Reasoning on the Web: A minimal set of RDFS Axioms.

Andrei Sili, Saranots Tzortzis

[1] andrei.sili@student.uva.nl
[2] sarantos.tzortzis@student.uva.nl
https://github.com/sarantinio/Semantic-Web-RDFs.git
May 26th, 2018

Abstract. In this paper, we try to find the minimal set of RDF(S) axioms, that result in the same closure according to the RDF(S) reasoning rules, called a reduction. In doing so, we also provide a rather thorough analysis of the Stardog RDFS reasoning profile, while also providing theoretical proofs for any empirically derived redundancies in the RDF(S) axiom set.

1 Introduction

The Resource Description Framework (RDF, RDFS) is a standard maintained by the World Wide Web Consortium (W3C). It is described in a set of specification documents, publicly available online [1][2][3][4]. In these documents, specifications are given for RDF concepts, abstract syntax, schema, and semantics.

The main motivation behind the RDF framework is to create a minimally constraining standard that allows for sharing data across web applications in a consistent and reusable format that is process-able by machines [3]. The resource description framework is composed of the following:

1. a graph data model of the form 1
2. a set of formal semantics
3. a rigorously defined notion of entailment

Our focus in this paper is on points 2 and 3 above. That is, we would like to investigate the semantics of the Resource Description Framework. Before doing so, let us first define what is to be understood by the RDF semantics.

The Resource Description Framework provides normative guidelines which describe how entailment can be computed for a given graph. That is, given a set of triples (s, p, o), what other triples (s', p', o') can be deduced. This is achieved in RDF(S) by defining a set of triples called axioms, and a set of rules according to which more triples can be deduced. The pair of axioms and rules is what we call the semantics of RDF(S). The axioms can be though of as a vocabulary, and the rules define the logic that can be applied to this vocabulary.

In this paper, we try to accomplish 2 goals. The main goal is to investigate the minimality of the RDF(S) axioms. That is, given the RDF(S) reasoning rules,

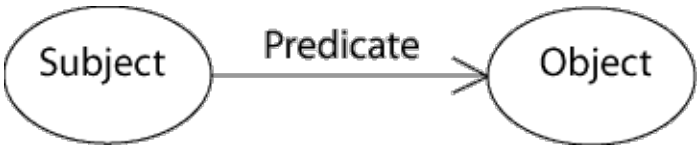

Fig. 1. An example graph data model. *Each statement consists of a triple. In this example, the subject and object are represented as node and the predicate is an edge. However, RDF does not impose a distinction here. A predicate can also be a node. The only normative concepts relating to the data model are: URIs, blank nodes, and Literals.*

can we infer a subset of the axioms, thus proving that they are redundant. In doing so, we will first take an empirical approach using Stardog and its reasoning capabilities. This gives rise to a secondary goal, that of investigating to which extent does Stardog implement all the reasoning rules defined by RDF(S). We also take care to provide formal proofs of the empirically observed redundancies later on.

2 Resource Description Framework: Axioms and Inference Rules

To understand the issue that we are trying to tackle, we start by defining the RDF concepts as described in [3] and its abstract syntax that we will be using throughout this paper.

Graph Data Model This is a collection of triples in the form (s, p, o). The triples are represented as a directed graph, where the arrow always points to the object, like in Figure 1. It states that some relation identified by the predicate exists between the subject and the object.

Node Identification In general, a node in the graph is identified by a URI reference, a blank node, or a Literal. A URI reference uniquely identifies a particular resource on the web. A blank node is a unique resource that is not identified explicitly. A literal expresses some constant value (e.g. a string , an integer, a date, so on). Subjects are either URI references, or blank nodes. Predicates are always URI references, and objects are any of the 3.

Literals Literals identify values such as strings, dates, or numbers. RDF defines 2 types of literals: plain and typed. Plain literals are treated as simple strings and can be used for natural language representation. Typed literals are represented as strings that are tagged with the URI of a datatype.

Datatypes Datatypes are used to represent values of a specific type. They are identified by URIs and provide a value space, a lexical space, and a lexical-to-value mapping. The value space are the actual values that are being represented.

The lexical space holds the representation of the values. The lexical-to-value mapping defines an injective correspondence from the lexical space to the value space.

