Chapter 4
RDF: Resource Description Framework

Recall (cf. Slide 37):

The 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")
  - descriptions about classes and properties allow to draw conclusions for their
    instances.

- 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.

⇒ 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.
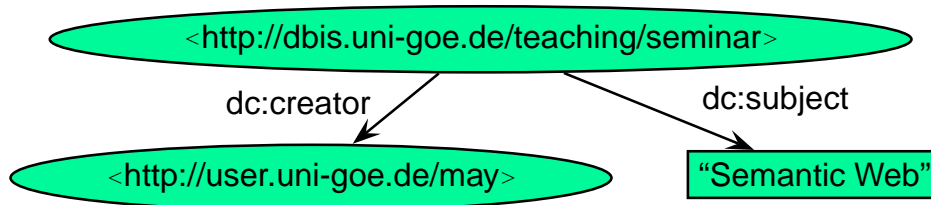
Model: RDF Graph + derived knowledge

- 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" Unicode 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 ...

```
         ⟨http://dbis.uni-goe.de/teaching/seminar⟩
          dc:creator                    dc:subject
     ⟨http://user.uni-goe.de/may⟩         "Semantic Web"
```

81

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/⟩

82

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 annote 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)

⇒ 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 (URI)*, 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.

- 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!

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
    (this becomes more concrete with "Linked Open Data" (since $\sim$2010))

  – defining a URI that is completely different from the file's URL

- all members of a "community" can describe the resources in RDF.

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.

  - similarity with *vertically partitioned* storage of relational databases (cf. Slide 37): each predicate is stored in a binary relation.

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" $\boxed{subj\ pred\ obj\ .}$

  (Alternatives: $\boxed{subj\ \text{has}\ pred\ obj\ .}$ or $\boxed{obj\ \text{is}\ pred\ \text{of}\ subj\ .}$)

- URIs: as $<uri>$

- Literals may occur as objects: as numbers (e.g. 42) or strings (e.g., "John Doe").

- Short form for $\boxed{x\ p\ y_1\ .}$ and $\boxed{x\ p\ y_2\ .}$: $\boxed{x\ p\ y_1,\ y_2.}$

  $<\#john>\ <\#child>\ <\#alice>,\ <\#bob>\ .$

- Short form for $\boxed{x\ p_1\ x_1\ .}$ and $\boxed{x\ p_2\ x_2\ .}$: $\boxed{x\ p_1\ y_1;\ p_2\ y_2.}$

  $<\#alice>\ <\#age>\ 10;\ <\#name>\ \text{"Alice"}\ .$

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 exponent/mantissa).

- full syntax "*string*"^^*datatype-url* ,

  e.g. "42"^^<http://www.w3.org/2001/XMLSchema#int>
  (note: the "..." must also be present for numeric datatypes)

- 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

- simplest "full" form: <schema:name>; typical for examples:

```
<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: LOCAL IDENTIFIERS

- if only a local part of the URI is given, it is by default extended with the document URL
  (yielding a *local* uri):

```
<#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: LOCAL IDENTIFIERS IN QUERIES

- files with local URIs can be queried via HTTP

- 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]

- change the default expansion: 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]

- result e.g,
  X/<http://dbis.informatik.uni-goettingen.de/Teaching/SemWeb/RDF/john-local.n3#bob>,
  Y/"Bob"

- Note: "base" is used in the application (here: SPARQL),
  not in N3 (although, RDF/XML has xml:base!)

