TECHNISCHE UNIVERSITÄT DRESDEN

International Center for Computational Logic

KNOWLEDGE GRAPHS

Lecture 2: Representing Graphs with RDF

Markus Krötzsch

Knowledge-Based Systems

TU Dresden, 22 Oct 2024

More recent versions of this slide deck might be available.
For the most current version of this course, see
https://iccl.inf.tu-dresden.de/web/Knowledge_Graphs/en

Encoding Graphs

We have seen that graphs can be encoded in several ways:

- Adjacency matrix (and variants)
- Adjacency list (and variants)
- Other derived representations

Encoding Graphs

We have seen that graphs can be encoded in several ways:

- Adjacency matrix (and variants)
- Adjacency list (and variants)
- Other derived representations

This is enough to store and manipulate graphs in software,
but it is not enough to exchange graphs across applications.

Encoding Graphs

We have seen that graphs can be encoded in several ways:

- Adjacency matrix (and variants)
- Adjacency list (and variants)
- Other derived representations

This is enough to store and manipulate graphs in software,
but it is not enough to exchange graphs across applications.

Open questions:

- What kind(s) of graph do we want to exchange?
- How are vertices given (numbers? strings? specific ids? . . .)?
- Are edge labels supported and what are they?
- Can the graph include values of data types (integer? float? string? times? . . .)?
- How exactly are these things encoded in bytes in a file?

The Resource Description Framework

RDF is a W3C[1] standard for exchanging graphs

- First proposed in 1999
- Updated in 2004 (RDF 1.0) and in 2014 (RDF 1.1)
- Originally built for Web data exchange
- Meanwhile used in many graph database applications
- Supported by many other W3C standards (RDFa, SPARQL, OWL, SHACL, . . .)

[1] World Wide Web Consortium

The Resource Description Framework

RDF is a W3C[1] standard for exchanging graphs

- First proposed in 1999
- Updated in 2004 (RDF 1.0) and in 2014 (RDF 1.1)
- Originally built for Web data exchange
- Meanwhile used in many graph database applications
- Supported by many other W3C standards (RDFa, SPARQL, OWL, SHACL, . . .)

In this course: focus on graph representation features of RDF 1.1

[1] World Wide Web Consortium

The Resource Description Framework

RDF is a W3C[1] standard for exchanging graphs

- First proposed in 1999
- Updated in 2004 (RDF 1.0) and in 2014 (RDF 1.1)
- Originally built for Web data exchange
- Meanwhile used in many graph database applications
- Supported by many other W3C standards (RDFa, SPARQL, OWL, SHACL, . . .)

In this course: focus on graph representation features of RDF 1.1

W3C creates open standards: patent-free & freely accessible

- Gentle RDF 1.1 introduction: `https://www.w3.org/TR/rdf11-primer/`
- Specification of graph model: `https://www.w3.org/TR/rdf11-concepts/`
- Specific file formats are defined in other documents, linked from those

[1] World Wide Web Consortium

Graphs in RDF

RDF allows us to specify graphs that are:

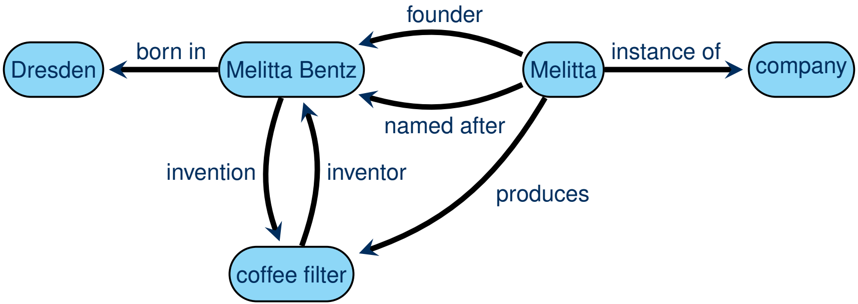
- directed (edges have a source and a target)
- edge-labelled (edges have one label)
- a restricted form of multi-graphs (multiple edges can exist between same vertices, but only if they have different labels)

Graphs in RDF

RDF allows us to specify graphs that are:

- directed (edges have a source and a target)
- edge-labelled (edges have one label)
- a restricted form of multi-graphs (multiple edges can exist between same vertices, but only if they have different labels)

Example of such a graph:



Elements of RDF Graphs

Question 1: What are the vertices in RDF graphs?

Elements of RDF Graphs

Question 1: What are the vertices in RDF graphs?

