RDF Summary

- more or less simple data model,

- possibility to use property and class names that are agreed Web-wide,

- the idea of URIs,

- the contents of RDF files can be integrated in a natural way via URIs and names.

4.10 Further Topics

- some additional RDF syntax: Reification and Collections. Later.

- Schema information: RDF Schema

- RDF/RDFS Model Theory + Reasoning

- More than schema information: Ontology specification by Description Logics and OWL

- How to provide RDF data and metadata on the Web?
  Later: RDF/XML. It's just another representation of RDF, RDF Schema and OWL data in a special XML syntax.

- What's missing?

- Rules?

Chapter 5
RDF Schema

Schema Information and Reasoning in an Open World

ONTOLOGIES

Schema languages, metadata languages, modeling languages, ontologies ...

Classical Data Models: seen as Specification and Constraints

- every schema description defines a (more or less complete) ontology:

- ER Model (1976, entity types, attributes, relationships with cardinalities),

- UML (1997, classes with subclasses, associations with cardinalities, OCL assertions to schema components etc.).

Knowledge Representation

Metadata provides additional information about resources of a type, or about a property.

- F-Logic signatures (1989),

- ... RDFS and OWL (Web Ontology Language)

SCHEMA INFORMATION IN AN OPEN WORLD

- schema describes
  - **–** allowed properties for an object,
  - **–** datatype constraints for literal properties [Here: XSD literal types],
  - **–** allowed types/classes for reference properties,
  - **–** cardinality constraints.

Closed World: Schema as Constraints

- a database must satisfy the constraints. It must be a *model* of the formulas – *the given data alone must be a model*.

Open World: potentially incomplete knowledge

- schema information as *additional information*

- since the world must be a model of the schema, some information can be *derived* from the schema.

- complain only if information is *contradictory* to the schema.

METADATA INFORMATION: TYPES, PROPERTIES, AND ONTOLOGIES

- Types and properties (i.e., everything that is used in a namespace) are not only "names", but are resources "somewhere in the Web", identified by a URI (used in RDF or in XML via namespaces).

$\Rightarrow$ a *domain ontology* describes the notions used in a namespace.

Schema and Ontology Information

- what types/classes are there,

- subclass information,

- what properties objects of a given type must/can have,

- to what types some property is applicable and what range it has,

- cardinalities of properties,

- default values,

- that some properties are transitive, symmetric, subproperties of another or excluding each other etc.

5.1   RDF Schema Notions - Overview

- RDF is the instance level

- cf. XML: DTDs and XML Schema for describing the structure/schema of the instance (DTD: no atomic datatypes, only tree structure; XSD has atomic datatypes)

- RDF Schema: stronger than DTD/XML – "semantic-level"
  - describe the structure of the RDF instance (i.e. the "schema" of the RDF graph, not of the RDF/XML file):
  - describes the schema *semantically* in terms of an (lightweight) ontology (OWL provides then much more features):
    * class/subclass
    * property/subproperty, domains and ranges
  - atomic datatypes for literal properties.

PREDEFINED RDFS CLASSES

The obvious ones

rdfs:Resource is "everything". All things described by RDF are called resources, and are instances of the class rdfs:Resource. This is the class of everything. All other classes are subclasses of this class. rdfs:Resource is an instance of rdfs:Class.

rdfs:Class : all things (resources and literals) are of rdf:type of some rdfs:Class. rdf:Properties have an rdfs:Class as domain and another rdfs:Class or rdfs:Datatype as range.
mon:Country rdf:type rdfs:Class.

An rdfs:Class is simply a resource X that is of (X rdf:type rdfs:Class). Usually, class names start with a capital letter.

Later, **owl:Class** will provide more interesting concepts of *intensionally defined* classes – like "the class father is the class of things that are male and have children".

rdf:Property is a subset of rdfs:Resource that contains all properties.

mon:capital rdf:type rdf:Property.

Usually, property names start with a non-capital letter.

[note: it's rdf:Property, not rdfs:Property!]

PREDEFINED RDFS CLASSES

rdfs:Datatype is the class of datatypes.

rdfs:Literal is the subclass of rdfs:Resource that contains all literals (i.e., values of
rdfs:Datatypes).
Literals do (usually) not have a URI, but a literal representation (as already discussed for
integers and strings).

E.g. the following holds

@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
xsd:int rdf:type rdfs:Datatype .

