Chapter 4
RDF: Resource Description Framework

Recall:

Relational Model is a specialization of First-Order Logic:

- no function symbols. Only constant literals.
- entity-type relations:
  collect data about instances of a certain entity type,
  property names given by the attributes of the relation schema
- relationship-type relations:
  represent instances (binary or n-ary) relationship types,
  relationship name given by the table name

THE SEMANTIC WEB DATA AND KNOWLEDGE MODEL

Combination of several concepts:

- Data Model: RDF (Resource Description Framework)
  - simple logical data model
  - things: resources; Web aspect: Web-wide *Uniform Resource Identifiers*
  - statements express properties: subject-predicate-object
- Metadata: RDF Schema
  - conceptual model: classes, subclasses and properties
  - classes and properties are also resources and can be described.
    (i.e., “second-order”)
- database-style format + XML representation
- so far: more database-style than knowledge-base or ontology
- OWL: higher-level conceptual model
  based on description logic (a fragment of FOL), “sliced” in complexity levels

Design and Use of a Domain Ontology

Design

- define the concepts (using RDFS): names of classes, class hierarchy, properties
- define a schema for URIs
- all people then use this schema and these names

Use

- make statements with the given vocabulary about things identified by the URIs

Global Semantics

- ... just collect all statements.
- since all use the same URIs things will easily fit together.

$\Rightarrow$ The Web as a global data source

- query evaluation?
- incompleteness, inconsistencies, junk & fakes.

RDF: Resource Description Framework

RDF is another specialization of First-Order Logic without function symbols.

Extension of the ideas of conceptual modeling (ER-Model, UML) and a restricted subset of first-order logic:

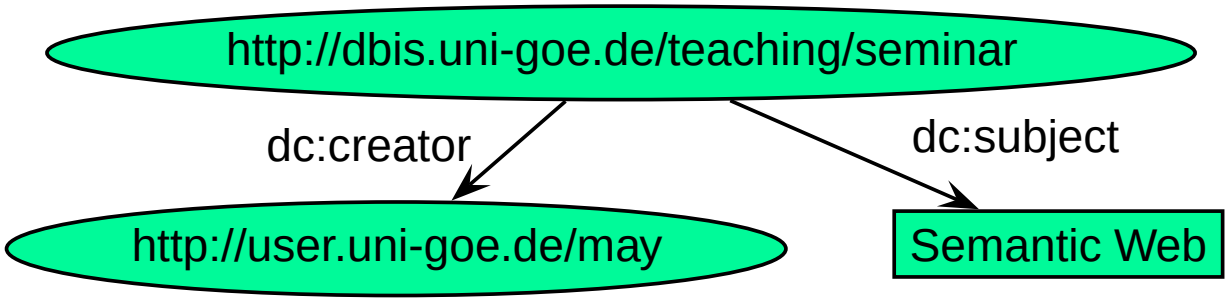
- (globally) agreed identifiers of elements of the domain as constants,
- literal constants,
- concepts (as unary predicates, e.g., Country(*germany*); usually with capital first letter),
- properties as binary relationships, e.g.,
  capital(*germany*,*berlin*) , or (*germany*, capital, *berlin*); usually with non-capital first letter.

Model: RDF Graph + derived knowledge

- derived knowledge: by RDFS and OWL built-in notions

4.1 RDF: Idea and Overview

- first Working Draft: 1997
- triple-based “RDF statements”: (*subject*, *predicate*, *object*)
- graphical representation ($\Rightarrow$ RDF graph), “N3” ASCII representation, XML representation
- *subject* and *object* are *resources* (cf. terminology for XLink)
  the object can also be a literal value (of an XML Schema datatype).
- a resource is everything ... that can be described by a URI
  (graphical notation: an ellipse)
  - a Web page, a book, a lecture ...



EXAMPLE: DUBLIN CORE

"Dublin Core": a set of properties for describing (HTML Web) documents (defined at a metadata workshop in Dublin/Ohio 1995).

- Title: A name given to the resource.
- Creator: An entity primarily responsible for making the content of the resource.
- Subject: The topic of the content of the resource.
- Description: An account of the content of the resource.
- Publisher: An entity responsible for making the resource available.
- Contributor: An entity responsible for making contributions to the content of the resource.
- Date
- Type: The nature or genre of the content of the resource.
- Language: A language of the intellectual content of the resource.

etc ... see `http://dublincore.org/documents/dces/`

RDF Model and Triples

- Initial goal: semantic description of things on the Web (i.e., Web pages and parts of them)
  author, keywords, *annotation* of links
  goal: semantics-driven Web indexing and search engines
- Information about real resources (*Triples, Statements*):

  (<http://dbis.uni-goe.de> <dc:creator> “Wolfgang May”)
  (<http://dbis.uni-goe.de> <dc:subject> “Database Group”)
  (<http://dbis.uni-goe.de/teaching/semweb/> <dc:type> “Lecture”)

The Semantic Web intention was not only to annotate things, but also to *relate* them:

(<http://dbis.uni-goe.de> <uni:offers-lecture>
<http://dbis.uni-goe.de/teaching/semweb/ >)

- This requires *relationships* from other domains.
- some “things” are also not real HTTP Web pages (e.g., persons)

$\Rightarrow$ generic notion of *resources*
(note: properties are also resources)

URIs and URLs

Things that are not “real” resources that have a URL (e.g. in HTTP) get a virtual *Unified Resource Identifier*, e.g.

(<http://user.cs.uni-goe.de/∼may> <dc:creator> <de:person-D-12345678>)

and can then be described:

(<de:person-D-12345678> <uni:position> <uni:Professor>)
(<de:person-D-12345678> <bla:lives> <geo://country-de/city-goettingen>)
(<de:person-D-12345678> <bla:e-mail> <mailto:may@cs.uni-goe.de>)

RDF for Data Integration

- information contained in the triples is independent from where they are located, and from their order.
- RDF files (knowledge bases) in different places that use the same URIs for “things” and the same “names” for properties can be easily combined.
- these “names” are taken from *domain ontologies*.

URIs and URLs

URI according to RFC 2396 (http://www.ietf.org/rfc/rfc2396.txt)

- a URI can be a locator (URL), a “name” (URN), or a “general” URI.
  General form: *scheme*:*hierarchical_part*
- URIs: a sequence of characters; often with a hierarchical structure
- URLs are those URIs that identify resources via their physical access mechanism (i.e., http, ftp, gopher, file, ...), in general their *schema* part names the protocol.
  http://www.informatik.uni-goettingen.de/people.html
  mailto:may@informatik.uni-goettingen.de
  news:de.comp.text.tex
- URI schemes can be “registered” at www.IANA.org (Internet Assigned Numbers Authority).
  example (non-registered): isbn:3-89722-153-5
- URIs serve as agreed *identifiers* in a certain community.
- some of them are valid URLs, some of them not.