RDF allows three kinds of vertices:

- Vertices with an identifier (technical term: IRIs)
- Vertices without an identifier (technical term: blank node)
- Vertices that are data values (technical term: literal)

Elements of RDF Graphs

Question 1: What are the vertices in RDF graphs?

RDF allows three kinds of vertices:

- Vertices with an identifier (technical term: IRIs)
- Vertices without an identifier (technical term: blank node)
- Vertices that are data values (technical term: literal)

Question 2: What are the edge labels in RDF graphs?

Elements of RDF Graphs

Question 1: What are the vertices in RDF graphs?

RDF allows three kinds of vertices:

- Vertices with an identifier (technical term: IRIs)
- Vertices without an identifier (technical term: blank node)
- Vertices that are data values (technical term: literal)

Question 2: What are the edge labels in RDF graphs?

RDF requires all edges to be labelled with identifiers (IRIs).

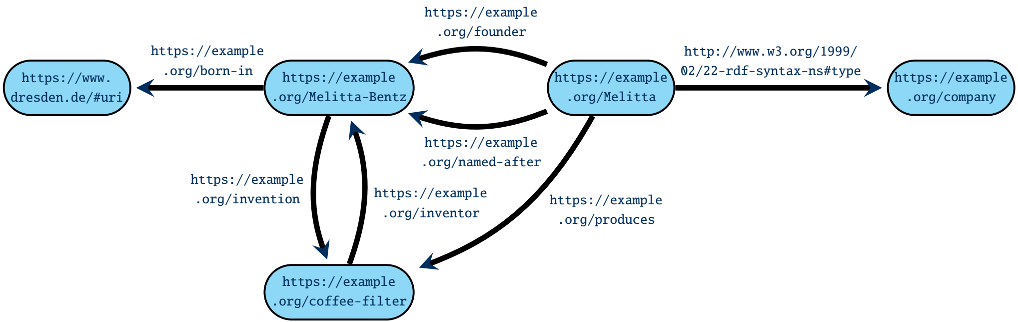
IRIs in RDF

IRIs (Internationalized Resource Identifiers) are string ids that resemble the widely known URLs. Technical details are defined in RFC 3987.

RDF uses IRIs in two ways:

- IRIs can be vertices in the graph
- IRIs are used as edge labels

Example graph with IRIs:



IRIs vs. URLs

Example 2.1: URLs are a well-known special form of IRI. For example:

$$\underbrace{\texttt{https}}_{\text{scheme}}\texttt{:}\underbrace{\texttt{//example.org}}_{\text{authority}}\underbrace{\texttt{/some/page}}_{\text{path}}\underbrace{\texttt{?get=something\&lang=en}}_{\text{query}}\underbrace{\texttt{\#results}}_{\text{fragment}}$$

URLs are used to locate resources, especially on the Web:

- Various schemes exist (`http`, `https`, `ftp`, `file`, `...`), many with own technical rules and protocols
- Various forms of escaping of special characters exist (e.g., é can occur in IRIs or be encoded as `%C3%A9` or as `é`)
- Web URLs have a domain, and domain registration is governed by dedicated organisations and contracts

IRIs vs. URLs

Example 2.1: URLs are a well-known special form of IRI. For example:

$$\underbrace{\texttt{https}}_{\text{scheme}}\texttt{:}\underbrace{\texttt{//example.org}}_{\text{authority}}\underbrace{\texttt{/some/page}}_{\text{path}}\underbrace{\texttt{?get=something\&lang=en}}_{\text{query}}\underbrace{\texttt{\#results}}_{\text{fragment}}$$

URLs are used to locate resources, especially on the Web:

- Various schemes exist (`http`, `https`, `ftp`, `file`, `...`), many with own technical rules and protocols
- Various forms of escaping of special characters exist (e.g., é can occur in IRIs or be encoded as `%C3%A9` or as `é`)
- Web URLs have a domain, and domain registration is governed by dedicated organisations and contracts

IRIs in RDF are just a special form of string identifiers:

- Identifiers just have to start with something like a scheme, e.g., `myiri:myid42`
- Character escaping plays no role: IRIs are only equal in RDF if they are the same sequence of Unicode symbols
- Domain registration is not needed (but it is good to use own domains for your IRIs)

Data values in RDF

Many applications require concrete data values (numbers, strings, dates, etc.).

RDF supports data values from many domains:

- decimal numbers, including the special case of integers
- floating-point numbers (32bit or 64bit)
- boolean values (true and false)
- strings
- strings with a language tag
- dates and times
- . . .

RDF features an extensible datatype system:

- Applications may not support all types of data
- Applications can introduce new types of data

Encoding data values

To use a data value as a vertex in a graph, we must give its value **and** type.

