Contents

1 Resource Description Framework 3
  1.1 URIs ............................................................. 3
  1.2 Literals .......................................................... 4
  1.3 RDF graphs ...................................................... 4
  1.4 Blank nodes ..................................................... 4
  1.5 Vocabularies ..................................................... 4
    1.5.1 Examples with RDF and RDFS ......................... 5
  1.6 RDF Serializations ............................................ 5
  1.7 Turtle ............................................................ 5
    1.7.1 URI references ............................................. 5
    1.7.2 Namespaces ............................................... 6
    1.7.3 Literals ................................................... 6
    1.7.4 Sharing elements ....................................... 6
    1.7.5 Blank nodes .............................................. 7
    1.7.6 Other ..................................................... 7

2 SPARQL Protocol And RDF Query Language 7
  2.1 SPARQL fundamentals ........................................ 8
  2.2 Groups and options ......................................... 9
    2.2.1 SPARQL keywords ...................................... 10
    2.2.2 Functions and operators .............................. 10
  2.3 SPARQL 1.1 ..................................................... 11
    2.3.1 Graph operations ...................................... 11
    2.3.2 Inserting and deleting triples ...................... 11
    2.3.3 Aggregation ........................................... 12

3 RDFS Reasoning 12
  3.1 Syntactic reasoning with deduction rules ................. 13
    3.1.1 Soundness and completeness .......................... 13
  3.2 Interpretations ............................................. 13
    3.2.1 Blank node semantics ................................ 14
### 4 Description Logic (ALC)

### 5 The Web Ontology Language (OWL)
- **5.1** OWL vocabulary in OWL/RDF
- **5.2** Properties in OWL
  - **5.2.1** Characteristics of OWL properties
- **5.3** Manchester OWL Syntax
- **5.4** Assumptions
  - **5.4.1** World assumptions
  - **5.4.2** Name assumptions
- **5.5** OWL 2
  - **5.5.1** ABox axioms: equality
  - **5.5.2** Creating concepts using individuals
  - **5.5.3** Axioms involving individuals
  - **5.5.4** ABox axioms: negative
  - **5.5.5** Cardinality restrictions
  - **5.5.6** Value restrictions
  - **5.5.7** Self restriction
  - **5.5.8** Restrictions, NUNA and OWA
  - **5.5.9** Unexpected results
  - **5.5.10** Role modeling in OWL 2
- **5.6** Mathematical properties and operations

---

**Introduction**

This is my notes for repetition in the course INF3580–Semantic technologies in front of the exam. This document is based on lectures, own notes and the course books.
1 Resource Description Framework

An Resource Description Framework (RDF) document describes a directed graph. Both nodes and edges are labeled with identifiers to distinguish them. RDF was intended to serve as a description language for data on the WWW, and being a graph structure—it’s easy to combine RDF data from multiple sources.

All information in RDF is expressed using a triple pattern. A triple consist of a subject, a predicate and an object.

<table>
<thead>
<tr>
<th>subject</th>
<th>predicate</th>
<th>object</th>
</tr>
</thead>
<tbody>
<tr>
<td>Norway</td>
<td>has capital</td>
<td>Oslo</td>
</tr>
<tr>
<td>Oslo</td>
<td>has mayor</td>
<td>Fabian Stang</td>
</tr>
<tr>
<td>Fabian Stang</td>
<td>born year</td>
<td>1955</td>
</tr>
</tbody>
</table>

A another word for an RDF triple is a statement or fact.

1.1 URIs

RDF uses Uniform Resource Identifiers (URIs) as names to clearly distinguish resources from each other. URIs that are not URLs are sometimes called Uniform Resource Name (URNs). Details of protocols (as http) are not relevant when using URI only as a name, as in RDF. RDF allow encoding of data which are not URIs, and it features blank nodes which do not carry any name. Data values in RDF are represented by literals.

Almost everything is a resource. Resources are identified by URIs. For example dbpedia.org:

- Norway: http://dbpedia.org/resource/Norway
• has capital: http://dbpedia.org/ontology/capital

In RDF-syntax (with prefixes) the triple "Norway has capital Oslo" will be.

@prefix dbp: <http://dbpedia.org/resource/>
@prefix dbp-ont: <http://dbpedia.org/ontology/>

dbp:Norway dbp-ont:capital dbp:Oslo .

1.2 Literals

Data values are specified by a literal string in quotation marks, followed by "^^ and the URI of the datatype. Language information can be provided as semantically untyped literals. Supplied by means of the symbol @.

1.3 RDF graphs

An RDF graph is a graph containing of at least two triples.

dbp:Norway dbp-ont:capital dbp:Oslo .
dbp:Oslo dbp-ont:population "629313"^^xsd:integer .

1.4 Blank nodes

RDF allow us to introduce nodes without any URI, called blank nodes. This feature is only available for subject and object of RDF triples. Blank nodes cannot be addressed globally by means of URIs, and they do not carry additional information within RDF graphs.

