Introduction to the Semantic Web

Lecture 2: The Resource Description Framework (RDF)

Moritz Blum

Semantic Computing Group, Bielefeld University, Germany

About myself

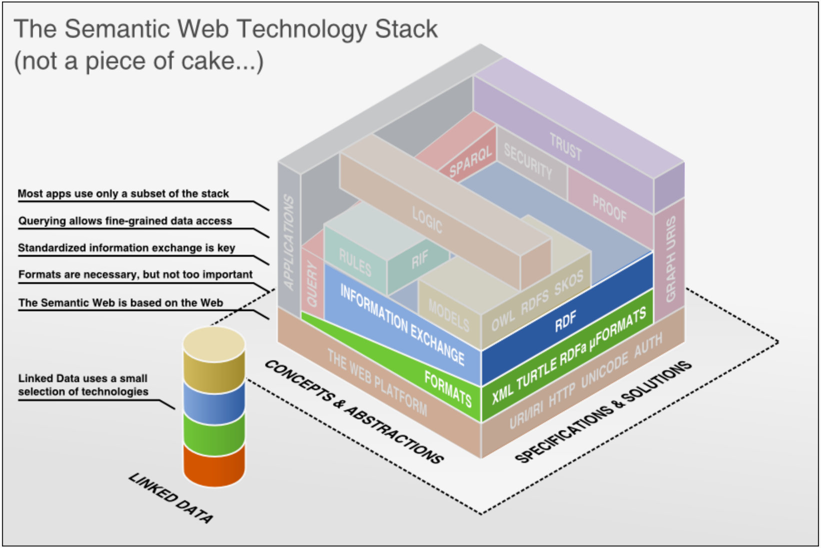
Figure 1: Me

- Ph.D. student at @ the Semantic Computing Group
- research on KGs & NLP: joint embedding space, KGC, RE, GNNs

Table of contents

Introduction

Resource Description Framework (RDF)



Semantic Web technology stack

Resource Description Framework (RDF)

The Resource Description Framework (RDF) is a data model that allows us to **describe resources** on the Web in **a structured manner**.

Resource Description Framework (RDF)

RDF …

- **Resource** can be everything (must be uniquely identified and referenceable via a URI)
- **Description** of resources via representing properties and relationships among resources as a graph
- **Framework** combination of web based protocols (URIs, HTTP, XML, Turtle, JSON)

URIs & Content Negotiation

Resources

As the name reveals, RDF is mainly concerned with the description of **resources**. But what is a resource? A resource is anything that we might want to talk about in the context of the Web. Examples are:

- The lecturer, Moritz Blum, is a resource.
- Bielefeld University is a resource.
- Anyone of you is a resource.
- The audience of this lecture as a collection is a resource.
- Albert Einstein’s general theory of relativity is a resource.
- The letter ”A” is a resource
- ”Nothing” is a resource.

Uniform Resource Identifiers

In RDF, we introduce URIs to identify and talk about resources.

- *https://www.wikidata.org/wiki/Q24382*
- *http://www.uni-bielefeld.de*
- *https://moritzblum.github.io*
- *https://moritzblum.github.io/moritz_foaf.rdf#me*

URIs are like symbols: they are a substitute for an instance in the real world.

URL vs. URI

URL: identify what exists on the web
(*https://moritzblum.github.io/*)
vs.
URI: identify on the web what exists
(*https://moritzblum.github.io/moritz_foaf.rdf#me*)

Attention: My web page is not the resource, it just contains data about a resource.

URI Design Principles

A URI defines a simple and extensible schema for worldwide unique identification of abstract or physical resources with a clear identity.

1. Use URIs as names for things.
2. Avoid overlap: use HTTP URIs so that people can look up those names.
3. Include links to other URIs. so that they can discover more things.
4. When someone looks up a URI, provide useful documentation at the same place. $\rightarrow$ Content Negotiation

Content Negotiation

What do you get if you enter URIs in your Browser?

