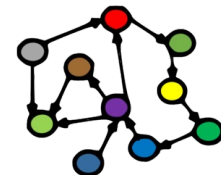# Welcome to INFO216:
Knowledge Graphs
Spring 2023

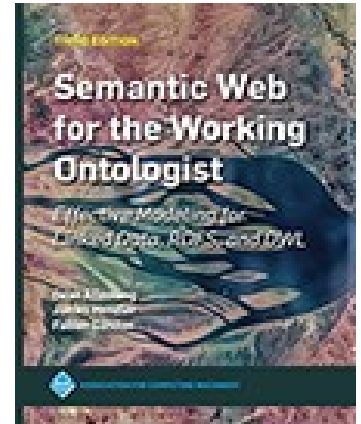## Andreas L Opdahl
<Andreas.Opdahl@uib.no>

Session 7: Rules (SHACL and RDFS)

- Themes:
  - why SHACL (SHapes Constraint Language)?
    - node and property constraints
  - why RDFS (RDF Schema)?
    - utility properties
    - classes and subclasses
    - properties and subproperties
    - entailments and axioms
  - motivation for OWL (the Web Ontology Language)

Readings

- Sources:

  - Allemang, Hendler & Gandon (2020):
    Semantic Web for the Working Ontologist, 3rd edition:
    chapters 7-8, but chapter 6-7 in the 2nd edition (no SHACL)

  - Chapter 5: SHACL in Validating RDF (available online)

  - Blumauer & Nagy (2020):
    The Knowledge Graph Cookbook – Recipes that Work:
    e.g., pages 101-106, 136-139 *(supplementary)*

- Resources in the wiki <http://wiki.uib.no/info216>:

  - Interactive SHACL Playground

  - W3C's RDF Schema 1.1 (sections 1-3 and 6)
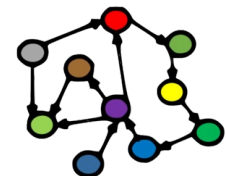
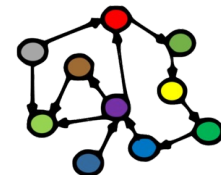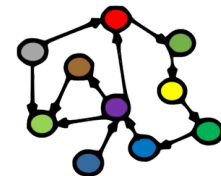 - Shapes Constraint Language (SHACL) (Editor's Draft)
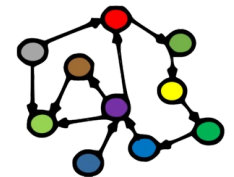
SHACL

SHACL (SHapes Constraint Language)

- Constraining the structure of RDF (and RDFS, OWL...) graphs
  - another W3C standard (from around 2018)
- Used to validate:
  - open and other KGs we want to *reuse*
  - graphs resulting from *user input*
  - the KGs *we make ourselves*
- *SHACL constraints are written in RDF*
  - a shapes graph is used to validate
  - a data graph
- *SHACL Core* and several extensions
  - we will focus on *SHACL Core*

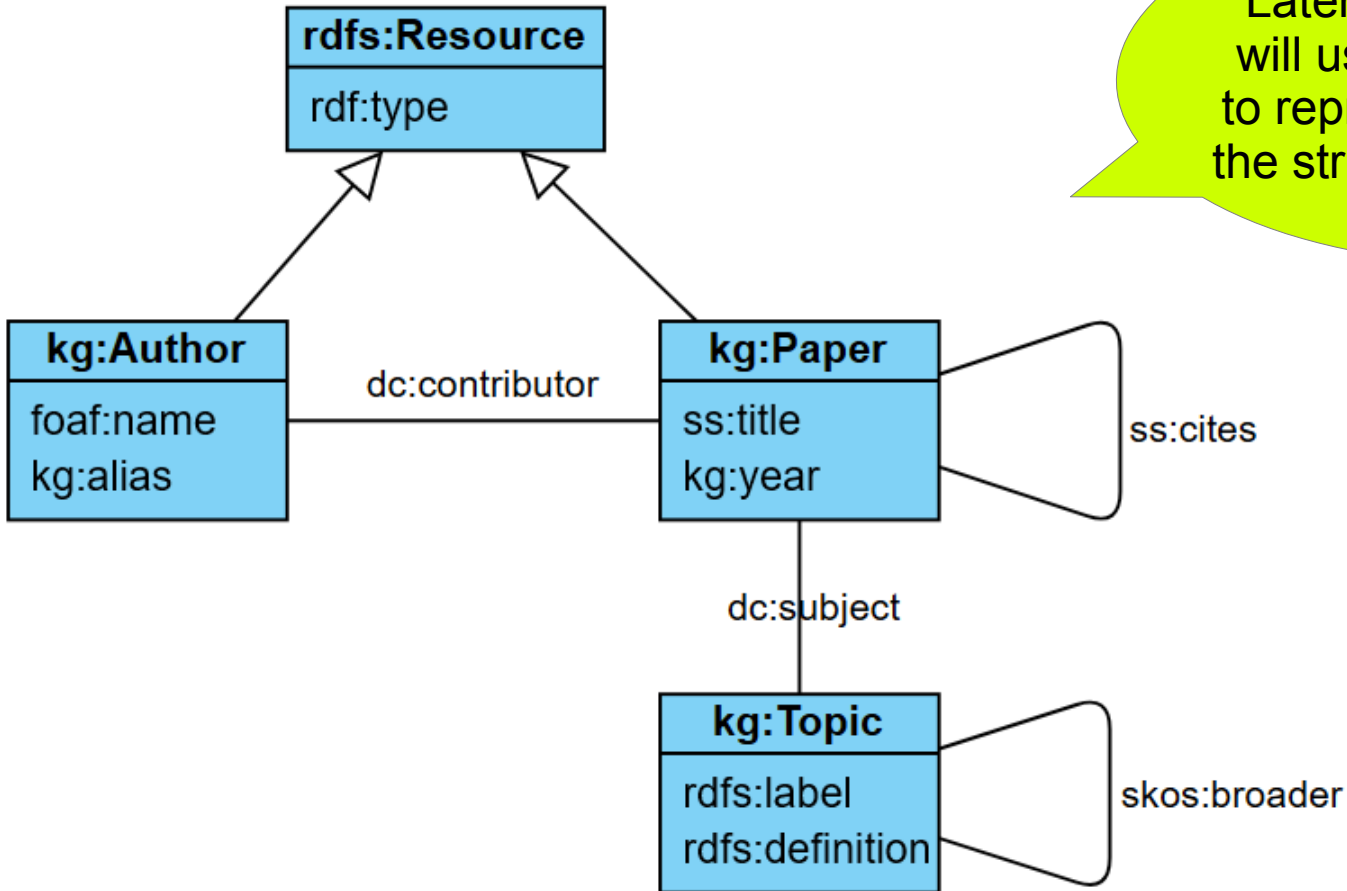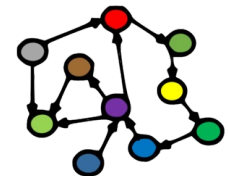# Example KG (←S03)

