

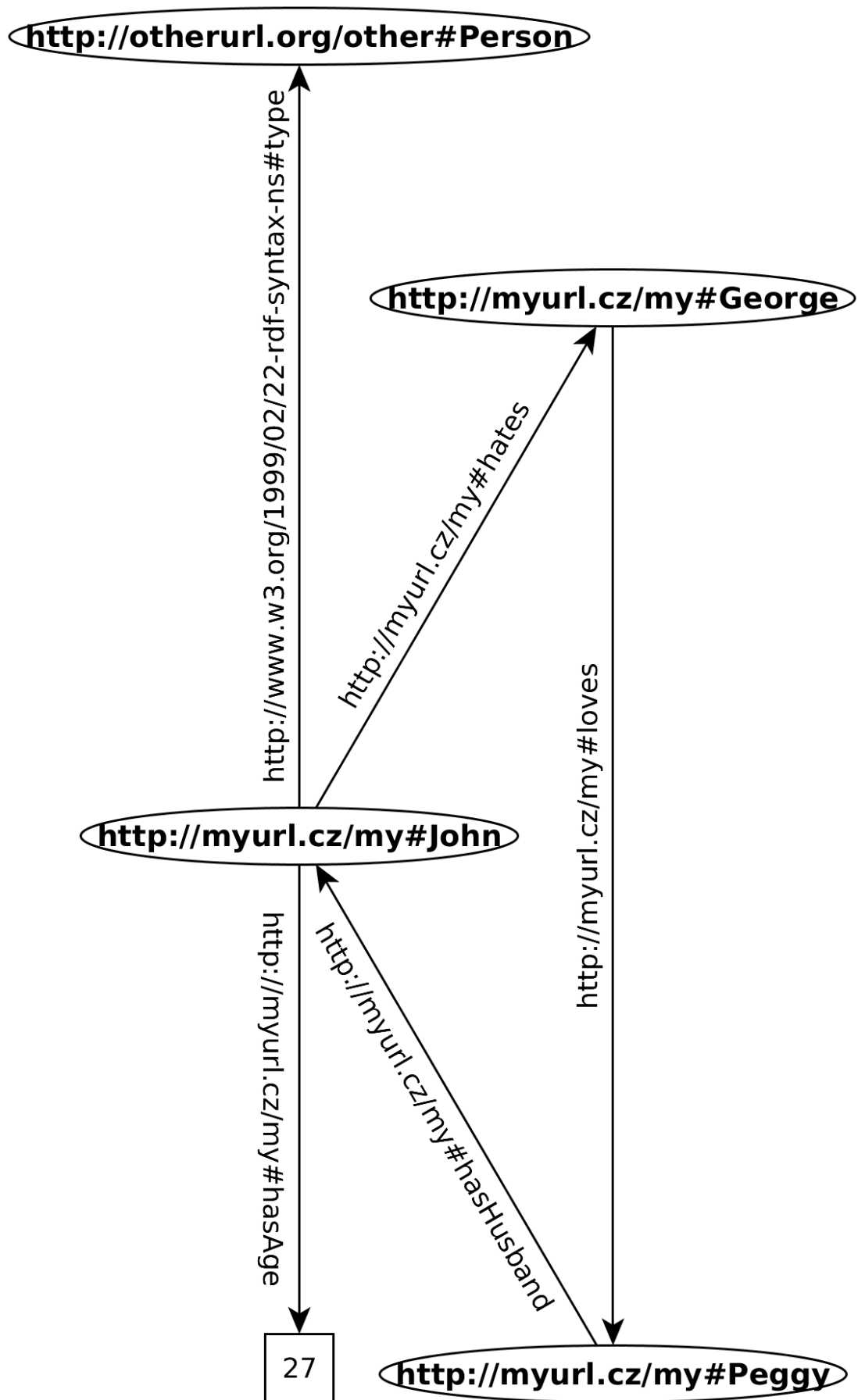
1 Introduction

1.1 RDF(S)

1.1.1 Core RDF

RDF Basics

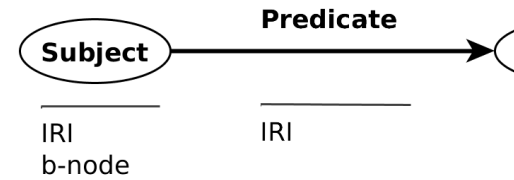
- RDF 1.0 – W3C Recommendation in 2004,
- Intuitively, RDF document is a graph, where each node is either (1) an IRI (ellipse), or (2) a literal (rectangle), or (3) a blank node (blank ellipse)
- RDF document is a set of triples (graph edges) in the form (Subject, Predicate, Object)



Main Definitions

Definitions

RDF Graph is a set of RDF triples,



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

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>`
 - `<http://myurl.cz/my/document-about-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`.