URI: *`https://www.wikidata.org/entity/Q24382`*

Content Negotiation

Depending on the request, we get the needed data.

We use curl that does a HTTP request to a web server.

Try out via command line:

- curl -L -H ”Accept: text/text” https://www.wikidata.org/entity/Q24382
- curl -L -H ”Accept: text/plain” https://www.wikidata.org/entity/Q24382

Triples

The RDF data model essentially builds on triples of the form.

(subject, predicate, object)

Triples

where

- **subject** has to be a URI representing a resource
- **predicate** has to be a URI representing a resource (yes, predicates are also resources!)
- **object** can be either a URI representing a resource or a so called ”literal”

Example triples

```
(https://moritzblum.github.io/resource/Moritz,
    http://example.org/worksAt,
        http://www.uni-bielefeld.de/resource/uni)

(https://moritzblum.github.io/resource/Moritz,
    http://example.org/teaches,
        http://www.uni-bielefeld.de/resource/studium/lehre/sw_ss2023)

(https://moritzblum.github.io/resource/Moritz,
    http://www.w3.org/2000/01/rdf-schema#label,
        "Moritz Blum"@en)

(https://moritzblum.github.io/resource/Moritz,
    http://xmlns.com/foaf/0.1/room,
        "CITEC 2-310")
```

Serializing RDF

Serializing RDF

The goal of RDF is to allow us to exchange information on the Web. Therefore, we need a way to publish RDF information, we need a syntax for RDF documents. There have been defined a number of serializations of RDF:

- Turtle family of RDF languages (N-Triples, Turtle, TriG and N-Quads)
- RDF/XML (XML syntax for RDF)
- JSON-LD (JSON-based RDF syntax)
- RDFa (for HTML and XML embedding)

N-Triples

```
<https://moritzblum.github.io/resource/Moritz>
    <http://example.org/worksAt>
        <http://www.uni-bielefeld.de/resource/uni>.

<https://moritzblum.github.io/resource/Moritz>
    <http://example.org/teaches>
        <http://www.uni-bielefeld.de/resource/studium/lehre/sw_ss2023>.

<https://moritzblum.github.io/resource/Moritz>
    <http://example.org/teaches>
        <http://www.uni-bielefeld.de/resource/studium/lehre/sw_ss2024>.

<https://moritzblum.github.io/resource/Moritz>
    <http://example.org/memberOf>
        <http://www.sc.cit-ec.uni-bielefeld.de/resource/group>.
```

Turtle

```
@prefix blum: <https://moritzblum.github.io/resource/> .
@prefix unibi: <http://www.uni-bielefeld.de/resource/> .
@prefix unibi_teach: <http://www.uni-bielefeld.de/studium/lehre/> .
@prefix sc: <http://www.sc.cit-ec.uni-bielefeld.de/resource/> .
@prefix ex: <http://example.org/> .

blum:Moritz ex:worksAt unibi:uni ;
             ex:teaches unibi_teach:sw_ss2023 ,
                        unibi_teach:sw_ss2024 ;
             ex:memberOf sc:group.
```

RDF/XML Syntax

```
<?xml version="1.0" encoding="utf-8"?>

<rdf:RDF
   xmlns:rdf="http://www.w3.org/199/02/22-rdf-syntax-ns#"
   xmlns:ex="http://example.org#"
   >

<rdf:Description rdf:about="https://moritzblum.github.io/resource/Moritz">
   <ex:worksAt>
       <rdf:Description rdf:about="http://www.uni-bielefeld.de/resource/uni">
       </rdf:Description>
   </ex:worksAt>
</rdf:Description>

</rdf:RDF>
```

RDF datatypes

So far, we have only looked at untyped literals. However, in RDF, literals can also be typed using XML datatypes, e.g.

```
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

<http://www.w3.org/TR/rdf-primer>
    <http://example.org/title> "RDF Primer"^^xsd:string;
    <http://example.org/publicationDate> "2004-02-10"^^xsd:date .
```