- A knowledge graph of research literature related to "Knowledge Graphs for the News"
  - built to support a recent literature study
  - 78 main papers with 291 authors
  - 4086 other papers with 8990 authors
  - 100s of topics and themes, >300k triples
- Accessible at http://bg.newsangler.uib.no/
  - runs on a Blazegraph triple store
  - Blazegraph's simple web front end, read only

Example KG: graph structure (←S03)

rdfs:Resource
rdf:type

kg:Author
foaf:name
kg:alias

dc:contributor

kg:Paper
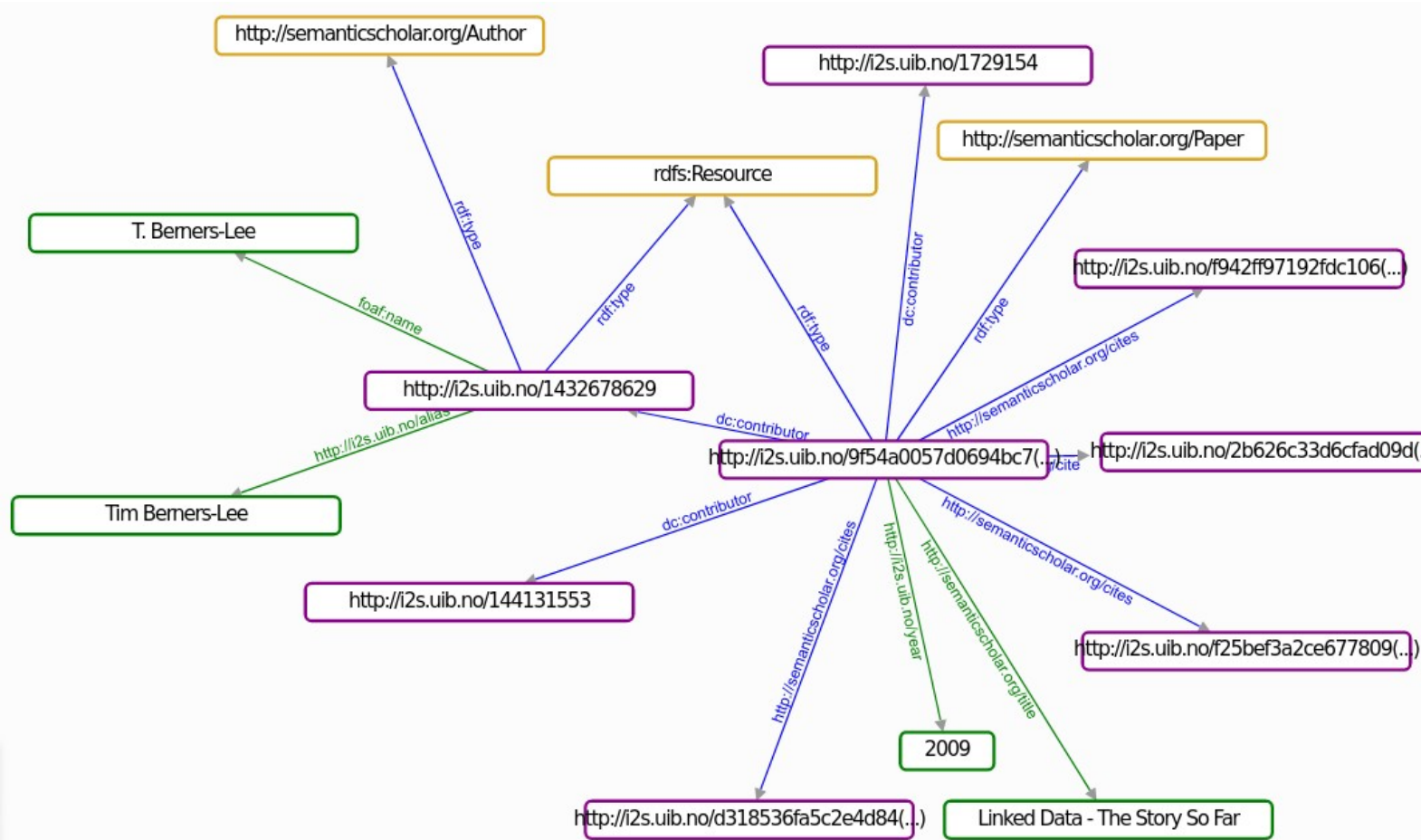ss:title
kg:year

ss:cites

dc:subject

kg:Topic
rdfs:label
rdfs:definition

skos:broader

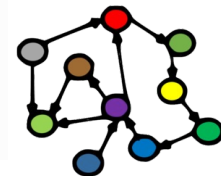Later in the course, we will use RDFS and OWL to represent and visualise the structure of ontologies.