RDF uses general URIs, not only URLs.

Semantics of URIs and URLs

URIs: mainly just identifiers, can be URLs

"Typical" Objects

- Objects are resources. Their URIs can be URLs or purely virtual.
- applications that use them load some given RDF files that describe them and evaluate these files.

Properties and Class Names

- They are also resources
- for querying, their URIs are just identifiers
- for an application, there *can* be an actual description of a notion at the place that is identified by the URL,
- then, an application can find this information.

SYNTAX OF URIS AND URLS

- General form: *scheme*:*hierarchical_part*
- short form: when only a single document is used, *#name* is sufficient – it is expanded to *url-of-the-document#name*.
- all representations (N3, RDF/XML) allow to use *prefixes* (cf. XML Namespaces) that hide the actual schema, e.g.,
  - declare `prefix monmeta: “http://www.semwebtech.org/mondial/10/meta#”` `prefix mon: “http://www.semwebtech.org/mondial/10/”`
  - use `monmeta:Country` as class name (stands for “http://www.semwebtech.org/mondial/10/meta#Country)
  - use `monmeta:capital` as property name
  - use `http://www.semwebtech.org/mondial/10/countries/D` as URI for Germany
  - use `mon:countries/D/provinces/Berlin/cities/Berlin` as URI for Berlin
- there is no commitment that these URLs actually exist or what can be found there!

RDF Representations and Notations

- as triple “database”: “N-triple” (next section)
- graphical: as a graph (illustrated in the next section)
- an RDF/XML representation that represents the RDF data, used for data exchange (for every RDF database there are multiple RDF/XML serializations)
- note that although RDF data looks like a large table with three columns, it cannot be stored directly in SQL: the *subject* and *object* column must hold URIs, strings and numbers.
  (which could be done e.g. using SQLX)

Tools

The W3C RDF Validator

- `http://www.w3.org/RDF/Validator/`
- input: RDF/XML
- output: Graph and/or triples

A lightweight JENA-based locally installed tool

- JENA (`http://jena.sourceforge.net`) is a Java-based Semantic Web Framework that supports RDF/RDFS, several types of OWL reasoning, and the SPARQL language; optionally an underlying relational database system can be used.
- see Web page of the lecture for further information.

4.2 The “N3” RDF Notation

- Triple-“Statements” *subj pred obj .*
  (Alternatives: *subj* has *pred obj* . or *obj* is *pred* of *subj* .)
- URIs: as <*uri*>
- Literals may occur as objects: as numbers (e.g. 42) or strings (e.g., “John Doe”).
- Short form for x p y_1 . and x p y_2 .: x p y_1, y_2.
  <#john> <#child> <#alice>, <#bob> .
- Short form for x p_1 x_1 . and x p_2 x_2 .: x p_1 y_1; p_2 y_2.
  <#alice> <#age> 10; <#name> “Alice” .

EXAMPLE: N3

- example: use most simple “URI”s

```
<family:john> <family:name> "John";  <family:age> 35;
              <family:child> <family:alice>, <family:bob> .
<family:alice> <family:name> "Alice"; <family:age> 10 .
<family:bob> <family:name> "Bob"; <family:age> 8 .
```

[Filename: RDF/family.n3]

Query language: SPARQL (details later)

```
select ?X ?N
from <file:family.n3>
where {?X <family:name> ?N}
```

[Filename: RDF/family.sparql]

- user interface of the local Jena-based tool: see Web page
  here: jena -q -qf family.sparql
- more on the use of URIs later.

LITERALS IN RDF

- Literals are strings (“John”) and numbers (42, 3.1415, 1.23E26)

```
select ?X ?N ?A
from <file:family.n3>
where {?X <family:name> ?N . ?X <family:age> ?A }
```

[Filename: RDF/family-age.sparql]

- Literals can be associated with *XML Schema Datatypes* xsd:int (numbers), xsd:decimal (with dec. point) , xsd:double (with exp).
- full syntax *value*^^*datatype-url*,

  e.g. 42^^<http://www.w3.org/2001/XMLSchema#int>
- String literals can be associated with languages, e.g. “München”@DE, “Munich”@EN, “Monaco”@IT.

URIs as Identifiers across RDF Files

- URIs: allow for “referencing” things across RDF files

```
<family:john> <family:name> "John";  <family:age> 35;
              <family:child> <family:alice>, <family:bob> .
<family:alice> <family:name> "Alice"; <family:age> 10 .
<family:bob> <family:name> "Bob"; <family:age> 8 .
```

[Filename: RDF/family.n3]

```
<family:mary> <family:name> "Mary";  <family:age> 32;
   <family:married> <family:john>;
   <family:child> <family:alice>, <family:bob> .
```

[Filename: RDF/family2.n3]

```
select ?X ?C ?A
from <file:family.n3>
from <file:family2.n3>
where {?X <family:child> ?C . ?C <family:age> ?A }
```

[Filename: RDF/family-both.sparql]

URIs as Web-Wide Identifiers

- The identifiers carry non-logical semantics as “names” throughout the (Semantic) Web.
- URIs are agreed in the same way as property names by the domain ontology designer. Real URLs and virtual URIs can be arbitrarily mixed.
- URIs to be used Web-wide can be defined freely
  - referring to the URL of the file where they are defined
  - defining an URI that is completely different from the file’s URL
- all members of a “community” can describe the resources by RDF.