1 Introduction

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

- $\underbrace{\text{"dolphin"}}_{\text{lex. form}} \underbrace{\text{@en}}_{\text{lang. tag}}$
- $\underbrace{\text{"dolphin"}}_{\text{lex. form}} \underbrace{\text{^^xsd:string}}_{\text{datatype IRI}}$

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

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

lexical-to-value mapping $L2V$, e.g. $L2V(\text{datatype for } \text{xsd:integer}) = \{ \langle "01", 1 \rangle, \dots \}$.

- most XML Schema built-in datatypes – see [**Lanthaler:14:RCA**] for complete list:
 - `xsd:string`, `xsd:boolean`, `xsd:integer`, `xsd:decimal`, `xsd:dateTimeStamp`, `xsd:duration`, `xsd:gYear`, `xsd:gYearMonth`, `xsd:gMonth`, `xsd:gMonthDay`, `xsd:gDay`, `xsd:gDayTime`, `xsd:gTime`, `xsd:gDateTime`, `xsd:gTimeInstant`, `xsd:gDayTimeInstant`, `xsd:gDateTimeInstant`, `xsd:gDayTimeDuration`, `xsd:gDuration`, `xsd:dateTime`, `xsd:time`, `xsd:date`, `xsd:timeInstant`, `xsd:dateTime`, `xsd:timeDuration`, `xsd:dateDuration`, `xsd:duration`, `xsd:gYearMonthDuration`, `xsd:gMonthDuration`, `xsd:gDayDuration`, `xsd:gDayTimeDuration`, `xsd:gTimeDuration`, `xsd:gDateTimeDuration`, `xsd:gDayTimeInstant`, `xsd:gTimeInstant`, `xsd:dateTime`, `xsd:timeInstant`, `xsd:dateTimeInstant`, `xsd:timeDuration`, `xsd:dateDuration`, `xsd:duration`, `xsd:gYearMonthDuration`, `xsd:gMonthDuration`, `xsd:gDayDuration`, `xsd:gDayTimeDuration`, `xsd:gTimeDuration`, `xsd:gDateTimeDuration`
- `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 predefined 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)

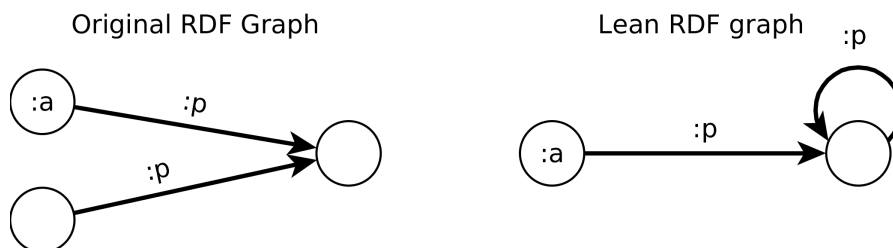
- denote existentially quantified variables,
- have local scope to the RDF document and cannot be reused,
- in Turtle/N-TRIPLES/SPARQL have `_:` prefix, e.g. `_:x`,

Definition

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

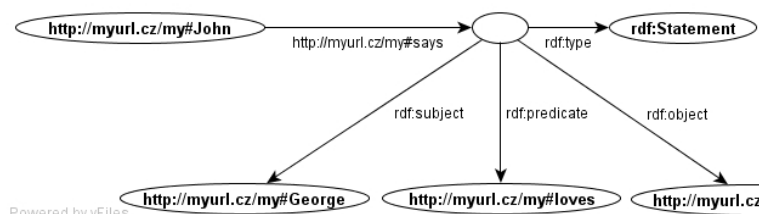
Instance of RDF Graph G_1 is an RDF Graph in which some b-nodes are replaced by an arbitrary RDF Term.