- Note that *reification* takes place here: rdfs:Datatype is both an instance of and a subclass
of rdfs:Class! Each instance of rdfs:Datatype is a subclass of rdfs:Literal.

PROPERTIES (OF CLASSES AND PROPERTIES) IN THE RDFS VOCABULARY

rdfs:subClassOf specifies that one rdfs:Class is an rdfs:subClassOf another:
:Cat rdfs:subClassOf :Animal .

rdfs:subPropertyOf specifies that one rdf:Property is an rdfs:subPropertyOf another:
:hasCat rdfs:subPropertyOf :hasAnimal .

rdfs:domain specifies that the domain of an rdf:Property is a certain rdfs:Class:
:hasCat rdfs:domain :Person .

rdfs:range specifies that the range of an rdf:Property is a certain rdfs:Class
(note that rdfs:Datatype is a subclass (and an instance) of rdfs:Class):
:hasCat rdfs:range :Cat .
:age rdfs:range xsd:int .

INFERENCE RULES

- until now, the SPARQL query language was applied to pure RDF facts (*extensional knowledge*)

- RDFS adds *inference rules* (= *intensional knowledge*)

- Queries are then not evaluated against the *fact base*, but against the *model* of the factbase and the rules.

- for this, a *reasoner* is required.

⇒ underlying *entailment relationship* based on *model theory*.

5.2   RDF/RDFS Model Theory

- FOL: Recall the definition of an *interpretation*: constants are mapped to the domain, while predicate symbols are mapped to relations over the domain.

- In RDF, predicates (predicate symbols) are also objects of discourse. [note that classes are just unary predicates while properties are binary predicates]
  One of the important goals of RDF and the Semantic Web is to make statements *about* classes and predicates!

Further reading

1. "Three Theses of Representation in the Semantic Web",
   Ian Horrocks and Peter Patel-Schneider.
   In *World-Wide-Web Conference (WWW 2003)*
   Note: can be found by google or via `http://www.dblp.de`
   (discusses three ways to define a model theory for RDFS)

RDFS Axiomatic Triples

Some axioms that are expected to hold in any RDFS model can be expressed inside RDF itself independent from the chosen logic (cf. `http://www.w3.org/TR/rdf-mt`):

```
rdf:type rdfs:domain rdfs:Resource .
rdfs:domain rdfs:domain rdf:Property .
rdfs:range rdfs:domain rdf:Property .
rdfs:subPropertyOf rdfs:domain rdf:Property .
rdfs:subClassOf rdfs:domain rdfs:Class .

rdf:type rdfs:range rdfs:Class .
rdfs:domain rdfs:range rdfs:Class .
rdfs:range rdfs:range rdfs:Class .
rdfs:subPropertyOf rdfs:range rdf:Property .
rdfs:subClassOf rdfs:range rdfs:Class .

rdfs:Datatype rdfs:subClassOf rdfs:Class .
```

... and some more.

The interesting things with RDFS are not these (rather trivial) axioms, but the built-in semantics of rdfs:domain/range/subClassOf/subPropertyOf.

For them, some model theory is required.

GENERAL CONSIDERATIONS

Needed: a *domain* (the things talked about), and a signature (the predicate names (that are used in the *logic formalization*)).

Straightforward (and intuitive) idea

- the domain are the resources and the blank nodes,
  classes are mapped to unary properties,
  (e.g. *person(john)*), properties are mapped to binary predicates,
  (e.g. *age(john,32)* and child(john,alice)) .

- problem: rdfs:subClassOf, rdfs:range etc. talk *about* classes and properties

Alternatives (see subsequent slides)

1. FOL with reification, i.e., handling classes and properties also as domain items, and having only one meta-predicate "holds",

2. resorting to Second-Order-Logic,

3. a special model theory not based on any other logic (as in
   `http://www.w3.org/TR/rdf-mt`) .

- The set of constant symbols consists of all IRIs, Blank node identifiers RDF-B, and Literals RDF-L as constant symbols,

- there is a a single "holds" predicate that represents the triples.

- Herbrand-style interpretation: all terms are "interpreted by themselves", i.e., since there are no function symbols in RDF, the domain $\mathcal{D}$ *is* just the set of constant symbols (which include everything: individuals, literals, classes, properties).