URIs: LOCAL IDENTIFIERS

- if only a local part of the URI is given, it is extended with the document URL:

```
<#john> <#name> "John";  <#age> 35;  <#child> <#alice>, <#bob> .
<#alice> <#name> "Alice"; <#age> 10 .
<#bob> <#name> "Bob"; <#age> 8 .
```

[Filename: RDF/john-local.n3]

```
select ?X ?Y ?N
from <file:john-local.n3>
where {?X ?Y ?N}
```

[Filename: RDF/john-local.sparql]

Result (among others):

```
X / <file:///homewap1/may/teaching/SemWeb/RDF/john-local.n3#alice>
Y / <file:///homewap1/may/teaching/SemWeb/RDF/john-local.n3#name>
N / "Alice"
```

URIs: RDF on the Web

- such files can be queried via HTTP
- ... and the notion http://.../...#name becomes globally known,
- resource URIs combine the URI of the file + local part

```
select ?X ?N
from <http://dbis.informatik.uni-goettingen.de/Teaching/SemWeb/RDF/john-local.n3>
where {?X <http://dbis.informatik.uni-goettingen.de/Teaching/SemWeb/RDF/john-local.n3#name> ?N}
```

[Filename: RDF/john-http.sparql]

- use “base” for a relative addressing in the query:

```
base <http://dbis.informatik.uni-goettingen.de/Teaching/SemWeb/RDF/john-local.n3>
select ?X ?Y
from <http://dbis.informatik.uni-goettingen.de/Teaching/SemWeb/RDF/john-local.n3>
where {?X <#name> ?Y}
```

[Filename: RDF/john-base.sparql]

- Note: “base” only in SPARQL, not in N3

USING WEB-WIDE URIS

- other RDF files can then also make statements about resources in the “scope” of the remote URL, and even add resources to that scope.

```
<http://dbis.informatik.uni-goettingen.de/Teaching/SemWeb/RDF/john-local.n3#mary>
  <http://dbis.informatik.uni-goettingen.de/Teaching/SemWeb/RDF/john-local.n3#married>
  <http://dbis.informatik.uni-goettingen.de/Teaching/SemWeb/RDF/john-local.n3#john>.
```

[Filename: RDF/mary-remote.n3]

```
base <http://dbis.informatik.uni-goettingen.de/Teaching/SemWeb/RDF/john-local.n3>
select ?X ?Y
from <http://dbis.informatik.uni-goettingen.de/Teaching/SemWeb/RDF/john-local.n3>
from <file:mary-remote.n3>
where {?X <#married> ?Y}
```

[Filename: RDF/mary-http.sparql]

- note that adding resources or facts to things in “foreign” scopes is also a danger in RDF and in the Semantic Web.

$\Rightarrow$ “Trust” is an important keyword for SW applications.

URIs in the Semantic Web

Base URIs are often used for distinguishing ontologies

- e.g., <http://purl.org/dc/elements/1.1/> is the base URL for the Dublin Core Ontology that defines notions for annotating documents with authors etc.
- <http://www.semwebtech.org/mondial/> could be a base URI for defining the notions used in Mondial
  - entity types (Country, City) and property names (has-capital)
  - recall that entity instances are also resources that have a URI in this scope.
  - see later for details how to design this.
- ... first continue with syntax.

N3 Prefix Notation

define @prefix *pre*: <*uri*>.
and then use expressions *pre*:*qname*

These expressions expand to <*uri qname*>.

```
@prefix : <foo://bla/names#> .
@prefix person: <foo://bla/persons/> .
  person:john :name "John";  :age 35;  :child person:alice, person:bob.
  person:alice :name "Alice"; :age 10.
  person:bob :name "Bob"; :age 8.
```

[Filename: RDF/john.n3]

Equivalent:

```
<foo://bla/persons/john> <foo://bla/names#name> "John"; <foo://bla/names#age> 35;
     <foo://bla/names#child> <foo://bla/persons/alice>, <foo://bla/persons/bob>.
<foo://bla/persons/alice> <foo://bla/names#name> "Alice"; <foo://bla/names#age> 10.
<foo://bla/persons/bob> <foo://bla/names#name> "Bob"; <foo://bla/names#age> 8.
```

[Filename: RDF/john-expanded.n3]

- Note: hierarchically structures URLs begin with "//"
  (the Jena SPARQL tool otherwise complains)

EXPANSION OF PREFIXES

illustrate the equivalence by stating the same query against both N3 “fact bases”:

```
prefix : <foo://bla/names#>    # we need only this for the :name predicate
select ?X ?N
from <file:john.n3>
where {?X :name ?N}
```

[Filename: RDF/john.sparql]

```
prefix : <foo://bla/names#>
select ?X ?N
from <file:john-expanded.n3>
where {?X :name ?N}
```

[Filename: RDF/john-expanded.sparql]

- all positions are subject to expansion.

N3 Prefix Notation: Comments

- empty prefix: define "@prefix : *uri*" and just use ":*qname*"
- additionally, a set of non-empty prefixes can be defined as "@prefix *pre*: *uri*" and use "*pre*:*qname*"
- the results are URIs "below" the URI that is defined for the prefix
- usually, the prefix uris end with "#", or with "/"
  - "#": generates expressions like ("hash-namespace") foo://bla/names#age (which is a property), or foo://bla/names#Country (which is a class), or foo://bla/names#alice (which is an individual).
  - "/": generates expressions like ("slash-namespace") <foo://bla/persons/alice> (which is an individual).
  - without these that results just in a string concatenation.
- note that after *prefix*: only qnames are allowed. Usage with paths like *pre*:$qname_1$/$qname_2$ is not allowed.
- note the similarity with the use of *namespaces* in XML:
  xmlns = *uri* use <*elementname*>
  xmlns:*ns* = *uri* use <*uri*:*elementname*>

MERGING RDF DATA

- easy Web-wide data integration when using agreed *URIs* and *notions*.
- just load several RDF files into one model: same URIs are identified.

```
@prefix : <foo://bla/names#> .
@prefix person: <foo://bla/persons/> .
  person:jack :name "Jack";  :age 65;  :child person:john .
  person:kate :name "Kate";  :child person:john .
```

[Filename: RDF/parents.n3]

- foo://bla/names# is the same URL base as earlier in john.n3:

```
prefix : <foo://bla/names#>
select ?P ?N
from <file:john.n3>
from <file:parents.n3>
where {?X :name ?P . ?X :child ?Y . ?Y :name ?N }
```

[Filename: RDF/parents.sparql]

URIs and Ontologies

- the URIs will usually be organized in an *ontology*.
- note: these URIs can also be URLs that refer to something useful – the RDF+Metadata world is a complex one.
  - later with RDFS and OWL.

