Resource Description Framework

Jan Wilken Dörrie

January 10, 2013

Contents

1 Introduction **2**

2 Concepts **2**
2.1 Graph data model 2
2.2 URI-based vocabulary 2
2.3 Datatypes 3
2.4 Literals 3
2.5 XML serialization syntax 3
2.6 Expression of simple facts 3
2.7 Entailment 4

3 Syntax **4**
3.1 RDF Schema 4

4 Semantics **5**
4.1 Entailment Rules 7
4.1.1 Simple Entailment Rules 7
4.1.2 RDF Entailment Rules 8
4.1.3 RDFS Entailment Rules 8

5 Conclusion **10**

1 Introduction

The Resource Description Format (RDF) is a semantic web standard introduced by the W3C designed to represent information in the web. RDF has an abstract syntax that reflects a simple graph-based data model, and formal semantics with a rigorously defined notion of entailment providing a basis for well founded deductions in RDF data. RDF was designed with the following goals in mind [KC04]:

- having a simple data model
- having formal semantics and provable inference
- using an extensible URI-based vocabulary
- using an XML-based syntax
- supporting use of XML schema datatypes
- allowing anyone to make statements about any resource

The modeling of information is inspired by object orientated programming languages, with the difference that anybody can add information without altering already existing information. For example extending an existing class with another property does not lead to an update of all previous usages of this class. The restrictive syntax and semantics make the problem of RDF entailment decidable and tractable.

2 Concepts

RDF makes use of the following concepts [KC04]:

- Graph data model
- URI-based vocabulary
- Datatypes
- Literals
- XML serialization syntax
- Expression of simple facts
- Entailment

2.1 Graph data model

Any expression in RDF can be understood as a collection of triples, each consisting of a subject, a predicate and an object. Each such set of triples is also called an RDF graph. This makes intuitive sense, because one can think of the subject and the object as nodes and the predicate as a directed, labeled arc connecting the two.

2.2 URI-based vocabulary

A Uniform resource identifier (URI) is a string of characters used to identify a name or a resource. Although they look similar to URLs, they should not be confused with them. Examples for URI references are `http://xmlns.com/foaf/0.1/Person` and `http://xmlns.com/foaf/0.1/Person/name`, being the identifiers for a person and the name of a person. Subjects and objects of RDF triples can be, and predicates have to be URI references. Additionally it is allowed for subjects and objects to be blank nodes which can be thought of variables without a special meaning. To be able to reference the same blank node in a given graph it is common to give them a blank node identifier. However, when graphs are merged, one needs to pay attention that the identifiers of different graphs stay distinct and a re-allocation might be necessary. Finally objects are also allowed to be strings of characters, called literals.

Putting this together an example of a valid RDF triple is the following:
`http://xmlns.com/foaf/0.1/Person` `http://xmlns.com/foaf/0.1/Person/name` “Jan Wilken Dörrie”.
stating there exist a person with the name Jan Wilken Dörrie.

2.3 Datatypes

RDF introduces datatypes, in order to be able to describe numerical values, such as integers, floating point numbers and dates. Each datatype consist of a lexical space, a value space and a lexical-to-value mapping. For example to describe the XML Schema datatype `xsd:boolean` where each member of the value space has two lexical representations the following is used:

Value Space	{T, F}
Lexical Space	{“0”, “1”, “true”, “false”}
Lexical-to-Value Mapping	{<“true”, T>, <“1”, T>, <“0”, F>, <“false”, F>}

RDF only predefines the datatype `rdf:XMLLiteral` which is used for embedding XML into RDF.

2.4 Literals

As mentioned in 2.2 literals in RDF are strings of characters and only objects can possibly be literals. There are two types of literals one has to distinguish, plain literals and typed literals. Plain literals are strings optionally equipped with a language tag and represent plain text in a natural language. Typed literals have to be equipped with a datatype URI and denote the member of the value space of the given datatype one obtains when applying the lexical-to-value mapping to the string. As an example the string “true” only denotes the corresponding boolean value when written <`xsd:boolean`, “true”>.