Example KG: resources (←S03)

(The URIs are simplified.)

Example constraints

- Every main paper is the subject of exactly one year property.

```
kg:MainPaperShape
    a sh:NodeShape ;
    sh:targetClass kg:MainPaper ;
    sh:property [
        sh:path kg:year ;
        sh:minCount 1 ;
        sh:maxCount 1
    ] .
```
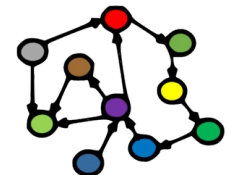
Example constraints

- Every main paper is the subject of exactly one year property.

- Every year value (literal object) of a main paper is an integer.

```
kg:MainPaperShape
    a sh:NodeShape ;
    sh:targetClass kg:MainPaper ;
    sh:property [
        sh:path kg:year ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
        sh:datatype xsd:integer
    ] .
```
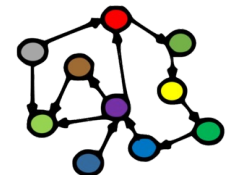
Example constraints

- Every main paper has at least one contributor

```
kg:MainPaperShape
    a sh:NodeShape ;
    sh:targetClass kg:MainPaper ;
    sh:property [
        sh:path dcterm:contributor ;
        sh:minCount 1
    ] .
```
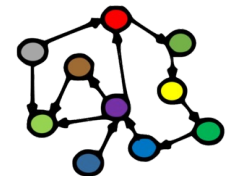
Example constraints

- Every main paper has at least one contributor
  - who is a main author

```
kg:MainPaperShape
    a sh:NodeShape ;
    sh:targetClass kg:MainPaper ;
    sh:property [
        sh:path dcterm:contributor ;
        sh:minCount 1 ;
        sh:class kg:MainAuthor
    ] .
```

Example constraints

- Every main paper has at least one contributor
  - who is a main author
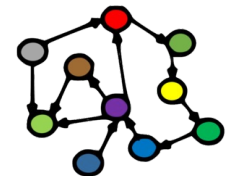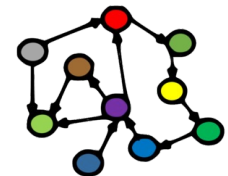 - and is represented by a URI

```
kg:MainPaperShape
    a sh:NodeShape ;
    sh:targetClass kg:MainPaper ;
    sh:property [
        sh:path dcterm:contributor ;
        sh:minCount 1 ;
        sh:class kg:MainAuthor ;
        sh:nodeKind sh:IRI
    ] .
```

Example constraints

- Every main paper has at least one subject
  - whose value is a SKOS concept (→S09) which is represented by a URI

```
kg:MainPaperShape
    a sh:NodeShape ;
    sh:targetClass kg:MainPaper ;
    sh:property [
        sh:path dcterm:subject ;
        sh:minCount 1 ;
        sh:class skos:Concept ;
        sh:nodeKind sh:IRI ;
    ] .
```

Example constraints

- Every main paper has at least one subject
  - whose value is either
    - a theme,
    - a topic

```
kg:MainPaperShape
    a sh:NodeShape ;
    sh:targetClass kg:MainPaper ;
    sh:property [
        sh:path dcterm:subject ;
        sh:minCount 1 ;
        sh:or ( [sh:class th:Theme]
                [sh:class ss:Topic] ) ;
        sh:nodeKind sh:IRI ;
    ] .
```

SHACL constraint structure

- SHACL constrains
  - node shapes
  - property shapes
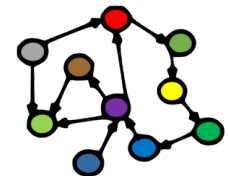- The node shapes can act as collections of property shapes belonging to the same class
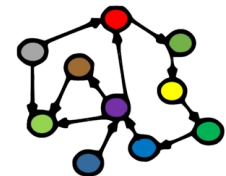


```
kg:MainPaperShape
   a sh:NodeShape ;
   sh:targetClass kg:MainPaper ;
   sh:property [
      sh:path dcterm:subject ;
      sh:minCount 1 ;
      sh:or ( [sh:class th:Theme]
              [sh:class ss:Topic] ) ;
      sh:nodeKind sh:IRI ;
   ] .
```

SHACL constraint structure

- SHACL constrains
  - node shapes
  - property shapes
- The node shapes are mostly collections of property shapes pertaining to the same class


```
kg:MainPaperShape
    a sh:NodeShape ;
    sh:targetClass kg:MainPaper ;
    sh:property kg:SubjectShape .

kg:SubjectShape
    a sh:PropertyShape ;
    sh:path dcterm:subject ;
    sh:minCount 1 ;
    sh:or ( [sh:class th:Theme]
            [sh:class ss:Topic] ) ;
    sh:nodeKind sh:IRI .
```
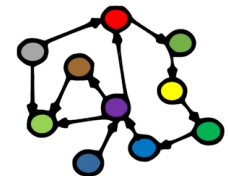
SHACL constraint structure

