TECHNISCHE UNIVERSITÄT DRESDEN

KNOWLEDGE GRAPHS

Lecture 2: Representing Graphs with RDF

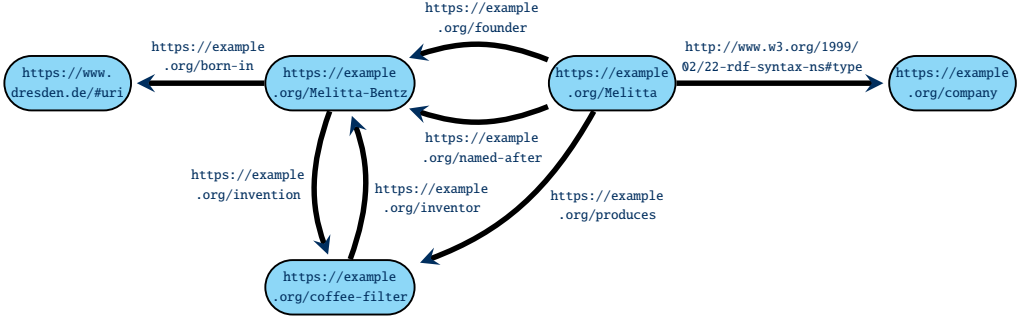
Markus Krötzsch

Knowledge-Based Systems

TU Dresden, 22th Oct 2019

Review: Basic Graphs in RDF

- RDF graphs can be viewed as directed, edge-labelled graphs
- Vertices are IRIs
- Edges are labelled by IRIs



Which IRIs to use?

Where do the IRIs that we use in graphs come from?

Which IRIs to use?

Where do the IRIs that we use in graphs come from?

- They can be newly created for an application
  ⇝ avoid confusion with resources in other graphs
- They can be IRIs that are already in common use
  ⇝ support information integration and re-use across graphs

Which IRIs to use?

Where do the IRIs that we use in graphs come from?

- They can be newly created for an application
  $\rightsquigarrow$ avoid confusion with resources in other graphs
- They can be IRIs that are already in common use
  $\rightsquigarrow$ support information integration and re-use across graphs

Guidelines for creating new IRIs:

1. Check if you could re-use an existing IRI $\rightsquigarrow$ avoid duplication if feasible
2. Use http(s) IRIs $\rightsquigarrow$ useful protocols, registries, resolution mechanisms
3. Create new IRIs based on domains that you own $\rightsquigarrow$ clear ownership; no danger of clashing with other people's IRIs
4. Don't use URLs of existing web pages, unless you want to store data about pages $\rightsquigarrow$ avoid confusion between pages and more abstract resources
5. Make your IRIs return some useful content via http(s) $\rightsquigarrow$ helps others to get information about your resources

Why IRIs?

URIs may seem a bit complicated

- They look a bit technical and complex
- They are hard to display or draw in a graph
- The guidelines just given may seem quite demanding to newcomers

Why IRIs?

URIs may seem a bit complicated

- They look a bit technical and complex
- They are hard to display or draw in a graph
- The guidelines just given may seem quite demanding to newcomers

However, it's not that hard:

- RDF can work with any form of IRI (most tools would probably accept any Latin-letter string with a colon inside!)
- The guidelines help sharing graphs across applications – a strength of RDF
- Internet domain name registration is a very simple way to define ownership in a global data space
- IRIs should not be shown to users (we will introduce human-readable labels soon)

Excursion: Resolvable IRIs

We asked for IRIs that are different from URLs of Web pages while at the same time returning some useful content via http(s).

How is this possible?

Excursion: Resolvable IRIs

We asked for IRIs that are different from URLs of Web pages while at the same time returning some useful content via http(s).

How is this possible?

(1) **Use fragments.** IRIs with fragments are different from the IRIs (URLs) without the fragment, but resolving them will return the content of this fragment-less IRI.

> **Example 2.1:** RDF uses some own IRIs for special purposes, e.g., `http://www.w3.org/1999/02/22-rdf-syntax-ns#type` for denoting the relation between resources and their type (class). This resolves to `http://www.w3.org/1999/02/22-rdf-syntax-ns` but is not the identifier of this document.