RDF datatypes

A list of the RDF-compatible XSD types, with short descriptions"

	Datatype	Value space (informative)
Core types	xsd:string	Character strings (but not all Unicode character strings)
	xsd:boolean	true, false
	xsd:decimal	Arbitrary-precision decimal numbers
	xsd:integer	Arbitrary-size integer numbers
IEEE floating-point numbers	xsd:double	64-bit floating point numbers incl. ±Inf, ±0, NaN
	xsd:float	32-bit floating point numbers incl. ±Inf, ±0, NaN
Time and date	xsd:date	Dates (yyyy-mm-dd) with or without timezone
	xsd:time	Times (hh:mm:ss.sss…) with or without timezone
	xsd:dateTime	Date and time with or without timezone
	xsd:dateTimeStamp	Date and time with required timezone
Recurring and partial dates	xsd:gYear	Gregorian calendar year
	xsd:gMonth	Gregorian calendar month
	xsd:gDay	Gregorian calendar day of the month
	xsd:gYearMonth	Gregorian calendar year and month
	xsd:gMonthDay	Gregorian calendar month and day
	xsd:duration	Duration of time
	xsd:yearMonthDuration	Duration of time (months and years only)
	xsd:dayTimeDuration	Duration of time (days, hours, minutes, seconds only)

XML schema datatypes

RDF language tags

For character strings, there are also language tags available. They denote the (natural) language of the text.

```
@prefix xsd: <http://www.w3.org/2001/XMLSchema> .
<http://bielefeld-university.de>
<http://example/label> "Bielefeld University"@en;
<http://example/label> "Universität Bielefeld"@de;
```

Language tags automatically imply the data type xsd:string.

RDF language tags

In RDF, untyped, typed and untyped literals with language specification are not the same:

```
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>

<http://crcpress.com/uri> <http://example.org/name> "CRC Press" ,
                                                    "CRC Press"@en ,
                                                    "CRC Press"^^xsd:string .
```

Application: Cooking with RDF

Cooking with RDF (I)

In order to prepare Mango Chutney, we need 450g Green Mango, one teaspoon of Cayenne pepper, ...

Cooking with RDF (II)

A first try with Turtle Syntax:

```
@prefix ex: <http://www.cooking_with_rdf.de/> .

ex:Chutney ex:hasIngredient "450g Green Mango" .
ex:Chutney ex:hasIngredient "1 tbsp Cayenne Pepper" .
```

Cooking with RDF (III)

A second try:

```
@prefix ex: <http://www.cooking_with_rdf.de/> .

ex:Chutney ex:hasIngredient     "Green Mango";
           ex:hasQuantity       "450g";
           ex:hasIngredient     "Cayenne Pepper";
           ex:hasQuantity       "1 tbsp" .
```

Cooking with RDF (IV)

```
@prefix ex: <http://www.cooking_with_rdf.de/> .

ex:Chutney  ex:hasIngredient    ex:Ingredient_1 ;
            ex:hasIngredient    ex:Ingredient_2 .

ex:Ingredient_1 ex:ingredient   ex:GreenMango .
ex:Ingredient_1 ex:hasQuantity  "450g" .
ex:Ingredient_2 ex:ingredient   ex:CayennePepper .
ex:Ingredient_2 ex:hasQuantity  "1 tbsp" .
```

Cooking with RDF (V)

Final version:

```
@prefix ex: <http://www.cooking_with_rdf.de/> .

ex:Chutney  ex:hasIngredient ex:Ingredient_1;
            ex:hasIngredient ex:Ingredient_2 .

ex:Ingredient_1 ex:ingredient   ex:GreenMango;
                ex:hasQuantity  "450";
                ex:hasUnit      "g" .
ex:Ingredient_2 ex:ingredient   ex:CayennePepper;
                ex:hasQuantity  "1";
                ex:hasUnit      "tbsp" .
```

Blank Nodes Reification

Blank Nodes (1)

You want to make a statement about something you can not identify directly. $\rightarrow$ Blank Nodes can be used to claim the existence of things.

