ground RDF graph, then $I(E) = true$ iff $I(E') = true$ for each triple $E' \in E$
- If $E$ is an RDF graph, then $I(E) = true$ iff there exists a mapping $A : N_{bnod} \rightarrow IR$, such that $I(A(E)) = true$, where $A(E)$ is $E$, where each blank node $B$ is replaced by $A(B)$.

Simple Entailment

- Graph $G_1$ (simply) entails graph $G_2$ (denoted $G_1 \models G_2$) if $I(G_2) = true$ whenever $I(G_1) = true$.
- If $G_1 \models G_2$ and $G_2 \models G_1$ then they are logically equivalent.
How to Check Simple Entailment?

Interpolation lemma

Graph $G_1$ simply entails graph $G_2$ iff a subgraph of $G_1$ is an instance of $G_2$.

Simple entailment is NP in the size of $G_1$ and $G_2$. 
D-Entailment

In addition to blank nodes, $D$-entailment ($|=D$) interprets datatypes in the set $D$ of recognized datatypes. Literals with non-recognized datatypes are treated as uninterpreted.

Semantic Conditions on D-Entailment

- if rdf:langString $\in D$, then for each literal lex@lang:
  $IL(lex@lang) = \langle lex, \text{lowercase}(lang) \rangle$
- if dIRI $\in D$, then for each literal lex ^^ dIRI:
  $IL(lex ^^ dIRI) = L2V(I(dIRI))(lex)$, where
  - $I(dIRI)$ is a datatype identified by dIRI
  - $L2V(d)$ transforms a lexical value to the value space of $d$. 
RDF-Entailment

In addition to $D$-entailment, RDF-entailment w.r.t $D$ interprets properties in the RDF vocabulary.

Entailment rules

<table>
<thead>
<tr>
<th>rule</th>
<th>$G$ contains</th>
<th>$t_i$, s.t. $G \models_{RDF-D} t_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>GrdfD1</td>
<td>$(s, p, lex^{d})$ $d \in D$</td>
<td>$(lex^{d}, rdf: type, d)$</td>
</tr>
<tr>
<td>rdfD2</td>
<td>$(s, p, o)$</td>
<td>$(p, rdf: type, rdf: Property)$</td>
</tr>
</tbody>
</table>

For example:

```
:a →:p →:b
"abc"^{xsd:string}
```

```
:q ← "abc"^{xsd:string}

:q ← rdf:Property
```

```
:q ← rdf:type
```

```
:q ← rdf:Property
```

```
:q ← rdf:type
```

```
:q ← rdf:Type
```
RDFS-Entailment

RDFS-entailment w.r.t $D$ interprets most RDF and RDFS vocabulary.

<table>
<thead>
<tr>
<th>Entailment rules</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>rule</strong></td>
</tr>
<tr>
<td>rdfs1</td>
</tr>
<tr>
<td>rdfs2</td>
</tr>
<tr>
<td>rdfs3</td>
</tr>
<tr>
<td>rdfs4</td>
</tr>
<tr>
<td>rdfs5</td>
</tr>
<tr>
<td>rdfs6</td>
</tr>
<tr>
<td>rdfs7</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>rdfs8</td>
</tr>
<tr>
<td>rdfs9</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>rdfs10</td>
</tr>
<tr>
<td>rdfs11</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>rdfs12</td>
</tr>
<tr>
<td>rdfs13</td>
</tr>
</tbody>
</table>
RDFS-Entailment Example

For example:

```
:q
   ^ rdfs:subPropertyOf
   |    rdfs:range
   |      :c
   :p
   :a
   :q

:q
   ^ rdfs:range
   |      :c
   :p
   :b
```

\[ \models \text{RDFS-\{\}} \]
Entailment Checking

All discussed entailments can be checked by applying the entailment rules on *generalized RDF graphs*, i.e. graphs that allow all RDF Terms in all positions – subject, predicate, object.

Entailment checking procedure

\[ G_1 \models_X G_2, \text{ iff } \text{Clos}_X(G_1) \text{ simply entails } G_2, \text{ where } \text{Clos}_X(G_1) \text{ is constructed as follows:} \]

1. Add to \( G_1 \) all axiomatic triples for \( X \in \{\text{RDF-D, RDFS-D}\} \) (visualized in Figure 1, resp. Figure 2)

2. For each container membership property IRI \( p \) occuring in \( G_1 \), add to \( G_1 \) corresponding axiomatic triples for \( X \) containing \( p \).

3. If no triples were added in the previous step, add axiomatic triples for \( X \) containing \( \text{rdf:1} \).

4. Apply rules for \( X \) (i.e. \( \{\text{GrdfD1, rdfD2}\} \) for \( X = \text{RDF} \), or \( \{\text{Grdf1, rdfD2, rdfs1, \ldots, rdfs13}\} \) for \( X = \text{RDFS} \)) with \( D = \{\text{rdf: langString, xsd: string}\} \), until exhaustion.
Entailment Checking Complexity

- the previous procedure is finite and polynomial
- simple entailment checking itself is NP
- the less blank nodes, the more efficient