1.5 Vocabularies

Families of related notions are grouped into vocabularies. Usually the same namespace or prefix is shared. A well-known namespace (and prefix) is for example rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

A description is usually published at the namespace base URI. The prefix is not standardized, so you may give the prefix rdf:<http://xmlns.com/foaf/0.1/>. However, this would be highly irregular.
1.5.1 Examples with RDF and RDFS

```
rdf:Statement  rdfs:Class
rdf:subject    rdfs:subClassOf
    rdfs:subPropertyOf
rdf:predicate  rdfs:domain
rdf:object     rdfs:range
rdf:type       rdfs:label
```

Example of use

```
dbp:Oslo rdf:type dpt-ont:Place .
dbp:Norway rdfs:label "Norge"@no .
```

1.6 RDF Serializations

There are several serializations for the RDF data model.

**RDF/XML** this is the W3C standard.

```
<?xml version="1.0" encoding="utf-8" ?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:foaf="http://xmlns.com/foaf/0.1/">
    <foaf:Person rdf:about="http://example.org/ifi/veronahe">
        <foaf:Person>
            </rdf:RDF>
```

**Turtle** is convenient, human readable and writable. This is used in the course.

```
@prefix foaf: <http://xmlns.com/foaf/0.1/>
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
@prefix ifi: <http://example.org/ifi/>

ifi:veronahe rdf:type foaf:Person .
```

**N-triples** one triple per line. No abbreviations.

```
```

1.7 Turtle

1.7.1 URI references

URIs are surrounded by `<` and `>`.
• <http://dbpedia.org/resource/Oslo>

Statements are triples terminated by a period


Use a to abbreviate rdf:type


1.7.2 Namespaces

Namespace prefixes are declared with @prefix, may also declare default namespaces.

@prefix dbp: <http://dbpedia.org/resource/> .
@prefix : <http://dbpedia.org/ontology/> .

dbp:Oslo a :Place .

1.7.3 Literals

Literals are enclosed in double quotes " ... ". Possibly with type or language information. Numbers and booleans may be written without quotes.

dbp:Oslo :officialName "Oslo" .
dbp:Norway rdfs:label "Norge"@no .
dbp:Oslo :population "629313"^^xsd:integer .
dbp:Oslo :population 629313 .
dbp:Oslo :isCapital true .

1.7.4 Sharing elements

May list statements that share subject using ;.

```
dbp:Oslo :officialName "Oslo" ;
:population 629313 ;
:leaderName dbp:Fabian_Stang .
```

And we may list statements that share both subject and predicate using ,.

```
dbp:Norway rdfs:label "Norway"@en ,
"Norwegen"@de ,
"Norge"@no .
```
1.7.5 Blank nodes

Blank nodes are designated with underscore or [ ... ].

```
dbp:Norway :capital _:someplace .
_:someplace :population 629313 .
```

There is a place with official name Oslo

```
[] a :Place ;
 :officialName "Oslo" .
```

UiO has address Problemveien 7, 0313 Oslo

```
:UiO :address [ :street "Problemveien" ;
 :streetNumber "7" ;
 :place "Oslo" ;
 :postcode "0313" ] .
```

![Diagram of UiO address](image)

The blank node in this example has no name. In turtle you can name blank nodes with blank node identifiers, like _:someplace. This makes it easier to re-use blank nodes in several triples.

1.7.6 Other

- Use # to comment.
- Use \ to escape special characters.

```
:someGuy foaf:name "James "Mr. Man" Olson" .
```

- Turtle specification: [http://www.w3.org/TR/turtle/](http://www.w3.org/TR/turtle/)

2 SPARQL Protocol And RDF Query Language

As the title says, SPARQL is a query language for RDF. Documentation can be found at these locations

- Queries [http://www.w3.org/TR/rdf-sparql-query/](http://www.w3.org/TR/rdf-sparql-query/)
- Protocol [http://www.w3.org/TR/rdf-sparql-protocol/](http://www.w3.org/TR/rdf-sparql-protocol/)
- Results [http://www.w3.org/TR/rdf-sparql-XMLres/](http://www.w3.org/TR/rdf-sparql-XMLres/)
2.1 SPARQL fundamentals

PREFIX ex: <http://example.org/>
SELECT ?title ?author
  ?book ex:author ?author . }

The actual query is initiated by the key word WHERE. It is followed by a simple graph pattern, enclosed in curly braces.

Exam exercise  Consider the following RDF

@prefix nasa: <http://www.nasa.gov/vocab#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

nasa:voyager a nasa:SpaceProbe;

nasa:mariner9 a nasa:SpaceProbe;
  nasa:objective nasa:mars .

nasa:jupiter a nasa:Planet;
  nasa:moon nasa:io, nasa:ganume, nasa:callisto;
  nasa:distanceFromSun "778"^^xsd:double .

nasa:saturn a nasa:Planet;
  nasa:distanceFromSun "1426"^^xsd:double .

nasa:neptune a nasa:Planet;
  nasa:moon nasa:triton, nasa:proteus, nasa:naiad;
  nasa:distanceFromSun "4503"^^xsd:double .

List all spaceprobes.

SELECT ?probe WHERE {
  ?probe a nasa:SpaceProbe .
}

For each planet that has one or more moons, list the planet and its moons.

1Example taken from exam exercise 2, 2010.
SELECT ?planet ?moon WHERE {
    ?planet a nasa:Planet;
    nasa:moon ?moon .
}

List all planets that lie further than 2000 million kilometers from the sun.

SELECT ?planet WHERE {
    ?planet a nasa:Planet;
    nasa:distanceFromSun ?distance .
    FILTER(?distance > 2000)
}

All planets that have been visited by more than one space probe.