They allow us to introduce resources without assigning them a URI.

Blank Nodes are not unique as they are not a uri. Therefore, they can not be referenced from outside (e.g. the web).

Blank Nodes in Turtle

```
@prefix ex: <http://example.org/> .
ex:Chutney   ex:hasIngredient   _:id1 .
_:id1        ex:ingredient      ex:greenMango ;
             ex:amount          "1lb." .
```

Blank Nodes in Turtle (Method 2)

```
@prefix ex: <http://example.org/> .
ex:Chutney   ex:hasIngredient [
                   ex:ingredient    ex:greenMango ;
                   ex:amount        "1lb."
             ] .
```

RDF reification

How to make statements about statements?

We might for example say that Sherlock Holmes said that Jack the Ripper killed Mary:

Part 1:

```
ex:Jack_The_Ripper ex:killed ex:Mary .
```

Part 2:

```
ex:Sherlock_Holmes ex:says ??? .
```

How to connect these two parts?

RDF reification

There is a mechanism in RDF which allows us to make assertions about assertions.

```
@prefix ex: <http://example.org/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
ex:Sherlock_Holmes   ex:says        ex:Statement .
ex:Statement         rdf:subject    ex:Jack_The_Ripper .
ex:Statement         rdf:predicate  ex:killed .
ex:Statement         rdf:object     ex:Mary .
```

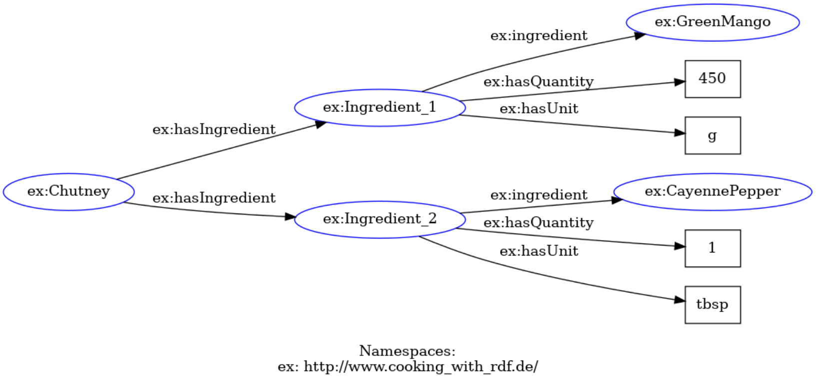
RDF as a graph

RDF as a graph

Formally, an RDF graph G is a set of triples $(s, p, o) \in (\mathcal{U} \cup \mathcal{B}) \times \mathcal{U} \times (\mathcal{U} \cup \mathcal{B} \cup \mathcal{L})$, where $\mathcal{U}$, $\mathcal{B}$ and $\mathcal{L}$ are pairwise disjoint sets of URIs, blank nodes and literal values, respectively.

RDF as a graph

All the triples together represent a **directed edge-labeled node-labeld multigraph**. Nodes represent resources or literals, and properties are represented as directed edges.

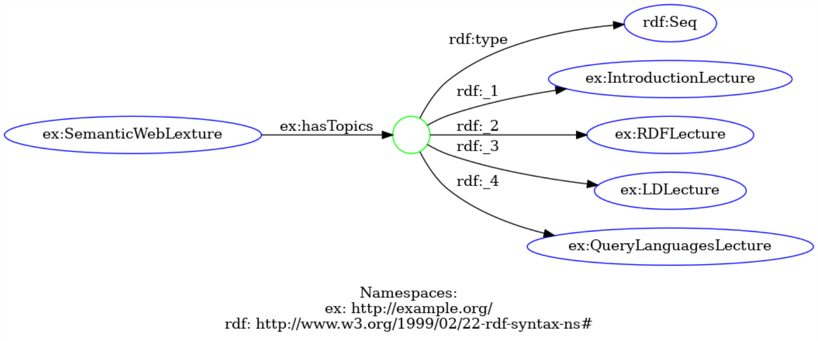


RDF data structures: RDF Lists

RDF Lists

How do we represent Lists or Sets in RDF?
Two options: $\rightarrow$ Container, Collection