LOCAL URIs: ADDING STATEMENTS TO "REMOTE" SCOPES

Even the use of *local* URIs cannot prevent others (by other RDF files) from adding statements 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 ?A
from <http://dbis.informatik.uni-goettingen.de/Teaching/SemWeb/RDF/john-local.n3>
from <file:mary-remote.n3>
where {?X <#married> ?Y . ?Y <#age> ?A}
```
[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.

⇒ "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/10/> could be a base URI for defining the notions used in Mondial
    - entity types (Country, City) and property names (has-capital) as e.g.
      .../10/meta#Country
    - recall that entity instances are also resources that have a URI in this scope:
      e.g. .../10/countries/D/cities/Berlin
    - 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 family: <foo://bla/persons/> .
  family:john :name "John";  :age 35;  :child family:alice, family:bob.
  family:alice :name "Alice"; :age 10 .
  family: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 structured 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 family: <foo://bla/persons/> .
  family:jack :name "Jack";  :age 65;  :child family:john .
  family:kate :name "Kate";  :child family: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]

- 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

- syntactically very similar and equivalent to Datalog queries over triples.

- Conjunctive queries over triples,

- variables with multiple occurrences act as *join variables*,

- 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)

BASIC 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: BASICS OF 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 ∪ RDF-L ∪ RDF-B

- Var: set of variables

- *Triple Pattern P*: an element of (RDF-T ∪ Var) × (IRI ∪ Var) × (RDF-T ∪ Var)
  (this allows literals at subject position which does not match anything)

- *Basic Graph Pattern*: a set of triple patterns.

... consider again BGPs over RDF-Terms and variables:

- a *mapping* μ from Var to RDF-T is a *partial* function $\mu : \text{Var} \to \text{RDF-T}$.
  (also called "(answer) substitutions" or "tuples of variable bindings"; cf. Datalog)
  for (?X mon:capital ?Y), e.g. $\mu_1 = \{X \mapsto \text{<germany>}, Y \mapsto \text{<berlin>}\}$

- $\text{dom}(\mu) := \{v \in \text{Var} : \mu(v) \text{ is defined}\}$

- for a BGP P, let $\text{var}(P) \subseteq \text{Var}$ denote the set of variables that occur in P.

- for a BGP P and a mapping μ s.t. $\text{dom}(\mu) \supseteq \text{var}(P)$, $\mu(P)$ denotes the replacement of every $v \in \text{Var}$ in P by $\mu(v)$.
  - $\mu(P)$ is a (ground) instance of P, i.e. a small RDF graph;
  - standard logic terminology: here the mapping is seen as a substitution (i.e., a syntactic mapping) that yields a ground instance of an expression.

- for a BGP P and an RDF graph G, an *answer substitution* is any substitution $\mu : \text{var}(P) \to \text{RDF-T}$ such that $\mu(P) \subseteq G$
  (modulo blank node renaming, which can be done by using don't care variables in the query).

- The answer set to a BGP wrt. a graph G is the set of all answer substitutions.

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 family: <foo://bla/persons/>
select ?P
from <file:john.n3>
where { family: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 family: <foo://bla/persons/>
select distinct ?P
from <file:john.n3>
where { family:john ?P ?Y}
```
[Filename: RDF/john-distinct-properties.sparql]

• correctly removes duplicates.

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 } }
```
[Filename: RDF/optional-age.sparql]

will always bind ?N, but ?A will be bound only if the age is known.

SPARQL: OPTIONAL (Cont'd)

- note: Filter inside OPTIONAL have local scope:

```
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-2.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 ≥ 60, the OPTIONAL part will simply bind nothing, but it will not evaluate to "false".

SPARQL: OPTIONAL (Cont'd)

Example

OPTIONAL is bound all-or-nothing:

```
prefix : <foo://bla/names#>
SELECT ?N1 ?N2 ?A1 ?A2
FROM <file:john.n3>
FROM <file:parents.n3>
WHERE { ?X1 :name ?N1 . ?X2 :name ?N2 . FILTER (?X1 != ?X2)
        OPTIONAL { ?X1 :age ?A1 . ?X2 :age ?A2}}
```

[Filename: RDF/age-pairs.sparql]

The result bindings contain the ages only if for *both* persons, the age is given.

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 returned.

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

SPARQL FILTER CONDITIONS

- atomic comparisons ($=, <, \leq, >, \geq$) with variables and constants as usual, string, date etc. predicates (see manual),

- logical connectives NOT ("!"), AND ("&&"), OR ("||") as usual,

- bound(?X) ... evaluates to true if ?X is bound in an answer,

- special predicates isIRI(?X), isLiteral(?X), etc. (see manual)

Semantics of Predicates on unbound variables

- (?X=?Y) when one of them is not bound: unknown (W3C Doc: error)

```
prefix mon: <http://www.semwebtech.org/mondial/10/meta#>
select ?N ?P
from <file:mondial-europe.n3>
where { {?C a mon:City; mon:name ?N OPTIONAL { ?C mon:population ?P } }
        # FILTER ((?P > 10000))   # try this also
        FILTER (!(?P > 10000)) }
```

[Filename: RDF/negatedcomparison.sparql]

- returns only those where population is known and < 10000.