Excursion: Resolvable IRIs

We asked for IRIs that are different from URLs of Web pages while at the same time returning some useful content via http(s).

How is this possible?

(1) **Use fragments.** IRIs with fragments are different from the IRIs (URLs) without the fragment, but resolving them will return the content of this fragment-less IRI.

(2) **Use HTTP redirects.** Web servers can be configured to transparently redirect one URL to another; for IRIs it is common to use HTTP response codes 302 (temporary redirect) or 303 (see other)

> **Example 2.1:** The Wikidata IRI `http://www.wikidata.org/entity/Q5` redirects permanently (301) to `https://www.wikidata.org/entity/Q5`, which redirects (303) to `https://www.wikidata.org/wiki/Special:EntityData/Q5`, which by default redirects (303) to a JSON document `https://www.wikidata.org/wiki/Special:EntityData/Q5.json`.
>
> Real-world server configurations can be complicated.

Excursion: Content negotiation

The redirect mechanism has another useful application:
Web servers (and the software they run) can dynamically decide where to redirect to.

Content negotiation is the practice of offering several documents under the same URL by redirecting to the version that is most suitable for the request of a client. The HTTP protocol allows clients and servers to exchange extra information with their request that can be used to state preferences.

Example 2.2: The `Accept` header is commonly used by clients to select preferred result formats for a request. For example, the command

```
curl -L -H "Accept: text/turtle" http://www.wikidata.org/entity/Q5
```

makes Wikidata return RDF data in Turtle format rather than JSON.

Data values

IRIs can represent anything, but data values (numbers, strings, times, . . .) should not be represented by IRIs!

Data values

IRIs can represent anything, but data values (numbers, strings, times, . . .) should not be represented by IRIs!

Why not use IRIs here too?

Data values

IRIs can represent anything, but data values (numbers, strings, times, . . .) should not be represented by IRIs!

Why not use IRIs here too?

(1) Data values are the same everywhere $\rightsquigarrow$ no use in application-specific IRIs

Data values

IRIs can represent anything, but data values (numbers, strings, times, . . .) should not be represented by IRIs!

Why not use IRIs here too?

(1) Data values are the same everywhere $\rightsquigarrow$ no use in application-specific IRIs

(2) Many RDF-based applications need a built-in understanding of data values (e.g., for sorting content)

Data values

IRIs can represent anything, but data values (numbers, strings, times, . . .) should not be represented by IRIs!

Why not use IRIs here too?

(1) Data values are the same everywhere $\rightsquigarrow$ no use in application-specific IRIs

(2) Many RDF-based applications need a built-in understanding of data values (e.g., for sorting content)

(3) Data values are usually more “interpreted” than IRIs.

> **Example 2.3:** Using a hypothetical scheme “`integer`,” the IRIs `integer:42` and `integer:+42` would be different, but intuitively they should represent the same number.

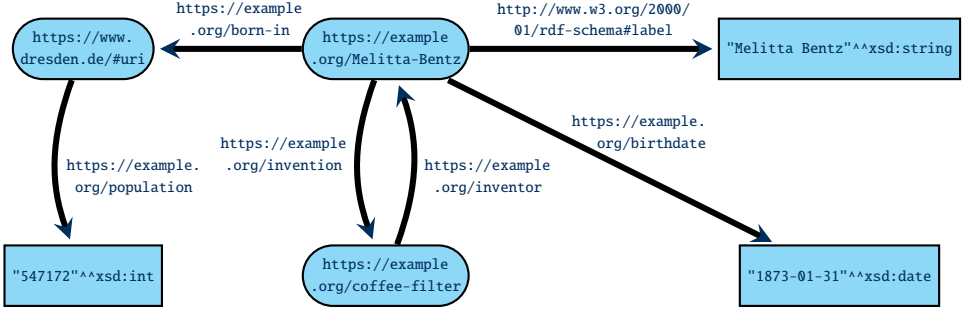
Encoding data values

- Data values in RDF are written in the format `"lexical value"^^datatype-IRI`
- They are drawn as rectangular nodes in graphs

Encoding data values

- Data values in RDF are written in the format `"lexical value"^^datatype-IRI`
- They are drawn as rectangular nodes in graphs

Example graph with data values:



RDF supports many different datatypes, most of which are based on XML Schema (“xsd”)

RDF datatypes

Definition 2.4: A datatype in RDF is specified by the following components:

- The value space is the set of possible values of this type.
- The lexical space is a set of strings[a] that can be used to denote values of this type.
- the lexical-to-value mapping is a function that maps each string from the lexical space to an element of the value space.

