

# KNOWLEDGE GRAPHS

## Lecture 2: Encoding Graphs with RDF

Markus Krötzsch

Knowledge-Based Systems

TU Dresden, 23th Oct 2018

# 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<sup>1</sup> 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, ...)



---

<sup>1</sup>World Wide Web Consortium

# The Resource Description Framework

RDF is a W3C<sup>1</sup> 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 lecture: focus on graph representation features of RDF 1.1



---

<sup>1</sup>World Wide Web Consortium

# The Resource Description Framework



RDF is a W3C<sup>1</sup> 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 lecture: 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

---

<sup>1</sup>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)

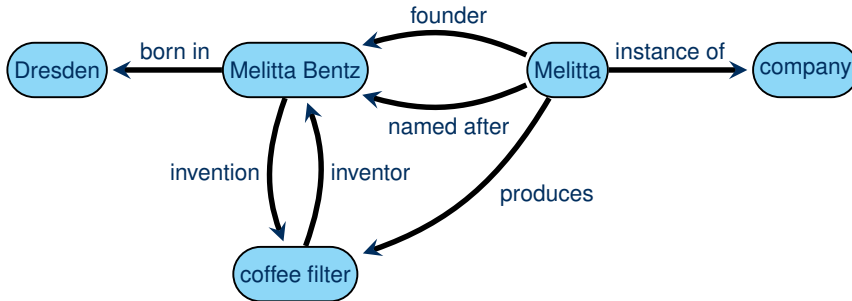


# Graphs in RDF

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:**



# Identifiers in RDF

How should we refer to vertices? What kind of labels are allowed?

# Identifiers in RDF

How should we refer to vertices? What kind of labels are allowed?

**Definition 2.1:** A **Uniform Resource Identifier** (URI) is a sequence (string) of a subset of ASCII characters as defined in RFC 3986 (link). Every (absolute) URI consists of a string that defines a **scheme**, followed by a colon (:) and another sequence of characters specifying an **authority**, **path**, **query**, and **fragment**, where all parts other than the path are optional.

A **International Resource Identifier** (IRI) is a generalised form of URI that allows for an expanded range of Unicode glyphs in part of its syntax.

# Identifiers in RDF

How should we refer to vertices? What kind of labels are allowed?

**Definition 2.1:** A **Uniform Resource Identifier** (URI) is a sequence (string) of a subset of ASCII characters as defined in RFC 3986 (link). Every (absolute) URI consists of a string that defines a **scheme**, followed by a colon (:) and another sequence of characters specifying an **authority**, **path**, **query**, and **fragment**, where all parts other than the path are optional.

A **International Resource Identifier** (IRI) is a generalised form of URI that allows for an expanded range of Unicode glyphs in part of its syntax.

**Example 2.2:** URLs are a well-known form of URI, for example:

`https://example.org/some/page?get=something&lang=en#results`

scheme                      authority                      path                      query                      fragment

# Identifiers in RDF

How should we refer to vertices? What kind of labels are allowed?

**Definition 2.1:** A **Uniform Resource Identifier** (URI) is a sequence (string) of a subset of ASCII characters as defined in RFC 3986 (link). Every (absolute) URI consists of a string that defines a **scheme**, followed by a colon (:) and another sequence of characters specifying an **authority**, **path**, **query**, and **fragment**, where all parts other than the path are optional.

A **International Resource Identifier** (IRI) is a generalised form of URI that allows for an expanded range of Unicode glyphs in part of its syntax.

**Example 2.2:** URLs are a well-known form of URI, for example:

`https://example.org/some/page?get=something&lang=en#results`

          scheme            authority            path            query            fragment

**Convention:** We ignore the differences between URIs and IRIs in this course.

(The lecturer will often say "URI" when he should say "IRI." Please ignore.)

# URIs vs. URLs

The widely used term **Uniform Resource Locator** (URL) is an informal way to refer to URIs that specify the location of a digital document. Not all URIs support this.