2.5 XML serialization syntax

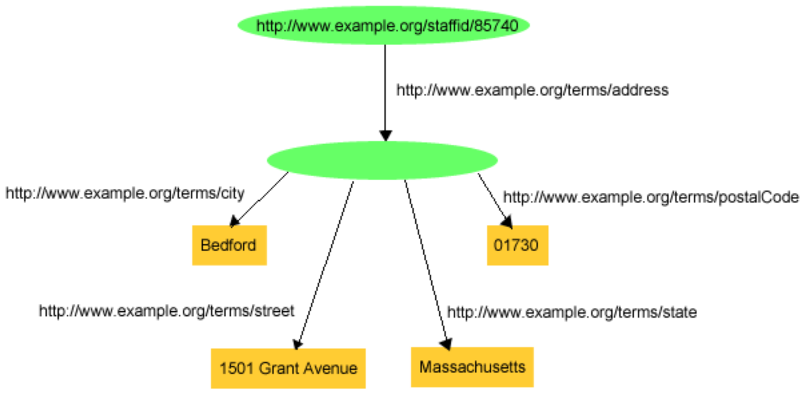
Alongside with RDF the W3C introduced a specification for an XML format used to serialize RDF statements. Its MIME type is `application/rdf+xml` and it comes with a full specified context-free grammar [Bec04].

In XML serialization RDF also makes use of namespaces, which are useful to abbreviate notation and prevent name clashing when a given word has several meanings. When writing the previous example in this syntax one obtains the following:

```
<rdf:RDF xmlns:foaf="http://xmlns.com/foaf/0.1/">
  <foaf:Person>
    <foaf:name>Jan Wilken Dörrie</foaf:name>
  </foaf:Person>
</rdf:RDF>
```

2.6 Expression of simple facts

The fact that RDF is built on a very simple structure makes it very easy to extend a given RDF graph and to transform in and from other ways of knowledge and data representation, as for example relational databases. To transform a table of a relational database into an RDF graph one can represent every row in the table as a blank node. Then add one triple for each column of the table with the corresponding blank node as the subject and the data as an object. Consider the following table entry and the corresponding RDF graph as an example:


http://www.example.org/staffid/85740
http://www.example.org/terms/address
http://www.example.org/terms/city
http://www.example.org/terms/postalCode
Bedford
01730
http://www.example.org/terms/street
http://www.example.org/terms/state
1501 Grant Avenue
Massachusetts

Staff ID	Street	City	Postal Code	State
85740	1501 Grant Avenue	Bedford	1730	Massachusetts

2.7 Entailment

RDF is designed to allow for entailment and interference. Its strict syntax makes the entailment problem decidable, i.e. it is possible to say if a given RDF graph entails another one. See section 4 for more details.

3 Syntax

As already mentioned in section 2.5 there exists a file format for RDF based on XML. It was developed together with the RDF specification and is widely used. However, due to the verbosity of XML, it is often easier to write down RDF statements in a shorter and for humans more readable form. Because of this there is another file format to serialize RDF graphs, N-triples [GB04]. It was designed as a fixed subset of N3 (Notation 3) and it is recommended to use the .nt file extension. Its MIME type is text/plain. Each line is of the form `subject predicate object .`, an example of this was given earlier: `http://xmlns.com/foaf/0.1/Person` `http://xmlns.com/foaf/0.1/Person/name` "Jan Wilken Dörrie".

3.1 RDF Schema

In order to be able to make statements about classes, containers, properties and their range and domain, etc. RDF comes with a vocabulary description language called RDF Schema [BG04]. Not only it provides basic classes and properties, but also defines them and how they have to be used. Examples are general purpose containers such as lists, bags and sequences, or statements about membership, and contents of those containers. Also it defines classes for the previously mentioned Datatypes and (XML)Literals. A summary of all provided classes and properties along side with explanations and domain and ranges can be found in the following two tables. The prefixes `rdf` and `rdfs` denote the RDF namespace (`http://www.w3.org/1999/02/22-rdf-syntax-ns#`) and the RDF Schema namespace (`http://www.w3.org/2000/01/rdf-schema#`).