4.3 SPARQL: An RDF Query Language

- has already been used above in the examples
- syntactically very similar and equivalent to Datalog queries over triples.
- Conjunctive queries over triples,
- mixed with a bit SQL style syntax.

```
// SPARQL
select ?P ?N
from <url>                  # optional, multiple input graphs
where {?X :name ?P . ?X :child ?Y . ?Y :name ?N }
```

is the same as

```
// DATALOG
?-  name(_X, P), child(_X, _Y), name(_Y, N).
```

- instead `?X`, also `$X` can be written.
- SPARQL's FROM does not correspond to SQL's selection of input relations, but to accessing multiple databases (cf. SchemaSQL)

SPARQL Syntax

SELECT *result variables*

FROM *input*

WHERE { dot-separated sequence of *triple-patterns* and *FILTER expressions* }

- triple-pattern: $x\ y\ z$ as above
- FILTER expression: FILTER (*predicate expression over variables and literals*)
  (see also Functions & Operators in SPARQL)

```
prefix : <foo://bla/names#>
select ?P ?A
from <file:john.n3>
where {?X :name ?P . ?X :age ?A . FILTER ( ?A > 30 ) }
```

[Filename: RDF/age-test.sparql]

- Object lists: x p y_1, y_2
- Predicate lists: x p_1 y_1; p_2 y_2;

SPARQL Formal Semantics

- RDF Terms: ("terms" is a bad terminology; actually these are the constant symbols)
  - IRI: the set of all IRIs (Internationalized Resource Identifiers)
  - RDF-L: the set of all RDF Literals
  - RDF-B: the set of all blank nodes (see later)
  - RDF-T = IRI $\cup$ RDF-L $\cup$ RDF-B
- V: set of variables
- Triple Pattern P: a member of (RDF-T $\cup$ V) $\times$ (IRI $\cup$ V) $\times$ (RDF-T $\cup$ V)
  (allows literals at subject position; fails matching anything)
- Pattern Solution S: substitution from (a subset of) V to RDF-T.
- $S(P)$: replace every $v \in V$ in P by $S(v)$.
- Basic Graph Pattern: a set of triple patterns with *value constraints* [see above query examples]
- *answer substitution* to P wrt. an RDF Graph G: any $S : V_P \rightarrow$ RDF-T such that $S(P) \subseteq G$.
  (modulo blank node renaming – which can be done by using don't care variables in the query)

SPARQL: Querying Metadata

It is totally natural in RDF and SPARQL to query also metadata:

- which properties does John have?

```
prefix : <foo://bla/names#>
prefix person: <foo://bla/persons/>
select ?P
from <file:john.n3>
where { person:john ?P ?Y }
```

[Filename: RDF/john-properties.sparql]

- cf. SchemaSQL and F-Logic (“history” part of the SSD&XML lecture)
- especially, F-Logic (1989) had a strong influence on later languages for semistructured data, knowledge representation and the Semantic Web.

SPARQL: DISTINCT

- which properties does John have, remove duplicates?

```
prefix : <foo://bla/names#>
prefix person: <foo://bla/persons/>
select distinct ?P
from <file:john.n3>
where { person:john ?P ?Y}
```

[Filename: RDF/john-distinct-properties.sparql]

- (currently) correctly removes duplicates but then behaves weird.

SPARQL: OPTIONAL Conjuncts

{ ··· . OPTIONAL { *group pattern* . ··· } }
binds the variables in the inner pattern if it is satisfied

```
prefix : <foo://bla/names#>
SELECT ?N ?A
FROM <file:john.n3>
FROM <file:parents.n3>
WHERE { ?X :name ?N .
        OPTIONAL { ?X :age ?A . FILTER (?A < 60) } }
```

[Filename: RDF/optional-age.sparql]

will always bind ?N, but ?A will be bound only if the age is known, and is less than 60.
If the age of ?X is known, but $\geq$ 60, the OPTIONAL part will simply bind nothing, but it will not evaluate to “false”.

SPARQL: OPTIONAL Conjuncts

Nested OPTIONAL:

```
prefix : <foo://bla/names#>
SELECT ?N ?A ?C
FROM <file:john.n3>
FROM <file:parents.n3>
WHERE { ?X :name ?N .
        OPTIONAL { ?X :age ?A . FILTER (?A < 60)
                   OPTIONAL { ?X :child ?Y . ?Y :name ?C }}}
```

[Filename: RDF/age-and-children.sparql]

Only when an age less than 60 is given, then also the children are optionally given.

- How to express “list ?X if no age is known or if its age is less than 60”?

SPARQL: NEGATION

OPTIONAL together with filtering can be used for checking that a property is *not* satisfied:

```
prefix : <foo://bla/names#>
 SELECT ?N ?A
 FROM <file:john.n3>
 FROM <file:parents.n3>
 WHERE { ?X :name ?N .
         OPTIONAL { ?X :age ?A . FILTER (?A > 60) }
         FILTER (!bound(?A)) }
```

[Filename: RDF/optional-age-negated.sparql]

- optionally binds ?A if an age is given and $>$ 60, and then removes those where ?A is bound.
- people without an age or with age $<$ 60 are returned.
- equivalent to “Negation as Failure” in Prolog.
- equivalent semantics in *defeasible logics*: “people where it is consistent to assume that their age could be $<$ 60”.

Another Example: Closed World Negation

- all countries that are not (known to be) neighbors of Germany
- Closed Predicate “neighbor”: “neighbor does *only* hold for the pairs where it is explicitly known”.

```
prefix  : <ex:pl#>
prefix  rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
prefix  owl: <http://www.w3.org/2002/07/owl#>
prefix  mon: <http://www.semwebtech.org/mondial/10/meta#>

SELECT  ?C1 ?C2
FROM <file:mondial-europe.n3>
WHERE
  { ?C1 a mon:Country . ?C2 a mon:Country
 OPTIONAL
  { ?C1 mon:neighbor ?X . FILTER ( ?X = ?C2 ) }
 FILTER ( ! bound(?X) ) }
```

[Filename: RDF/no-neighbor.sparql]

- The closing of the predicate is restricted to the *query*, but cannot be used in the OWL part.

SPARQL: UNION

- { *subpattern*$_1$ } UNION { *subpattern*$_2$ }
- *subpattern*s can bind different variables (recall that safety in the relational calculus (Slide 84) requires that both disjuncts bind the same variables),

```
@prefix : <foo://bla/names#> .        @prefix person: <foo://bla/persons/> .
 person:paul :name "Paul"; :age 30; :mother person:kate; :father person:jack.
 person:sue :name "Sue"; :age 32; :mother person:kate.
 person:peter :name "Peter"; :age 28; :father person:jack; :child person:andy.
 person:andy :name "Andy"; :age 4.   person:kate :name "Kate".
```

[Filename: RDF/father-mother.n3]

```
prefix : <foo://bla/names#>
select ?N ?M ?MN ?F ?FN
from <file:father-mother.n3>
where { ?X :name ?N .
   { ?X :mother ?M . OPTIONAL { ?M :name ?MN }} UNION
   { ?X :father ?F . OPTIONAL { ?F :name ?FN }}}
```

[Filename: RDF/father-mother.sparql]

SPARQL: Named Graphs

The RDF model can distinguish between stored graphs:

- Default Graph and Named Graphs
- Default Graph specified by FROM <*uri*>
- Named Graphs specified by FROM NAMED <*uri*>.

```
prefix : <foo://bla/names#>
SELECT ?X ?N ?Y ?M
FROM NAMED <file:john.n3>
FROM NAMED <file:father-mother.n3>
WHERE { { GRAPH <file:john.n3> { ?X :name ?N }}
        UNION
        { GRAPH <file:father-mother.n3> { ?Y :name ?M }}}
```

[Filename: RDF/read-graph.sparql]

- Can be used e.g. for loading multiple graphs and constraining properties to the data of only some of them.

