# Data with Semantics – RDF(S)

Petr Křemen

petr.kremen@fel.cvut.cz

October 1, 2020

# Outline

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

# RDF(S)

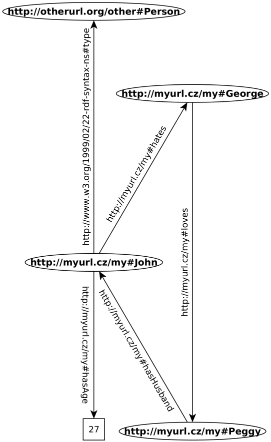
```
@prefix qb:     <http://purl.org/linked-data/cube#> .
@prefix cssz-measure: <https://data.cssz.cz/ontology/measure/> .
@prefix rdf:    <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix y-onto: <https://data.cssz.cz/ontology/years/> .
@prefix xsd:    <http://www.w3.org/2001/XMLSchema#> .
@prefix skos:   <http://www.w3.org/2004/02/skos/core#> .
@prefix cssz:   <https://data.cssz.cz/ontology/> .
@prefix rdfs:   <http://www.w3.org/2000/01/rdf-schema#> .
@prefix cssz-dimension: <https://data.cssz.cz/ontology/dimension/> .

<https://data.cssz.cz/resource/observation/invalidita/2011/pk_id/t/60-64>
  rdf:type qb:Observation ;
  qb:dataSet <https://data.cssz.cz/resource/dataset/invalidita> ;
  cssz-dimension:druh-duchodu
    <https://data.cssz.cz/resource/pension-kind/PK_ID_2010> ;
  cssz-dimension:pohlavi
    <https://data.cssz.cz/ontology/sdmx/code/sex-T> ;
  cssz-dimension:refPeriod
    <https://data.cssz.cz/resource/reference.data.gov.uk/id/gregorian-year
  cssz-dimension:skupina-diagnoz-dle-who
    <https://data.cssz.cz/resource/icd-10/chapter/C_T> ;
  cssz-dimension:vekova-kategorie
    <https://data.cssz.cz/generated/resource/age/Y60T64> ;
  cssz-measure:pocet-nove-priznanych-duchodu 234.0 .
```

# Core RDF

# RDF

- RDF = **R**esource **D**escription **F**ramework
- RDF 1.0 – W3C Recommendation in 2004,



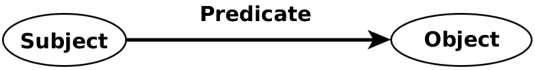

## RDF Document

is a graph, where each

- **node** is either an IRI (ellipse), a literal (rectangle), or a blank node (blank ellipse)
- **edge** is labeled with IRI

# RDF Triple

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




| Subject | Predicate | Object |
|---|---|---|
| IRI<br>b-node | IRI | IRI<br>b-node<br>literal |

**Definitions**

RDF Graph is a set of RDF triples

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

# IRIs

- IRI = International Resource Identifier
- denotes a *document*, or a real *thing*

  `<http://myurl.cz/my#Peggy>`
  `<http://myurl.cz/my/document-about-peggy>`
- using hash (#) or slash (/) for delimiting particular entities in a namespace
- mapped to URIs = backward-compatibility

## Note

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

# Namespaces

can be abbreviated using prefixes to improve readability

`rdf:type` (can be also abbreviated as `a`) instead of `http://www.w3.org/1999/02/22-rdf-syntax-ns#type`.

rdf: `http://www.w3.org/1999/02/22-rdf-syntax-ns#`. This namespace defines basic resources, like `rdf:type`, `rdf:Property`.

rdfs: `http://www.w3.org/2000/01/rdf-schema#`. This namespace is used for metamodeling, like `rdfs:Class`, or `rdfs:subPropertyOf`.

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

**Note**

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.

# 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.

- `"dolphin"@en` — "dolphin" = lex. form, @en = lang. tag
- `"dolphin"^^xsd:string` — "dolphin" = lex. form, ^^xsd:string = datatype IRI

other examples:

`"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 $\{"0", "01", \ldots\}$ of strings made of numbers 0-9.

value space , e.g. a set of integers $\{0, 1, \ldots, \infty\}$,

lexical-to-value mapping $L2V$, e.g.

$$L2V(\text{datatype for } \mathtt{xsd:integer}) = \{\langle "01", 1\rangle, \ldots\}.$$

- most XML Schema built-in datatypes:
  - `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, ...