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

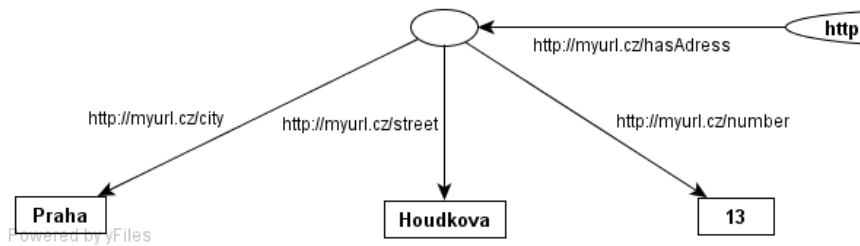


Blank Nodes Usage

- describing higher-order statements (reification)



1 Introduction

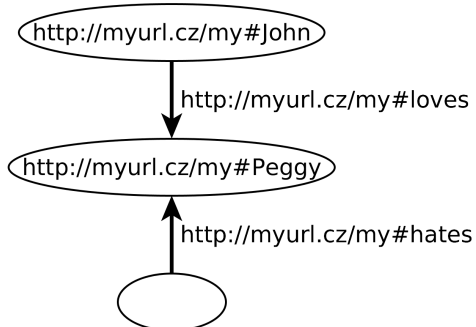


- expressing complex values
- container description – multisets, sequences, alternatives
- modeling n-ary relations (e.g. birth)

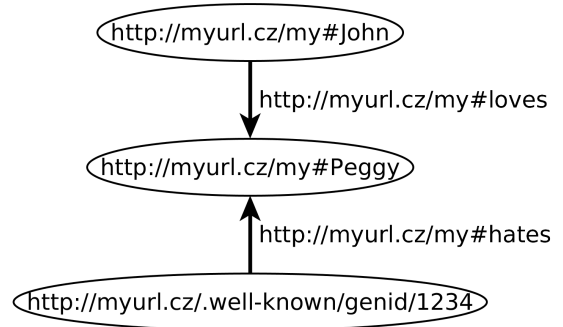
Blank Node Skolemization

- 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.