SELECT DISTINCT ?planet WHERE {
    ?planet a nasa:Planet .
    ?probe1 a nasa:SpaceProbe ;
    nasa:objective ?planet .
    ?probe2 a nasa:SpaceProbe ;
    nasa:objective ?planet .
    FILTER(probe1 != ?probe2)
}

2.2 Groups and options

Group patterns can be used to restrict the scope of query conditions to certain parts of the pattern. It is possible to define sub-patterns as being optional, or to provide multiple alternative patterns.

SELECT ?title ?author
{} ?book ex:author ?author }

Optional patterns are not required to occur in all retrieved results, but if they are found, may produce bindings of variables and thus extend the query result.
Every occurrence of \texttt{OPTIONAL} must be followed by a group pattern.

\subsection*{2.2.1 SPARQL keywords}

\textbf{PREFIX} Lets you use a prefix in the query instead of the whole URI.

\textbf{SELECT} Return clause. Returns data that fit some condition.

\textbf{FROM} Defines the RDF dataset which is being queried.

\textbf{WHERE} Specifies the query graph pattern to be matched.

\textbf{ORDER BY} One solution modifier. Used to rearrange query results. Other solution modifiers are \texttt{LIMIT} and \texttt{OFFSET}.

\textbf{ASK} Return clause. Checks if there is at least one result for a given pattern. Returns true or false.

\textbf{DESCRIBE} Returns a RDF graph that describes a resource.

\textbf{CONSTRUCT} Returns a RDF graph that is created from a template specified as a part of the query itself.

\textbf{DISTINCT} Lists equal entries once.

\textbf{FILTER} Filters the result with numerical functions, regular expressions, string operations etc.

\textbf{REDUCE} Allows to remove some or all duplicate solutions.

\textbf{UNION} Matches if any or some alternative matches. Combining graph patterns.

\textbf{OPTIONAL} Optional parts of a pattern.

\subsection*{2.2.2 Functions and operators}

- Usual binary operators: $|\, |$, $\&\&$, $=$, $\neq$, $<$, $>$, $\leq$, $\geq$, $+$, $-$, $\ast$, $/$.

- Usual unary operators: $!$, $+$, $-$.

- Unary tests: \texttt{BOUND(?var)}, \texttt{isURI(?var)}, \texttt{isBLANK(?var)}, \texttt{isLITERAL(?var)}.

- Accessors: \texttt{STR(?var)}, \texttt{LANG(?var)}, \texttt{DATATYPE(?var)}.
Some other tests are among others \texttt{sameTerm(?var)} which is used with unsupported data types. \texttt{langMatches} is used with \texttt{lang} to test for a language, like \texttt{langMatches(lang(?title), "no")}. Another one is \texttt{regex} which is used to match a variable with a regular expression. Always use \texttt{regex} with \texttt{str(?var)}!

2.3 SPARQL 1.1

Became a recommendation in March 2013.

2.3.1 Graph operations

\texttt{LOAD (SILENT)? IRIref\_from (INTO GRAPH IRIref\_to)?} Loads the graph at \texttt{IRIref\_from} into the specified graph, or the default graph if graph not given.

\texttt{CLEAR (SILENT)? (GRAPH IRIref | DEFAULT | NAMED | ALL)} Removes the triples from the specified graph, the default graph, all named graphs or all graphs respectively.

\texttt{CREATE (SILENT)? GRAPH IRIref} Creates a new graph in stores that record empty graphs.

\texttt{DROP (SILENT)? (GRAPH IRIref | DEFAULT | NAMED | ALL)} Same as \texttt{CLEAR}, but removes all triples as well.

2.3.2 Inserting and deleting triples

Inserting

\texttt{INSERT DATA {}
\hspace{1em} \texttt{GRAPH </graph/courses/> {}
\hspace{2em} \texttt{<course/inf3580> :takenBy <student/veronika> .}
\hspace{2em} \texttt{<student/veronika> foaf:name "Veronika Heimsbakk" ;}
\hspace{3em} \texttt{owl:sameAs <http:// ... > .}
\hspace{1em} \texttt{}}
\texttt{}}

Inserting conditionally

\texttt{INSERT {}
\hspace{1em} \texttt{<http:// ... /geo/inndeling/03> a gd:Fylke ;}
\hspace{2em} \texttt{gn:name "Oslo" ;}
\hspace{3em} \texttt{?p ?o .}
\hspace{1em} \texttt{}}
\texttt{WHERE {}
\hspace{2em} \texttt{<http:// ... /geo/inndeling/03/0301> a gd:Kommune ;}
Deleting

```
DELETE DATA {
    GRAPH </graph/courses/> {
        <exercise/oblig6> rdfs:label "Mandatory Exercise 6" .
    }
}
```

Deleting conditionally

```
DELETE {
}
WHERE {
    FILTER (?date < "2000-01-01T00:00:00"^^xsd:datetime)
}
```

2.3.3 Aggregation

Solutions can optionally be grouped according to one or more expressions. To specify the group, use `GROUP BY`. And to filter solutions resulting from grouping, use `HAVING`.