RDF Lists: Collection

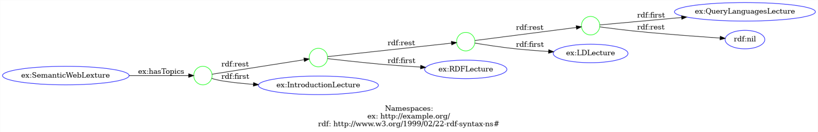


RDF Lists: Container

```
@prefix ex: <http://example.org/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

ex:SemanticWebLexture ex:hasTopics [
    a rdf:Seq;
    rdf:_1 ex:IntroductionLecture;
    rdf:_2 ex:RDFLecture;
    rdf:_3 ex:LDLecture;
    rdf:_4 ex:QueryLanguagesLecture
] .
```

RDF Lists: Collection



RDF Lists: Collection

```
@prefix ex: <http://example.org/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

ex:SemanticWebLexture ex:hasTopics [
    rdf:first ex:IntroductionLecture; rdf:rest [
        rdf:first ex:RDFLecture; rdf:rest[
            rdf:first ex:LDLecture; rdf:rest [
                rdf:first ex:QueryLanguagesLecture;
                rdf:rest rdf:nil
            ]
        ]
    ]
]
```

RDF Lists: Collection (simple)

```
@prefix ex: <http://example.org/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

ex:SemanticWebLexture ex:hasTopics (
    ex:IntroductionLecture ex:RDFLecture ex:LDLecture ex:QueryLanguagesLecture
) .
```

RDF Schema

All we need is... RDF?

RDF provides us with a language to define triples, thus allowing us to mainly state factual knowledge.

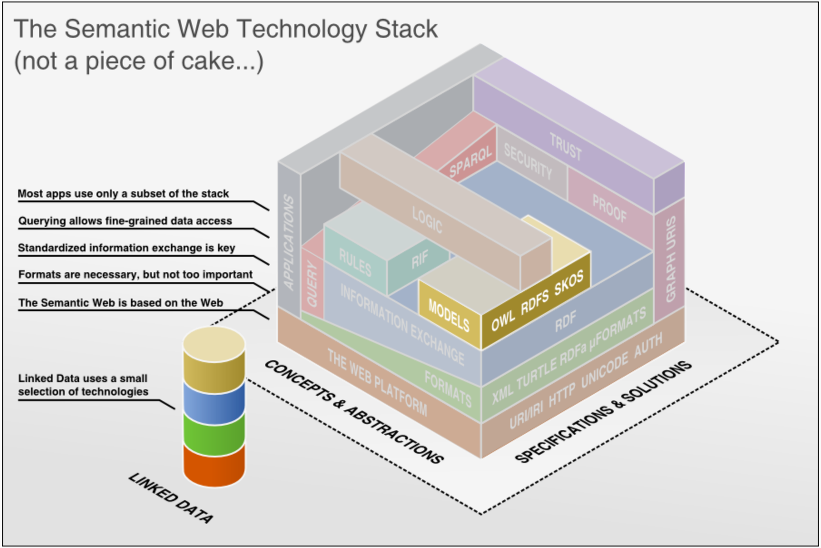
Example:

- *Flipper is a dolphin.*
  `ex:Flipper rdf:type ex:Dolphin .`
- *Sandy likes Flipper.*
  `ex:Sandy ex:like ex:Flipper .`

All we need is... RDF?