# Blank Nodes (b-nodes)

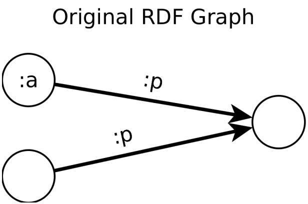
- denote existential variables,

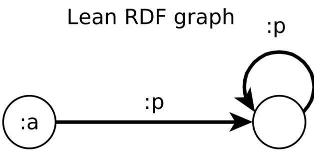
## Definition

**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)

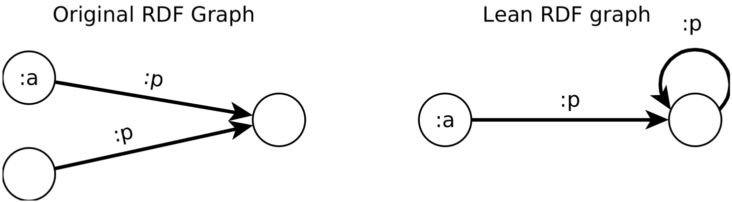
- denote existential variables,
- are local to the RDF document (cannot be reused outside),

**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)

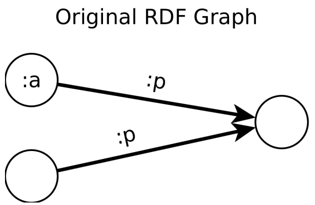
- denote existential variables,
- are local to the RDF document (cannot be reused outside),
- in Turtle/N-TRIPLES/SPARQL have `_:` prefix, e.g. `_:x`,

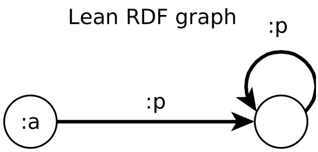
## Definition

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

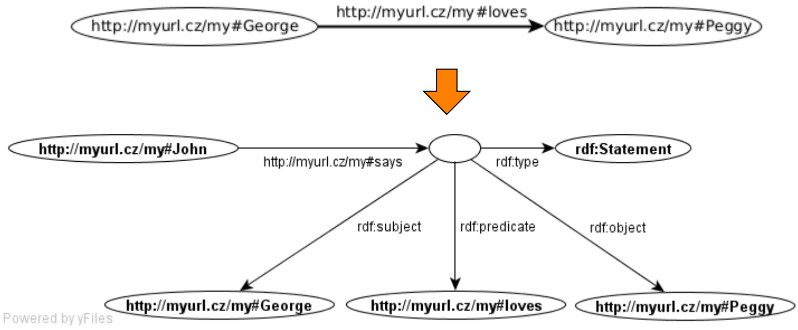
**Instance** of RDF Graph $G_1$ is an RDF Graph in which some b-nodes are be replaced by an arbitrary RDF Term.

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

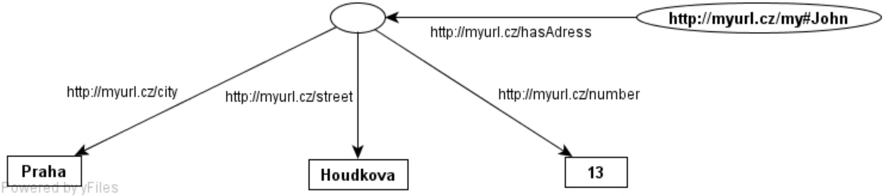


# Blank Nodes for statement reification



# Blank Nodes for expressing complex values



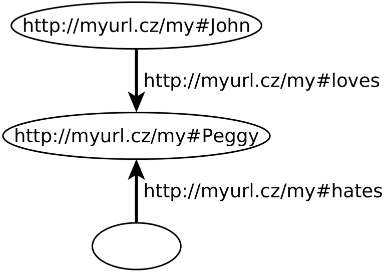
# Blank Nodes for other use-cases

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

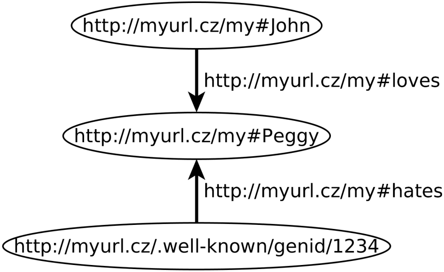
# Blank Node Skolemization

- 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.