Example using `HAVING`

```
SELECT ?name (count(?kommune) AS ?kcount)
WHERE {
    ?fylke a gd:Fylke ;
        gn:officialName ?name ;
    ?kommune a gd:Kommune .
    FILTER (langMatches(lang(?name), 'no'))
} GROUP BY ?name HAVING (?kcount < 15)
```

Returns counties of Norway with less than 15 municipalities.

3 RDFS Reasoning

Interpretations need in no way comply with actual reality. One usually choose a certain mathematical structure as interpretation in order to work in a formally correct way.
3.1 Syntactic reasoning with deduction rules

One option for describing a syntactic method consists of providing deduction rules.

\[
p_1 \ldots p_n \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \qu
3.2.1 Blank node semantics

Given an interpretation $\mathcal{I}$ with domain $\Delta^\mathcal{I}$, a blank node valuation $\beta$

- gives a domain element or literal value $\beta(b) \in \Delta^\mathcal{I} \cup \Delta$.
- for every blank node ID $b$.

Interpretation $\mathcal{I}, \beta \models r(x, y)$ iff $\langle x^{\mathcal{I},\beta}, y^{\mathcal{I},\beta} \rangle \in r^\mathcal{I}$, for any legal combination of URIs, literals or blank nodes $x$, $y$ and object or datatype property $r$.

3.2.2 Example on an interpretation

Let $\Gamma$ be following RDF graph.

```
:Tweety rdf:type :Bird .
:Nixon rdf:type :Republican .
:Nixon :listensTo :Tweety .
[] :likes :Nixon .
:Nixon :hasNickname "Ric" .
:Tweety :hasNickname "Mr. Man" .
:Tweety :likes :Tux .
```

Create an interpretation $\mathcal{I}_1$, such that $\mathcal{I}_1 \models \Gamma$.

- $\Delta^\mathcal{I} = \{1, 2, 3, 4_b, 5_b\}$
- $\text{Tweety}^\mathcal{I} = 1$, $\text{Nixon}^\mathcal{I} = 2$, $\text{Tux}^\mathcal{I} = 3$
- $\text{Bird}^\mathcal{I} = \{1, 4_b\}$, $\text{Republican}^\mathcal{I} = \{2\}$, $\text{Quacker}^\mathcal{I} = \{2\}$
- $\text{listensTo}^\mathcal{I} = \{(2, 1)\}$, $\text{likes}^\mathcal{I} = \{(2, 4_b), \langle 5_b, 2 \rangle, \langle 1, 3 \rangle\}$, $\text{hasNickname}^\mathcal{I} = \{(2, "Ric"), \langle 1, "Mr. Man" \rangle\}$

Let $\beta(b_1) = 4_b$ and $\beta(b_2) = 5_b$, then $\mathcal{I}, \beta \models \Gamma$, so we have that $\mathcal{I} \models \Gamma$.

Create an interpretation $\mathcal{I}_2$ such that $\mathcal{I}_2 \not\models \Gamma$. Let $\mathcal{I}$ be as above, but let $\text{Bird}^\mathcal{I} = \emptyset$. Then there is nothing we can send $b_1$ to have $\langle \text{Nixon} : \text{likes} [ a : \text{Bird} ] \rangle$. So this interpretation satisfy $\mathcal{I} \not\models \Gamma$. 

14
If S contains:

<table>
<thead>
<tr>
<th>rule</th>
<th>statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>rdfs1</td>
<td>any IRI aaa in D</td>
</tr>
<tr>
<td>rdfs2</td>
<td>aaa rdfs:domain xxx . yyy aaa zzz .</td>
</tr>
<tr>
<td>rdfs3</td>
<td>aaa rdfs:range xxx . yyy aaa zzz .</td>
</tr>
<tr>
<td>rdfs4a</td>
<td>xxx aaa yyy .</td>
</tr>
<tr>
<td>rdfs4b</td>
<td>xxx aaa yyy .</td>
</tr>
<tr>
<td>rdfs5</td>
<td>xxx rdfs:subPropertyOf yyy . yyy rdfs:subPropertyOf zzz .</td>
</tr>
<tr>
<td>rdfs6</td>
<td>xxx rdf:type rdf:Property . xxx rdfs:subPropertyOf zzz .</td>
</tr>
<tr>
<td>rdfs7</td>
<td>xxx rdf:type rdf:Property . xxx rdfs:subPropertyOf zzz .</td>
</tr>
<tr>
<td>rdfs8</td>
<td>xxx rdfs:subClassOf yyy . xxx rdfs:subClassOf rdf:Resource .</td>
</tr>
<tr>
<td>rdfs9</td>
<td>xxx rdfs:subClassOf yyy . zzz rdf:type yyy .</td>
</tr>
<tr>
<td>rdfs10</td>
<td>xxx rdf:type rdf:Class . xxx rdfs:subClassOf zzz .</td>
</tr>
<tr>
<td>rdfs11</td>
<td>xxx rdfs:subClassOf yyy . xxx rdfs:subClassOf zzz .</td>
</tr>
<tr>
<td>rdfs12</td>
<td>xxx rdf:type rdfs:ContainerMembershipProperty . xxx rdfs:subPropertyOf rdfs:member .</td>
</tr>
<tr>
<td>rdfs13</td>
<td>xxx rdf:type rdfs:Datatype . xxx rdfs:subClassOf rdfs:Literal .</td>
</tr>
</tbody>
</table>