SPARQL: NEGATION

- !bound(?X) is the only form of negation in SPARQL.

- 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]
```

- OPTIONAL binds ?A if an age is given and > 60, and then FILTER removes those where ?A is bound.

- people without an age or with age < 60 are returned.

- equivalent to "Negation as Failure" in Logic Programming.

- 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 each other

- Close the predicate "neighbor" for the query: "neighbor does *only* hold for the pairs where it is explicitly known".

```
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-neighbor1.sparql]

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

- all countries that are not (known to be) neighbors of each other.

- closed predicate "neighbor": "neighbor does *only* hold for the pairs where it is explicitly known".

```
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 ?C2 . ?C1 mon:name ?N }
 FILTER ( ! bound(?N) ) }
```
[Filename: RDF/no-neighbor2.sparql]

- note: the additional property used in the OPTIONAL must be present for every ?C1 (and should be easy to compute)

117

SPARQL: UNION

- { *subpattern*$_1$ } UNION { *subpattern*$_2$ }

- *subpattern*s can bind different variables (recall that safety in the relational calculus requires that both disjuncts bind the same variables):

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

- Exercise: if both parents are given, put both into the same answer.

118

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]

SPARQL: Named Graphs

- which individuals are listed with properties in both graphs?

```
prefix : <foo://bla/names#>
SELECT ?X ?P1 ?P2
FROM NAMED <file:parents.n3>
FROM NAMED <file:father-mother.n3>
WHERE { { GRAPH <file:parents.n3> { ?X ?P1 ?Y1 }}
          .
        { GRAPH <file:father-mother.n3> { ?X ?P2 ?Y2 }}}
```
[Filename: RDF/both-graphs.sparql]

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

- all cities that are the capital of a country or the headquarter of an organization, of both. Give also the respectove country and/or organization

```
prefix mon: <http://www.semwebtech.org/mondial/10/meta#>
select ?XN ?CC ?A
from <file:mondial-europe.n3>
where { ?X a mon:City; mon:name ?XN
        OPTIONAL { ?C a mon:Country ; mon:carCode ?CC; mon:capital ?X }
         OPTIONAL { ?O mon:hasHeadq ?X; mon:abbrev ?A }
        filter (bound(?C) || bound(?O)) }
```
[Filename: RDF/disjunction.sparql]

SPARQL RESULT MODIFIERS

- formal semantics: yields a set of *answer substitutions* $\mu : \text{var}(P) \to$ 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.)

VARIABLE ALIASING (E.G. FOR ARITHMETIC EXPRESSIONS)

- similar as in SQL; but always in parentheses:
  (*expression* AS *variable*)

```
prefix  mon: <http://www.semwebtech.org/mondial/10/meta#>


SELECT  ?C (?P/?A AS ?Density)
FROM <file:mondial-europe.n3>
WHERE
  { ?C a mon:Country; mon:population ?P; mon:area ?A }
ORDER BY DESC(?Density)
```
[Filename: RDF/density.sparql]

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 difference/negation operation, only via filter and !bound(X),

- 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.

- 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 SPARQL: Formal Semantics

SPARQL is based on graph pattern matching:

- input: a graph G,

- query: consists of
    - Basic Graph Patterns (where clause) – conjunctive query (join)
    - Filters (where clause)
    - Optional (outer join)
    - Union

  result modifiers:
    - order by (evaluated after query evaluation)
    - projection (in the select clause)
    - duplicate elimination (in the select clause)

- Similarity with the relational model and a similar algebraic characterization.

- Note that similar to SQL, actual evaluation does not use the algebra directly, but applies optimized algorithms).

Formal Semantics – Considerations

- strong similarity with relational model

- algebraic, compositional semantics desirable (cf. SQL, OQL, XQuery)

- SPARQL W3C Candidate Recommendation until v.20061004 gave a non-algebraic semantics informally in "natural language" – which is problematic for some cases.

- Strongly Recommended: Jorge Pérez, Marcelo Arenas, Claudio Gutierrez: Semantics and Complexity of SPARQL. International Semantic Web Conference 2006: 30-43 (use `http://www.dblp.org`) [PAG06]

  - develops an algebraic, compositional semantics,

  - gives a formalization of the W3C Semantics, compares both semantics.