- a single "holds" property that represents the triples:

$$\mathcal{I}(\mathsf{holds}) = \{(s,p,o)|(s,p,o) \text{ holds in the given RDF ontology}\}$$

Advantages

- mapping to a well-known and well-investigated formalism,

- RDFS semantics can easily be specified by logical axioms, e.g., rdfs:range specifies that the range of an rdf:Property is a certain rdfs:Class:

$$\mathcal{M} \models \forall c, p : (\ \mathsf{holds}(p, \mathsf{rdfs:range}, c) \rightarrow$$
$$(\forall x, y : \mathsf{holds}(x, p, y) \rightarrow \mathsf{holds}(y, \mathsf{rdf:type}, c)))$$

Mapping of RDF/RDFS to FOL (cont'd)

$$\mathcal{I}(\mathsf{holds}) = \{(s,p,o)|(s,p,o) \text{ holds in the given RDF ontology}\}$$

Problems

This is just a *mapping* to an artificial predicate.

1. FOL in general is undecidable (i.e. there are no complete reasoners for it), while RDFS reasoning itself decidable, even only polynomial.

2. later: OWL builds on RDF and is based on the DL $\mathcal{SHOIN}(D)$, which is a decidable subset of FOL.
   The original translation of $\mathcal{SHOIN}(D)$ to FOL has also to be mapped to the "holds" predicate.
   Again, this would be a mapping from a decidable formalism to an undecidable one.

3. Unrestricted reification can lead to paradoxes (cf. Slide 224).
   $\Rightarrow$ Sorted FOL can be used to partition the domain into "sorts" (individuals, classes, properties) and to constrain the usage of the quantifiers.

4. Equality: for properties p_1, p_2, $p_1 = p_2$ holds by definition only if $I(p_1) = I(p_2)$ which means identity
   $\Rightarrow$ needs actually FOL+Equality

A MAPPING TO HIGHER-ORDER LOGIC

A second-order-logic domain $\mathcal{D}$ consists of two *disjoint* subsets:

- first-order objects $\mathcal{D}_1$: Let IRI_{obj} and RDF-B_{obj} denote all IRIs and blank nodes that denote objects. Literals also belong to that partition.

- second-order objects $\mathcal{D}_1$: predicates (and functions).
  For RDF, the predicates are classes IRI_{cls} and properties IRI_{prop}.
  (only when defining derived classes (in OWL), there will be blank nodes RDF-B_{cls} that represent classes.)

- 1st-order predicates are interpreted by relationships over the object domain.

- general: predicates of order n are interpreted over the domain of order n and are objects of the domain of order $n+1$.

- Quantifiers range either over 1st-order objects or over 2nd-order objects, e.g.
  rdfs:range is now a 2nd-order predicate:

$$\mathcal{M} \models \forall C, P : \text{rdfs:range}(P, C) \rightarrow (\forall x, y : P(x, y) \rightarrow C(y))$$

Mapping to Higher-Order Logic (cont'd)

Assuming a given alphabet of rdfs:Classes and rdf:Properties, each of them induces a unary or binary predicate, respectively.

- Objects: $\text{IRI}_{obj} \cup \text{RDF-B}_{obj}$

- Predicates: IRI_p (note that rdf:type and rdf:Property are excluded)

- for class symbols c in IRI_c: $\mathcal{I}(c) \subseteq \text{IRI}_{obj} \cup \text{RDF-B}_{obj}$
  E.g. ‹foo://bla/meta#Person›(‹foo://bla/persons/john›)

- for property symbols p in IRI_p ($p \neq$ rdf:Type):
  $\mathcal{I}(p) \subseteq (\text{IRI}_{obj} \cup \text{RDF-B}_{obj}) \times (\text{IRI}_{obj} \cup \text{RDF-B}_{obj} \cup \text{RDF-L})$
  E.g. ‹foo://bla/meta#child›(‹foo://bla/persons/john›, ‹foo://bla/persons/alice›)

- the class rdf:Property is mapped to a 3rd-order unary predicate s.t.
  $I(\text{rdf:Property}) = \text{IRI}_{prop}$.

- rdf:type is only implicitly represented by the *interpretation* of IRI_{cls}.

Mapping to Higher-Order Logic (cont'd)

Advantages

- intuitive mapping of properties and classes

- equality: for properties p_1, p_2, $p_1 = p_2$ holds $I(p_1) = I(p_2)$ which is the case if both have the same extension,

- can also express OWL notions like transitivity of properties.

