TECHNISCHE UNIVERSITÄT DRESDEN

KNOWLEDGE GRAPHS

Lecture 3: Modelling in RDF/Introduction to SPARQL

Markus Krötzsch

Knowledge-Based Systems

TU Dresden, 30th Oct 2018

Review: RDF Graphs

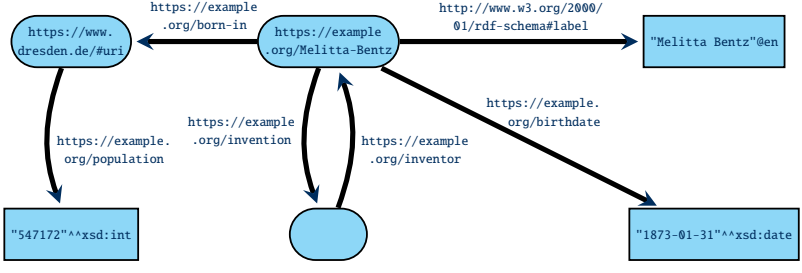
The W3C Resource Description Framework considers three types of RDF terms:

- IRIs, representing a resource using a global identifier
- Blank nodes, representing an unspecified resource without giving any identifier
- Literals that represent values of some datatype
  - either typed literals, such as
    `"2018-10-21"^^<http://www.w3.org/2001/XMLSchema#date>`
  - or language-tagged strings, such as `"Wir sind mehr"@de`

RDF graphs are sets of triples consisting of

- a subject, which can be an IRI or bnode,
- a predicate, which can be an IRI,
- an object, which might be an IRI, bnode, or literal