Table 2: The RDFS deduction rules.
4 Description Logic (\(ALC\))

The basic building blocks of \(ALC\) are classes, roles and individuals, which can be put into relationships with each other.

\[
\text{Student(veronahe)}
\]

This describes that the individual \(\text{veronahe}\) belongs to the class \(\text{Student}\).

\[
\text{hasAffiliation(veronahe, ifi)}
\]

Describes that \(\text{veronahe}\) is affiliated with \(\text{ifi}\). Subclass relations are expressed by using \(\sqsubseteq\).

\[
\text{Professor} \sqsubseteq \text{FacultyMember}
\]

Equivalence between classes is expressed by using \(\equiv\).

\[
\text{Professor} \equiv \text{Prof.}
\]

\(ALC\) provides logical class constructors.

\[
\begin{align*}
\text{Conjunction} & \quad \sqcap \\
\text{Disjunction} & \quad \sqcup \\
\text{Negation} & \quad \neg
\end{align*}
\]

Complex classes can also be defined by using quantifiers. If \(R\) is a role and \(C\) a class expression, then \(\forall R.C\) and \(\exists R.C\) are also class expressions.

The empty class \(\text{owl:Nothing}\), denoted \(\bot\), can be expressed by

\[
\bot \equiv C \sqcap \neg C
\]

\(\top\) corresponds to \(\text{owl:Thing}\), can be expressed by

\[
\top \equiv C \sqcup \neg C
\]

, or equivalently by

\[
\top \equiv \neg \bot
\]

Disjointness of class (\(\text{owl:disjointWith}\) \(C\) and \(D\), expressed by

\[
C \sqcap D \subseteq \bot \Rightarrow C \subseteq \neg D
\]

\(\text{rdfs:range}\) can be expressed by

\[
\top \subseteq \forall R.C
\]

and \(\text{rdfs:domain}\) by

\[
\exists R.\top \subseteq C
\]
Class expressions of $\mathcal{ALC}$: $C, D ::= A | \top | \bot | \neg C | C \sqcap D | C \sqcup D | \forall R.C | \exists R.C$

$\mathcal{ALC}$ semantics: An interpretation $\mathcal{I}$ fixes a set $\Delta^\mathcal{I}$, the domain, $\Delta^\mathcal{I} \subseteq \Delta^\mathcal{I}$ for each atomic concept $A$, $R^\mathcal{I} \subseteq \Delta^\mathcal{I} \times \Delta^\mathcal{I}$ for each role $R$, and $a^\mathcal{I} \in \Delta$ for each individual $a$.

Statements in $\mathcal{ALC}$ are divided into $TBox$ statements and $ABox$ statements. The TBox is considered to contain terminological (schema) knowledge, while the ABox contains knowledge about instances (i.e. individuals). An $\mathcal{ALC}$ knowledge base consists of an ABox and a TBox.

5 The Web Ontology Language (OWL)

Became a W3C recommendation in 2004 and is now the undisputed standard ontology language. It is built on Description Logics (DL) and combines DL expressiveness with RDF technology (URIs, namespaces...). OWL extends RDFS with boolean operations, universal/existential restrictions and more.

OWL is defined as set of things that can be said about classes, properties and instances.

5.1 OWL vocabulary in OWL/RDF

<table>
<thead>
<tr>
<th>New</th>
<th>From RDFS</th>
<th>From RDF</th>
</tr>
</thead>
<tbody>
<tr>
<td>owl:Ontology</td>
<td>rdfs:Class</td>
<td>rdf:type</td>
</tr>
<tr>
<td>owl:Class</td>
<td>rdfs:subClassOf</td>
<td>rdfs:Property</td>
</tr>
<tr>
<td>owl:Thing</td>
<td>rdfs:subPropertyOf</td>
<td></td>
</tr>
<tr>
<td>Properties</td>
<td>rdfs:domain</td>
<td></td>
</tr>
<tr>
<td>Restrictions</td>
<td>rdfs:range</td>
<td></td>
</tr>
<tr>
<td>Annotations</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3: Vocabulary of OWL.

Table 3 including XSD datatypes.

5.2 Properties in OWL

There are three kinds of mutually disjoint properties in OWL.

- **owl:DatatypeProperty**
  - Link individuals to data values as $\text{xsd:string}$.
  - Example: :hasAge, :hasSurname

- **owl:ObjectProperty**
- Link individuals to individuals.
  - Example: :hasFather, :driveAxle

- owl:AnnotationProperty
  - Has no logical implication, ignored by reasoners.
  - Anything can be annotated.
  - Use for human readable-only data.
  - Example: rdfs:label, dc:creator

5.2.1 Characteristics of OWL properties

Object properties link individuals to individuals, so all characteristics and operations are defined for them.

Datatype properties link individuals to data values, so they cannot be

- reflexive, or they would not be datatypes properties
- transitive, since no property takes data values in first position
- symmetric, as above
- inverses, as above
- inverse functional, for computational reasons
- part of chains, as above

So what remains is functionality, subsumption, equivalence and disjointness.

Annotation properties have no logical implication, so nothing can be said about them.

Example: Universal Restrictions in OWL/RDF

- TwoCV ⊑ ∀ driveAxle.FrontAxle

  ![Turtle syntax diagram]

  - In Turtle syntax:
:TwoCV rdfs:subClassOf [ rdf:type owl:Restrictions ;
   owl:onProperty :driveAxle ;
   owl:allValuesFrom :FrontAxle ]
.