- optional, more complex: Renzo Angles, Claudio Gutierrez: The Expressive Power of SPARQL. International Semantic Web Conference 2008: 114-129 [AG08]

- optional, lots of proofs: Jorge Pérez, Marcelo Arenas, Claudio Gutierrez: Semantics and Complexity of SPARQL. ACM Trans. on Database Systems, 34(3):16, 45 pages; 2009.

- title says it all: Jorge Pérez: Semantic Web Research Inspired by W3C Standards, or the Hell of the Practice without Theory. AMW 2012: 19 (tutorial available at `http://users.dcc.uchile.cl/~jperez/talks/amw2012-tutorial.pdf`)

Formal Semantics – Comparison with SQL

- both are basically conjunctive queries (CQs) – joins;

- SQL: base relations and views: schema: $(A_1, \ldots, A_n)$ – attributes are (usually) DB column names;

- SPARQL: tuples of variable bindings (cf. Datalog: matching variables with patterns): $(X_1, \ldots, X_n)$;

- FILTER is a selection;

- select clause (SPARQL: only finally) applies a projection;

- SPARQL: no nesting of subqueries (but nesting of {. . .}-groups with CQs, OPTIONAL and FILTER);

- OPTIONAL is an outer join (with was a rather rarely used operator in SQL)
  $\Rightarrow$ introduces lots of NULL values
  (which are much more frequent in the scenario of incomplete knowledge in the Web than in SQL)

- details of NULL values have only been sketched (pragmatically) in the DB lecture (and in the SQL lab)!

NULL VALUES IN SQL

- NULLs do not violate any integrity constraint, e.g.,

  ```
  CREATE TABLE Country
        ( ..., population NUMBER CHECK (population >= 0), ...)
  ```

- note that this also holds for referential integrity constraints! e.g.

  ```
  INSERT INTO located VALUES('Berlin',null,'A','Rhein',null,null);
  ```

  is not forbidden!

- NULLs do not satisfy any WHERE condition (except IS NULL)

- Join conditions in SQL are *null-rejecting*;
  a join including a null-containing column must use separate IS NULL checks or
  COALESCE.
  (example see next slide)

NULL Values in SQL - Join Example

```
CREATE VIEW OneProvCountry AS
SELECT country FROM province GROUP BY country HAVING COUNT(*) = 1;

CREATE TABLE city2 as
(     (SELECT name, province, country, population FROM city
        WHERE country NOT IN (SELECT country FROM OneProvCountry))
 UNION (SELECT name, NULL, country, population FROM city
        WHERE country IN (SELECT country FROM OneProvCountry)));

CREATE TABLE located2 AS
(     (SELECT city, province, country, river, lake, sea FROM located
        WHERE country NOT IN (select country from OneProvCountry))
 UNION (SELECT city, NULL, country, river, lake, sea FROM located
        WHERE country IN (SELECT country FROM OneProvCountry)));

SELECT c.name, c.population, l.river FROM city2 c, located2 l
WHERE c.name=l.city AND c.province=l.province AND c.country=l.country;
-- 958 results, the ones with NULL province are missing.

SELECT c.name, c.population, l.river FROM city2 c, located2 l
WHERE c.name=l.city
  AND COALESCE(c.province,'bla')=COALESCE(l.province,'bla')
-- note that the condition is not satisfied if one value is null, and the other is not!
  AND c.country=l.country;
-- 1157 results
```

4.4.1 SPARQL Algebra

(according to [PAG06])

Algebraic Syntax of SPARQL Queries

- deals with the graph pattern part, not the result modifiers (which contains projection and some operators that are also not covered by the relational algebra, like DISTINCT, ORDER BY, LIMIT, OFFSET)

- fully parenthesized

- Triple patterns (cf. Slide 106) in (RDF-T $\cup$ Var) $\times$ (IRI $\cup$ Var) $\times$ (RDF-T $\cup$ Var) are graph patterns;

- For graph patterns P_1 and P_2, the expressions (P_1 OPT P_2), and (P_1 UNION P_2) are graph patterns;

- For a graph pattern P and a condition R, (P FILTER R) is a graph pattern. (Require filter-safeness: var(R) $\subseteq$ var(P); cf. Slide 134)

- var(P) denotes the variables occurring in a graph pattern P.