Problems

- some RDF/RDFS notions don't even fit:
  - the class rdf:Property is mapped to a 3rd-order unary predicate,
  - rdf:type would have one first-order argument and one second-order argument.

- Usage of Higher-Order Logics:
  - can be used to axiomatize complex domains, like mathematics
  - highly intractable (= non-decidable, often even no heuristics-based incomplete proof methods)
  - HOL provers exist: they are used for *interactively* proving correctness, safety etc. (e.g., HOL (1993) and Isabelle (1994))

REIFICATION

Reification means to treat a higher-order object like a lower-order object:

- treat a class as an object, or

- treat a property as an object

i.e., to break the partitioning of the sorts/orders (which puts the mapping to Sorted FOL into FOL).

REIFICATION CAN LEAD TO PARADOXES

Reasoning with things that are both classes and instances reveals a famous paradox:

- define p as the set of all sets that do not contain themselves as an element:

  $\forall s : (p(s) \leftrightarrow \neg s(s))$

- is p in p?

  $p(p) \leftrightarrow \neg p(p)$

- any set of formulas that contains this definition has no model!

W3C RDF/RDFS MODEL THEORY

RDF Semantics, W3C Recommendation 10 February 2004

(`http://www.w3.org/TR/rdf-mt`)

- a semantics and model theory for RDFS (which borrows some features from higher-order ideas) (see next slide)

- without resorting to an encoding in any other logic
  $\Rightarrow$ no possibility to use theoretical results or reasoning algorithms.

Advantages

- handles intensional equality of classes or properties,

- expresses the specific RDF/RDFS ideas

Problems

- RDF constructs are not axiomatized logically as formulas,

- but incorporated into the semantics/model theory.

- no support by any reasoner,

- not extensible/adaptable to OWL.

Aside: formal details of `http://www.w3.org/TR/rdf-mt`

An RDF/RDFS interpretation I consists of the following:

- Universe = set IR of Resources: includes resources and literals (!?)

- IS interprets URIs (= constant symbols) into the universe IR
  (`http://www.w3.org/TR/rdf-mt` defines and uses $I(X) := IS(x)$ here).

- mappings $ICEXT$ (for classes) and $IEXT$ (for properties) from IR to 2^{IR} and $2^{(IR \times IR)}$:

- the set of classes is $IC = ICEXT(IS(\text{rdfs:Class})) \subset IR$,

- for each class URI y and each URI x,
  $IS(x) \in ICEXT(IS(y)) \Leftrightarrow (IS(x), IS(y)) \in IEXT(IS(\text{rdf:type}))$

- the set of properties, $IP \subset IR$.
  for each URI x, $IS(x) \in IP \Leftrightarrow (IS(x), IS(\text{rdf:Property})) \in IEXT(IS(\text{rdf:type}))$

- for each property URI p, $IEXT(IS(p)) \subset IR \times IR$ models the triples.

- RDFS notions are not expressed by formulas, but as "semantic conditions" in the model theory, e.g.,
  - (for rdfs:range): if $(IS(x), IS(y)) \in IEXT(IS(\text{rdfs:range}))$ and
    $(IS(u), IS(v)) \in IEXT(IS(x))$ then $IS(v) \in ICEXT(IS(y))$.

RDFS ENTAILMENT

An RDF Graph G *RDFS-entails* another RDF Graph H (which extends G with some edges)

- if every RDFS-interpretation which satisfies G also satisfies H.

For the *translation to FOL*, the following holds,

- if $\phi_G \cup \phi_{RDFS} \models_{FOL} \phi_H$ where ϕ_G and ϕ_H denote the FOL-translations of G and H, and ϕ_{RDFS} encodes the RDFS axioms (e.g. by rules), then $G \models_{RDFS} H$.
  [this is what rule-based reasoners do]

SPARQL on RDFS

- input: Turtle triples, seen as an RDF Graph G (including RDFS statements)

- query: a SPARQL graph pattern P

- answers: the answer bindings of all ground instances $\beta(P)$ of P s.t. $G \models_{RDFS} \beta(P)$.

$\Rightarrow$ query wrt. RDFS answering requires (a bit) more than only algebraic evaluation of conjunctive queries.