- SHACL constrains
  - node shapes
  - property shapes
- The node shapes are mostly collections of property shapes pertaining to the same class
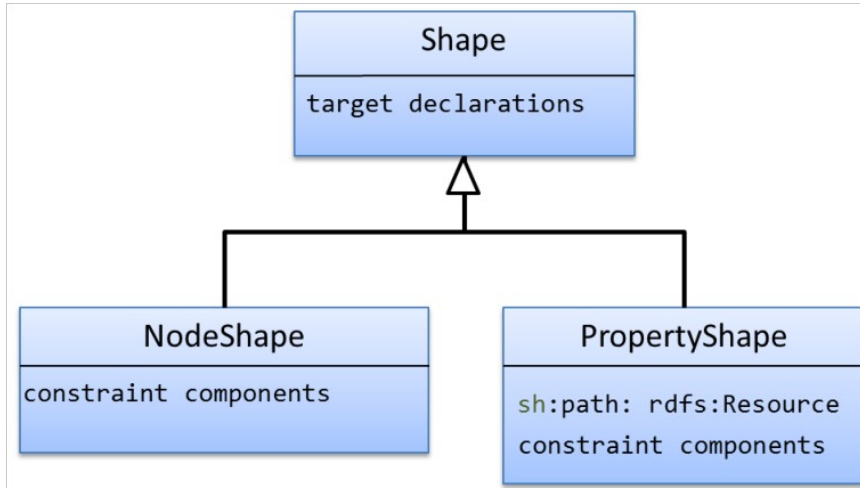
Almost the same, but now we can import shapes from other files and several node shapes can refer to the same property shape.
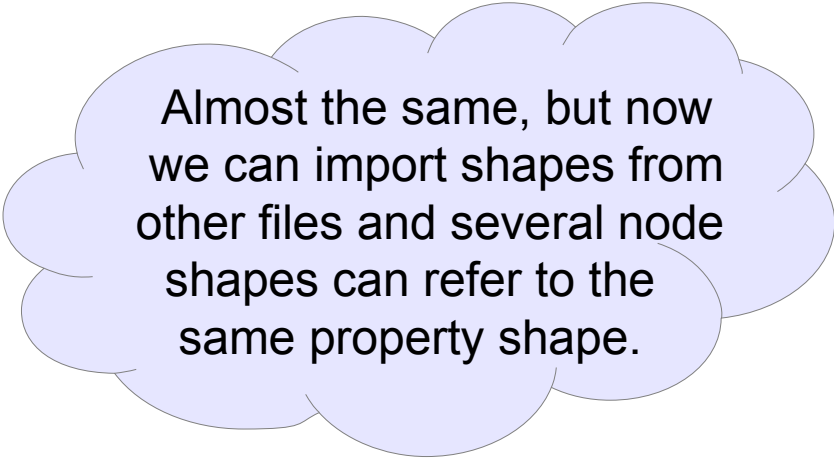
kg:MainPaperShape
  a sh:NodeShape ;
  sh:targetClass kg:MainPaper ;
  sh:property kg:SubjectShape .

kg:SubjectShape
  a sh:PropertyShape ;
  sh:path dcterm:subject ;
  sh:minCount 1 ;
  sh:or ([sh:class th:Theme]
         [sh:class ss:Topic]) ;
  sh:nodeKind sh:IRI .

Node shapes

- Node shapes specify constraints on focus nodes
  - \<node_shape_URI\> a sh:NodeShape
  - the focus nodes are often specified as sh:targetClass \<class_URI\>
    - the node constraints apply to all instances of the target class
    - alternatives: sh:targetNode, sh targetSubjectsOf, sh:targetObjectsOf
  - constraints *on each focus node itself*:
    - sh:class \<class_URI\>   or: sh:datatype \<datatype_URI\>
    - sh:in (...list of URIs/values...)   or: sh:hasValue ...URI/value...
    - sh:nodeKind can be one of
      - sh:IRI, sh:BlankNode, sh:Literal, sh:IRIOrLiteral, sh:BlankNodeOrIRI, sh:BlankNodeOrLiteral
      - sh:pattern \<regular_expression\>
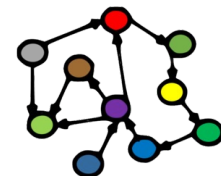
Node shapes

- Node shapes specify constraints on focus nodes
  - constraints on properties *from the focus node*
    (with the focus node as *subject*):
    - by shape URI:
      sh:property <property_shape_URI>

    - by anonymous node:
      sh:property [
         a sh:PropertyShape ; # not necessary
         sh:path <property_URI> ;
         ... specific property constraints go here ...
      ] ;

Property shapes

- Property shapes specify constraints about the values that can be reached from a focus node by some path
  - \<property_shape_URI\> a sh:PropertyShape # usually implicit
  - the property is often specified as sh:path \<property_URI\>
    - alternatively, SPARQL-like *property paths* can be used
    - the property constraints apply to:
      - all uses of the property path from the focus nodes
      - all values reached by the property path from the focus nodes
  - property constraints:
    - sh:minCount, sh:maxCount, ...
  - node constraints about the property *value* (the *object* resource or literal):
    - sh:class, sh:datatype, sh:nodeKind, sh:pattern, ...

Property paths in SPARQL and SHACL

SPARQL path...	...corresponds to SHACL path
schema:name	schema:name
^schema:knows	[sh:inversePath schema:knows]
schema:knows / schema:name	(schema:knows schema:name)
schema:knows | schema:follows	[sh:alternativePath (schema:knows schema:follows)]
schema:knows?	[sh:zeroOrOnePath schema:knows]
schema:knows+	[sh:oneOrMorePath schema:knows]
schema:knows* / schema:name	([sh:zeroOrMorePath schema:knows] schema:name)

Examples use the schema.org vocabulary <https://schema.org>

Validation reports

- Reports the results applying a SHACL shapes graph to a data graph
  - a sh:ValidationReport
  - three components:
    - sh:conforms (either true or false)
    - a results_text (from pySHACL)
    - zero or more sh:ValidationResults

SHACL validation result properties

Result property	Explanation
sh:focusNode	The focus node that was being validated when the error happened.
sh:resultPath	The path from the focus node. This property is optional usually corresponds to the sh:path declaration of property shapes.
sh:value	The value that violated the constraint, when available .
sh:sourceShape	The shape that the focus node was validated against when the constraint was violated.
sh:source ConstraintComponent	The IRI that identifies the component that caused the violation.
sh:detail	May point to further details about the cause of the error. This property can be used for reporting errors in nested nested shapes.
sh:resultMessage	Textual details about the error. This message can be affected by the sh:message property.
sh:resultSeverity	A value which is equal to the sh:severity value of the shape that caused the violation error, if present. Otherwise the default value will be sh:Violation.