[a] RDF is based on Unicode strings, but this is inessential here.

Datatypes for RDF must be identified by IRIs (known to software that supports them).

RDF datatypes

Definition 2.4: A datatype in RDF is specified by the following components:

- The value space is the set of possible values of this type.
- The lexical space is a set of strings[a] that can be used to denote values of this type.
- the lexical-to-value mapping is a function that maps each string from the lexical space to an element of the value space.

[a] RDF is based on Unicode strings, but this is inessential here.

Datatypes for RDF must be identified by IRIs (known to software that supports them).

Example 2.5: The W3C standard XML Schema defines the datatype **integer**, identified by the IRI `http://www.w3.org/2001/XMLSchema#integer`. It has the value space of all integer numbers (of arbitrarily large absolute value), the lexical space of finite-length strings of decimal digits (0–9) with an optional leading sign (− or +), and the expected lexical-to-value mapping.

Important datatypes in RDF

Many standard datatypes are defined by XML Schema (currently in version 1.1 of 2012):

Name	Value space	Lexical space
`string`	XML-defined subset of Unicode strings	Value space (mapping is identity)
`boolean`	The set {true, false}	The set {`true`, `false`, `1`, `0`}
`decimal`	Arbitrary-precision decimal numbers	Sequences of decimal digits, optionally signed and with one `.`
`integer`	Arbitrary integer numbers	Like `decimal` but without `.`
`int`	$\{-2147483648, \ldots, 2147483647\}$ (32bit)	Integer strings that map to values within this range
Similar types exist for `long` (64bit), `short` (16bit), `byte` (8bit), unsigned versions, etc.		
`double`	IEEE double-precision 64-bit floating point numbers	decimal strings, exp notation numbers, special values (e.g., `NaN`)
`float`	single-precision 32-bit floating point numbers	same as `double`

Important datatypes in RDF (2)

There are also many types for dates and times:

Name	Value space	Lexical space
`dateTime`	dates with times represented by a seven-tuple consisting of year, month, day, hours, minutes, seconds, timezone offset (constrained to time-like ranges)	ISO-style date-time strings, e.g., `2018-10-23T11:45:30+01:00`; one can write `Z` for offset `00:00`

There are also partial time types, such as `date`, `time`, `gYear` (Gregorian year), etc.

Many further types exist: a full list can be found at `https://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/#section-Datatypes`

RDF datatype literals

Definition 2.6: An RDF literal is an expression of form "lexical value"^^datatype, where lexical value is a string and datatype is an IRI.

Literals are semantically interpreted to denote a value in the value space as defined by the datatype's lexical-to-value mapping. If the given string is not a valid lexical value, the literal is ill-typed.

Note: RDF 1.1 also allows for string literals "lexical value"; in this case XML Schema `string` is assumed as datatype.

Special case: strings in a language

Furthermore, RDF supports strings that have a specific language:

Definition 2.7: A language-tagged string is an expression of the form "string"@language where string is a Unicode string and language is a well-formed language tag (after BCP47). They are interpreted as pairs of strings with a (lower-cased) language tag.

The datatype of these literals is defined to be `http://www.w3.org/1999/02/22-rdf-syntax-ns#langString` (but this is never used in syntax).

Example 2.8: The strings `"Pommes Frites"@de`, `"chips"@en-UK`, and `"French fries"@en-US` are language-tagged.

This special case of literal is widely used in practice to encode human-readable labels.

Blank nodes

RDF also supports vertices that are not identified by a IRI, called blank nodes or bnodes.