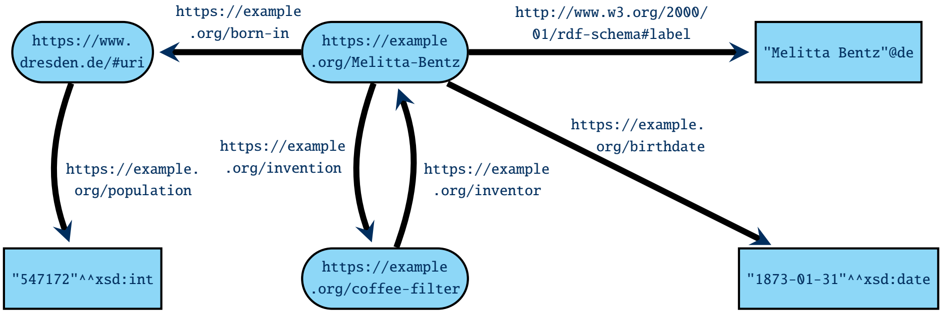
- Most data values in RDF are written in the format `"lexical value"^^datatype-id`
- Strings with language tag use the special form `"string"@language`
- Shortcuts will sometimes be supported, e.g., for strings and numbers

Encoding data values

To use a data value as a vertex in a graph, we must give its value **and** type.

- Most data values in RDF are written in the format `"lexical value"^^datatype-id`
- Strings with language tag use the special form `"string"@language`
- Shortcuts will sometimes be supported, e.g., for strings and numbers

Example graph with data values:



We depict data value vertices as rectangular boxes when drawing RDF graphs

RDF datatype literals

Definition 2.2: 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: Shortcuts are allowed in some formats. For example, literals of type string can simply be written as "lexical value".

RDF datatypes

Definition 2.3: 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 2.3: 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 2.4: 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 $\{\text{true}, \text{false}\}$	The set $\{\texttt{true}, \texttt{false}, \texttt{1}, \texttt{0}\}$
`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`

Special case: strings in a language

Furthermore, RDF supports strings that have a specific language:

Definition 2.5: 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 2.6: 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 an IRI, called blank nodes or bnodes.

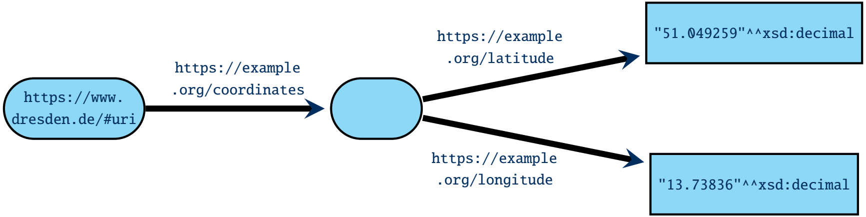
- Intuitively, bnodes are placeholder for some specific (but unspecified) node
- Their use makes the claim: “there is something at this position”
- Similar to existentially quantified variables in logic

Blank nodes

RDF also supports vertices that are not identified by an 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 2.7: 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

Interpreting RDF

What the W3C standard says about graphs

- Sets of RDF triples are considered (abstract) syntax
- Literals in such triple sets are not interpreted
  $\rightsquigarrow$ multiple ways of writing the same value lead to multiple graphs
  $\rightsquigarrow$ ill-formed literals are allowed in graphs

Interpreting RDF according to the W3C standard

- W3C standards defines several "official" ways of interpreting RDF
- Complicated definitions, rarely used formally in practice, ignored in theory

What is common in practice

- Many applications use RDF to describe graph structures
- Literals are interpreted as data values (different ways of writing the same value are normalised, ill-formed literals rejected)