There is a lot more...

- Logical expressions:
  - sh:or, sh:and, sh:xone (exactly one), and sh:not
- String and language constraints:
  - sh:length, sh:minLength, sh:maxLength, sh:pattern
  - sh:uniqueLang, sh:languageIn
- Value-range constraints on integers value nodes
- sh:severity of constraints
- Non-validating (informational) constraints:
  - sh:name, sh:description, sh:order, sh:group
- One shapes graph can owl:imports another

Programming pySHACL

```python
# pip install pyshacl

from pyshacl import validate
from rdflib import Graph

data_graph = Graph()
data_graph.parse('...')

shacl_str = """ ... """

shacl_graph = Graph()
shacl_graph.parse(
    data=shacl_str, format='ttl'
)
```
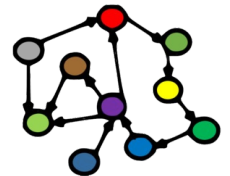
```python
results = validate(
    data_graph,
    shacl_graph=shacl_graph,
    inference='both'
)

(conforms,
    results_graph,
    results_text) = results

print(results_text)
```
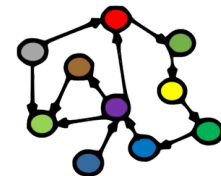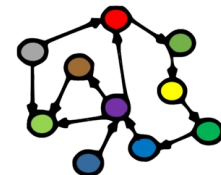
RDF Schema
(RDFS)

From RDF to RDFS

- RDF is a good start
  - excellent as a normal form for facts about individuals
  - less suitable for complex concept systems
    - e.g., vocabularies, ontologies
- RDF Schema (RDFS):
  - small RDF vocabulary for more expressive graphs
    - in particular for defining other vocabularies
  - many vocabularies defined in plain RDFS
    - also the foundation for SKOS and OWL
  - conventional prefix:
    - rdfs: http://www.w3.org/2000/01/rdf-schema#

Why RDF Schema (RDFS)

- *More expressive RDF graphs*, for example:
  - more specific types of resources
  - more specific types of predicates
  - which types of resources are the subjects and objects of which properties?
  - predefined *axioms*
  - data sets can *entail* additional triples (*inference*)
- *RDFS is expressed in RDF*
  - a small extension of the basic RDF vocabulary
  - RDF tools can be used on RDFS
    - ...but the semantics may get lost if inference is not supported

Utility properties in RDFS

- *Straightforward and much used:*
  - rdfs:label:
    - a human-readable label
  - rdfs:comment:
    - a human-readable comment
  - rdfs:seeAlso:
    - reference to further information
  - rdfs:isDefinedBy:
    - a human-readable definition
    - is a rdfs:subPropertyOf rdfs:seeAlso
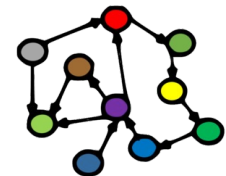 - *...often take "language-tagged"@en strings as objects*
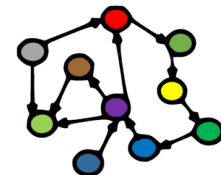
Resource classes (types)

- Classes are resources that represent a type of similar resources, which are the individuals in the class
  - e.g., dbpedia:Person, schema:Person, foaf:Person
  - belonging to a class is expressed by rdf:type:
    <RDF individual> rdf:type <RDFS class> .
    <RDF individual> a <RDFS class> .
  - an individual can belong to several classes (this is common!)
- *By convention, classes are named with an upper-case initial letter, properties with lower-case initial letters...*

Resource classes (types)

- Why resource classes?
  - the type (class) of a resource is an important part of its semantics
  - we can describe the class further
    - as natural-language text
    - formally using RDFS and OWL DL
  - knowing the type (class) of a resource often means we can infer additional information about it (entailment)
 - classes are important for defining and using other RDFS concepts

Not (quite) like object-oriented programming

- *Typical* OOP:
  - classes are templates for instantiating objects
  - objects with fixed class
  - adding a property restricts a class
  - information hiding

  - property overriding
  - local value ranges:
    "Tigers only have parents that are Tigers and Humans only have parents that are Humans"

- RDFS *is different*:
  - the properties of a resource determines its class
  - resources change class
  - adding a property enriches a resource
  - global visibility

  - no property overriding
  - no local value ranges
    (but you can use subproperties or use RDFS along with a rule language)
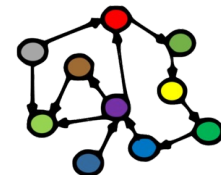
Subclasses (rdfs:subClassOf)

- Whenever an individual resource belongs to some class, it necessarily belongs to another class too, e.g.,
    - dbpedia:Physician rdfs:subClassOf dbpedia:Person .
- Why subclasses?
    - subclass hierarchies make the classes' semantics more precise!
    - subclasses facilitate more complete query answering
    - knowing the type of a resource often means we can infer additional information about it (entailment)
    - subclasses are important because different vocabularies may define overlapping, but not identical, classes
        - introduce a new class in the merged data set
        - make the old classes subclasses of the new class
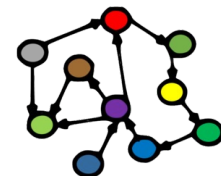
RDFS entailment

- The meaning of rdfs:subClassOf and the other RDFS concepts is defined by *entailment rules...*

- Example: classical *syllogism:*
    - *"All men are mortal."*   (Major Premise)
    - *"Socrates is a man."*   (Minor Premise)
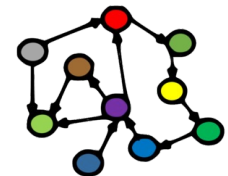   - *"Socrates is a mortal."*   (Valid conclusion)