Class name	comment
`rdfs:Resource`	The class resource, everything.
`rdfs:Literal`	The class of literal values, e.g. textual strings and integers.
`rdf:XMLLiteral`	The class of XML literals values.
`rdfs:Class`	The class of classes.
`rdf:Property`	The class of RDF properties.
`rdfs:Datatype`	The class of RDF datatypes.
`rdf:Statement`	The class of RDF statements.
`rdf:Bag`	The class of unordered containers.
`rdf:Seq`	The class of ordered containers.
`rdf:Alt`	The class of containers of alternatives.
`rdfs:Container`	The class of RDF containers.
`rdfs:ContainerMembershipProperty`	The class of container membership properties, `rdf:_1`, `rdf:_2`, ..., all of which are sub-properties of 'member'.
`rdf:List`	The class of RDF Lists.

Table 1: RDF classes

Property name	**comment**	**domain**	**range**
`rdf:type`	The subject is an instance of a class.	`rdfs:Resource`	`rdfs:Class`
`rdfs:subClassOf`	The subject is a subclass of a class.	`rdfs:Class`	`rdfs:Class`
`rdfs:subPropertyOf`	The subject is a subproperty of a property.	`rdf:Property`	`rdf:Property`
`rdfs:domain`	A domain of the subject property.	`rdf:Property`	`rdfs:Class`
`rdfs:range`	A range of the subject property.	`rdf:Property`	`rdfs:Class`
`rdfs:label`	A human-readable name for the subject.	`rdfs:Resource`	`rdfs:Literal`
`rdfs:comment`	A description of the subject resource.	`rdfs:Resource`	`rdfs:Literal`
`rdfs:member`	A member of the subject resource.	`rdfs:Resource`	`rdfs:Resource`
`rdf:first`	The first item in the subject RDF list.	`rdf:List`	`rdfs:Resource`
`rdf:rest`	The rest of the subject RDF list after the first item.	`rdf:List`	`rdf:List`
`rdfs:seeAlso`	Further information about the subject resource.	`rdfs:Resource`	`rdfs:Resource`
`rdfs:isDefinedBy`	The definition of the subject resource.	`rdfs:Resource`	`rdfs:Resource`
`rdf:value`	Idiomatic property used for structured values.	`rdfs:Resource`	`rdfs:Resource`
`rdf:subject`	The subject of the subject RDF statement.	`rdf:Statement`	`rdfs:Resource`
`rdf:predicate`	The predicate of the subject RDF statement.	`rdf:Statement`	`rdfs:Resource`
`rdf:object`	The object of the subject RDF statement.	`rdf:Statement`	`rdfs:Resource`

Table 2: RDF properties

In addition to these classes and properties, RDF also uses properties called `rdf:_1`, `rdf:_2`, `rdf:_3`... etc., each of which is both a sub-property of `rdfs:member` and an instance of the class `rdfs:ContainerMembershipProperty`. There is also an instance of `rdf:List` called `rdf:nil` that is an empty `rdf:List`.

4 Semantics

In order to be able to talk about entailment of RDF graphs, we need to define what an interpretation of an RDF graph is. There exist three different kinds of interpretation: Simple interpretation, RDF interpretation and RDFS interpretation. We will give the definition of a simple interpretation [HKR09], the definitions for RDF and RDFS interpretation are a bit more involved, but can be found here: `http://www.w3.org/TR/2004/REC-rdf-mt-20040210/#InterpVocab`, `http://www.w3.org/TR/2004/REC-rdf-mt-20040210/#rdfs_interp`.