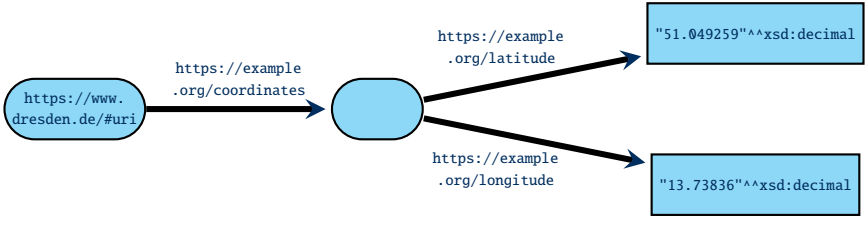
- Intuitively, bnodes are placeholder for some specific (but unspecified) node
- Their use makes the claim: “there is something at this position”
- Similar to existentially quantified variables in logic

Blank nodes

RDF also supports vertices that are not identified by a IRI, called blank nodes or bnodes.

- Intuitively, bnodes are placeholder for some specific (but unspecified) node
- Their use makes the claim: “there is something at this position”
- Similar to existentially quantified variables in logic

Example: Blank nodes have historically been used for auxiliary vertices



Note: Today, bnodes are largely avoided. They still occur in the RDF-encoding of the OWL Web Ontology Language, but specialised tools are used in this application anyway.

RDF Graphs

We now have defined all necessary kinds of RDF terms: IRIs, blank nodes, and literals.

RDF Graphs

We now have defined all necessary kinds of RDF terms: IRIs, blank nodes, and literals. The formal definition of RDF graph is maybe slightly different from expectations:

Definition 2.9: An RDF graph is a set of triples consisting of the following parts:

- a subject that is an IRI or blank node
- a predicate that is an IRI
- a object that is an IRI, blank node, or literal

RDF Graphs

We now have defined all necessary kinds of RDF terms: IRIs, blank nodes, and literals. The formal definition of RDF graph is maybe slightly different from expectations:

Definition 2.9: An RDF graph is a set of triples consisting of the following parts:

- a subject that is an IRI or blank node
- a predicate that is an IRI
- a object that is an IRI, blank node, or literal

Notes:

- This view resembles a (labelled) adjacency list encoding
- The restrictions on the use of blank nodes and literals in triples is a bit arbitrary
- RDF graphs are mostly syntactic (rather what we write than what we mean)
- In particular, literals are not interpreted when defining graphs
  $\rightsquigarrow$ multiple ways of writing the same value lead to multiple graphs
  $\rightsquigarrow$ ill-formed literals are allowed in graphs

Serialisations

RDF Serialisations

What we outlined so far is the abstract sytnax of RDF. To exchange graphs, we need concrete syntactic forms to encode RDF graphs.

RDF Serialisations

What we outlined so far is the abstract sytnax of RDF. To exchange graphs, we need concrete syntactic forms to encode RDF graphs.

There are numerous syntactic formats available:

- N-Triples as a simple line-based format
- Turtle adds convenient abbreviations to N-Triples
- JSON-LD for encoding RDF graphs in JSON
- RDF/XML for encoding RDF graphs in XML
- RDFa for embedding RDF graphs into HTML

Further historic/unofficial formats exist but are hardly relevant today.

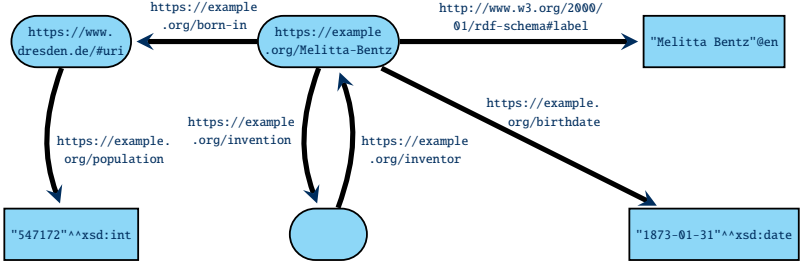
N-Triples

N-Triples is almost the simplest format conceivable:

- Each line encodes one triple:
  - IRIs are written in pointy brackets, e.g., <https://www.tu-dresden.de/>
  - Literals are written as usual with a given type IRI, e.g., "2019-10-22"^^<http://www.w3.org/2001/XMLSchema#date> or with a language-tag, e.g., "knowledge graph"@en
  - Blank nodes are written as _:stringId, where stringId is a string that identifies the blank node within the document (it has no global meaning)
  - Parts are separated by whitespace, and lines end with .
- Unicode is supported, but various escape sequences also work
- Comments are allowed after triples (nowhere else); they start with #

Full specification at https://www.w3.org/TR/n-triples/

Example



could be encoded as (line-breaks within triples for presentation only):

```
<https://example.org/Melitta-Bentz>  <http://www.w3.org/2000/01/rdf-schema#label>  "Melitta Bentz"@en .
<https://example.org/Melitta-Bentz>  <https://example.org/birthdate>
                                        "1873-01-31"^^<http://www.w3.org/2001/XMLSchema#date> .
<https://example.org/Melitta-Bentz>  <https://example.org/invention>  _:1 .
<https://example.org/Melitta-Bentz>  <https://example.org/born-in>  <https://www.dresden.de/#uri> .
<https://www.dresden.de/#uri>  <https://example.org/population>
                                                  "547172"^^<http://www.w3.org/2001/XMLSchema#int> .
_:1  <https://example.org/inventor>  <https://example.org/Melitta-Bentz> .
```

N-Triples: Summary

Advantages:

- Very simple
- Fast and easy to parse
- Processable even with basic text-processing tools, e.g., grep

Disadvantages:

- Somewhat inefficient in terms of storage space
- Not particularly human-friendly (reading and writing)

Turtle

The Turtle format extends N-Triples with several convenient abbreviations:

- Prefix declarations and base namespaces allow us to shorten IRIs
- If we terminate triples with `;` (respectively with `,`) then the next triple is assumed to start with the same subject (respectively with the same subject and predicate)
- Blank nodes can be encoded using square brackets; they might contain predicate-object pairs that refer to the blank node as subject
- More liberal support for comments (possibly on own line)
- Simpler forms for some kinds of data values

There are several other shortcuts and simplifications. Full specification at `https://www.w3.org/TR/turtle/`.

PREFIX and BASE by example

- `BASE` is used to declare a base IRI, so that we can use relative IRIs
- `PREFIX` is used to declare abbreviations for IRI prefixes

Example 2.10: We can write the previous example as follows:

```
BASE <https://example.org/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

<Melitta-Bentz>  rdfs:label  "Melitta Bentz"@en .
<Melitta-Bentz>  <birthdate>  "1873-01-31"^^xsd:date .
<Melitta-Bentz>  <invention>  _:1 .
<Melitta-Bentz>  <born-in>  <https://www.dresden.de/#uri> .
<https://www.dresden.de/#uri>  <population>  "547172"^^xsd:int .
_:1  <inventor>  <Melitta-Bentz> .
```

Note: Relative IRIs are still written in < and > (e.g., <`birthdate`>); prefixed names are written without brackets (e.g., `rdfs:label`).

Use of semicolon by example

- If we terminate triples with ; then the next triple is assumed to start with the same subject
- If we terminate triples with , then the next triple is assumed to start with the same subject and predicate

Example 2.11: We can write the previous example as follows:

```
BASE <https://example.org/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

<Melitta-Bentz>  rdfs:label  "Melitta Bentz"@en ;
                 <birthdate>  "1873-01-31"^^xsd:date ;
                 <invention>  _:1 ;
                 <born-in>  <https://www.dresden.de/#uri> .
<https://www.dresden.de/#uri>  <population>  "547172"^^xsd:int .
_:1  <inventor>  <Melitta-Bentz> .
```

Brackets for bnodes by example

- The expression `[]` represents a bnode (without id)
- predicate-object pairs within `[...]` are allowed to give further triples with the bnode as subject

Example 2.12: We can write the previous example as follows:

```
BASE <https://example.org/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

<Melitta-Bentz>  rdfs:label  "Melitta Bentz"@en ;
                 <birthdate>  "1873-01-31"^^xsd:date ;
                 <invention>  [ <inventor>  <Melitta-Bentz> ] ;
                 <born-in>  <https://www.dresden.de/#uri> .
<https://www.dresden.de/#uri>  <population>  "547172"^^xsd:int .
```