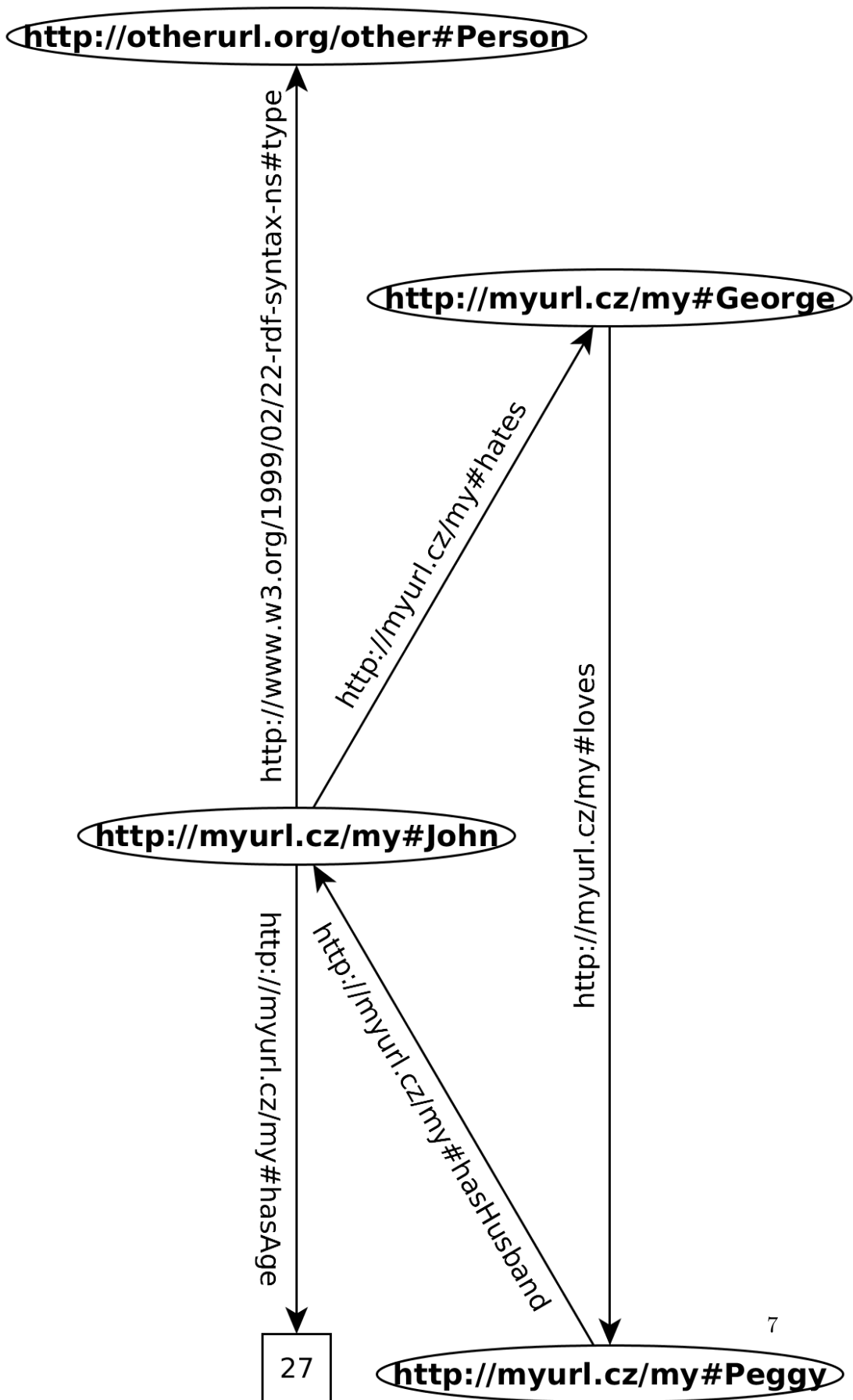
Original RDF Graph



Skolemized RDF Graph



Different syntaxes



1 Introduction

- Turtle syntax :

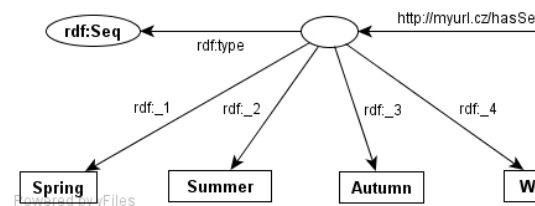
```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix my: <http://www.myurl.cz/my#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
my:George my:loves my:Peggy .
my:Peggy my:hasHusband my:John .
my:John rdf:type <http://www.otherurl.org/other#Person> ;
        my:hates my:George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

- RDF/XML syntax :

```
<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: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: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



RDF Model – Axiomatic Triples

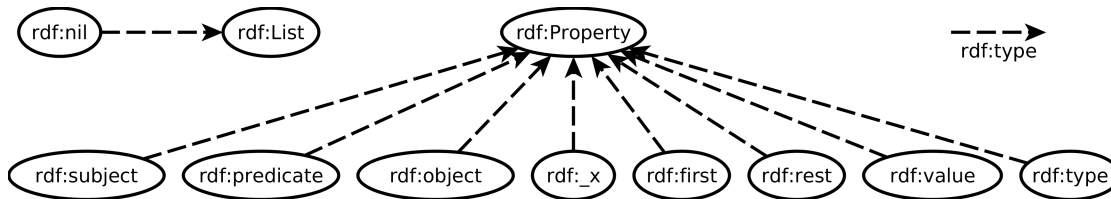


Figure 1.1: Visualization of axiomatic triples of RDF. Precise definition can be found in [Patel-Schneider:14:RS]