RDFS REASONING

- expressible in a decidable fragment of FOL: positive recursive Datalog
  - naive implementation: bottom-up graph completion by rules
  - querying: top-down Datalog evaluation (of any Datalog/Prolog system)
  - only issue: existentials from blank nodes (blank nodes mapped by skolemization, but it must be considered that two blank nodes describe the same individual)

$\Rightarrow$ use the FOL mapping

$\Rightarrow$ the following slides give the semantics of RDFS notions wrt. the FOL mapping.

REASONING WITH RDF, RDF SCHEMA AND OWL

- theoretical details will be discussed later. The underlying thing is either
  - graph completion by rules (RDFS, OWL-RL profile),
    (can be translated to positive Datalog)
  - *Description Logic (DL) Reasoning* (OWL DL)
    (requires a DL reasoner, based on Tableaux techniques)

- there are reasoners available for the Jena Framework:
  - an internal one:
    `jena -q -inf -qf` *sparql-file*
    for invoking SPARQL with its internal reasoner
  - an external one:
    (integrated into the semweb.jar used in the lecture as plug-in)
    `jena -q -pellet -qf` *sparql-file*
    for invoking SPARQL with the Pellet DL reasoner class
  - external ones as Web Services ...

USE OF THE JENA TOOL

- option "-t": transform (between Turtle and RDF/XML)
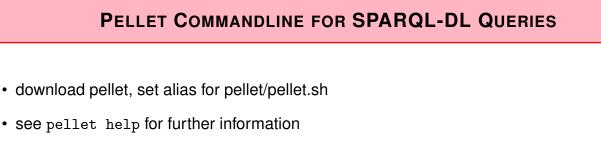   jena -t -pellet -if *rdf-file* .
  (-t is not complete for checking inconsistencies)

- option "-q": query
      jena -q -pellet [-if rdf-input-file] -qf *query-file* .

- option "-e": export the class tree (available only when the pellet reasoner is activated).
  Input is an RDF or OWL file:
      jena -e -pellet -if *rdf-file*.
  (for checking consistency, use -e)

PELLET COMMANDLINE FOR SPARQL-DL QUERIES

- download pellet, set alias for pellet/pellet.sh

- see `pellet help` for further information

- `pellet query -q` *query-file input-file*
  - does not use FROM line(s) in SPARQL, input file must be given explicitly,
  - only one input file possible.

ASIDE: DIG INTERFACE - DESCRIPTION LOGIC IMPLEMENTATION GROUP

- Web page: http://dl.kr.org/dig/

- agreed "tell-and-ask-interface" of DL Reasoners as Web Service:

- tell them the facts and ask them queries, or for the whole inferred model

- e.g. supported by "Pellet"

- URL for download see Lecture Web page

  ```
  may@dbis01:~/SemWeb-Tools/pellet-1.3$ ./pellet-dig.sh &
  PelletDIGServer Version 1.3 (April 17 2006)
  Port: 8081
  ```

- invoke the SPARQL Jena interface by
  `jena -q -qf` *sparql-file* `-inf -r` *reasoner-url*
  (e.g.: `http://localhost:8081`)

- note: the tell-functionality seems to transfer only part of the knowledge $\rightarrow$ incomplete reasoning $\rightarrow$ currently not recommended.

5.3 RDFS Vocabulary

SEMANTICS OF SUBCLASSES AND SUBPROPERTIES

rdfs:subClassOf specifies that one rdfs:Class is an rdfs:subClassOf another:

for any model $\mathcal{M}$ of the RDFS model theory,

$$\mathcal{M} \models \forall C_1, C_2 : (\ \mathsf{holds}(C_1, \mathsf{rdfs:subClassOf}, C_2) \rightarrow$$
$$(\forall x : (\mathsf{holds}(x, \mathsf{rdf:type}, C_1) \rightarrow \mathsf{holds}(x, \mathsf{rdf:type}, C_2))))$$

rdfs:subPropertyOf specifies that one rdf:Property is an rdfs:subPropertyOf another:

$$\mathcal{M} \models \forall P_1, P_2 : (\ \mathsf{holds}(P_1, \mathsf{rdfs:subPropertyOf}, P_2) \rightarrow$$
$$(\forall x, y : (\mathsf{holds}(x, P_1, y) \rightarrow \mathsf{holds}(x, P_2, y))))$$

SEMANTICS OF DOMAIN AND RANGE

rdfs:domain specifies that the domain of an rdf:Property is a certain rdfs:Class:

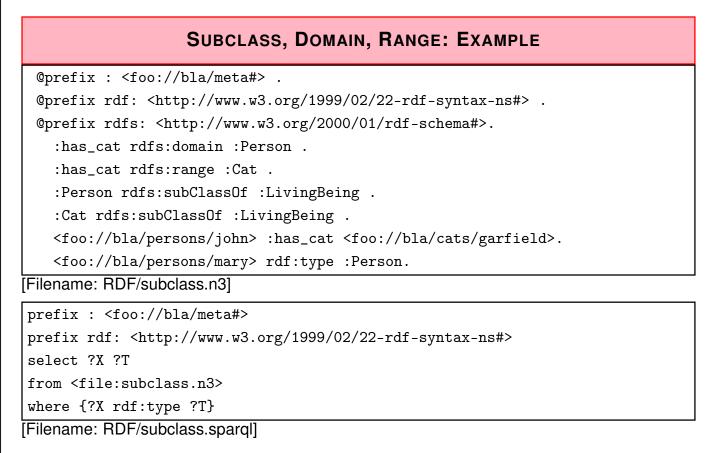$$\mathcal{M} \models \forall C, P : (\ \mathsf{holds}(P, \mathsf{rdfs:domain}, C) \rightarrow$$
$$(\forall x : (\exists y : \mathsf{holds}(x, P, y)) \rightarrow \mathsf{holds}(x, \mathsf{rdf:type}, C)))$$

rdfs:range specifies that the range of an rdf:Property is a certain rdfs:Class
(note that rdfs:Datatype is a subclass (and an instance) of rdfs:Class):

$$\mathcal{M} \models \forall C, P : (\ \mathsf{holds}(P, \mathsf{rdfs:range}, C) \rightarrow$$
$$(\forall y : (\exists x : \mathsf{holds}(x, P, y)) \rightarrow \mathsf{holds}(y, \mathsf{rdf:type}, C)))$$

Exercise

- Give an implementation by Datalog Rules for RDFS constructs.

SUBCLASS, DOMAIN, RANGE: EXAMPLE

```
@prefix : <foo://bla/meta#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
  :has_cat rdfs:domain :Person .
  :has_cat rdfs:range :Cat .
  :Person rdfs:subClassOf :LivingBeing .
  :Cat rdfs:subClassOf :LivingBeing .
  <foo://bla/persons/john> :has_cat <foo://bla/cats/garfield>.
  <foo://bla/persons/mary> rdf:type :Person.
```

[Filename: RDF/subclass.n3]

```
prefix : <foo://bla/meta#>
prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
select ?X ?T
from <file:subclass.n3>
where {?X rdf:type ?T}
```

[Filename: RDF/subclass.sparql]

- activate the reasoner (internal or pellet) when invoking Jena.

SUBCLASS, DOMAIN, RANGE: EXAMPLE (CONT'D)

Recall the previous example. Given the following facts:

```
:has_cat rdfs:domain :Person .
:has_cat rdfs:range :Cat .
:Person rdfs:subClassOf :LivingBeing .
:Cat rdfs:subClassOf :LivingBeing .
<foo://bla/persons/john> :has_cat <foo://bla/cats/garfield>.
<foo://bla/persons/mary> rdf:type :Person.
```

The domain/range information does not act as a constraint, but as information. From that knowledge, the following facts can be *inferred*:

- :has_cat implies that the subject (John) is a Person, and the object (Garfield) is a cat,

- both are thus LivingBeings.

SUBPROPERTIES

- outlook: combine it with owl:TransitiveProperty.

```
@prefix : <foo://bla/meta#> .
@prefix p: <foo://bla/persons/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
  p:john :hasChild p:alice, p:bob.
  p:kate :hasChild p:john.
  :hasChild rdfs:subPropertyOf :descendant.
  :descendant rdf:type owl:TransitiveProperty.
```
[Filename: RDF/descendants.n3]

```
prefix : <foo://bla/meta#>
select ?X ?Y
from <file:descendants.n3>
where {?X :descendant ?Y}
```
[Filename: RDF/descendants.sparql]

237

COMPARISON

SQL

- queries only against the database (no intensional knowledge),

- equivalent to tree expressions in relational algebra, based on set theory,

- formal semantics can be given purely syntactically with the algebra,

⇒ in the DB lecture, we did not need logic.