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

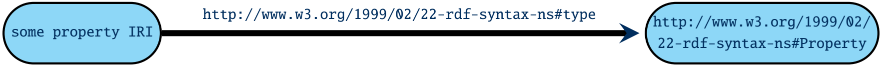
RDF properties

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

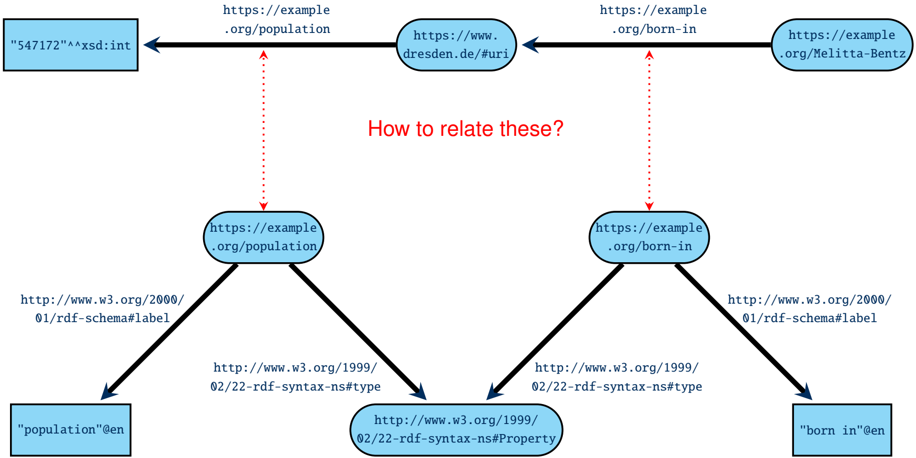
We can declare a resource as a property using special RDF vocabulary:



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!

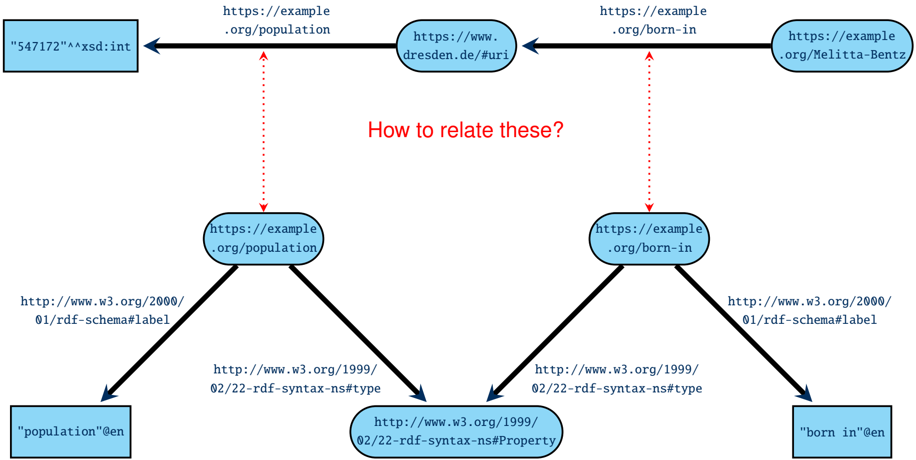


RDF graphs as hypergraphs

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).

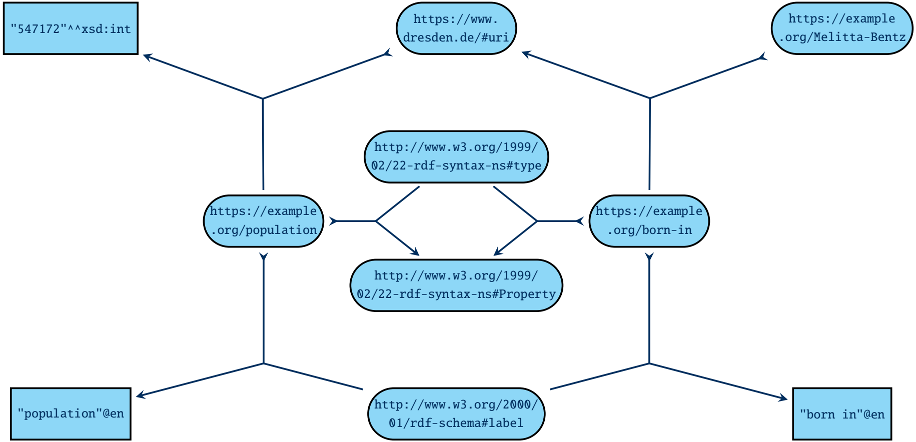
RDF graphs as hypergraphs

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).



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.

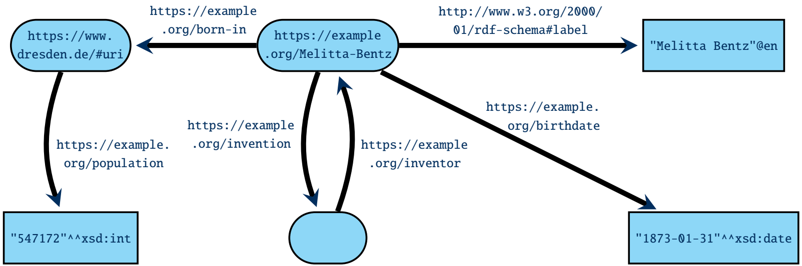
N-Triples

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 2.9: 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 2.10: 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 ;
                 <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 2.11: 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>  [ <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 2.12: 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 .
```

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 2.13: 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)

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

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

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

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?