RDFS entailment

- The meaning of rdfs:subClassOf and the other RDFS concepts is defined by *entailment rules...*

- Example: classical *syllogism* in RDFS:

  - *ex:Man rdfs:SubclassOf ex:Mortal .*   *(Major Premise)*

  - *ex:Socrates rdf:type ex:Man .*   *(Minor Premise)*

  - *ex:Socrates rdf:type ex:Mortal .*   *(Valid conclusion)*

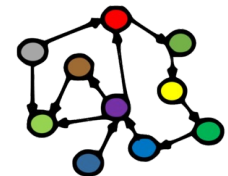*Entailment means that some triples are there in our RDFS models even when we have not asserted them*

RDFS entailment

- The meaning of rdfs:subClassOf and the other RDFS concepts is defined by *entailment rules...*

- Example: *pattern* for classical syllogism in RDFS:
  - *?c1 rdfs:SubclassOf ?c2 .*   (Major Premise)
  - *?s rdf:type ?c1 .*   (Minor Premise)
  - *?s rdf:type ?c2 .*   (Valid conclusion)

Entailment means that some triples are there in our RDFS models even when we have not asserted them

This rule is built into all RDFS models!

RDFS entailment [rdfs9]

- The meaning of rdfs:subClassOf and the other RDFS concepts is defined by *entailment rules* [rdfs9]:

- "The triples   ?s rdf:type ?c1 .
               ?c1 rdfs:subClassOf ?c2 .
entail that   ?s rdf:type ?c2 ."

```
PREFIX ...
INSERT {
    ?s rdf:type ?c2 .
} WHERE {
    ?s rdf:type ?c1 .
    ?c1 rdfs:subClassOf ?c2 .
}
```

Here, we express the rule using SPARQL. But RDFS rules are usually not implemented with SPARQL.
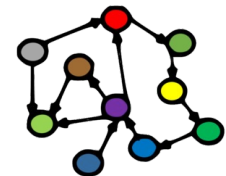
What does entailment mean?

- Entailment means that *some triples are there in our RDFS graphs even when we have not asserted them*
  - full list at http://www.w3.org/TR/rdf-mt/
  - 14 entailment rules in RDFS (2 in RDF)
- *Different RDFS tools may support entailment rules in different ways*, e.g.:
  - strategy 1: always add entailed triples when possible
  - strategy 2: only extract entailed triples when needed
  - with RDFLib you can use *RDFSClosure* in the *OWL-RL* package for this:
    - https://github.com/RDFLib/OWL-RL
    - https://owl-rl.readthedocs.io/en/latest/owlrl.html
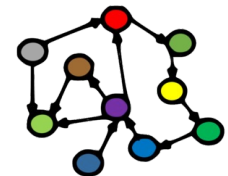
Transitive properties

- rdfs:subClassOf is *transitive*:
  - ex:Dolphins rdfs:subClassOf ex:Whales .
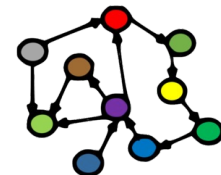 - ex:Whales rdfs:subClassOf ex:Mammals .
  - ex:Dolphins rdfs:subClassOf ex:Mammals .
- Entails new rdf:type triples about which classes an individual belongs to

Transitive properties

- rdfs:subClassOf is *transitive*:
  - ?c1 rdfs:subClassOf ?c2 .
  - ?c2 rdfs:subClassOf ?c3 .
  - ?c1 rdfs:subClassOf ?c3 .
- Entails new rdf:type triples about which classes an individual belongs to

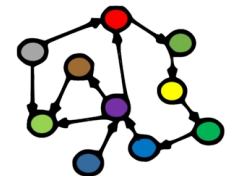# RDFS entailment [rdfs11]

- rdfs:subClassOf is *transitive*:

- "The triples   ?c1 rdfs:subClassOf ?c2 .

                     ?c2 rdfs:subClassOf ?c3 .

 entail that   ?c1 rdfs:subClassOf ?c3 ."

```
PREFIX ...
INSERT {
    ?c1 rdfs:subClassOf ?c3 .
} WHERE {
    ?c1 rdfs:subClassOf ?c2 .
    ?c2 rdfs:subClassOf ?c3 .
}
```
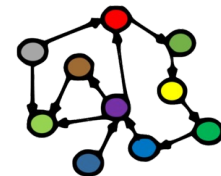
RDFS entailment [rdfs10]

- rdfs:subClassOf is also *reflexive*:

- "The triple   ?c rdf:type rdfs:Class .
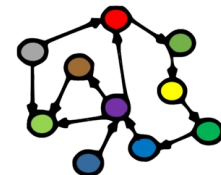 entails that   ?c rdfs:subClassOf ?c ."

```
PREFIX ...
INSERT {
      ?c rdfs:subClassOf ?c .
} WHERE {
      ?c rdf:type rdfs:Class .
}
```

- "Every class is its own subclass..."

Properties

- All predicates have *rdf:type rdf:Property*
    - this is expressed by an *entailment (next slide!)*
    - properties have *domains* and *ranges*, i.e., their subjects and objects belong to particular classes
    - properties can be *transitive*
- Why properties?
    - needed in RDF (along with rdf:type) to express that only certain resources act as predicates in triples
    - as for classes/subclasses:
        - clearer semantics, entailment, complete answers to queries and defining other concepts*, e.g.,*
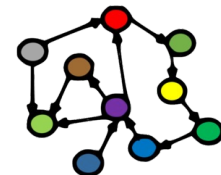   - *most classes are defined by their properties...*

RDF entailment [rdf1]

- "The triple   ?s ?p ?o .
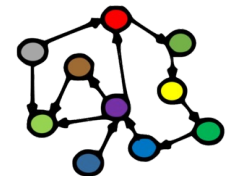 entails that   ?p rdf:type rdf:Property ."