- equivalent to the relational calculus, semantics of queries can be given by the calculus.
  Equivalent to *nonrecursive Datalog* (cf. Database Theory Lecture) with "negation as
  failure" (top-down) stratification (bottom-up).

RDFS + SPARQL

- only restricted negation

- RDFS: built-in rules (positive, recursive Datalog)

- SPARQL: positive, nonrecursive Datalog

- intuitive bottom-up semantics

238

USING RDF IN THE WORLD WIDE WEB

- The (Semantic) Web is not seen as a collection of documents, but as a collection of correlated information (described via documents)

- using RDF, everybody can make statements about any resource
  (cf. link-bases in XLink)

  – incremental, world wide data and meta-data

  – distributed RDFS,

  – distributed RDF,

  – real URLs (Linked Open Data; cf. Slides 280 ff) vs. only virtual resources (URIs).

- not assumed that complete information about any resource is available.

- Open world, no notion of (implicit) negation.

- potentially inconsistent information;

- statements can be equipped with probabilities or labeled as opinions;
  fuzzy reasoning, belief revision ...

- ... lots of artificial intelligence applications ...

REASONING BASED ON RDFS

- RDF/RDFS *model theory* as above,

- rather simple Datalog rules, graph completion,

- queries: against the (completed) graph by matching (SPARQL).

- incomplete knowledge when reasoning: "open world assumption"

- only very restricted negation (none in RDFS; "!bound" in SPARQL 1.0 and "FILTER NOT EXISTS" in SPARQL 1.1 allow for negation of failure)

Preview: OWL

- based on DLs

- OWL-DL includes constructs that are not expressible by Datalog (union/disjunction) – requires "Disjunctive Datalog"
  $\Rightarrow$ totally different complexity and reasoning algorithms

- OWL-Lite is a simpler fragment (e.g., only 0-1-*-cardinalities) ... turned out that tools are not much simpler to design

- The OWL-RL profile: can be translated to positive recursive Datalog.

EXAMPLE/EXERCISE

Consider again the employee-manages-departments example (Slide 22).

- Give the RDF Graph.

- give the Turtle triples and feed them into the Jena tool.

ADDITIONAL RDF/RDFS VOCABULARY

The rdf/rdfs namespaces provide some more vocabulary:

Like most data models, RDF provides a representation for *Collections*:

- Collections: `rdf:Alt`, `rdf:Bag`, `rdf:Seq`, `rdf:List` are collections.
  Lists have properties `rdf:first` (a resource) and `rdf:rest` (a list). Others have
  properties `_1`, `_2`, ... that refer to their members.

- (rdfs:Container, rdfs:member, rdfs:ContainerMembershipProperty)

... these are partially used implicitly (e.g., collections in owl:intersectionOf, owl:OneOf), but
often not supported by OWL reasoners if used explicitly (see Slides 421 ff.).

EXAMPLE: THE MONDIAL ONTOLOGY

See mondial.n3, mondial-europe.n3 and mondial-meta.n3 on the Web page.

Note that it is highly redundant: defining just rdfs:domain and rdfs:range of properties implies most of the classes (and also most of the rdfs:type relationships in mondial.n3).

```
prefix mon: <http://www.semwebtech.org/mondial/10/meta#>
prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
select ?X
from <file:mondial.n3>
from <file:mondial-meta.n3>
where {?X rdf:type mon:Country}
```

[Filename: RDF/mondial-meta-query.sparql]

• activate Jena with reasoner (if mondial.n3 is too big, use mondial-europe.n3 instead)

Mondial is not an interesting example for RDFS (and OWL):

• it's mainly data, no intensional knowledge, no complex ontology

• for that reason it is a good example for SQL and XML.

• RDFS and OWL is interesting when information is *combined* and additional knowledge can be derived.

Developing Ontologies

• have an idea of the required concepts and relationships (ER, UML, ...),

• generate a (draft) n3 or RDF/XML instance,

• write a separate file for the metadata,

• load it into Jena with activating a reasoner.

• If the reasoner complains about an inconsistent ontology, check the metadata file alone. If this is consistent, and it complains only when also data is loaded:

  – it may be due to populating a class whose definition is inconsistent and that thus must be empty.

  – often it is due to wrong datatypes. Recall that datatype specification is not interpreted as a constraint (that is violated for a given value), but as additional knowledge.