**Example 2.3:** Many URI schemes have been defined. Examples include

- `http`, `https`, `ftp` for transferring data using various protocols
- `mailto` for emails (=path)
- `irc` for specifying IRC channels
- `file` for referring to locations on some file system
- `urn` for naming resources without defining a protocol or resolution mechanism; used, e.g., for ISBNs
- ...

Registered schemes usually provide some additional syntax requirements, access protocols, and resolution mechanisms, and they often relate to a registration procedure based on some authority.

# IRIs in RDF

RDF uses IRIs in two ways:

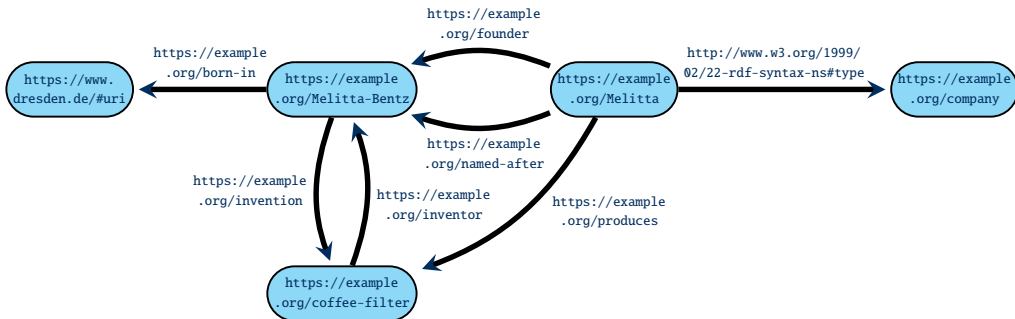
- IRIs define resources that appear as vertices in the graph
- IRIs are used as property labels

# IRIs in RDF

RDF uses IRIs in two ways:

- IRIs define resources that appear as vertices in the graph
- IRIs are used as property labels

**Example of such a graph:**



**Note:** It is not always obvious what an IRI is supposed to refer to, and many IRIs may refer to the same thing – we cannot assume that all RDF data in the world is integrated.



# 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  
    ↪ 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

## Guidelines for creating new IRIs:

1. Check if you could re-use an existing IRI ↪ avoid duplication if feasible
2. Use http(s) IRIs ↪ useful protocols, registries, resolution mechanisms
3. Create new IRIs based on domains that you own ↪ clear ownership; no danger of clashing with other peoples' IRIs
4. Don't use URLs of existing web pages, unless you want to store data about pages ↪ avoid confusion between pages and more abstract resources
5. Make your IRIs return some useful content via http(s) ↪ 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 2.4:** 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 2.4:** 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 2.5:** 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  $\leadsto$  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  $\leadsto$  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  $\leadsto$  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 2.6:** Using a hypothetical scheme “integer,” the IRIs `integer:42` and `integer:+42` would be different, but intuitively they should represent the same number.

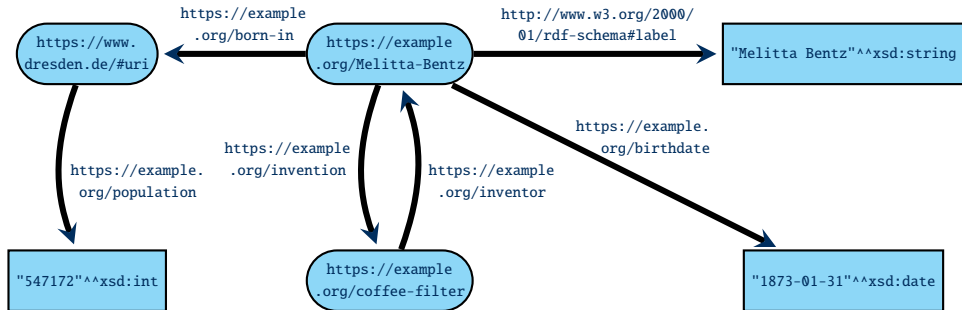
# Encoding data values

- 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 2.7:** 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<sup>a</sup> 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.