Original RDF Graph




Skolemized RDF Graph



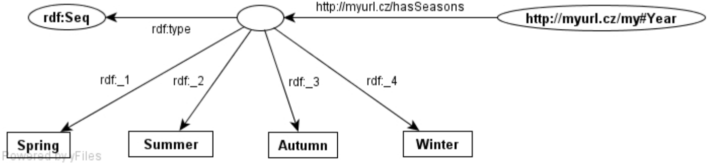
# Vocabularies

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/`. `https://www.dublincore.org/specifications/dublin-core/dcmi-terms/`
- FOAF – `http://www.foaf-project.org/`
- VOID – `http://www.w3.org/TR/void/`
- . . . and many others

# RDF features

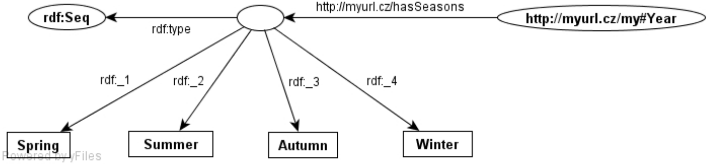
# RDF containers




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

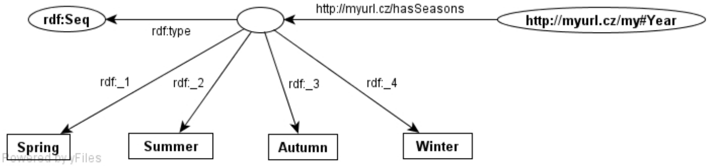
# RDF containers




- `rdf:Bag` denotes an unordered sets of possibly repeating elements (multiset),
- `rdf:Seq` denotes an ordered seequence,
- `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 adding elements to our container,

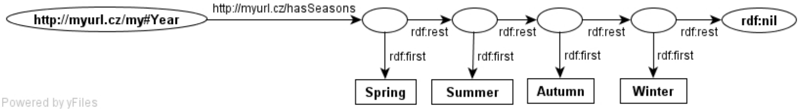
# RDF containers




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

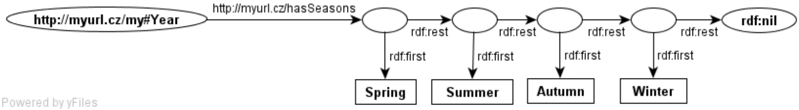
- Container elements can be addressed by means of the `rdf:_x` property, where 'x' is a positive number,
- Containers are **not closed** – someone else can assert statements adding elements to our container,
- Containers can be modeled by means of blank nodes.

# RDF collections




- represent **closable** containers, similarly as LISP/Prolog lists

# RDF collections




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

# RDF Model – Axiomatic Triples

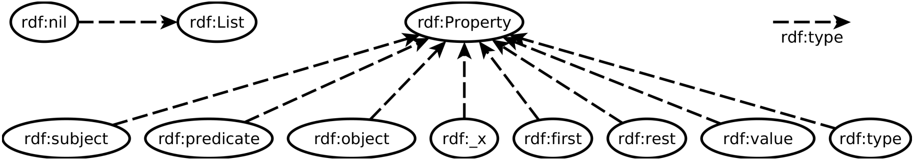



Figure: Visualization of axiomatic triples of RDF. Precise definition can be found in [**Patel-Schneider:14:RS**]

# RDF 1.1

- RDF 1.1 Primer – W3C Working Group Note [**Schreiber:13:RP**]

# RDF 1.1

- RDF 1.1 Primer – W3C Working Group Note [**Schreiber:13:RP**]
- a set of W3C Recommendations in February 2014

# RDF 1.1

- RDF 1.1 Primer – W3C Working Group Note [**Schreiber:13:RP**]
- a set of W3C Recommendations in February 2014
- main differences to RDF 1.0:

# RDF 1.1

- RDF 1.1 Primer – W3C Working Group Note [**Schreiber:13:RP**]
- a set of W3C Recommendations in February 2014
- main differences to RDF 1.0:
  - identification of resources by IRIs