• In OWL/XML syntax:

<SubClassOf>
   <Class URI=":TwoCV"/>
   <ObjectAllValuesFrom>
      <ObjectProperty URI=":driveAxle"/>
      <Class URI=":FrontAxle"/>
   </ObjectAllValuesFrom>
</SubClassOf>

• In OWL Functional syntax:

SubClassOf(TwoCV ObjectAllValuesFrom(driveAxle FrontAxle))

5.3 Manchester OWL Syntax

Manchester OWL syntax is used in Protége for concept descriptions. It also has syntax for axioms, but this is less used. Correspondence to DL constructs.

<table>
<thead>
<tr>
<th>DL</th>
<th>Manchester</th>
</tr>
</thead>
<tbody>
<tr>
<td>C ⊓ D</td>
<td>C AND D</td>
</tr>
<tr>
<td>C ⊔ D</td>
<td>C OR D</td>
</tr>
<tr>
<td>¬D</td>
<td>NOT C</td>
</tr>
<tr>
<td>∀R.C</td>
<td>R ONLY C</td>
</tr>
<tr>
<td>∃R.C</td>
<td>R SOME C</td>
</tr>
</tbody>
</table>

With Protége, everything can be mapped to DL axioms.

5.4 Assumptions

5.4.1 World assumptions

Closed World Assumption (CWA)

• Complete knowledge.

• Any statement that is now known to be true is false.

• Typical semantics for database systems.
Open World Assumption (OWA)

- Potential incomplete knowledge.
- Any statement that is not known to be true is false, does not hold.
- Typical semantics for logic-based systems.

5.4.2 Name assumptions

Under any assumptions, equal names (individual URIs, DB constants) always denote the same "thing". Cannot have $a^I \neq a^J$.

Unique Name Assumption (UNA)  Under UNA, different names always denote different things. E.g., $a^I \neq b^J$. This is common in relational databases.

Non-unique Name Assumption (NUNA)  Under NUNA, different named need not denote different things. We can have $a^I = a^J$, or dpedia:Oslo$^I$ = geo:34521$^J$.

5.5 OWL 2

OWL 2 is based on the DL $SHOIN(D)$.

5.5.1 ABox axioms: equality

This axiom express equality and non-equality between individuals.

<table>
<thead>
<tr>
<th>RDF/OWL</th>
<th>DL</th>
<th>Manchester</th>
</tr>
</thead>
<tbody>
<tr>
<td>:a owl:sameAs :b</td>
<td>$a = b$ iff $a^I = b^J$</td>
<td>SameAs</td>
</tr>
<tr>
<td>:a owl:differentFrom :b</td>
<td>$a \neq b$ iff $a^I \neq b^J$</td>
<td>DifferentFrom</td>
</tr>
</tbody>
</table>

Semantics

- $\mathcal{I} \models a = b$ iff $a^I = b^J$
- $\mathcal{I} \models a \neq b$ iff $a^I \neq b^J$

Remember that with non-unique name assumptions in semantic web, you must sometimes use $=$ and $\neq$ to get expected results.

5.5.2 Creating concepts using individuals

Create anonymous concepts by explicitly listing all members. This is called closed classes in OWL.

<table>
<thead>
<tr>
<th>RDF/OWL</th>
<th>DL</th>
<th>Manchester</th>
</tr>
</thead>
<tbody>
<tr>
<td>owl:oneOf {a,b,...}</td>
<td>{a, b,...}</td>
<td></td>
</tr>
<tr>
<td>rdf:List</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>


Example  SimpsonFamilyMember ≡ {Homer, Marge, Bart, Lisa, Maggie}

Note that the individuals does not necessarily represent different objects. We still need = and ≠ to say that members are the same or different. "Closed classes of data values" are datatypes.

5.5.3 Axioms involving individuals

Using closed classes we can exclude individuals from classes.

Example  {NedFlanders} ⊑ ¬SimpsonFamilyMember

- Ned Flanders is not a family member of the Simpsons.
- Better solution: FlandersFamilyMember ≡ {Ned Flanders, ...} and FlandersFamilyMember ⊑ ¬SimpsonFamilyMember

Closed properties does not exist in OWL. However, it can be done with closed classes, but there is negated role assignment to exclude relationships from relations and roles.

5.5.4 ABox axioms: negative

<table>
<thead>
<tr>
<th>RDF/OWL</th>
<th>DL</th>
<th>Manchester</th>
</tr>
</thead>
<tbody>
<tr>
<td>NegativePropertyAssertion</td>
<td>¬R(a, b)</td>
<td>a NOT R b</td>
</tr>
</tbody>
</table>

Semantics

- $\mathcal{I} \models ¬R(a, b)$ iff $\langle a^\mathcal{I}, b^\mathcal{I} \rangle \not\in R^\mathcal{I}$

This works both for object properties and datatype properties.

Example  :Bart not :hasFather :NedFlanders

5.5.5 Cardinality restrictions

Cardinality restrictions are introduced through $\mathcal{SHOIN}(D)$.