But RDF does **not** provide us with a formalism to describe more precisely the meaning of and relation between different vocabulary elements, e. g. to make statements about classes and properties.

Resource Description Framework (RDF)



Semantic Web technology stack

Example

```
ex:Flipper rdf:type ex:Dolphin .
ex:Flipper rdf:type ex:Mammal .
ex:Flipper rdf:type ex:Animal .
```

But how to express that all dolphins are mammals and all mammals are animals?

Example

```
ex:Mango_Chutney ex:goes_well_with ex:Curry .
```

But how to express that *`ex:goes_well_with`* relates food with food?

RDF Schema

- RDF Schema is part of the set of W3C Recommendations for RDF.
- It supports the specification of schema or terminological knowledge by providing a vocabulary for this purpose (a meta-vocabulary so to speak).
- It uses an own namespace, i. e. the RDFS-namespace, i. e. *http://www.w3.org/2000/01/rdf-schema#*

All those namespaces!

prefix.cc

(For example: prefix.cc/rdfs, prefix.cc/foaf, prefix.cc/xsd)

Relation between RDFS and RDF

- RDFS essentially introduces additional vocabulary on top of RDF using the RDFS namespace.
- Every RDFS document is a legal RDF document.

Typing

```
ex:Flipper rdf:type ex:Dolphin .
```

So far there is nothing to distinguish typed from instances of those types. They are all resources.

rdf:type corresponds to the standard set operator $\in$.

Classes in RDFS

In RDFS we can define types (classes) explicitly:

```
ex:Dolphin rdf:type rdfs:Class .
```

Here, ***rdfs:Class*** is the class of all classes (and again a resource, of course).

Motivation for subclasses

Imagine we want to query for all mammals. Would we also get all dolphins?

Motivation for subclasses

Imagine we want to query for all mammals. Would we also get all dolphins?

Not necessarily, unless they are typed also as a mammal. We would therefore need to introduce one triple for each type:

```
ex:Flipper rdf:type ex:Dolphin .
ex:Flipper rdf:type ex:Mammal .

ex:Fungie rdf:type ex:Dolphin .
ex:Fungie rdf:type ex:Mammal .

ex:Finchen rdf:type ex:Dolphin .
ex:Finchen rdf:type ex:Mammal .
```

Subclasses in RDFS

The solution in RDFS is to define the class *`Dolphin`* as a subclass of the class *`Mammal`*.

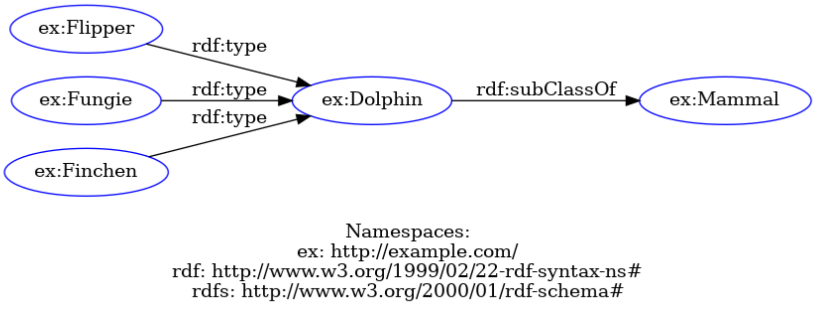
```
ex:Dolphin rdf:subClassOf ex:Mammal .

ex:Flipper rdf:type ex:Dolphin .
ex:Fungie rdf:type ex:Dolphin .
ex:Finchen rdf:type ex:Dolphin .
```

- The interpretation here is extensional, i.e. any member of the class *`Dolphin`* is also a member of the class *`Mammal`*.
- The subclass construct can also be used to declare two classes as being equivalent. (How?)

Subclasses in RDFS

The solution in RDFS is to define the class ***Dolphin*** as a subclass of the class ***Mammal***.