  PREFIX rdf: <...>

  INSERT {
      ?p rdf:type rdf:Property .
  } WHERE {
      ?s ?p ?o .
  }

- Resources *become properties* by *being used* as predicates in triples!

Domain and range of properties

- The subjects and objects that occur in triples along with some property belong to certain classes

- Example:

  - *<subject>* ex:hasPassportNumber *<object>* .

  - when we see this triple, we know that:

    - the *<subject>* has rdf:type ex:Person

    - the *<object>* has rdf:type xsd:int

  - this is part of the semantics of ex:hasPassportNumber

  - ...can be expressed as follows:

    - ex:hasPassportNumber rdfs:domain ex:Person .

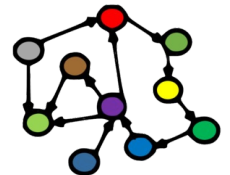   - ex:hasPassportNumber rdfs:range xsd:int .

RDFS entailment [rdfs2]

- "The triples   ?s ?p ?o .
                     ?p rdfs:domain ?t .
   entail that   ?s rdf:type ?t ."

```
PREFIX rdf: <...>
PREFIX rdfs: <...>

INSERT {
      ?s rdf:type ?t .
} WHERE {
      ?s ?p ?o .
      ?p rdfs:domain ?t .
}
```

RDFS entailment [rdfs3]

- "The triples   ?s ?p ?o .
                   ?p rdfs:range ?t .
   entail that   ?o rdf:type ?t ."

```
PREFIX rdf: <...>
PREFIX rdfs: <...>

INSERT {
      ?o rdf:type ?t .
} WHERE {
      ?s ?p ?o .
      ?p rdfs:range ?t .
}
```
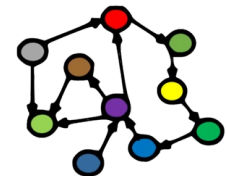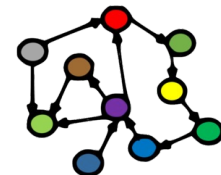
RDFS axioms

- RDFS axioms:
    - triples that are "built into" RDFS
    - predefined in any RDFS graph
    - essential part of the semantics of RDFS
    - full list at http://www.w3.org/TR/rdf-mt/
    - 40 axioms and 3 axiom schemas
- Example axioms for *rdf:type*:
    - rdf:type rdfs:range rdfs:Class .
    - rdf:type rdfs:domain rdfs:Resource .

RDFS entailment [rdfs3]

- "The triples   ?s ?p ?o .

               ?p rdfs:range ?t .

 entail that   ?o rdf:type ?t ."

```
PREFIX rdf: <...>
PREFIX rdfs: <...>

INSERT {
    ?o rdf:type ?t .
} WHERE {
    ?s ?p ?o .
    ?p rdfs:range ?t .
}
```

Remember:
rdf:type rdfs:range rdfs:Class .
is an axiom in RDFS. This axiom fits
straight into the rule:
    ?p = rdf:type
    ?t = rdfs:Class

RDFS entailment [rdfs3 + axiom]

- "The triples ?s rdf:type ?o .
                rdf:type rdfs:range rdfs:Class .
  entail that ?o rdf:type rdfs:Class ."

```
PREFIX rdf: <...>
PREFIX rdfs: <...>

INSERT {
    ?o rdf:type rdfs:Class .
} WHERE {
    ?s rdf:type ?o .
    rdf:type rdfs:range rdfs:Class .
}
```

Remember:
rdf:type rdfs:range rdfs:Class .
is an axiom in RDFS. This axiom fits
straight into the rule!

This is an axiom in RDFS!

RDFS entailment

- "The triples   ?s rdf:type ?o .
  ~~rdf:type rdfs:range rdfs:Class .~~
  entail that   ?o rdf:type rdfs:Class ."

```
PREFIX rdf: <...>
PREFIX rdfs: <...>

INSERT {
    ?o rdf:type rdfs:Class .
} WHERE {
    ?s rdf:type ?o .
    rdf:type rdfs:range rdfs:Class .
}
```

Because
rdf:type rdfs:range rdfs:Class .
is an axiom in RDFS, this rule
entails that every object in an
rdf:type-triple is an RDFS class.

RDFS entailment

- "The triples   ?s rdf:type ?o .
   entail that   ?o rdf:type rdfs:Class ."

```
PREFIX rdf: <...>
PREFIX rdfs: <...>

INSERT {
    ?o rdf:type rdfs:Class .
} WHERE {
    ?s rdf:type ?o .
}
```
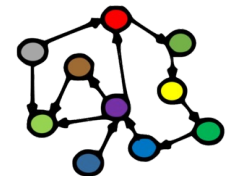
This rule entails that every object in an rdf:type-triple is an RDFS class.

It is not expressed explicitly in RDFS, but is always there in practice, because it is implied by the rule and the axiom we have just shown.

Subordinate properties (rdfs:subPropertyOf)

- Expresses that: whenever a subject resource and an object resource are related by a particular property, they are necessarily also related by another property, e.g.,

  - whenever this is a fact:
    dbpedia:Mathias_Dyngeland ex:goalkeeperFor dbpedia:SK_Brann .

  - then this is necessarily also a fact:
    dbpedia:Mathias_Dyngeland ex:playsFor dbpedia:SK_Brann .

  - *because* we have defined a *subproperty relationship*:
    ex:goalkeeperFor rdfs:subPropertyOf ex:playsFor .