Review: N-Triples



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> .
```

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 3.1: 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 3.2: 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 3.3: 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 3.4: 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 3.5: 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 3.6: 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 3.6: It is common to assign labels to properties, and many applications display these labels to show triples that use these properties as their predicate.

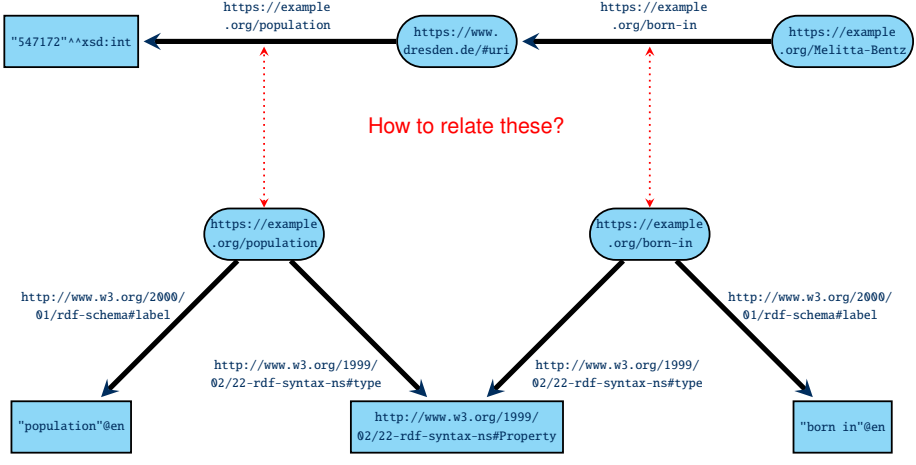
RDF also allows us to declare a resource as a property, using the following triple (in Turtle, using standard abbreviations):

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!

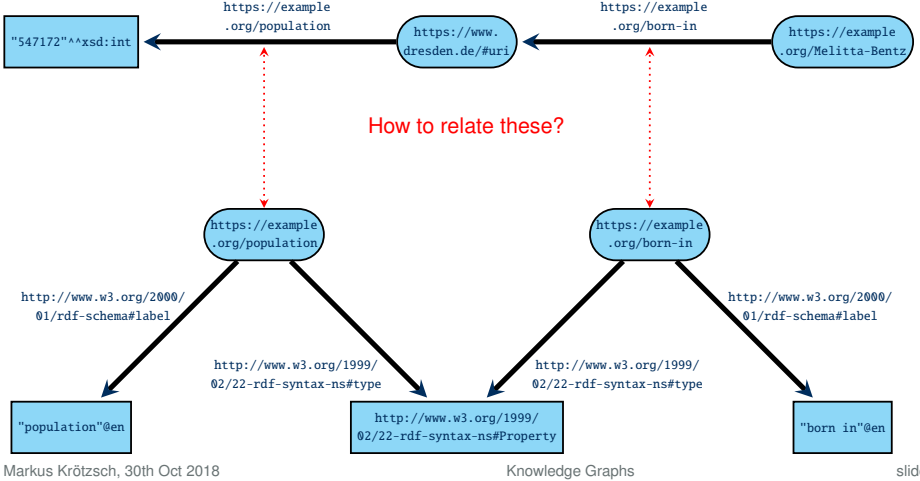


RDF graphs as hypergraphs

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).

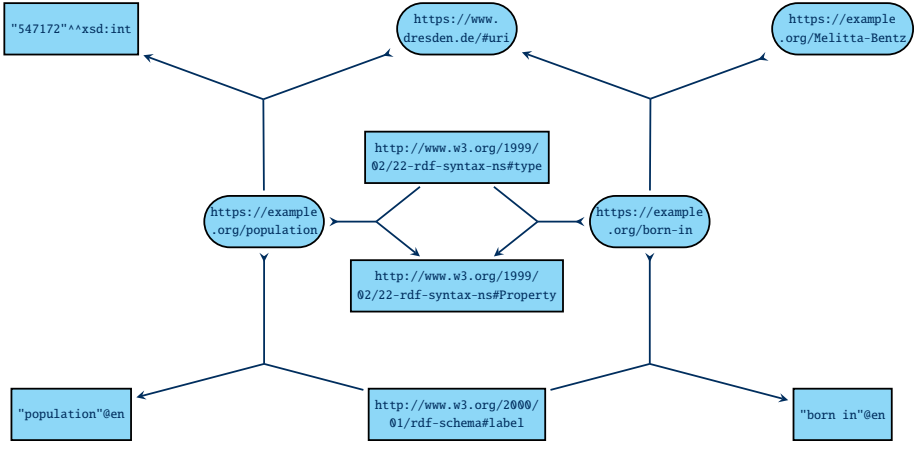
RDF graphs as hypergraphs

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 3.7: 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 3.7: 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 3.8: 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

Representing data in RDF

Goal: Let's manage more of our data in RDF (for interoperability, data integration, uniform tooling, use RDF software, . . .) – Is this possible?

Representing data in RDF

Goal: Let's manage more of our data in RDF (for interoperability, data integration, uniform tooling, use RDF software, ...) – Is this possible?

Any information can be encoded in RDF, in particular:

- Relational data, including CSV files
- XML documents
- Other kinds of graphs
- Other kinds of database formats
- (Abstractions of) program code
- ...

But not all data should be RDF (e.g., media files or executable binaries)

Representing data in RDF

Goal: Let's manage more of our data in RDF (for interoperability, data integration, uniform tooling, use RDF software, ...) – Is this possible?

Any information can be encoded in RDF, in particular:

- Relational data, including CSV files
- XML documents
- Other kinds of graphs
- Other kinds of database formats
- (Abstractions of) program code
- ...

But not all data should be RDF (e.g., media files or executable binaries)

Main steps that are needed:

- Define IRIs for relevant resources
- Define RDF graph structure from given data structure

Encoding data as graphs

Challenges when translating data to RDF:

Encoding data as graphs

Challenges when translating data to RDF:

- RDF has only triples – what to do with relations of higher arity (e.g., from a relational database)?
  $\rightsquigarrow$ covered, e.g., by RDB2RDF standard; fairly common

Encoding data as graphs

Challenges when translating data to RDF:

- RDF has only triples – what to do with relations of higher arity (e.g., from a relational database)?