A simple interpretation $\mathcal{I}$ of a vocabulary V consists of

- IR, a non-empty set of resources, alternatively called domain or universe of discourse of $\mathcal{I}$
- IP, the set of properties of $\mathcal{I}$ (which may overlap with IR),
- $\mathrm{I_{EXT}}$, a function assigning to each property a set of pairs form IR, i.e. $\mathrm{I_{EXT}} : IP \rightarrow 2^{IR\times IR}$, where $\mathrm{I_{EXT}}(p)$ is called the *extension* of the property p,
- $\mathrm{I_S}$, a function, mapping URIs from V into the union of the sets IR and IP, i.e. $\mathrm{I_S} : V \rightarrow IR \cup IP$,
- $\mathrm{I_L}$, a function from typed literals from V into the set IR of resources and
- LV, a particular subset of IR, called the set of *literal values*, containing (at least) all untyped literals from V.

Now define an interpretation function $\cdot^{\mathcal{I}}$ (written as exponent).

- every untyped literal “a” is mapped to a, formally: $(\text{“}a\text{”})^{\mathcal{I}} = a$,
- every untyped literal carrying language information “a”@ t is mapped to the pair $\langle a, t\rangle$, i.e. $(\text{“}a\text{”@ } t)^{\mathcal{I}} = \langle a, t\rangle$
- every typed literal l is mapped to $\mathrm{I_L}(l)$, formally: $l^{\mathcal{I}} = \mathrm{I_L}(l)$, and
- every URI u is mapped to $\mathrm{I_S}(u)$, i.e. $u^{\mathcal{I}} = \mathrm{I_S}(u)$.
- The truth value `s p o.`$^{\mathcal{I}}$ of a grounded triple (i.e. a triple not containing a blank node) `s p o.` is true exactly if `s`, `p`, `o` are contained in V and $\langle \mathtt{s}^{\mathcal{I}}, \mathtt{o}^{\mathcal{I}}\rangle \in \mathrm{I_{EXT}}(\mathtt{p}^{\mathcal{I}})$.

Furthermore a grounded RDF graph evaluates to true, iff all of its triples evaluateto true. With this simple interpretation model we are able to assign a truth value to a given grounded graph and can decide if a set of RDF graphs S simple entails another RDF graph E (every interpretation satisfying each member of S has to satisfy E as well). To be able to cover non-grounded graphs we need to extend our interpretation function. This again we will not cover, but give a link: `http://www.w3.org/TR/2004/REC-rdf-mt-20040210/#unlabel`. In addition RDF and RDFS entailment rely on axioms, which we will list in the following [Hay04]:

RDF axiomatic triples.

```
rdf:type rdf:type rdf:Property .
rdf:subject rdf:type rdf:Property .
rdf:predicate rdf:type rdf:Property .
rdf:object rdf:type rdf:Property .
rdf:first rdf:type rdf:Property .
rdf:rest rdf:type rdf:Property .
rdf:value rdf:type rdf:Property .
rdf:_1 rdf:type rdf:Property .
rdf:_2 rdf:type rdf:Property .
...
rdf:nil rdf:type rdf:List .
```

RDFS axiomatic triples.

```
rdf:type rdfs:domain rdfs:Resource .
rdfs:domain rdfs:domain rdf:Property .
rdfs:range rdfs:domain rdf:Property .
rdfs:subPropertyOf rdfs:domain rdf:Property .
rdfs:subClassOf rdfs:domain rdfs:Class .
rdf:subject rdfs:domain rdf:Statement .
rdf:predicate rdfs:domain rdf:Statement .
rdf:object rdfs:domain rdf:Statement .
rdfs:member rdfs:domain rdfs:Resource .
rdf:first rdfs:domain rdf:List .
rdf:rest rdfs:domain rdf:List .
rdfs:seeAlso rdfs:domain rdfs:Resource .
rdfs:isDefinedBy rdfs:domain rdfs:Resource .
rdfs:comment rdfs:domain rdfs:Resource .
rdfs:label rdfs:domain rdfs:Resource .
rdf:value rdfs:domain rdfs:Resource .

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