Filter-Safe Graph Patterns

Definition 4.1 ([PAG06, AG 08])
A SPARQL algebra expression is *filter-safe*, if for every subexpression of the form (P FILTER R), var(R) $\subseteq$ var(P). □

- for the *compositional* semantics the requirement is absolutely necessary – what should be the semantics of {(?X p ?Y) FILTER (?Y > ?Z)} ?

- The commonly used SPARQL pattern

```
{ ?P1 a :Person;  :age ?A1. ?P2 a :Person
  OPTIONAL { ?P2 :age ?A2 . FILTER ( ?A2 > ?A1 ) }}
```

is not filter-safe. The filter actually formulates a join condition on the outer join:

$$(P_1 \text{ a Person; age } A_1 . P_2 \text{ a Person}) \underset{A_1 < A_2}{\sqsupset\!\!\bowtie} (P_2 \text{ age } A_2)$$

- [AG08] gives an algorithm that transforms a non-filter-safe SPARQL query into a filter-safe algebra expression.
  (DBTheory: similar to the transformation of general formulas into RANF when moving conjuncts into negated subformulas to make them *self-contained*; cf. exercises).

- Non-compositional sideways-information-passing evaluation is often also much more efficient, cf. evaluation of SQL vs. relational algebra.

SPARQL Algebra: Mappings/Answer Substitutions

- a *mapping* μ from Var to RDF-T is a *partial* function $\mu : \text{Var} \to \text{RDF-T}$.
  (also called "answer substitutions" or "tuples of variable bindings"; cf. Datalog)
  for (?X mon:capital ?Y), e.g. $\mu_1 = \{X \mapsto \text{<germany>}, Y \mapsto \text{<berlin>}\}$

- $\text{dom}(\mu) := \{v \in \text{Var} : \mu(v) \text{ is defined}\}$

- for a SPARQL query P, let $\text{var}(P) \subseteq \text{Var}$ denote the set of variables that occur in P.

- for a triple pattern t, and a mapping μ s.t. $\text{dom}(\mu) \supseteq \text{var}(P)$, $\mu(t)$ denotes the triple obtained by replacing all variables $v \in \text{Var}$ in P by $\mu(v)$.

- two mappings μ_1 and μ_2 are *compatible* if for all $v \in \text{dom}(\mu_1) \cap \text{dom}(\mu_2)$, $\mu_1(v) = \mu_2(v)$,
  i.e., when $\mu_1 \cup \mu_2$ is also a mapping (with $\text{dom}(\mu) = \text{dom}(\mu_1) \cup \text{dom}(\mu 2)$).
  - cf. definition of joining tuples $\mu_1\mu_2$ in the relational algebra,
  - here using $\text{dom}(\mu)$ (which is flexible wrt. null values) instead of the fixed schema/format of an expression.

- now consider sets Ω of such mappings (instead of sets of fixed-format tuples in the relational algebra) ...

Semantics of Expressions and Operations on Sets of Mappings

Recall: Relational Algebra defines operators π, σ, $\bowtie$ etc. on sets of tuples.

Common general notation: $[\![P]\!]$ for "semantics of P"; $[\![P]\!]_D$ for "wrt. a given graph D".

- base case: triple patterns:
  $[\![t]\!]_D = \{\mu \mid \text{dom}(\mu) = \text{var}(t) \text{ and } \mu(t) \in D\}$.

- operations on sets of mappings:

- projection: not needed here; as a result modifier, it is external to the core algebra.

- selection: next slide.

- $[\![(P_1 \text{ UNION } P_2)]\!]_D = [\![P_1]\!]_D \cup [\![P_2]\!]_D$;

- $[\![(P_1 \text{ AND } P_2)]\!]_D = [\![P_1]\!]_D \bowtie [\![P_2]\!]_D$ with
  $\Omega_1 \bowtie \Omega_2 = \{\mu_1 \cup \mu_2 \mid \mu_1 \in \Omega_1 \text{ and } \mu_2 \in \Omega_2 \text{ are compatible}\}$;

- $[\![(P_1 \text{ OPT } P_2)]\!]_D = [\![P_1]\!]_D \sqsupset\!\bowtie [\![P_2]\!]_D$ with
  $\Omega_1 \sqsupset\!\bowtie \Omega_2 = (\Omega_1 \bowtie \Omega_2) \cup (\Omega_1 \setminus \Omega_2)$ where $\setminus$ means
  $\Omega_1 \setminus \Omega_2 = \{\mu_1 \in \Omega_1 \mid \text{ for all } \mu_2 \in \Omega_2, \mu_1 \text{ and } \mu_2 \text{ are not compatible}\}$;
  (note that "$\setminus$" here does not denote set difference, but has a different definition).