Semantic Web technology stack

rdfs:subClassOf corresponds to the standard set operator $\subseteq$

Predefined classes in RDFS

- *rdfs:Resource*
- *rdfs:Class*
- *rdfs:Property*
- *rdfs:Literal*
- *rdfs:Datatype*

Example

```
ex:Flipper rdf:type ex:Dolphin .

ex:Flipper ex:averageDepth ”47”^^ex:meter .
ex:Flipper ex:maximumDepth ”321”^^ex:meter .

ex:Flipper        rdf:type rdfs:Resource .
ex:Dolphin        rdf:type rdfs:Class .
ex:averageDepth   rdf:type rdfs:Property .
ex:maximumDepth   rdf:type rdfs:Property .
ex:meter          rdf:type rdfs:Datatype .
```

Subproperties

```
ex:averageDepth  rdfs:subPropertyOf ex:depth  .
ex:maximumDepth  rdfs:subPropertyOf ex:depth  .

ex:hasSon        rdfs:subPropertyOf ex:hasChild  .
ex:hasDaughter   rdfs:subPropertyOf ex:hasChild  .
```

Domain and range of properties

We can specify the types that we allow/expect at subject and object position of a property by defining its domain and range:

```
ex:breathHold  rdfs:domain  ex:Mammal  .
ex:breathHold  rdfs:range   xsd:duration  .

ex:goes_well_with  rdfs:domain  ex:Food  .
ex:goes_well_with  rdfs:range   ex:Food  .
```

Domain and range of properties

Important note: There is no type checking in RDF. Whether a triple adheres to domain/range specifications is an inference.

Domain and range of properties

Example: A Zoo is always located in a settlement.

Modeled via rdfs:range:

```
ex:city rdfs:domain ex:zoo .
ex:city rdfs:range ex:Settlement .
```

Conjunctive Interpretation

rdfs:domain and *rdfs:range* have a conjunctive interpretation.

Example:

```
ex:eat rdfs:domain ex:Mammal .
ex:eat rdfs:domain ex:Fish .
```

The interpretation here is that everything that eats something is/must be **both** a *Mammal* and a *Fish*.

T-Box & A-Box

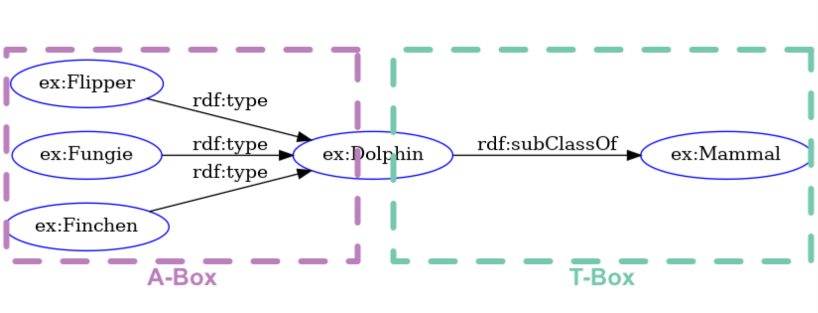
terminological knowledge (T-Box): class definitions and porperty definitions

assertional knowledge (A-Box): instance definitions

T-Box & A-Box

terminological knowledge (T-Box): class definitions and porperty definitions

assertional knowledge (A-Box): instance definitions



Additional information

In order to document the meaning of individuals, classes, and properties, we would like to add comments to the vocabulary defined. We can thus enhance the understanding of users of our vocabulary.

RDFS introduces different properties to capture the meaning of vocabulary elements without affecting their semantics, either through comments or by pointing to external resources.

Additional information

rdfs:label allows us to assign a name (string literal) to a resource.

Example:

```
ex:Cetacea rdfs:label "whales"@en .
ex:Cetacea rdfs:label "Wale"@de .
ex:Cetacea rdfs:label "cétacés"@fr .
```

Additional information

rdfs:comment allows us to assign a longer comment (string literal) to a resource.