# RDF 1.1

- RDF 1.1 Primer – W3C Working Group Note [**Schreiber:13:RP**]
- 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

# RDF 1.1

- RDF 1.1 Primer – W3C Working Group Note [**Schreiber:13:RP**]
- 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

- RDF 1.1 Primer – W3C Working Group Note [**Schreiber:13:RP**]
- 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

# RDF*

- extending RDF 1.1 to support more efficient reification
- `https://arxiv.org/pdf/1406.3399.pdf`
- statement `:man :hasSpouse :woman.`
- reified statement `<<:man :hasSpouse :woman>> :startDate "2020-02-11" .`

# Metamodeling 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 :

```
ex:John rdf:type ex:Person .
```

- define classes (class `rdfs:Class`) :

```
ex:Person rdf:type rdfs:Class .
```

- create class hierarchies (property `rdfs:subClassOf`) :

```
ex:Woman rdfs:subClassOf ex:Person .
```

# Classes

- define instances :

```
ex:John rdf:type ex:Person .
```

- define classes (class rdfs:Class) :

```
ex:Person rdf:type rdfs:Class .
```

- create class hierarchies (property rdfs:subClassOf) :

```
ex:Woman rdfs:subClassOf ex:Person .
```

- multiple inheritance :

```
ex:Woman rdfs:subClassOf ex:Person .
ex:Woman rdfs:subClassOf ex:Female.
```

# Properties

- property definitions (resource `rdf:Property`) :

```
ex:hasParent rdf:type rdf:Property .
```

- creation of property hierarchies (property `rdfs:subPropertyOf`) :

```
ex:hasMother rdfs:subPropertyOf ex:hasParent.
```

- multiple inheritance
- domain and range definition :

```
ex:hasMother rdfs:domain ex:Person .
ex:hasMother rdfs:range ex:Woman
```

- domains/ranges considered as conjunction :

```
ex:hasMother rdfs:range ex:Person .
ex:hasMother rdfs:range ex:Female .
```

# RDFS Model – Axiomatic Triples

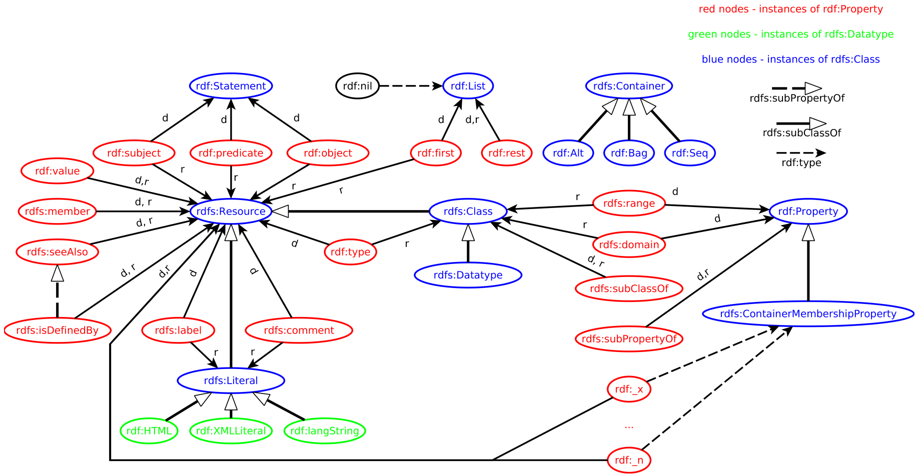



Figure: Visualization of axiomatic triples of RDFS. Precise definition can be found in [**Patel-Schneider:14:RS**]

# RDF Syntaxes

# Syntaxes

Turtle family

- 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)

- RDF/XML , a frame-based syntax
- JSON-LD , JSON syntax for RDF 1.1
- RDF-A , syntax for embedding RDF 1.1 into HTML

# N-TRIPLES

suitable for loading large data volumes

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

# TURTLE

## extension of N-TRIPLES, allowing shortcuts

```
@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.
```

```
:a :p1 :o1 ;
   :p2 :o2 .
```

```
:a :p1 :o1 .
:a :p2 :o2 .
```

```
:a :p :o1, :o2 .
```

```
:a :p :o1 .
:a :p :o2 .
```