<table>
<thead>
<tr>
<th>RDF/OWL</th>
<th>OWL</th>
<th>Manchester</th>
</tr>
</thead>
<tbody>
<tr>
<td>owl:minCardinality</td>
<td>$\geq_n R.D$</td>
<td>MIN</td>
</tr>
<tr>
<td>owl:maxCardinality</td>
<td>$\leq_n R.D$</td>
<td>MAX</td>
</tr>
<tr>
<td>owl:cardinality</td>
<td>$=_n R.D$</td>
<td>EXACTLY</td>
</tr>
</tbody>
</table>
Semantics

- \( (\leq_n R.D)^I = \{a \in \Delta^I | \{b : (a, b) \in R^I \land b \in D^I\} \# \leq n\} \)
- \( (\geq_n R.D)^I = \{a \in \Delta^I | \{b : (a, b) \in R^I \land b \in D^I\} \# \geq n\} \)

Restricts the number of relations a type of object can/must have.

Example  Car \( \sqsubseteq_{\leq 2} \) driveAxle. \( \top \) — "A car has at most two drive axles."

5.5.6 Value restrictions

Existential and universal restrictions are called value restrictions. Restrictions of the form \( \forall R.D, \exists R.D, \leq_n R.D, \geq_n R.D \) are called qualified when \( D \) is not \( \top \). Can also qualify with a closed class.

<table>
<thead>
<tr>
<th>RDF/OWL</th>
<th>DL</th>
<th>Manchester</th>
</tr>
</thead>
<tbody>
<tr>
<td>owl:hasValue</td>
<td>( \exists R{a} ) R VALUE a</td>
<td></td>
</tr>
</tbody>
</table>

Example  Norwegian \( \equiv \) Person \( \sqcup \exists \text{citizenOf.\{}\text{Norway}\}\)

5.5.7 Self restriction

Local reflexivity restriction. Restricts to objects which are related to themselves.

<table>
<thead>
<tr>
<th>RDF/OWL</th>
<th>DL</th>
<th>Manchester</th>
</tr>
</thead>
<tbody>
<tr>
<td>owl:hasSelf</td>
<td>( \exists R.\text{Self} ) SELF</td>
<td></td>
</tr>
</tbody>
</table>

Semantics

- \( (\exists R.\text{Self})^I = \{x | \langle x, x \rangle \in R^I \} \)

Example  \( \exists \text{hasBoss.\text{Self}} \sqsubseteq \text{Selfemployed} \)

5.5.8 Restrictions, NUNA and OWA

[Diagram of Ensemble, ChamberEnsemble, Orchestra, \( \leq_n \text{firstViolin} \) relationships]
TBox

Orchestra ⊑ Ensemble

ChamberEnsemble ⊑ Ensemble

ChamberEnsemble ⊑≤\text{ firstViolin}. \top

ABox

Ensemble(oslo)

firstViolin(oslo, skolem)

firstViolin(oslo, lie)

Orchestras and Chamber ensembles are Ensembles. Chamber ensembles have only one instrument on each voice, in particular, only one first violin.

5.5.9 Unexpected results

It does not follow from TBox + ABox that oslo is an Orchestra. An ensemble need neither be an orchestra nor a chamber ensemble, it’s ”just” an ensemble. Add a ”covering axiom” Ensemble ⊑ Orchestra ⊔ ChamberEnsemble — an ensemble is an orchestra or a chamber ensemble.

It still does not follow that oslo is an Orchestra. This is due to NUNA. We cannot assume that skolem and lie are distinct. The statement skolem owl:differentFrom lie, makes oslo an orchestra.

5.5.10 Role modeling in OWL 2

This can get excessively complicated. For instance, transitive roles cannot be irreflexive or asymmetric. Role inclusions are not allowed to cycle.

\text{hasParent} \circ \text{hasHusband} ⊑ \text{hasFather}

\text{hasFather} ⊑ \text{hasParent}

Transitive roles \text{R} and \text{S} cannot be declared disjoint. These restrictions can be hard to keep track of, the reason they exist are computational, not logical.

5.6 Mathematical properties and operations

A relation \text{R} over a set \text{X}(\text{R} ⊆ \text{X} \times \text{X}) is
Reflexive if \( \langle a, a \rangle \in R \) for all \( a \in X \)
Irreflexive if \( \langle a, a \rangle \notin R \) for all \( a \in X \)
Symmetric if \( \langle a, b \rangle \in R \) implies \( \langle b, a \rangle \in R \)
Asymmetric if \( \langle a, b \rangle \in R \) implies \( \langle b, a \rangle \notin R \)
Transitive if \( \langle a, b \rangle, \langle b, c \rangle \in R \) implies \( \langle a, c \rangle \in R \)
Functional if \( \langle a, b \rangle, \langle a, c \rangle \in R \) implies \( b = c \)
Functional if \( \langle a, b \rangle, \langle c, b \rangle \in R \) implies \( a = c \)

Relations from ordinary language

- Symmetric: hasSibling, differentFrom
- Non-symmetric: hasBrother
- Asymmetric: olderThan, memberOf
- Transitive: olderThan, hasSibling
- Functional: hasBiologicalMother
- Inverse functional: gaveBirthTo