SPARQL Result Modifiers

- formal semantics: yields an *answer substitution* $S : V_P \rightarrow$ RDF-T.
- cf. SQL: SELECT *expression*, XQuery: return *expression*
- ORDER BY: SELECT ... WHERE ... ORDER BY *directives*
  where *directives* is a sequence of expressions of the forms
  - *variable*, equivalent ASC(*variable*), and
  - DESC(*variable*)
- Projection: SELECT *variables* FROM ... WHERE ...
- DISTINCT: SELECT DISTINCT *variables*
- LIMIT n: only a given number of results
- OFFSET k: start with the k-th solution
- Syntax: SELECT ... WHERE ... [ORDER BY ...] LIMIT n OFFSET k.
  (note that combination of ORDER BY, LIMIT and OFFSET can be used for e.g. returning the 3rd to 6th largest items etc.)

SPARQL Graph Constructors

- use CONSTRUCT { *dot-separated sequence of triple patterns* }
  instead of simple SELECT
- returns an RDF Graph instead of variable bindings
- result triples that contain unbound variables (e.g. due to an OPTIONAL) are not added to the graph

```
prefix : <foo://bla/names#>
construct { ?G :grandchild ?X . ?X :name ?N}
from <file:john.n3>
from <file:parents.n3>
where {?G :child ?P . ?P :child ?X . ?X :name ?N}
```

[Filename: RDF/construct.sparql]

- just to test RDF/XML (see later for details):

jena -q -qf construct.sparql -ol RDF/XML -of grandchildren.rdf

SHORTCOMINGS

Some can be solved by extended clause syntax

- No explicit negation, only via Filter
- no grouping/aggregation

Some are inherent to the language design

- SPARQL is not a closed language: input model (RDF) different from result model (sets of variable bindings).

$\Rightarrow$ no nested queries.

SPARQL as a Closed Language?

... would look like this:

```
prefix : <foo://bla/names#>
select ?X
from
 { construct { ?G :grandchild ?X . ?X :name ?N}
   from <file:john.n3>
   from <file:parents.n3>
   where {?G :child ?P . ?P :child ?X . ?X :name ?N}
 }
where { ?X ?P ?Y }
```

[Filename: RDF/construct-nested.sparql]

- Note: this is not yet possible in SPARQL
- Possible Master's Thesis: extend the JENA SPARQL Module accordingly.

COMPARISON WITH MULTIDATABASES

- Multidatabases in the “old” times: a set of autonomous databases that provide correlated contents
- cf. the section on SchemaSQL in the SSD&XML lecture:
  SELECT ... FROM db_1::rel_1, db_2::rel_2, ...
  (and similar expressions)
- the SPARQL FROM clause also selects multiple input RDF data sources

Current Situation

Seeing the RDF Web as a multidatabase, RDF and SPARQL provide a unified model and language for data integration (via the URIs).

Semantic Web Vision

The actual origin of the RDF data is *not* specified by the user: users query “the Semantic Web” (via a portal) and relevant data sources will be selected transparently.

4.4 N3/RDF vs. FOL

- So far, RDF/N3 is really just a restricted variant of ground (sorted) First-Order logic atoms:
  - only binary predicates
  - literal values and relationships carry semantics
  - URIs as constant symbols
  - no function symbols other than constants
- Sorted domain: URIs and Literals

Existential Knowledge

Sometimes the existence of things is relevant, without actually naming and giving an ID to an object.

- “A parent is a person that has at least one child”
- “John has a child of 12 years”

BLANK NODES – EXISTENTIAL QUANTIFICATION

- Short form for x p y_1 ., y_1 q_1 z_1 . and x p y_2 ., y_2 q_2 z_2 . when the URIs for y_1 and y_2 are not relevant (purely existential "blank nodes"):
  x p [q_1 z_1], [q_2 z_2] .

  <#john><#name> "John"; <#age> 35 ;

  <#child> [<#name> "Alice"; <#age> 10] , [<#name> "Bob"; <#age> 8] , [<#age> 12] .

- note that also #john is just an identifier that contains no actual information (we could also have chosen #pers123 instead). It can also be replaced by using a blank node:

  [<#name> "John"; <#age> 35;

  <#child> [<#name> "Alice"; <#age> 10] , [<#name> "Bob"; <#age> 8] , [<#age> 12]] .

RDF "Model"

Equivalent expression in FOL:

$$\exists j : (\text{name}(j, \text{“John”}) \wedge \text{age}(j, 35) \wedge$$
$$\wedge \exists a, b, c : (\text{child}(j, a) \wedge \text{child}(j, b) \wedge \text{child}(j, c) \wedge$$
$$\wedge \text{name}(a, \text{“Alice”}) \wedge \text{age}(a, 10) \wedge \text{name}(b, \text{“Bob”}) \wedge \text{age}(b, 8) \wedge \text{age}(c, 12)))$$

Example: Blank Nodes

```
@prefix : <foo://bla/names#>.
 [ :name "John";  :age 35;
   :child [:name "Alice"; :age 10] ,
                [:name "Bob"; :age 8] , [:age 12 ]] .
```

[Filename: RDF/john-blank.n3]

```
prefix : <foo://bla/names#>
select ?X ?N
from <file:john-blank.n3>
where {?X :name ?N ; :child [:age 12] }
```

[Filename: RDF/john-blank.sparql]

- generated by “[...]” above: “something that satisfies”
- nested “term” syntax for implicit conjunction of atoms, without using identifiers or key references.
  [Note that F-Logic terms have a very similar structure]

Example:

Extend the database such that John is married to Mary, who is the mother of both children.