Example:

```
ex:Cetacea rdfs:comment "carnivorous, finned,
    aquatic marine mammals"@en .
```

Inference

RDFS is based on formal semantics. That allows to draw valid and sound logical inferences.

Which conclusions can we deduce with RDFS?

- deduction from the class hierarchy
- deduction of entity class membership from domain and range
- deduction from the property hierarchy

Deduction from the class hierarchy

$$(e, rdf : type, c_1) \wedge (c_1, rdfs : subClassOf, c_2) \rightarrow (e, rdf : type, c_2)$$

```
ex:Dolphin rdf:subClassOf ex:Mammal .
ex:Flipper rdf:type ex:Dolphin .
```

$$\downarrow$$

Deduction from the class hierarchy

$$(e, rdf : type, c_1) \wedge (c_1, rdfs : subClassOf, c_2) \rightarrow (e, rdf : type, c_2)$$

```
ex:Dolphin rdf:subClassOf ex:Mammal .
ex:Flipper rdf:type ex:Dolphin .
```

$$\downarrow$$

```
ex:Flipper rdf:type ex:Dolphin .
ex:Flipper rdf:type ex:Mammal .
```

Deduction from domain and range

$$(e_1, p, e_2) \wedge (p, rdfs : domain, c_1) \wedge (p, rdfs : range, c_2)$$

$$\rightarrow (e_1, rdf : type, c_1) \wedge (e_2, rdf : type, c2)$$

```
ex:Tierpark_Olderdissen  ex:city  ex:Bielefeld  .
```

$$\downarrow$$

Deduction from domain and range

$$(e_1, p, e_2) \wedge (p, rdfs : domain, c_1) \wedge (p, rdfs : range, c_2)$$
$$\rightarrow (e_1, rdf : type, c_1) \wedge (e_2, rdf : type, c2)$$

```
ex:Tierpark_Olderdissen ex:city ex:Bielefeld .
```

$$\downarrow$$

```
ex:Tierpark_Olderdissen rdf:type ex:zoo .
ex:Bielefeld rdf:type ex:Settlement .
```

Deduction from subproperty relationships

$$(e_1, p_1, e_2) \wedge (p_1, rdfs: subPropertyOf, p_2) \rightarrow (e_1, p_2, e_2)$$

```
ex:Tierpark_Olderdissen ex:city ex:Bielefeld .
ex:city rdfs:subPropertyOf ex:location
```

$$\downarrow$$

Deduction from subproperty relationships

$$(e_1, p_1, e_2) \wedge (p_1, rdfs : subPropertyOf, p_2) \rightarrow (e_1, p_2, e_2)$$

```
ex:Tierpark_Olderdissen ex:city ex:Bielefeld .
ex:city rdfs:subPropertyOf ex:location
```

$$\downarrow$$

```
ex:Tierpark_Olderdissen ex:location ex:Bielefeld .
```

Summary

- RDF is based on triples (s,p,o), thereby forming a directed graph with labeled edges
- RDF can be serialized different languages
- Blank nodes allow making statements about something we can not identify
- Methods of representing different kind of data: RDF reification, RDF lists

Advertisement

Are you looking for a project, or do you intend to write a thesis?

$\rightarrow$ moritzblum.github.io or mblum@techfak.uni-bielefeld.de

Next Chapters

RDF exercise submission due to: May 15th

Upcoming lectures:

- May 15th - KG and LD (Basil)
- May 22nd- Query languages - SPARQL (Meisam)
- June 05th- Description Logics (Philipp)
- June 12th - Ontologies Web Ontology Language (Philipp)
- June 19th - Reasoning in Description Logics OWL (Philipp)
- June 26th - Ontology Engineering (Meisam)
- July 2nd - Foundational Ontologies (Meisam)
- July 10th - Ontolgoy Design Patterns (Moritz)
- July 17th - Knowledge Graph Completion (Moritz)
- July 24th - Data mining for the Semantic Web (Basil)