RDF 1.1

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

1.1.2 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 .
```

- 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

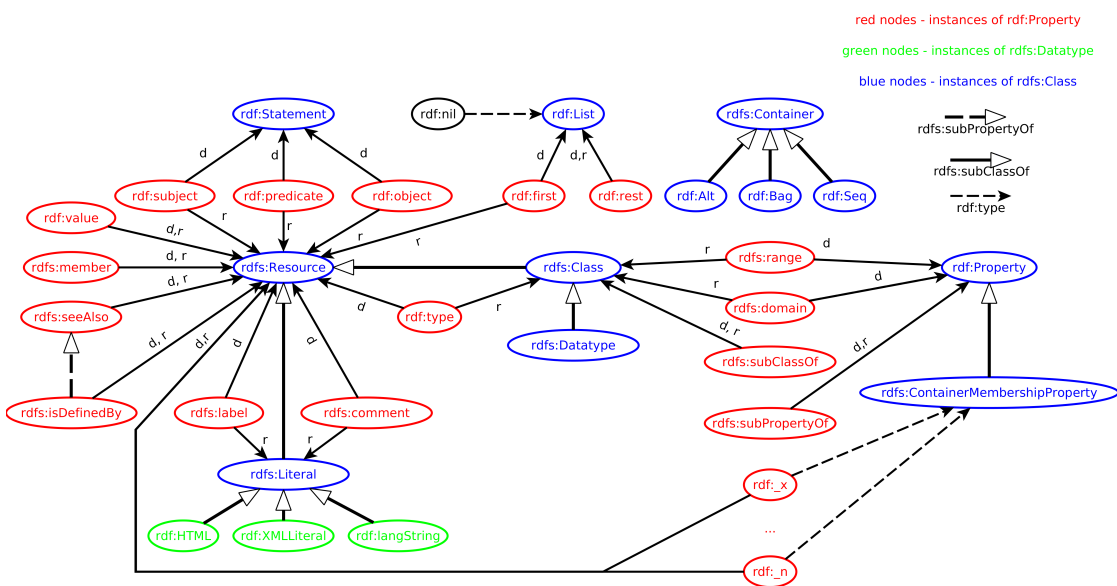


Figure 1.2: Visualization of axiomatic triples of RDFS. Precise definition can be found in [Patel-Schneider:14:RS]

1.1.3 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)

1 Introduction

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: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>
```

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

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

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

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

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

```

1.1.4 RDF Datasets

Definition

RDF dataset is a collection of RDF graphs:

$$DS = \{DG, (i_1, G_1), \dots, (i_n, G_n)\}$$

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 :

```

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

```

– G_2 :

1 Introduction