Named Blank Nodes

- blank nodes can also be named by local identifiers of the form "_:*localname*" that allow for "referencing" things again (existential variables).

```
@prefix : <foo://bla/names#>.
  [ :name "John";  :age 35;
    :child _:a, _:b ;
    :married [ :name "Mary";  :child  _:a, _:b ] ] .
  _:a :name "Alice"; :age 10 .
  _:b :name "Bob"; :age 8 .
```

[Filename: RDF/john-married.n3]

```
prefix : <foo://bla/names#>
select ?N ?C ?A
from <file:john-married.n3>
where {?X :child ?C . ?X :name ?N . ?C :age ?A }
```

[Filename: RDF/john-married.sparql]

PITFALLS OF EXISTENTIAL KNOWLEDGE

Consider again John's family and its logical formalization (Slides 181 and 180).

```
@prefix : <foo://bla/names#>.
 [ :name "John";  :age 35;
   :child [:name "Alice"; :age 10] ,
                [:name "Bob"; :age 8] , [:age 12 ]] .
```

[Filename: RDF/john-blank.n3]

How many children des John have?

- what does a human reader understand?
- is this entailed by the logical formalization?

Pitfalls of Existential Knowledge (Cont'd)

- Alice:

```
[ :name "John"; :child [:name "Alice"]]
```

- His health insurance:

```
[ :name "John"; :child [:age 10]]
```

- Grandmother:

```
[ :name "John"; :child [:nickname "My sunshine"] ]
```

- A neighbor:

```
[ :name "John"; :child [:nickname "The little beast"] ]
```

  How many children does John have?

  – merging this yields one RDF graph.

  – the *logical formalization* is very similar to the one from the previous slide.

Pitfalls of Existential Knowledge (Cont'd)

- from the first example, most people conclude that John has three children.
  (note that "at least three" would be more exact)
- from the second example (text) many people conclude that all statements describe Alice
- the RDF graph of the second example, most people conclude that there are four children
- formally in all cases: the knowledge bases entail only that John has at least one child!
- for the first example:
  - there is a logical model where there is only one child with three names and a property "age" that has three values.
  - people use meta knowledge that age and name are functional properties. This is not contained in the RDF data!
- OWL allows to express such additional information.
- the next slides anticipate some OWL stuff that will be discussed in detail later.

Pitfalls of Existential Knowledge (Cont'd)

Show that it is consistent to assume that John has exactly one child:

- define a class oneChildParent that restricts the cardinality of child to one and make John an instance of it (using OWL; see later):

```
@prefix : <foo://bla/names#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
 [ :name "John";  :age 35;
   :child [:name "Alice"; :age 10], [:name "Bob"; :age 8], [:age 12 ];
   rdf:type :oneChildParent].
:oneChildParent owl:equivalentClass [rdf:type owl:Restriction;
  owl:onProperty :child; owl:cardinality 1].
```

[Filename: RDF/john-silly-example.n3]

- how old is Alice (invoke with -inf -r pellet)?

```
prefix : <foo://bla/names#>
select ?X ?A  from <file:john-silly-example.n3>
where {?X :name "Alice" . ?X :age ?A}
```

[Filename: RDF/john-silly-example.sparql]

Pitfalls of Existential Knowledge (Cont'd)

- Assert that age is a functional property

```
@prefix : <foo://bla/names#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
 [ :name "John";  :age 35;
   :child [:name "Alice"; :age 10], [:name "Bob"; :age 8], [:age 12 ];
   rdf:type :oneChildParent].
:oneChildParent owl:equivalentClass [rdf:type owl:Restriction;
  owl:onProperty :child; owl:cardinality 1].
:age rdf:type owl:FunctionalProperty.
```

[Filename: RDF/john-silly-example-2.n3]

```
prefix : <foo://bla/names#>
select ?X ?A  from <file:john-silly-example-2.n3>
where {?X :name "Alice" . ?X :age ?A}
```

[Filename: RDF/john-silly-example-2.sparql]

- pellet complains about an inconsistent ontology.

Aside: Reasoning under Inconcistency

Consider the situation if the previous slide.

- Semantics of boolean implication:
  Any formula of the form false $\rightarrow \varphi$ is true in any interpretation (equivalent to $\neg$false $\vee \varphi$, which is in turn equivalent to true $\vee \varphi$.
- Given an inconsistent ontology formalization Φ, for *every* formula φ, $\Phi \models \varphi$ holds:
  for every model $\mathcal{M}$ such that $\mathcal{M} \models \Phi$ (note that there are no such $\mathcal{M}$), also $\mathcal{M} \models \varphi$ holds.
- Tableau Calculus: Given an ontology formalization Φ,
  prove $\Phi \models \varphi$ by starting a tableau over $\Phi \wedge \neg\varphi$ and trying to close it.
  If already Φ is inconsistent, any tableau over Φ can be closed without using $\neg\varphi$. Thus,
  $\Phi \vdash_{\text{Tableau}} \varphi$.

For that reason, it is reasonable that a prover even does not start working with an inconsistent ontology

- research issue: reduce an inconsistent ontology to a conistent subset and prove or refute φ from this.

Pitfalls of Existential Knowledge (Cont'd)

- Assert that age is a functional property and look what is entailed ...

```
@prefix : <foo://bla/names#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
 [ :name "John";  :age 35;
   :child [:name "Alice"; :age 10], [:name "Bob"; :age 8], [:age 12 ]].
:oneChildParent  owl:equivalentClass [rdf:type owl:Restriction;
  owl:onProperty :child; owl:cardinality 1].
:twoChildrenParent  owl:equivalentClass [rdf:type owl:Restriction;
  owl:onProperty :child; owl:cardinality 2].
:threeChildrenParent  owl:equivalentClass [rdf:type owl:Restriction;
  owl:onProperty :child; owl:minCardinality 3].
:age rdf:type owl:FunctionalProperty.
```

[Filename: RDF/john-three-children.n3]

(continue next slide)

PITFALLS OF EXISTENTIAL KNOWLEDGE (CONT'D)

```
prefix : <foo://bla/names#>
prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
select ?X  from <file:john-three-children.n3>
where { ?X :name "John" . ?X rdf:type :threeChildrenParent }
```

[Filename: RDF/john-three-children.sparql]

- Note: replacing the "assumption" threeChildrenParent by ?C does not yield the intended result (incomplete reasoning)

UNIQUE NAME ASSUMPTION (DATALOG)

Datalog makes the *Unique Name Assumption*: different constants denote different things, e.g.

$$\text{sibling}(X, Y) \text{ :- } \text{child}(\_Z, X),\ \text{child}(\_Z, Y),\ X \neq Y.$$

will derive sibling(alice,bob) when facts child(john,alice) and child(john,bob) are given.