---

<sup>a</sup>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 2.7:** 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<sup>a</sup> 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.

---

<sup>a</sup>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 2.8:** The W3C standard XML Schema defines the datatype **integer**, identified by the IRI `http://www.w3.org/2001/XMLSchema#integer`, 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):

<b>Name</b>	<b>Value space</b>	<b>Lexical space</b>
<code>string</code>	XML-defined subset of Unicode strings	Value space (mapping is identity)
<code>boolean</code>	The set {true, false}	The set {true, false, 1, 0}
<code>decimal</code>	Arbitrary-precision decimal numbers	Sequences of decimal digits, optionally signed and with one .
<code>integer</code>	Arbitrary integer numbers	Like <code>decimal</code> but without .
<code>int</code>	{-2147483648, ..., 2147483647} (32bit)	Integer strings that map to values within this range

Similar types exist for `long` (64bit), `short` (16bit), `byte` (8bit), unsigned versions, etc.

<code>double</code>	IEEE double-precision 64-bit floating point numbers	decimal strings, exp notation numbers, special values (e.g., NaN)
<code>float</code>	single-precision 32-bit floating point numbers	same as <code>double</code>

## Important datatypes in RDF (2)

There are also many types for dates and times:

<b>Name</b>	<b>Value space</b>	<b>Lexical space</b>
<code>dateTime</code>	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., <code>2018-10-23T11:45:30+01:00</code> ; one can write Z for offset <code>00:00</code>

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>:

[//www.w3.org/TR/2014/REC-rdf11-concepts-20140225/#section-Datatypes](https://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/#section-Datatypes)

# RDF datatype literals

**Definition 2.9:** 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 2.10:** 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 2.11:** 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**.

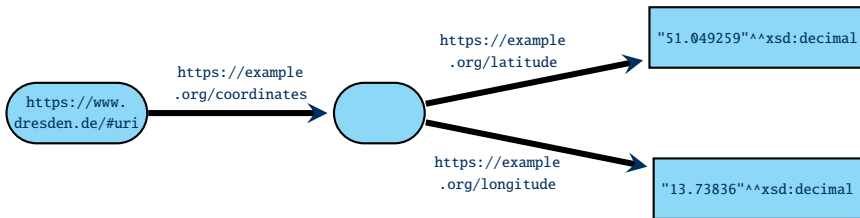
- Intuitively, bnodes are placeholder for some specific (but unspecified) node
- Their use makes the claim: “there is something at this position”
- Similar to existentially quantified variables in logic

# 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 2.12:** 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 2.12:** 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
  - ~> multiple ways of writing the same value lead to multiple graphs
  - ~> ill-formed literals are allowed in graphs

# Serialisations

# RDF Serialisations

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

# RDF Serialisations

What we outlined so far is the **abstract syntax** 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.

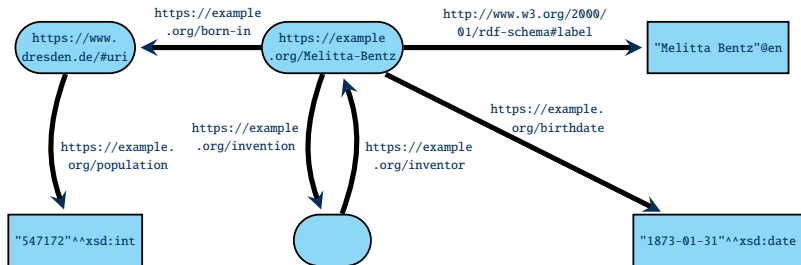
# N-Triples

N-Triples is almost the simplest format conceivable:

- Each line encodes one triple:
  - IRIs are written in pointy brackets, e.g., `<https://www.dresdenrespekt.de/>`
  - Literals are written as usual with a given type IRI, e.g., `"2018-10-21"^^<http://www.w3.org/2001/XMLSchema#date>` or with a language-tag, e.g., `"Wir sind mehr"@de`
  - 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)

# 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

## **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?