Three-valued logic for evaluating conditions when variables are not bound:

- truth values t (true, 1), u (undefined, 0.5), f (false, 0), ordered by $t > u > f$.

- All three-valued logics coincide in the definition of $\wedge$, $\vee$, and $\neg$:

$$A \wedge B = \min(A, B) \qquad A \vee B = \max(A, B) \qquad \neg A = 1 - a$$

A \\ B	f	u	t
f	f	f	f
u	f	u	u
t	f	u	t

A \\ B	f	u	t
f	f	u	t
u	u	u	t
t	t	t	t

A	$\neg A$
f	t
u	u
t	f

- selection: $[\![(P \text{ FILTER } R)]\!]_D = \{\mu \in [\![P]\!]_D \mid \mathsf{val}(R, \mu) = t\}$ with

  - $\mathsf{val}(\mathsf{bound}(?X), \mu) = (X \in \mathsf{dom}(\mu))$ with $(X \in \mathsf{dom}(\mu))$ as truth value $t, f \in \{t, u, f\}$;

  - $\mathsf{val}(?X =?Y, \mu) = \begin{cases} t & \text{if } X \in \mathsf{dom}(\mu), Y \in \mathsf{dom}(\mu), \text{ and } \mu(X) = \mu(Y) \\ u & \text{if } X \in \mathsf{dom}(\mu), Y \in \mathsf{dom}(\mu), \text{ and } \mu(X) \neq \mu(Y) \\ f & \text{if } X \notin \mathsf{dom}(\mu) \text{ or } Y \notin \mathsf{dom}(\mu); \end{cases}$

  - $\mathsf{val}(?X = c, \mu)$ analogously;

  - for $(R_1 \wedge R_2)$, $(R_1 \vee R_2)$, and $(\neg R_1)$ see above truth tables.

137

- SPARQL (like SQL) has a bag (=multiset) semantics.

- the above algebra (like the relational algebra) is defined wrt set semantics;

- can be generalized to bag semantics.

1. define a relational "null-tolerant join" that acts like $\bowtie$ above.

2. which SQL construct is similar to the "\\" operator in the above SPARQL algebra?

3. in the above algebra, OPT is expressed via left outer join, which is defined via "\\" (while a corresponding MINUS does not exist in the SPARQL syntax).
   Such a MINUS (cf. Exercise (2)) provides a more intuitive idea of negation than
   "! bound(x)". Give a general pattern how to express $(P_1 \text{ MINUS } P_2)$ in SPARQL syntax.

4. recall the definition of $\rtimes\!\!\!\bowtie$ in the in the relational algebra (DB lecture) and define SPARQL's $\rtimes\!\!\!\bowtie$ in a similar way.

138

Theorem 4.1 ([AG08])

- SPARQL (=version 1.0, = the above algebra) has the same expressive power as *non-recursive safe Datalog with negation*.
  (recall: non-recursive with negation is stratifiable)

- SPARQL (=version 1.0, = the above algebra) has the same expressive power as the relational algebra (under set semantics, and under bag semantics). □

Proof ([AG08]):

- mapping from SPARQL to the filter-safe algebra expressions;

- mappings between the (filter-safe) SPARQL algebra and Datalog/relational algebra.

Normal Form

Theorem 4.2 ([PAG06])
Every graph pattern is equivalent to a pattern in the *normal form*

$$(P_1 \text{ UNION} \cdots \text{ UNION } P_n)$$

where each P_i is UNION-free graph pattern. □

Proof: UNION is distributive wrt. each of the operators:

- $(P_1 \text{ AND } (P_2 \text{ UNION } P_3)) \equiv (P_1 \text{ AND } P_2) \text{ UNION } (P_1 \text{ AND } P_3)$,

- $((P_1 \text{ UNION } P_2) \text{ OPT } P_3) \equiv (P_1 \text{ OPT } P_3) \text{ UNION } (P_2 \text{ OPT } P_3)$,

- $((P_1 \text{ UNION } P_2) \text{ FILTER } R) \equiv (P_1 \text{ FILTER } R) \text{ UNION } (P_2 \text{ FILTER } R)$,

- note that the case for $(P_1 \text{ OPT } (P_2 \text{ UNION } P_3))$ is harder to prove [PAG09].