NO UNIQUE NAME ASSUMPTION IN RDF

- RDF model theory: objects described by different URIs may be the same!
- even if the children are explicitly described by URIs, the consequences are the same as in the previous example,
- in OWL, it can be explicitly stated that two URIs denote the same or different things.
- equivalence: e.g. used in ontology mapping.
- if no such information is explicitly given, both is assumed to be possible.
- equality or non-equality can be derived (e.g., by declaring a property to be functional) (Note that the same is the case for F-Logic model theory with functional methods.)

ASIDE: SOME THEORY

Consider the N3 fragment consisting of expressions as follows:

- no explicit subjects,
- no URIs at object positions (only literals and [...] expressions),
- no named blank nodes,
- i.e., constants are only literals and property name URIs.

Expressiveness:

- Only existentially quantified variables.
- can only represent tree-like relationship structures. No cycles (cf. the “mary” example).
- The expressiveness is exactly that of FOL formulas over an alphabet of only two variables. (note that these can be used several times)

Exercise: express the RDF/N3 “database” given on Slide 180 by a first-order fomula that uses only two variables.

SOME THEORY (CONT'D)

- FOL with two variables is decidable,
- satisfiability of first-order formulas that use three variables is in general undecidable ("3-SAT")!
- restriction to two variables: guarantees a tree-like "structure" of the models ("tree model property"). This locality guarantees decidability.
- for the same reason, also many variants of Description Logics (see later) are decidable. Proofs use the equivalence to modal logics over tree-like Kripke Structures.

This RDF fragment will be relevant later when considering decidable fragments of Description Logics and OWL.

SEMANTICS: NEGATION

- compare the FOL axiomatizations of John and his family – they always describe only the things that are there, and say nothing about the (non)-existence of other things:
  Has John a child of the age of 14?

  The database does not find any, but the formula

  $$\exists j : (... \wedge \exists d : \mathsf{child}(j, d) \wedge \mathsf{age}(d, 14))$$

  is consistent – thus it has models. It is actually possible that John has such a child which is simply unknown to the database ("Open-World-Assumption" (OWA)).
- another RDF Web source could add RDF triples that describe additional children of John.
- Minimal-Model-based reasoning or SQL databases would answer "no"
  "Closed-World-Assumption" (CWA).
- SPARQL has no negation except the restricted value-based negation shown on Slide 170.
- in general: OWA is more appropriate to the Web.
- so far: SPARQL does only matching on an RDF graph. Later: RDF/RDFS/OWL *model* that extends the graph by further knowledge that is also presented in RDF format.

4.5 RDF Conceptual Model and Built-in Predicates

... so far, RDF defines just an edge-labeled graph.

RDF provides also a simple type system (which is exteded later by RDF Schema):

- the RDF conceptual model knows about class/type membership and properties,
- rdf:type: is a special property that assigns a type (means: class membership) to something

Types and Properties as Resources

Extend the “Persons” running example from Slide 158:

```
@prefix : <foo://bla/names#> .
@prefix person: <foo://bla/persons/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
  person:john rdf:type :Person; :name "John";  :age 35;
        :child person:alice, person:bob.
  person:alice :name "Alice"; :age 10.
  person:bob :name "Bob"; :age 8.
```

[Filename: RDF/john.n3]

- <foo://bla/persons/john> (and others) are the URIs of objects that are described by the database,
- <foo://bla/names#Person> is the URI of the class “Person”
- <foo://bla/names#name> and <foo://bla/names#age> are the URIs of the property names.

Design *Ontologies* in RDF like this.

ONTOLOGY DESIGN

- separate part for “notions” (often as a “#”-namespace)
- separate part for objects (either as “#”-namespace or as “/”-namespace)
- note that after *prefix*: only qnames are allowed. Usage with paths like *pre*:$qname_1$/$qname_2$ is not allowed.
- [later in RDF/XML xml:base can be used together with paths]

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix : <foo://bla/terms#> .
@prefix persons: <foo://bla/persons/> .
@prefix cats: <foo://bla/cats/> .
 persons:john rdf:type :Person; :name "John"; :age 35;
    :has_cat cats:garfield.
 cats:garfield rdf:type :Cat; :name "Garfield"; :age 3.
```

[Filename: RDF/john-and-the-cat.n3]

- There are <foo://bla/persons/john> and <foo://bla/cats/garfield>.

Ontology Design and Query Formulation

```
prefix terms: <foo://bla/terms#>
prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
select ?X ?T ?A
from <file:john-and-the-cat.n3>
where { ?X rdf:type ?T . ?X terms:age ?A }
```

[Filename: RDF/john-and-the-cat.sparql]

- note that most queries only need the prefix for the metadata
- the prefix/base for data is only used when a query explicitly accesses an object by its URI

```
base <foo://bla/>
prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
select ?X ?T ?A
from <file:john-and-the-cat.n3>
where { <persons/john> rdf:type ?T . ?X <terms#age> ?A }
```

[Filename: RDF/sparql-base.sparql]

- Alternative: value-based selection of an object

```
{ ?X [ :name "John" ; :age ?A ] }
```

METADATA AS RESOURCES

... on the way to more expressive ontology languages:

- Some metadata notions are defined in the rdf namespace
  - @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
    @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
  - rdf:type as a property: `:john rdf:type :Person`

    (short syntax in N3: `:john a :Person`)
  - rdf:Property as a class: `:child rdf:type rdf:Property`
  - rdfs:Class (and owl:Class) as a class: `:Person rdf:type rdfs:Class`
- transition/not-so-well-defined language border to RDF Schema (RDFS)
- RDF tools know about rdf:type, rdf:Property, rdfs:Class etc.

note: rdf:type non-capitalized (is a property), rdf:Property (is a type) capitalized.

TYPES AND PROPERTIES

- these things are built-in notions that have in pure RDF still no meaning.
- Types are *not* XML Schema complex types (structural typing), but *semantical types*.
- Types/classes and properties are also resources that can be described this way.
- nevertheless, in pure RDF there is no kind of signature or schema; especially also no type constraints on properties.
- it's just data.

RDF Instance Data and Metadata Examples

- Consider
  reasonable: (http://.../geo#germany rdf:type http://.../geo#Country)
  reasonable: (http://.../geo#berlin rdf:type http://.../geo#City)
  non-reasonable: (http://.../geo#germany rdf:type http://.../geo#berlin)
  $\Rightarrow$ things that are of some type should not be types themselves?
- but this crashes with meta-metadata:
  reasonable: (http://.../bio#Penguin rdf:type http://.../bio#Species)
  reasonable: (http://.../bio#tweety rdf:type http://.../bio#Penguin)
- Consider
  reasonable: (http://...#has_capital rdf:type rdf:Property)
  not reasonable: (http://...#has_capital http://...#has_population 42)
  $\Rightarrow$ a property does not have other properties?
- but this crashes with meta-metadata:
  reasonable: (http://.../geo#has_capital rdf:type rdf:Property)
  reasonable: (http://.../geo#has_capital http://.../meta#has_range http://.../geo#City)

$\Rightarrow$ these "legal" and "non-legal" usages are not enforced by anything. Non-legal usage can lead to severe problems (see Slide 213).

4.6 Example: Mondial in RDF

- RDF level: structure of URIs, types and relationships
- later: RDFS and OWL add further metadata information and knowledge

Design of the Mondial Ontology

- we chose classes and properties of the Mondial ontology to be identified by

  http://www.semwebtech.org/mondial/10/meta#*classname*

- the URIs for identifying countries, organizations etc. are actual URLs formed like

  http://www.semwebtech.org/mondial/10/countries/*code*
  http://www.semwebtech.org/mondial/10/countries/*code*/cities/*name*

- Alternatives will be discussed later (with RDF/XML).

Example Fragment