Abbreviating numbers and booleans

- Numbers can just be written without quotes and type to denote literals in default types (integer, decimal, or double)
- Booleans can also be written as `true` or `false` directly

Example 2.13: We can write the previous example as follows:

```
BASE <https://example.org/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

<Melitta-Bentz>  rdfs:label  "Melitta Bentz"@en ;
                 <birthdate>  "1873-01-31"^^xsd:date ;
                 <invention>  [ <inventor>  <Melitta-Bentz> ] ;
                 <born-in>  <https://www.dresden.de/#uri> .
<https://www.dresden.de/#uri>  <population>  547172 .
```

Turtle: Summary

Advantages:

- Still quite simple
- Not hard to parse
- Human-readable (if formatted carefully)

Disadvantages:

- Not safely processable with grep and similar tools

Other syntactic forms

There are various further syntactic forms:

- RDF/XML: An XML-based encoding; historically important in RDF 1.0; hard-to-parse but unable to encode all RDF graphs; not human-readable either
- JSON-LD: A JSON-based encoding and away of specifying how existing JSON maps to RDF; can re-use fast JSON parsers (esp. those in browsers)
- RDFa: An HTML embedding of RDF triples; used for HTML document annotations (e.g., with schema.org); mostly for consumption by Web crawlers

Details are found online; we will not cover them here.

Common namespaces/prefixes

Many syntactic encodings of RDF support some abbreviation mechanism for IRIs by declaring some form of namespaces or prefixes.

While prefixes can usually be declared freely, there are some standard prefixes that are conventionally used and virtually always declared in the same way. They include:

Abbr.	Abbreviated IRI prefix	Usage
`xsd:`	`http://www.w3.org/2001/XMLSchema#`	XML Schema datatypes
`rdf:`	`http://www.w3.org/1999/02/22-rdf-syntax-ns#`	RDF vocabulary
`rdfs:`	`http://www.w3.org/2000/01/rdf-schema#`	"RDF Schema," more RDF vocabulary

Convention 2.14: We will henceforth assume that these abbreviations are used with the above meaning throughout this course.

Abbreviations such as `xsd:dateTime` are sometimes called qualified names (qnames)

RDF Datasets

RDF 1.1 also supports datasets that consist of several graphs:

- This is useful for organising RDF data, especially within databases
- Several named graphs are identified by IRIs; there is also one default graph without any IRI
- RDF dataset = RDF data that may have more than one graph

RDF Datasets

RDF 1.1 also supports datasets that consist of several graphs:

- This is useful for organising RDF data, especially within databases
- Several named graphs are identified by IRIs; there is also one default graph without any IRI
- RDF dataset = RDF data that may have more than one graph

Only some specialised syntactic forms can serialise RDF datasets with named graphs:

- N-Quads: Extension of N-Triples with optional fourth component in each line to denote graph.
- TriG: Extension of Turtle with a new feature to declare graphs (group triples of one graph in braces, with the graph IRI written before the opening brace)
- JSON-LD: Can also encode named graph

RDF Datasets

RDF 1.1 also supports datasets that consist of several graphs:

- This is useful for organising RDF data, especially within databases
- Several named graphs are identified by IRIs; there is also one default graph without any IRI
- RDF dataset = RDF data that may have more than one graph

Only some specialised syntactic forms can serialise RDF datasets with named graphs:

- N-Quads: Extension of N-Triples with optional fourth component in each line to denote graph.
- TriG: Extension of Turtle with a new feature to declare graphs (group triples of one graph in braces, with the graph IRI written before the opening brace)
- JSON-LD: Can also encode named graph

The semantics of named graphs was left open by the Working Group. Are all graphs' triples asserted to hold, or just those in the default graph? Do the IRIs of graphs denote the resource that is the set of triples given, or something else?
$\rightsquigarrow$ currently application-dependent

RDF properties

RDF uses IRIs in predicate positions

- Resources represented by predicates are called properties
- We can make statements about properties by using their IRIs as subjects in triples

Example 2.15: It is common to assign labels to properties, and many applications display these labels to show triples that use these properties as their predicate.