```
rdfs:member rdfs:range rdfs:Resource .
rdf:first rdfs:range rdfs:Resource .
rdf:rest rdfs:range rdf:List .
rdfs:seeAlso rdfs:range rdfs:Resource .
rdfs:isDefinedBy rdfs:range rdfs:Resource .
rdfs:comment rdfs:range rdfs:Literal .
rdfs:label rdfs:range rdfs:Literal .
rdf:value rdfs:range rdfs:Resource .

rdf:Alt rdfs:subClassOf rdfs:Container .
rdf:Bag rdfs:subClassOf rdfs:Container .
rdf:Seq rdfs:subClassOf rdfs:Container .
rdfs:ContainerMembershipProperty rdfs:subClassOf rdf:Property .

rdfs:isDefinedBy rdfs:subPropertyOf rdfs:seeAlso .

rdf:XMLLiteral rdf:type rdfs:Datatype .
rdf:XMLLiteral rdfs:subClassOf rdfs:Literal .
rdfs:Datatype rdfs:subClassOf rdfs:Class .

rdf:_1 rdf:type rdfs:ContainerMembershipProperty .
rdf:_1 rdfs:domain rdfs:Resource .
rdf:_1 rdfs:range rdfs:Resource .
rdf:_2 rdf:type rdfs:ContainerMembershipProperty .
rdf:_2 rdfs:domain rdfs:Resource .
rdf:_2 rdfs:range rdfs:Resource .
...
```

Most of those rules should seem familiar, as they formally describe what we already listed in the tables in the RDF Schema section.

4.1 Entailment Rules

For completeness we will list all the entailment rules there are in RDF graphs (simple, RDF and RDFS entailment) and give explanations [Hay04]. We make use of the following variables:

- a, b, ... denoting an arbitrary URI reference, i.e. any possible predicate of a triple
- u, v, ... denoting an arbitrary URI reference or blank node identifier, i.e. any possible subject of a triple
- x, y, ... denoting an arbitrary URI reference, blank node identifier or literal, i.e. any possible object of a triple
- l denoting an arbitrary literal
- $\_{:}n$ denoting a blank node identifier

4.1.1 Simple Entailment Rules

Simple Entailment Rules

se1 $\dfrac{u\ a\ x\ .}{u\ a\ \_{:}n\ .}$ where $\_{:}n$ identifies a blank node allocated to x by rule **se1** or **se2**.

se2 $\dfrac{u\ a\ x\ .}{\_{:}n\ a\ x\ .}$ where $\_{:}n$ identifies a blank node allocated to u by rule **se1** or **se2**.

Explanation: For every RDF triple $u\ a\ x$. we can entail another RDF triple where a blank node is allocated to either the subject or object of the triple. This blank node has to be either a new one, or was previously created by either **se1** or **se2**.

Although these are very simple rules, the problem of determining simple entailment between two RDF graphs is NP-complete [Hay04]. One reason for this is that both **se1** and **se2** can be applied to themselves again, resulting in an infinite amount of entailments. Because of this the following more restrictive rules are introduced.

Literal Generalization Rule

lg $\dfrac{u\ a\ l\ .}{u\ a\ \_{:}n\ .}$ where $\_{:}n$ identifies a blank node allocated to the literal l by this rule.

Literal Instantiation Rule

gl $\dfrac{u\ a\ \_{:}n\ .}{u\ a\ l\ .}$ where $\_{:}n$ identifies a blank node allocated to the literal l by rule **lg**.

4.1.2 RDF Entailment Rules

RDF Entailment Rules

rdf1 $\dfrac{u\ a\ y\ .}{a\ \texttt{rdf:type rdf:Property}\ .}$

rdf2 $\dfrac{u\ a\ l\ .}{\_{:}n\ \texttt{rdf:type rdf:XMLLiteral}\ .}$ where $\_{:}n$ identifies a blank node allocated to l by rule **lg**.

RDF Entailment Lemma

S rdf-entails E if and only if there is a graph which can be derived from S plus the RDF axiomatic triples by the application of rule **lg** and the RDF entailment rules and which simply entails E.

4.1.3 RDFS Entailment Rules