Complexity

[PAG06] Consider the problem "is $\mu \in [\![P]\!]_D$"?

So-called "data complexity" for fixed P as a function of the size of D:

- The problem is in LOGSPACE (which is a subset of P, i.e., it is polynomial).

So-called "combined complexity" as a function of P and D:

- Conjunctive queries: the problem is in $O(|P| \cdot |D|)$ for graph patterns containing only AND and FILTER. [PAG06]

- the problem is NP-complete for graph patterns containing only AND, FILTER, and UNION. [PAG06]

- the problem is PSPACE-complete for graph patterns in general. [PAG06]

- the problem is PSPACE-complete for graph patterns over OPT only. [SML10]
  [SML10]: M. Schmidt, M. Meier, and G. Lausen: Foundations of SPARQL Query Optimization. In *International Conference on Database Theory (ICDT)*, pp. 4–33. ACM, 2010.

Well-Designed Graph Patterns

Definition 4.2 ([PAG06])
A graph pattern is *well-designed* if for every subpattern $P' = (P_1 \text{ OPT } P_2)$: every variable $?X$ that occurs inside P_2 and also outside P', also occurs in P_1. □

- the combined complexity of well-designed graph patterns is coNP-complete [PAG09]

- Many non-well-designed graph patterns exhibit weird behavior (cf. exercises).

Theorem 4.3 ([PAG06])
Every *union-free* well-designed graph pattern is equivalent to a pattern in the *normal form*

$$(\cdots(t_1 \text{ AND } \cdots \text{ AND } t_k) \text{ OPT } O_1) \text{ OPT } O_2) \cdots) \text{ OPT } O_n)$$

where each t_i is a triple pattern and each O_j is also in this normal form. □

- The role of well-designed patterns for SPARQL is comparable to the role of conjunctive queries for the relational algebra.

Prove or show a counterexample:

The statement

> If OPT(A, B) is an optional graph pattern, where A and B are graph patterns, then S is
> a solution of OPT(A,B) if S is a pattern solution of A and of B otherwise if S is a
> solution to A, but not to A and B.

describes the same semantics as above.

4.5 SPARQL 1.1 - Syntactic Extensions

- path expressions (by "/") that consist of two or more subexpressions

- $^\wedge expr$ for inverse (and $expr^\wedge expr$ for $expr/^\wedge expr$),

- $expr \mid expr$ for alternative,

- $expr+$ for transitive closure, $expr\{n,m\}$ for n to m $expr$ steps.

```
prefix   mon: <http://www.semwebtech.org/mondial/10/meta#>
# http://jena.apache.org/documentation/query/property_paths.html


SELECT  ?C ?P ?O ?C2 ?NC ?T
FROM <file:mondial-europe.n3>
WHERE
  { ?C a mon:Country ;
    mon:capital/mon:population ?P ;
    mon:capital/^mon:hasHeadq ?O ;
    # mon:neighbor+ ?C2 ;
    mon:neighbor{2,4} ?C2 }
```
[Filename: RDF/mondial-path-exprs.sparql]

SPARQL 1.1 - Syntactic Extensions

- arithmetic and string expressions + aliasing in the select clause

- str(*expr*) for converting URIs into strings, uri(*string*) for creating URIs from strings.

```
prefix  mon: <http://www.semwebtech.org/mondial/10/meta#>
# http://jena.apache.org/documentation/query/select_expr.html


SELECT  ?C (?P/?A as ?D) (uri(concat(str(?CAP),'bla')) as ?U)
FROM <file:mondial-europe.n3>
WHERE
  { ?C a mon:Country ; mon:population ?P ; mon:area ?A;
    mon:capital ?CAP  }
```
[Filename: RDF/mondial-select-exprs.sparql]

- "if(a,b,c)" for "if a is true, then use b, else c",

- "coalesce(*expr*$_1$,...,*expr*$_n$)" returns the value of the first *expr*$_i$ that evaluated without error (unbound counts as error).