RDF properties

RDF uses IRIs in predicate positions

- Resources represented by predicates are called properties
- We can make statements about properties by using their IRIs as subjects in triples

Example 2.15: It is common to assign labels to properties, and many applications display these labels to show triples that use these properties as their predicate.

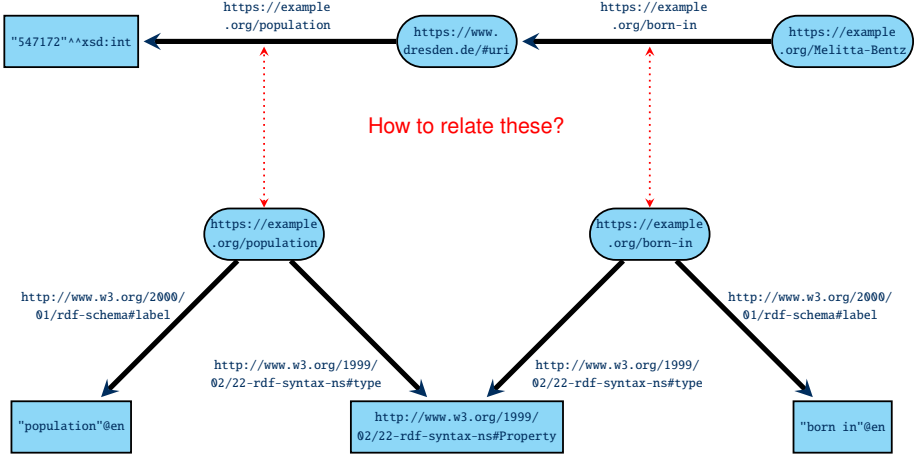
RDF also allows us to declare a resource as a property, using the following triple (in Turtle, using standard abbreviations):

Property-IRI rdf:type rdf:Property .

Much further information about a property can be specified using properties of RDF and other standard vocabularies (esp. OWL)

Describing properties

Problem: The relation of predicates (edge-labels) to certain resources is not convenient to show in a drawing!

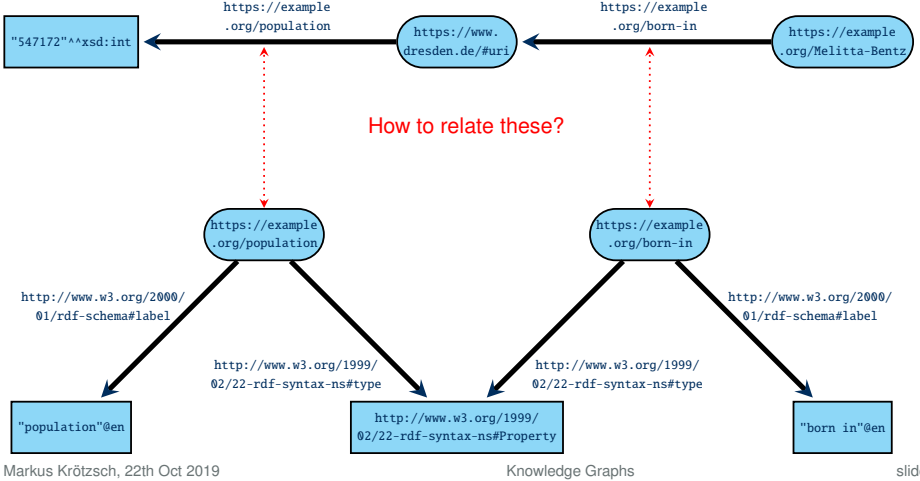


RDF graphs as hypergraphs

The drawing issue hints at a general observation: RDF graphs are really hypergraphs with ternary edges. The edges are unlabelled and directed (with three types of “tips”); the graph is simple (each triple at most once).