  ⇝ covered, e.g., by RDB2RDF standard; fairly common
- RDF graphs are (unordered) sets of triples – how to represent ordered lists (including, e.g., the ordered children of a node in XML)?
  ⇝ several possible ways around; no standard solution

Representing n-ary relations

Consider the following relational table:

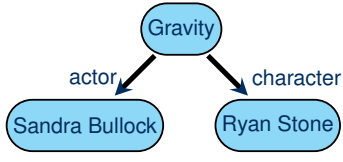
Film	Actor	Character
Arrival	Amy Adams	Louise Banks
Arrival	Jeremy Renner	Ian Donnelly
Gravity	Sandra Bullock	Ryan Stone

Representing n-ary relations

Consider the following relational table:

Film	Actor	Character
Arrival	Amy Adams	Louise Banks
Arrival	Jeremy Renner	Ian Donnelly
Gravity	Sandra Bullock	Ryan Stone

Possible encoding:

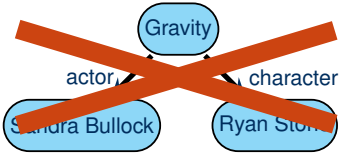


Representing n-ary relations

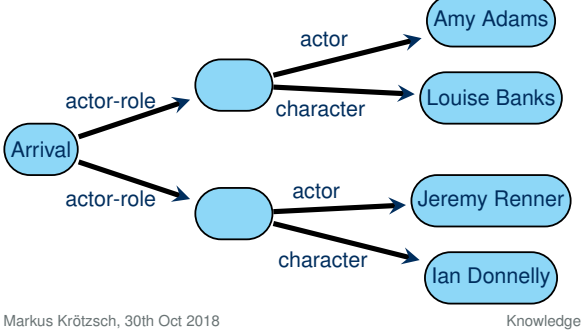
Consider the following relational table:

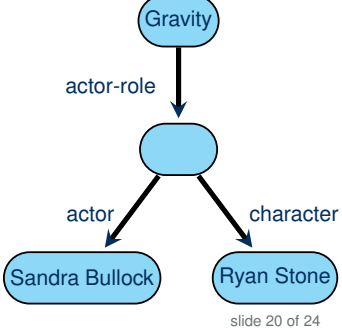
Film	Actor	Character
Arrival	Amy Adams	Louise Banks
Arrival	Jeremy Renner	Ian Donnelly
Gravity	Sandra Bullock	Ryan Stone

Insufficient encoding:



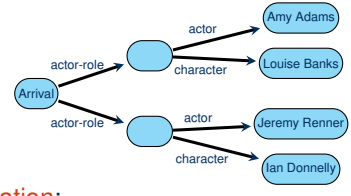
Possible encoding:

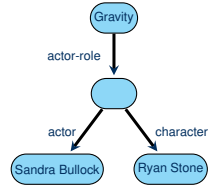




Reification

Film	Actor	Character
Arrival	Amy Adams	Louise Banks
Arrival	Jeremy Renner	Ian Donnelly
Gravity	Sandra Bullock	Ryan Stone





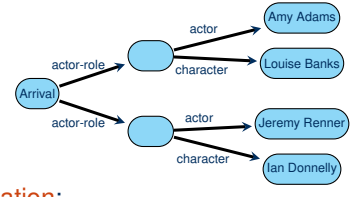
This type of encoding is known as reification:

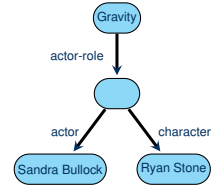
- Introduce auxiliary nodes to represent relationships
- Connect related objects and values with the auxiliary node

⇝ Can be realised in many specific ways (bnodes or not, direction of relations)

Reification

Film	Actor	Character
Arrival	Amy Adams	Louise Banks
Arrival	Jeremy Renner	Ian Donnelly
Gravity	Sandra Bullock	Ryan Stone





This type of encoding is known as reification:

- Introduce auxiliary nodes to represent relationships
- Connect related objects and values with the auxiliary node

$\rightsquigarrow$ Can be realised in many specific ways (bnodes or not, direction of relations)

RDF Reification: RDF even has a dedicated vocabulary to reify RDF triples:

- Reified triples x marked as x `rdf:type rdf:Statement .`
- Triple-pieces connected by properties `rdf:subject`, `rdf:predicate`, and `rdf:object`

However, this is not frequently used (reasons: specific to triples, inflexible, incompatible with OWL DL).

RDB2RDF

The W3C's "RDB2RDF" working group has published a direct mapping from relational databases to RDF graphs (`https://www.w3.org/TR/rdb-direct-mapping/`):

- Basic approach is to reify triples as shown
- Special guidelines for creating IRIs for auxiliary nodes (no bnodes!), taking table names into account
- Mapping from SQL table entries to RDF literals
- Additional properties and triples to capture foreign-key relations

More specific mappings can be defined using the R2RML RDB to RDF Mapping Language (`https://www.w3.org/TR/r2rml/`).