# TURTLE

extension of N-TRIPLES, allowing shortcuts

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

```
:a :p1 _:x .
_:x :p2 :o2 .
_:x :p3 :o3 .
```

```
:a :p (:o1 :o2 :o3).
```

```
:a :p _:a .
_:a rdf:first :o1 .
_:a rdf:rest _:b .
_:b rdf:first :o2 .
_:b rdf:rest _:c .
_:c rdf:first :o3 .
_:c rdf:rest rdf:nil .
```

# 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:hates rdf:about="http://myurl.cz/my#George"/>
  <my:hasAge rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">
    27
  </my:hasAge>
</other:Person>
</rdf:RDF>
```

# 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 RDF4J 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:
  - $G_1$ :

```
@prefix : <http://www.myurl.cz/my#> .
:a :p _:b .
:a :q _:c .
```

  - $G_2$ :

```
@prefix : <http://www.myurl.cz/my#> .
:a :s _:c .
:a :t _:d .
```

  - merge of $G_1$ and $G_2$ :

```
@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 $N = (N_{IRI}, N_{lit})$ is defined as follows:

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

# Simple Interpretation Example

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

- $IR = \{John, Mary, 2\}$ (real resources)
- $IP = \{loves, childcount\}$ (real properties)
- $IEXT = \{(loves, \langle John, Mary \rangle), (childcount, \langle John, 2 \rangle)\}$
- $IS = \{\langle \texttt{http://www.myurl.cz/my\#John}, John \rangle, \langle \texttt{http://www.myurl.cz/my\#Mary}, Mary \rangle, \langle \texttt{http://www.myurl.cz/my\#loves}, loves \rangle, \langle \texttt{http://www.myurl.cz/my\#childcount}, childcount \rangle\}$
- $IL = \{\langle \texttt{"2"\^{}\^{}http://www.w3.org/2001/XMLSchema\#integer}, 2 \rangle\}$

# 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 $A : N_{bnode} \rightarrow IR$, such that $I(A(E)) = true$, where $A(E)$ is $E$, where each blank node $B$ is replaced by $A(B)$.

## Simple Entailment

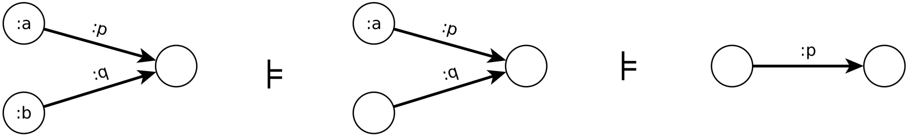
- graph $G_1$ (simply) entails graph $G_2$ (denoted $G_1 \models G_2$) if $I(G_2) = true$ whenever $I(G_1) = true$.
- if $G_1 \models G_2$ and $G_2 \models G_1$ then they are *logically equivalent*.

# 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 $G_2$.

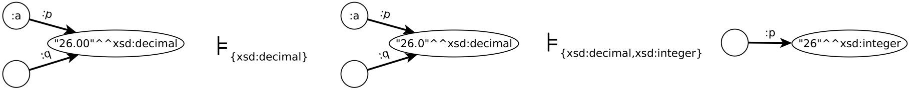


# D-Entailment

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

## Semantic Conditions on D-Entailment

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



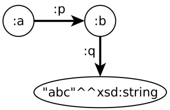
# RDF-Entailment

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

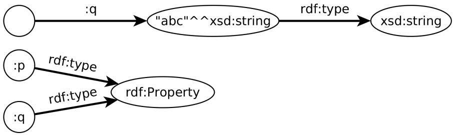
## Entailment rules

| rule | $G$ contains | $t_i$, s.t. $G \models_{RDF-D} t_i$ |
|---|---|---|
| GrdfD1 | $(s, p, lex\hat{}\hat{}d)$ <br> $d \in D$ | $(lex\hat{}\hat{}d, \mathtt{rdf:type}, d)$ |
| rdfD2 | $(s, p, o)$ | $(p, \mathtt{rdf:type}, \mathtt{rdf:Property})$ |

For example:




$\models_{\text{RDF-\{xsd:string,xsd:integer\}}}$



# RDFS-Entailment

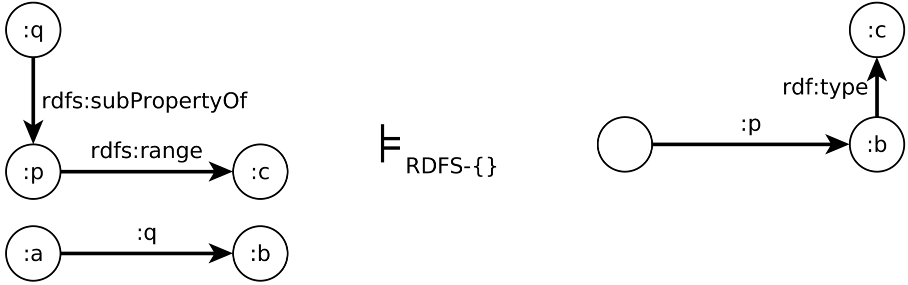
RDFS-entailment w.r.t $D$ interprets most RDF and RDFS vocabulary.

## Entailment rules

| rule | $G$ contains | $t_i$, s.t. $G \models_{RDFS-D} t_i$ |
|---|---|---|
| rdfs1 | any IRI $dIRI \in D$ in $G$ | $(dIRI, \texttt{rdf:type}, \texttt{rdfs:Datatype})$ |
| rdfs2 | $(s, p, o)$, $(p, \texttt{rdfs:domain}, w)$ | $(s, \texttt{rdf:type}, w)$ |
| rdfs3 | $(s, p, o)$, $(p, \texttt{rdfs:range}, w)$ | $(o, \texttt{rdf:type}, w)$ |
| rdfs4 | $(s, p, o)$ | $(s, \texttt{rdf:type}, \texttt{rdfs:Resource})$<br>$(o, \texttt{rdf:type}, \texttt{rdfs:Resource})$ |
| rdfs5 | $(p_1, \texttt{rdfs:subPropertyOf}, p_2)$<br>$(p_2, \texttt{rdfs:subPropertyOf}, p_3)$ | $(p_1, \texttt{rdfs:subPropertyOf}, p_3)$ |
| rdfs6 | $(p, \texttt{rdf:type}, \texttt{rdf:Property})$ | $(p, \texttt{rdfs:subPropertyOf}, p)$ |
| rdfs7 | $(p_1, \texttt{rdfs:subPropertyOf}, p_2)$<br>$(s, p_1, o)$ | $(s, p_2, o)$ |
| rdfs8 | $(s, \texttt{rdf:type}, \texttt{rdfs:Class})$ | $(s, \texttt{rdfs:subClassOf}, \texttt{rdfs:Resource})$ |
| rdfs9 | $(c_1, \texttt{rdfs:subClassOf}, c_2)$<br>$(s, \texttt{rdf:type}, c_1)$ | $(s, \texttt{rdf:type}, c_2)$ |
| rdfs10 | $(c, \texttt{rdf:type}, \texttt{rdfs:Class})$ | $(c, \texttt{rdfs:subClassOf}, c)$ |
| rdfs11 | $(c_1, \texttt{rdfs:subClassOf}, c_2)$<br>$(c_2, \texttt{rdfs:subClassOf}, c_3)$ | $(c_1, \texttt{rdfs:subClassOf}, c_3)$ |
| rdfs12 | $(p, \texttt{rdf:type},$<br>$\texttt{rdfs:ContainerMembershipProperty})$ | $(p, \texttt{rdfs:subPropertyOf},$<br>$\texttt{rdfs:member})$ |
| rdfs13 | $(d, \texttt{rdf:type}, \texttt{rdfs:Datatype})$ | $(d, \texttt{rdfs:subClassOf}, \texttt{rdfs:Literal})$ |

# RDFS-Entailment Example

For example:



# 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

$G_1 \models_X G_2$, iff $Clos_X(G_1)$ simply entails $G_2$, where $Clos_X(G_1)$ is constructed as follows:

1. Add to $G_1$ all axiomatic triples for $X \in \{$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 `rdf:_1`.
4. Apply rules for $X$ (i.e. $\{GrdfD1, rdfD2\}$ for $X = RDF$, or $\{Grdf1, rdfD2, rdfs1, \ldots, rdfs13\}$ for $X = RDFS$) with $D = \{rdf : langString, xsd : string\}$, 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