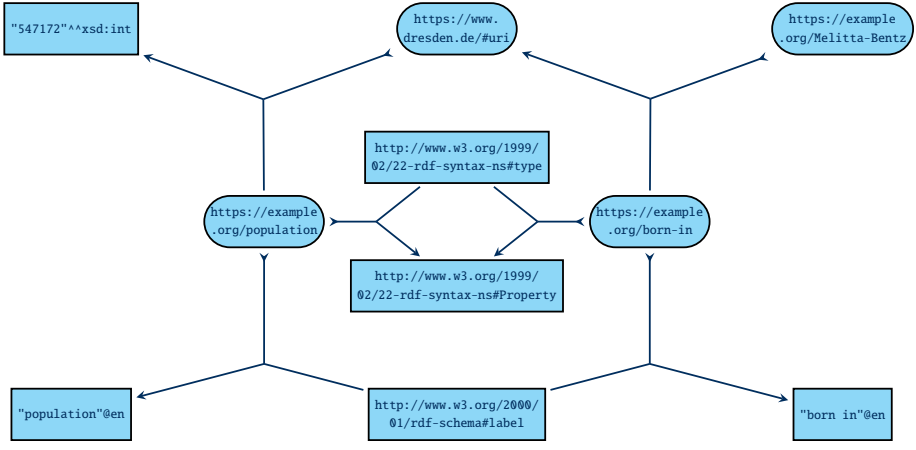
RDF graphs as hypergraphs

The drawing issue hints at a general observation: RDF graphs are really hypergraphs with ternary edges. The edges are unlabelled and directed (with three types of “tips”); the graph is simple (each triple at most once).



RDF graphs as hypergraphs

The drawing issue hints at a general observation: RDF graphs are really hypergraphs with ternary edges. The edges are unlabelled and directed (with three types of “tips”); the graph is simple (each triple at most once).



Describing schema information in RDF

Triples about properties can also be used to specify how properties should be used.

Example 2.16: RDF provides several properties for describing properties:

```
<PropertyIRI>  rdf:type  rdfs:Property .      # declare resource as property
<PropertyIRI>  rdfs:label  "some label"@en . # assign label
<PropertyIRI>  rdfs:comment  "Some human-readable comment"@en .
<PropertyIRI>  rdfs:range  xsd:decimal .      # define range datatype
<PropertyIRI>  rdfs:domain  <classIRI> .      # define domain type (class)
```

Describing schema information in RDF

Triples about properties can also be used to specify how properties should be used.

Example 2.16: RDF provides several properties for describing properties:

```
<PropertyIRI>  rdf:type  rdfs:Property .      # declare resource as property
<PropertyIRI>  rdfs:label  "some label"@en . # assign label
<PropertyIRI>  rdfs:comment  "Some human-readable comment"@en .
<PropertyIRI>  rdfs:range  xsd:decimal .      # define range datatype
<PropertyIRI>  rdfs:domain  <classIRI> .      # define domain type (class)
```

- There are many properties beyond those from the RDF standard for such purposes, e.g., `rdfs:label` can be replaced by `<http://schema.org/name>` or `<http://www.w3.org/2004/02/skos/core#prefLabel>`
- RDF defines how its properties should be interpreted semantically (see later lectures)
- There are more elaborate ways of expressions schematic information in RDF
  - The OWL Web Ontology Language extends the semantic features of RDF
  - Constraint languages SHACL and SHEX can restrict graphs syntactically

  (see later lectures)

No schema?

RDF supports properties without any declaration, even with different types of values

Example 2.17: The property `http://purl.org/dc/elements/1.1/creator` from the Dublin Core vocabulary has been used with values that are IRIs (denoting a creator) or strings (names of a creator).

- RDF tools can tolerate such lack of schema . . .
- . . . but it is usually not desirable for applications

The robustness of RDF is useful for merging datasets, but it's easier to build services around more uniform/constrained data

Summary

RDF is a W3C standard for describing directed, edge-labelled graphs in an interoperable way

It identifies vertices and edge-types using IRIs, and it can use many datatypes

Several syntactic formats exist for exchanging RDF data, the simplest being N-Triples

Edge labels (predicates) in RDF are represented by their respective properties, which can be used to add further information

RDF can most naturally be viewed as hypergraphs with ternary edges

What's next?

- How can we encode data in RDF?
- Where can we get RDF data? Application examples?
- How can we query and analyse such graphs?