SPARQL 1.1 - Syntactic Extensions

- NOT EXISTS keyword (syntactic sugar for the !bound(*var)* construct)

```
prefix  mon: <http://www.semwebtech.org/mondial/10/meta#>
# http://jena.apache.org/documentation/query/negation.html


SELECT  ?C
FROM <file:mondial-europe.n3>
WHERE
  { ?C a mon:Country ;
    NOT EXISTS { ?C mon:neighbor ?C2 } }
```
[Filename: RDF/mondial-not-exists.sparql]

```
prefix  mon: <http://www.semwebtech.org/mondial/10/meta#>
# http://jena.apache.org/documentation/query/group-by.html


SELECT  ?C count(?Cty)
FROM <file:mondial-europe.n3>
WHERE
  { ?C a mon:Country ;
    (mon:hasCity|(mon:hasProvince/mon:hasCity)) ?Cty
  }
  GROUP BY ?C
  HAVING (count(?Cty) > 1)
```

[Filename: RDF/mondial-group-by.sparql]

147

Subqueries

- WHERE { *basic graph pattern* . { SELECT ... [FROM ...] WHERE ... }}

- cf. SQL: subqueries in the SQL FROM clause contributing to the join,

- *implicit* join variables,

- no subqueries in the FILTER (SQL's where clause).

- recall from SQL/rel.Algebra: every SQL query with a subquery can be rewritten into an algebra query (which does not allow for subqueries in selection predicates).

- main use: compute some aggregate in the subquery:

```
prefix  mon: <http://www.semwebtech.org/mondial/10/meta#>
SELECT  ?X ?MAXPOP
FROM <file:mondial.n3>
WHERE {
        ?X a mon:Country; mon:population ?MAXPOP .
        { SELECT (MAX(?YP) AS ?MAXPOP)
          WHERE { ?Y a mon:Country; mon:population ?YP }  }
      }                              [Filename: RDF/biggestcountry.sparql]
```

148

- for each country, its bigges city, if this has more than 1000000 inhabitants

```
prefix  mon: <http://www.semwebtech.org/mondial/10/meta#>


SELECT  ?Y ?C ?CN ?MP
FROM <file:mondial.n3>
WHERE
  { ?Y a mon:Country ;  mon:carCode ?C ;
      mon:hasProvince/mon:hasCity
        [ mon:name ?CN ; mon:population ?MP] .
   { SELECT ?Y (MAX(?YP) AS ?MP)
     WHERE {
       ?Y a mon:Country;
          mon:hasProvince [ mon:hasCity [ mon:population ?YP ]].
          ## mon:hasProvince/mon:hasCity/mon:population ?YP. does not work!
     } GROUP BY ?Y
     HAVING (?MP > 1000000)
  }}
ORDER BY ?MP                            [Filename: RDF/biggestcities.sparql]
```

4.6   RDF vs. Object Identity

- So far, RDF is just a restricted variant of ground (sorted) relational First-Order Logic atoms:
  (cf. Slides 192 for a more detailed consideration)
  - 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"

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

  <#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 152 and 151).

```
@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 does 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.

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 -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.

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 $\lor \varphi$,
  which is in turn equivalent to true $\lor \varphi$.

- 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 $\Phi \land \neg\varphi$ 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.

- 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 ?C
from <file:john-three-children.n3>
where { ?X :name "John" . ?X rdf:type ?C }
```

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

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.)

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 151 by a first-order formula 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

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

  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 115.

- 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.

4.7   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 99:

```
@prefix : <foo://bla/names#> .
@prefix family: <foo://bla/persons/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
  family:john rdf:type :Person; :name "John";  :age 35;
        :child family:alice, family:bob.
  family:alice :name "Alice"; :age 10 .
  family: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/meta#> .
@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 4 .
```
[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/meta#>
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 ?T ?A
from <file:john-and-the-cat.n3>
where { <persons/john> rdf:type ?T ; <meta#age> ?A }
```
[Filename: RDF/sparql-base.sparql]

- Alternative: value-based selection of an object

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

169

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.

170

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)
  ⇒ 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)
  ⇒ 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)

⇒ these "legal" and "non-legal" usages are not enforced by anything. Non-legal usage can lead to severe problems (see Slide 200).

4.8 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.

4.9   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?