Handy tables

<table>
<thead>
<tr>
<th>OWL constructor</th>
<th>DL syntax</th>
<th>Manchester</th>
</tr>
</thead>
<tbody>
<tr>
<td>intersectionOf</td>
<td>( C \cap D )</td>
<td>C AND D</td>
</tr>
<tr>
<td>unionOf</td>
<td>( C \cup D )</td>
<td>C OR D</td>
</tr>
<tr>
<td>complementOf</td>
<td>( \neg C )</td>
<td>NOT C</td>
</tr>
<tr>
<td>oneOf</td>
<td>{a} \cap {b} \ldots</td>
<td>{a \ b \ldots}</td>
</tr>
<tr>
<td>someValuesFrom</td>
<td>( \exists RC )</td>
<td>R SOME C</td>
</tr>
<tr>
<td>allValuesFrom</td>
<td>( \forall RC )</td>
<td>R ONLY C</td>
</tr>
<tr>
<td>minCardinality</td>
<td>( \geq NR )</td>
<td>R MIN 3</td>
</tr>
<tr>
<td>maxCardinality</td>
<td>( \leq NR )</td>
<td>R MAX 3</td>
</tr>
<tr>
<td>cardinality</td>
<td>( = NR )</td>
<td>R EXACTLY 3</td>
</tr>
<tr>
<td>hasValue</td>
<td>( \exists R{a} )</td>
<td>R VALUE a</td>
</tr>
<tr>
<td>NegativePropertyAssertion</td>
<td>( \neg R(a, b) )</td>
<td>a NOT R b</td>
</tr>
<tr>
<td>hasSelf</td>
<td>( \exists R.Self )</td>
<td>SELF</td>
</tr>
</tbody>
</table>

Table 4: Class constructions in OWL
<table>
<thead>
<tr>
<th>Class</th>
<th>Class(individual)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relations</td>
<td>relation(subject, object)</td>
</tr>
<tr>
<td>Subclasses</td>
<td>Class ⊑ Class</td>
</tr>
<tr>
<td>Equivalence</td>
<td>Class ≡ Class</td>
</tr>
<tr>
<td>owl:Nothing</td>
<td>⊥ ≡ C∩ ≠ C</td>
</tr>
<tr>
<td>owl:Thing</td>
<td>⊤ ≡ C∪ ≠ C</td>
</tr>
<tr>
<td>Disjointness</td>
<td>C ∩ D ⊑ ⊥ ⇒ C ⊑ ¬D</td>
</tr>
<tr>
<td>rdfs:range</td>
<td>⊤ ⊑ ∀R.C</td>
</tr>
<tr>
<td>rdfs:domain</td>
<td>∃R.⊤ ⊑ C</td>
</tr>
</tbody>
</table>

Table 5: Description logics

<table>
<thead>
<tr>
<th>If S contains:</th>
<th>then S RDFS entails recognizing D:</th>
</tr>
</thead>
<tbody>
<tr>
<td>rdfs1</td>
<td>any IRI aaa in D aaa rdf:type rdfs:Datatype .</td>
</tr>
<tr>
<td>rdfs2</td>
<td>aaa rdfs:domain xxx . yyy rdf:type xxx . yyy aaa zzz .</td>
</tr>
<tr>
<td>rdfs3</td>
<td>aaa rdfs:range xxx . zzz rdf:type xxx . yyy aaa zzz .</td>
</tr>
<tr>
<td>rdfs4a</td>
<td>xxx aaa yyy . xxx rdf:type rdfs:Resource .</td>
</tr>
<tr>
<td>rdfs4b</td>
<td>xxx aaa yyy . yyy rdf:type rdfs:Resource .</td>
</tr>
<tr>
<td>rdfs5</td>
<td>xxx rdfs:subPropertyOf yyy . xxx rdfs:subPropertyOf zzz . yyy rdfs:subPropertyOf zzz .</td>
</tr>
<tr>
<td>rdfs6</td>
<td>xxx rdf:type rdf:Property . xxx rdfs:subPropertyOf xxx .</td>
</tr>
<tr>
<td>rdfs7</td>
<td>aaa rdfs:subPropertyOf bbb . xxx bbb yyy . xxx aaa yyy</td>
</tr>
<tr>
<td>rdfs8</td>
<td>xxx rdf:type rdfs:Class . xxx rdfs:subClassOf rdf:Resource .</td>
</tr>
<tr>
<td>rdfs9</td>
<td>xxx rdfs:subClassOf yyy . zzz rdf:type yyy . zzz rdf:type xxx</td>
</tr>
<tr>
<td>rdfs10</td>
<td>xxx rdf:type rdfs:Class . xxx rdfs:subClassOf xxx .</td>
</tr>
<tr>
<td>rdfs11</td>
<td>xxx rdfs:subClassOf yyy . xxx rdfs:subClassOf zzz . yyy rdfs:subClassOf zzz .</td>
</tr>
<tr>
<td>rdfs12</td>
<td>xxx rdf:ContainerMembershipProperty . xxx rdfs:subPropertyOf rdf:member .</td>
</tr>
<tr>
<td>rdfs13</td>
<td>xxx rdf:type rdfs:Datatype . xxx rdfs:subClassOf rdfs:Literal .</td>
</tr>
</tbody>
</table>

Table 6: The RDFS deduction rules.

| se1 | xxx aaa yyy . xxx aaa _:_n |
| se2 | xxx aaa yyy . _:_n aaa yyy |

Table 7: Simple entailment deduction rules.