```
@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 .
```

1.1.5 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 $IL : N_{lit} \rightarrow IR$

For example

- $IR = \{John, Mary, 2\}$ (real resources)
- $IP = \{loves, childcount\}$ (real properties)

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

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

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 $A(B)$.

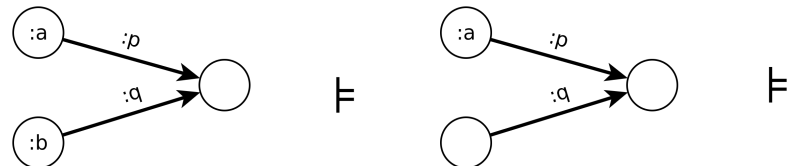
Simple Entailment

- 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

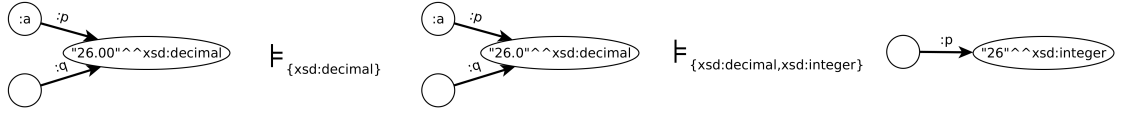
In addition to blank nodes, D -entailment (\models_D) interprets datatypes in the set D of recognized datatypes. Literals with non-recognized datatypes are treated as uninterpreted.

Semantic Conditions on D-Entailment

- if $rdf:langString \in D$, then for each literal $lex@lang$, $IL(lex@lang) = \langle lex, lowercase(lang) \rangle$

1 Introduction

- if $dIRI \in D$, then for each literal $lex \hat{\wedge} dIRI$: $IL(lex \hat{\wedge} dIRI) = L2V(I(dIRI))(lex)$, where
 - $I(dIRI)$ is a datatype identified by $dIRI$
 - $L2V(d)$ transforms a lexical value to the value space of d .

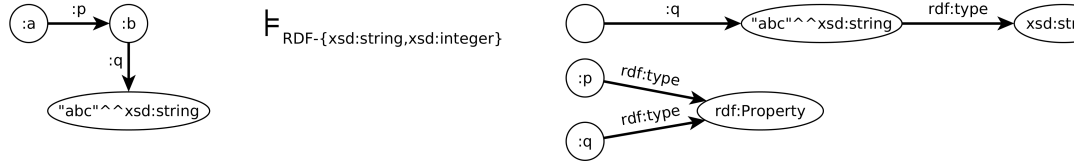


RDF-Entailment

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	$(s, p, lex \hat{\wedge} d)$ $d \in D$	$(lex \hat{\wedge} d, rdf : type, d)$
rdFD2	(s, p, o)	$(p, rdf : type, rdf : Property)$



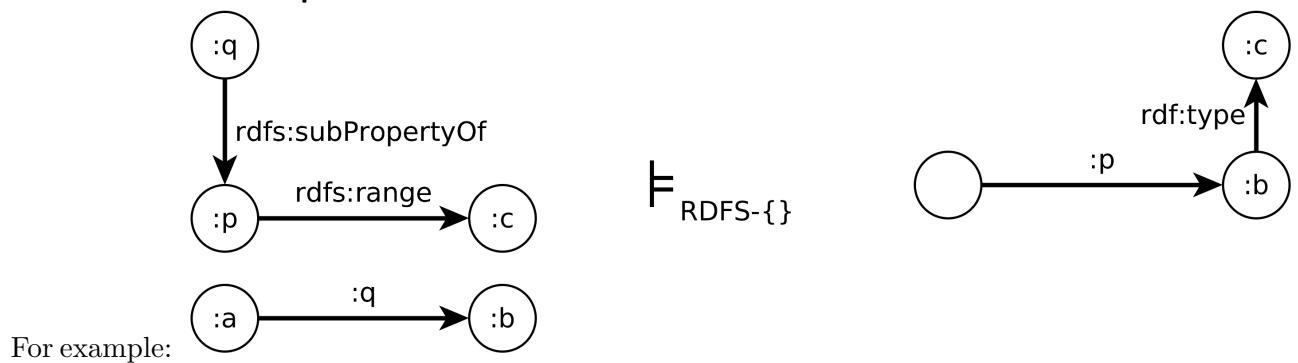
For example:

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	$(dIRI, rdf : type, rdfs : Datatype)$
rdfs2	$(s, p, o), (p, rdfs : domain, w)$	$(s, rdf : type, w)$
rdfs3	$(s, p, o), (p, rdfs : range, w)$	$(o, rdf : type, w)$
rdfs4	(s, p, o)	$(s, rdf : type, rdfs : Resource)$ $(o, rdf : type, rdfs : Resource)$
rdfs5	$(p_1, rdfs : subPropertyOf, p_2)$ $(p_2, rdfs : subPropertyOf, p_3)$	$(p_1, rdfs : subPropertyOf, p_3)$
rdfs6	$(p, rdf : type, rdf : Property)$	$(p, rdfs : subPropertyOf, p)$
rdfs7	$(p_1, rdfs : subPropertyOf, p_2)$ (s, p_1, o)	(s, p_2, o)
rdfs8	$(s, rdf : type, rdfs : Class)$	$(s, rdfs : subclassOf, rdfs : Resource)$
rdfs9	$(c_1, rdfs : subclassOf, c_2)$ $(s, rdf : type, c_1)$	$(s, rdf : type, c_2)$
rdfs10	$(c, rdf : type, rdfs : Class)$	$(c, rdfs : subclassOf, c)$
rdfs11	$(c_1, rdfs : subclassOf, c_2)$ $(c_2, rdfs : subclassOf, c_3)$	$(c_1, rdfs : subclassOf, c_3)$
rdfs12	$(p, rdf : type,$ $rdfs : ContainerMembershipProperty)$	$(p, rdfs : subPropertyOf,$ $rdfs : member)$
rdfs13	$(d, rdf : type, rdfs : Datatype)$	$(d, rdfs : subclassOf, rdfs : Literal)$

RDFS-Entailment 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 1.1, resp. Figure 1.2)
2. For each container membership property IRI p occurring 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. $\{Grdf D1, rdf D2\}$ for $X = RDF$, or $\{Grdf1, rdf D2, rdfs1, \dots, rdfs13\}$ 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