RDFS Entailment Rules

rdfs1 $\dfrac{u\ a\ l\ .}{\_{:}n\ \texttt{rdf:type rdf:Literal}\ .}$ where $\_{:}n$ identifies a blank node allocated to l by rule **lg**.

rdfs2 $\dfrac{\begin{array}{c} a\ \texttt{rdfs:domain}\ x\ . \\ u\ a\ y\ . \end{array}}{u\ \texttt{rdf:type}\ x\ .}$

Explanation: Given a predicate a with domain x, then if $\langle u, y\rangle \in \mathrm{I}_{\mathrm{EXT}}(a)$, $u \in x$

rdfs3 $\dfrac{\begin{array}{c} a\ \texttt{rdfs:range}\ x\ . \\ u\ a\ v\ . \end{array}}{v\ \texttt{rdf:type}\ x\ .}$

Explanation: Given a predicate a with range x, then if $\langle u, v\rangle \in \mathrm{I}_{\mathrm{EXT}}(a)$, $v \in x$

rdfs4a $\dfrac{u\ a\ x\ .}{u\ \texttt{rdf:type rdfs:Resource}\ .}$

Explanation: If there is a predicate a relating a URI reference u to an object x, then u is a resource.

rdfs4b $\dfrac{u\ a\ v\ .}{v\ \texttt{rdf:type rdfs:Resource}\ .}$

Explanation: If there is a predicate a relating a URI references u and v, then v is a resource.

rdfs5 $\dfrac{\begin{array}{c} u\ \texttt{rdfs:subPropertyOf}\ v\ . \\ v\ \texttt{rdfs:subPropertyOf}\ x\ . \end{array}}{u\ \texttt{rdfs:subPropertyOf}\ x\ .}$

Explanation: `rdfs:subPropertyOf` is transitive.

$$\textbf{rdfs6}\ \frac{u\ \texttt{rdf:type rdf:Property .}}{u\ \texttt{rdfs:subPropertyOf}\ u\ .}$$

Explanation: Every property is a sub-property of itself.

$$\textbf{rdfs7}\ \frac{\begin{array}{l} a\ \texttt{rdfs:subPropertyOf}\ b\ . \\ \quad u\ a\ y\ . \end{array}}{u\ b\ y\ .}$$

Explanation: If a predicate a is a sub-property of another predicate b, then if a relates subject u and object y b does as well.

$$\textbf{rdfs8}\ \frac{u\ \texttt{rdf:type rdfs:Class .}}{u\ \texttt{rdfs:subClassOf rdfs:Resource .}}$$

Explanation: Every class is a sub-class of `rdfs:Resource`.

$$\textbf{rdfs9}\ \frac{\begin{array}{l} u\ \texttt{rdfs:subClassOf}\ x\ . \\ \quad v\ \texttt{rdf:type}\ u\ . \end{array}}{v\ \texttt{rdf:type}\ x\ .}$$

Explanation: Every class is a sub-class of `rdfs:Resource`.

$$\textbf{rdfs10}\ \frac{u\ \texttt{rdf:type rdf:Class .}}{u\ \texttt{rdfs:subClassOf}\ u\ .}$$

Explanation: Every class is a sub-class of itself.

$$\textbf{rdfs11}\ \frac{\begin{array}{l} u\ \texttt{rdfs:subClassOf}\ v\ . \\ v\ \texttt{rdfs:subClassOf}\ x\ . \end{array}}{u\ \texttt{rdfs:subClassOf}\ x\ .}$$

Explanation: `rdfs:subClassOf` is transitive.

$$\textbf{rdfs12}\ \frac{u\ \texttt{rdf:type rdfs:ContainerMembershipProperty .}}{u\ \texttt{rdfs:subPropertyOf rdfs:member .}}$$

Explanation: Every container membership property is a sub-property of `rdfs:member`.

$$\textbf{rdfs13}\ \frac{u\ \texttt{rdf:type rdfs:Datatype .}}{u\ \texttt{rdfs:subClassOf rdfs:Literal .}}$$

Explanation: Every data-type is a sub-class of `rdfs:Literal`.

RDFS Entailment Lemma
S rdfs-entails E if and only if there is a graph which can be derived from S plus the RDF and RDFS axiomatic triples by the application of rule **lg**, rule **gl** and the RDF and RDFS entailment rules and which either simply entails E or contains an XML clash (an XML clash is a statement of the form `x rdf:type rdfs:Literal .` where xxx is allocated to an ill-typed XML literal by rule **lg**).

Note that although RDFS entailment is decidable, the rules are not complete. Consider the following RDF graph [tH05]:

$$\begin{array}{l} a\ \texttt{rdfs:subPropertyOf}\ \_{:}n. \\ \_{:}n\ \texttt{rdfs:domain}\ x. \\ u\ a\ y. \end{array}$$