- Is useful for connecting overlapping properties from distinct data sets, e.g.:

  - movie:actor_name, movie:film_name dc:name, rdfs:label
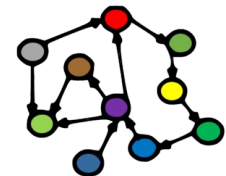
  - ...just like rdfs:subClassOf

RDFS entailment [rdfs7]

- "The triples   ?s ?p1 ?o .
                         ?p1 rdfs:subPropertyOf ?p2 .
   entail that   ?s ?p2 ?o ."

```
PREFIX rdfs: <...>

INSERT {
    ?s ?p2 ?o .
} WHERE {
    ?s ?p1 ?o .
    ?p1 rdfs:subPropertyOf ?p2 .
}
```

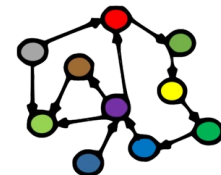# RDFS entailment [rdfs5]

- rdfs:subPropertyOf is *transitive*:

- "The triples   ?p1 rdfs:subPropertyOf ?p2 .
                    ?p2 rdfs:subPropertyOf ?p3 .
  entail that   ?p1 rdfs:subPropertyOf ?p3 ."

PREFIX rdfs: <...>

INSERT {
    ?p1 rdfs:subPropertyOf ?p3 .
} WHERE {
    ?p1 rdfs:subPropertyOf ?p2 .
    ?p2 rdfs:subPropertyOf ?p3 .
}

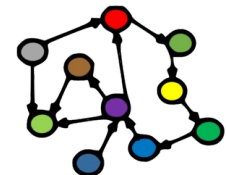# RDFS entailment [rdfs6]

- rdfs:subPropertyOf is *reflexive*:

- "The triple   ?p rdf:type rdf:Property .
 entails that   ?p rdfs:subPropertyOf ?p ."

```
PREFIX rdf: <...>
PREFIX rdfs: <...>

INSERT {
    ?p rdfs:subPropertyOf ?p .
} WHERE {
    ?p rdf:type rdf:Property .
}
```

Additional classes

- RDFS also defines rdfs:Class-es for:
  - resources: rdfs:Resource
    - the class of all resources
  - literals: rdfs:Literal
    - the class of all literals
    - rdfs:Literal rdfs:subClassOf rdfs:Resource .
  - datatypes: rdfs:Datatype
    - the class of all datatypes
    - rdfs:Datatype rdfs:subClassOf rdfs:Class .
  - all of them have rdf:type rdfs:Class
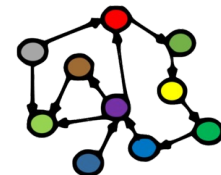 - all of them have entailment rules and axioms
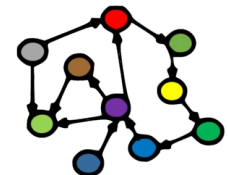
RDFS entailment [rdfs4a]

- *Every subject in a triple is a resource...*

- "The triple   ?s ?p ?o .
  entails that   ?s rdf:type rdfs:Resource ."

```
PREFIX rdf: <...>
PREFIX rdfs: <...>

INSERT {
        ?s rdf:type rdfs:Resource .
} WHERE {
        ?s ?p ?o .
}
```

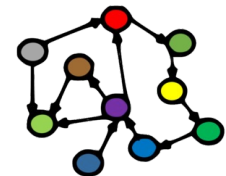# RDFS entailment [rdfs4b]

- *...and every object too*

- "The triple   ?s ?p ?o .
  entails that   ?o rdf:type rdfs:Resource ."

```
PREFIX rdf: <...>
PREFIX rdfs: <...>

INSERT {
      ?o rdf:type rdfs:Resource .
} WHERE {
      ?s ?p ?o .
}
```
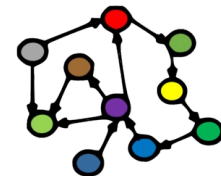
RDFS entailment [rdfs8]

- Every class corresponds to a set of resources.

  – ...or to a subset of the set of all resources.

- "The triple   ?c rdf:type rdfs:Class .
  entails that   ?c rdfs:subClassOf rdfs:Resource ."

```
PREFIX rdf: <...>
PREFIX rdfs: <...>

INSERT {
      ?c rdfs:subClassOf rdfs:Resource .
} WHERE {
      ?c rdf:type rdfs:Class .
}
```
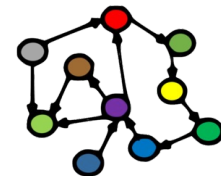
Overview

http://www-kasm.nii.ac.jp/~koide/SWCLOS2/Manual/05RDFSchema.htm

Summary of resources and properties

- Resources:
  - rdfs:Class rdfs:Resource rdfs:Datatype rdfs:Literal
  - rdf:Property, rdf:XMLLiteral, rdf:HTML
  - rdfs:Container rdfs:ContainerMembershipProperty
  - rdf:Alt rdf:Bag rdf:Seq rdf:List rdf:Statement

- Properties:
  - rdfs:subClassOf rdfs:subPropertyOf
  - rdfs:domain rdfs:range
  - rdfs:comment rdfs:seeAlso rdfs:isDefinedBy rdfs:label
  - rdfs:member rdf:_1 rdf:_2 ...
  - rdf:first rdf:rest rdf:nil
  - rdf:subject rdf:predicate rdf:object
  - rdf:value
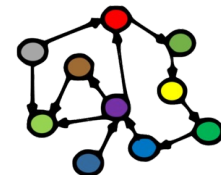
What we cannot express...

- RDFS has many limitations, e.g., it cannot say:
  - *"my ancestors' ancestors are also my ancestors"*
  - *"a Person has a unique birth number"*
  - *"a Person has exactly one father"*
  - *"a SoccerTeam has 11 players, but a BasketballTeam has 5"*
  - *"classes with different URIs actually represent the same class"*
  - *"resources with different URIs represent the same resource"*
  - *"properties with different URIs are actually the same"*
  - *"two individuals with different URIs are actually different"*
  - *"two classes cannot share individuals (they are disjoint)"*
  - *"a class is a combination (union or intersection) of other classes"*
  - *"a class is a negation of another class"*
- *Web Ontology Language (OWL) does all this and more!*

Next week: Ontologies (OWL)