Representing order

Problem: RDF can easily represent sets, but not lists

Representing order

Problem: RDF can easily represent sets, but not lists

Possible solutions:

- Represent lists as linked lists

  Supported by “RDF Collections” (`rdf:first`, `rdf:rest`, `rdf:nil`)
  Pro: Faithful, elegant graph representation; easy insert/delete
  Con: Inefficient access (many joins)

Representing order

Problem: RDF can easily represent sets, but not lists

Possible solutions:

- Represent lists as linked lists

  Supported by "RDF Collections" (`rdf:first`, `rdf:rest`, `rdf:nil`)
  Pro: Faithful, elegant graph representation; easy insert/delete
  Con: Inefficient access (many joins)

- Use order-indicating properties (one property per position)

  Supported by "RDF Containers" (`rdf:_1`, `rdf:_2`, ..., `rdfs:ContainerMembershipProperty`)
  Pro: direct access; length only detectable by absence of triples
  Con: list structure requires external knowledge; harder insert/delete

Representing order

Problem: RDF can easily represent sets, but not lists

Possible solutions:

- Represent lists as linked lists

  Supported by "RDF Collections" (`rdf:first`, `rdf:rest`, `rdf:nil`)
  Pro: Faithful, elegant graph representation; easy insert/delete
  Con: Inefficient access (many joins)

- Use order-indicating properties (one property per position)

  Supported by "RDF Containers" (`rdf:_1`, `rdf:_2`, ..., `rdfs:ContainerMembershipProperty`)
  Pro: direct access; length only detectable by absence of triples
  Con: list structure requires external knowledge; harder insert/delete

- Give up order and represent lists as sets

  Easy to do in RDF
  Pro: simple; may suffice for many use cases
  Con: no order

- Other encodings are conceivable ...

Representing order

Problem: RDF can easily represent sets, but not lists

Possible solutions:

- Represent lists as linked lists

  Supported by "RDF Collections" (`rdf:first`, `rdf:rest`, `rdf:nil`)
  Pro: Faithful, elegant graph representation; easy insert/delete
  Con: Inefficient access (many joins)

- Use order-indicating properties (one property per position)

  Supported by "RDF Containers" (`rdf:_1`, `rdf:_2`, ..., `rdfs:ContainerMembershipProperty`)
  Pro: direct access; length only detectable by absence of triples
  Con: list structure requires external knowledge; harder insert/delete

- Give up order and represent lists as sets

  Easy to do in RDF
  Pro: simple; may suffice for many use cases
  Con: no order

- Other encodings are conceivable ...

Possible compromise: Combine several encodings within one graph
(increased data volume – easier to read – harder to write)

Summary

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

RDF can express n-ary relations and (with some effort) lists

What's next?

- How to query RDF graphs with SPARQL
- Wikidata as a working example to try out our knowledge
- More powerful SPARQL features