```
@prefix monmeta: <http://www.semwebtech.org/mondial/10/meta#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
<http://www.semwebtech.org/mondial/10/countries/D>
  rdf:type monmeta:Country ;
  monmeta:name "Germany" ;
  monmeta:code "D" ;
  monmeta:population 83536115 ;
  monmeta:capital
    <http://www.semwebtech.org/mondial/10/countries/D/provinces/Berlin/cities/Berlin>.
```

[Filename: RDF/a-bit-mondial.n3]

- Mondial N3 available on the Web site
  (note: not yet stable ... still experimenting)
- the type “Country” is <http://www.semwebtech.org/mondial/10/meta#Country>
- the property “capital” is <http://www.semwebtech.org/mondial/10/meta#capital>
- it would be nice to be able to set @base <http://www.semwebtech.org/mondial/10/> for the individuals’ URIs, but this is not supported in N3; see later for RDF/XML.

```
base <http://www.semwebtech.org/mondial/10/>
prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
select ?Y
from <file:a-bit-mondial.n3>
where { <countries/D> rdf:type ?Y }
```

[Filename: RDF/a-bit-mondial.sparql]

Relationship with XML Namespaces

Consider the following XML fragment:

```
<mon:mondial xmlns:mon="http://www.semwebtech.org/mondial/10/meta#">
  <mon:Country car_code= "D">
    <mon:name>Germany<mon:name>
  </mon:Country>
</mon:mondial>
```

There's more behind these namespaces, which is not used in plain XML.

See later with RDF/XML.

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.

4.7 RDF Model Theory vs. FOL

- FOL: Recall from Slide 42: 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!

$\Rightarrow$ A model theory for RDF is likely to be different from that for FOL.

- further reading:
  "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)

General Considerations

Like in Datalog, use a Herbrand-Style interpretation:

- The alphabet consists of IRIs, Blank node identifiers RDF-B, and Literals RDF-L,
- the domain $\mathcal{D}$ consists of the same things,
- there are no function symbols,
- ... and only the interpretation of predicate symbols (i.e., class names and property names) is flexible.

Assume the domain to be partitioned in two sets:

- objects: Let IRI$_{obj}$ and RDF-B$_{obj}$ denote all IRIs and blank nodes that denote objects. Literals also belong to that partition.
- predicates: classes and properties. Denoted by IRI$_c$ and IRI$_p$, not containing the special property rdf:type and the special class rdf:Property.
  - only when defining derived classes, there will be blank nodes that represent classes
  - function symbols would also belong here, but these don't exist in RDF.

MAPPING TO SORTED FIRST-ORDER LOGIC

The domain is partitioned into (disjoint) *sorts*.

- $\mathcal{D} = \mathcal{O} \cup \mathcal{C} \cup \mathcal{P}$ (objects and literals, classes and predicates)
- the signature of predicates is expressed in terms of the sorts.
- rdf:type as "isa":
  $\mathcal{I}(\text{isa}) \subseteq (\text{IRI}_o \cup \text{RDF-B}_o \cup \text{RDF-L}) \times (\text{IRI}_c)$
  E.g. isa(<foo://bla/persons/john>, <foo://bla/names#Person>)
- all properties as "holds":
  $\mathcal{I}(\text{holds}) \subseteq ((\text{IRI}_o \cup \text{RDF-B}_o) \times \text{IRI}_p \times (\text{IRI}_o \cup \text{RDF-B}_o \cup \text{RDF-L}))$
  E.g. holds(<foo://bla/persons/john>, <foo://bla/names#married>, <foo://bla/persons/mary>)
- each quantifier in a formula ranges over *one* of the sorts.
- ignoring the sorts leads to plain first-order logic.

With this requirement, ontology reasoning using quantification over classes or predicates stays within FOL. But, so far there is no way to express metadata information (the first component of "holds" is restricted to individuals).

SECOND-ORDER LOGIC

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

- first-order objects: 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: predicates (and functions).
  For RDF, the predicates are classes IRI_c and properties IRI_p.
- predicates are interpreted by relationships over the object domain.
- $\Rightarrow$ no statements *about* predicates.
- Formulas may use quantification over predicate symbols.
  The induction axiom is second-order (simplified):

$$\forall pred : (pred(0) \wedge \forall n(pred(n) \rightarrow pred(n+1)) \rightarrow (\forall n : pred(n))$$

Higher-Order Logic

- predicates of order n are interpreted over the domain of order $n-1$.

A Second-Order Logic Style Mapping

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/names#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/names#child>(<foo://bla/persons/john>, <foo://bla/persons/alice>)
- rdf:type is represented by the set IRI_c,
- the class rdf:Property is represented by the set IRI_p.

This encoding is “syntactically second order”, but actually covers only first-order semantics.

ASIDE: 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))

Restrictions of these mappings

- As long as the objects and the classes and properties are disjoint, i.e., there are no statements about classes or properties, both mappings are sufficient.
- Definition of subclasses and subproperties leads to a *specific* domain logic with built-in second-order axioms.

Overcoming this Restriction: 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!

4.8 Further Topics

- some additional RDF syntax: Reification and Collections. Later.
- Schema information: RDF Schema
- 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?