This graph entails u `rdf:type` x (proof omitted), however when using the previously listed rules **rdfs7** and **rdfs2** we obtain the triple $u\ \_{:}n\ y$. as an intermediate step violating the rule that the predicate is always a URI reference (and never a blank node). In order to fix this the notion of RDF graphs needs to be extended to allow blank nodes in predicate position. [tH05] does this, showing that completeness can be achieved. Also it gives proofs that this extension as well as entailment for RDFS is decidable, NP-complete, and in P if the target graph does not contain blank nodes. This is a very remarkable result, since it allows for efficient queries of RDF graphs giving rise to the query language SPARQL doing exactly that (`http://www.w3.org/TR/rdf-sparql-query/`).

5 Conclusion

In this essay we have explained what RDF is, and with what goals it was designed. Also we listed and described its concepts Graph data model, URI-based vocabulary, Datatypes, Literals, XML serialization syntax, Expression of simple facts and Entailment. Furthermore we explained another serialization format called N-triples, and talked about the RDF Vocabulary Description Language, RDF Schema. Lastly we looked at the semantics of RDF, describing the different interpretations of an RDF graph (simple, RDF, RDFS), listed RDF axioms and also gave entailment rules together with entailment lemmas. We stated that simple, RDF and RDFS entailment is decidable, and that the problem of entailment is in NP or P, depending on if there are blank nodes in the graphs or not. Also we showed that the given RDF entailment rules are not complete and need to be extended in order to have completeness. RDF is an important standard for the semantic web, as it allows for easy writing down and querying of knowledge. Its wide usage is motivation to transform other type of information on the web into RDF, for example it is possible to extend HTML with RDF attributes, this technique is called RDFa.

References

[Bec04] Dave Beckett. RDF/XML Syntax Specification, W3C Recommendation. `http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/#section-Infoset-Grammar`, February 2004. [Online; accessed 8-January-2013].

[BG04] Dan Brickley and R.V. Guha. RDF Schema, W3C Recommendation. `http://www.w3.org/TR/2004/REC-rdf-schema-20040210/`, February 2004. [Online; accessed 8-January-2013].

[GB04] Jan Grant and Dave Beckett. RDF Test Cases, W3C Recommendation. `http://www.w3.org/TR/2004/REC-rdf-testcases-20040210/#ntriples`, February 2004. [Online; accessed 8-January-2013].

[Hay04] Patrick Hayes. RDF Semantics, W3C Recommendation. `http://www.w3.org/TR/2004/REC-rdf-mt-20040210/`, February 2004. [Online; accessed 8-January-2013].

[HKR09] Pascal Hitzler, Markus Krötzsch, and Sebastian Rudolph. *Foundations of Semantic Web Technologies.* Chapman & Hall/CRC, 2009.

[KC04] Graham Klyne and Jeremy J. Carrol. RDF Concepts and Abstract Syntax, W3C Recommendation. `http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/`, February 2004. [Online; accessed 8-January-2013].

[tH05] Herman J. ter Horst. Completeness, decidability and complexity of entailment for RDF Schema and a semantic extension involving the OWL vocabulary. `http://smtp.websemanticsjournal.org/index.php/ps/article/viewFile/66/64`, May 2005. [Online; accessed 8-January-2013].