Entailment In RDF, entailment takes the meaning from model theory, where a statement entails another if for all models in the world that make the first statement true, also the second statement is true. That is, the second can be inferred from the first.

2.1 Axioms

On the basis of these concepts, RDF defines a vocabulary that works at the level of relations between resources in the graph without any assumptions on what those resources are. The RDF vocabulary and axioms are given in Table 1. The RDFS(S) extensions to the vocabulary and axioms are presented in Table 2. It is important to note that the RDF standard does not impose any constraint on the meaning of the the elements of the vocabulary. The intended meaning of this vocabulary is provided as an informative topic, and following this meaning ensures consistency across graphs, but it is not formally enforced in any way. These intended meanings are given in appendix D of the RDF semantics specification [2]. The RDF axioms specify nothing more than the fact that everything in the vocabulary with the exception of $rdf : nil$ is a $rdf : Property$. The RDFS axioms are more involved, defining ranges, domains, multiple relations of type child-parent, and so on. We do not give a full account of what the intended meaning of each of those axioms means. For a full discussion, we refer the reader to the specification [2].

Vocabulary	Axioms
rdf:type	rdf:type rdf:type rdf:Property .
rdf:subject	rdf:subject rdf:type rdf:Property .
rdf:predicate	rdf:predicate rdf:type rdf:Property .
rdf:object	rdf:object rdf:type rdf:Property .
rdf:first	rdf:first rdf:type rdf:Property .
rdf:rest	rdf:rest rdf:type rdf:Property .
rdf:value	rdf:value rdf:type rdf:Property .
rdf:nil	rdf:nil rdf:type rdf:List .
rdf:List	rdf:_1 rdf:type rdf:Property .
rdf:langString	
rdf:Property	
rdf:_1	

Table 1. RDF Vocabulary and Axioms. *There is no row-wise relation between the vocabulary and the axioms. Note that the axioms are technically an infinite set, but for all practical purposes, a single 'underscore' triple can be included because all the rest are equivalent so any entailment for that one instance will hold in general.*

Taken as a unified pair, the RDF(S) vocabulary and axioms give the basis for deriving graph entailment according to a specified set of rules.

2.2 Rules

The RDF inference rules specify ways to derive new information from existing one. These are formed by a set of conditions and a conclusion. The full set of RDF(S) rules can be seen in Table 3. As a note on the notation, in the RDFS rules, $\mathcal{D}$ is considered to be the set of all recognised data types. So, rule RDF1 specifies how typed literals are decomposed to standard RDF triples with the use of a blank node. And the rule RDFS1 simply states that all data types should belong to the class rdf:DataType.

Rule	Statement(s)	Conclusion(s)
RDF1	xxx aaa "sss"^^ddd . for ddd in $\mathcal{D}$	xxx aaa _:nnn . _:nnn rdf:type ddd .
RDF2	xxx aaa yyy .	aaa rdf:type rdf:Property .
RDFS1	any IRI aaa in $\mathcal{D}$	aaa rdf:type rdfs:Datatype .
RDFS2	aaa rdfs:domain xxx . yyy aaa zzz .	yyy rdf:type xxx .
RDFS3	aaa rdfs:range xxx . yyy aaa zzz .	zzz rdf:type xxx .
RDFS4	xxx aaa yyy .	xxx rdf:type rdfs:Resource . yyy rdf:type rdfs:Resource .
RDFS5	xxx rdfs:subPropertyOf yyy . yyy rdfs:subPropertyOf zzz .	xxx rdfs:subPropertyOf zzz .
RDFS6	xxx rdf:type rdf:Property .	xxx rdfs:subPropertyOf xxx .
RDFS7	aaa rdfs:subPropertyOf bbb . xxx aaa yyy .	xxx bbb yyy .
RDFS8	xxx rdf:type rdfs:Class .	xxx rdfs:subClassOf rdfs:Resource .
RDFS9	xxx rdfs:subClassOf yyy . zzz rdf:type xxx .	zzz rdf:type yyy .
RDFS10	xxx rdf:type rdfs:Class .	xxx rdfs:subClassOf xxx .
RDFS11	xxx rdfs:subClassOf yyy . yyy rdfs:subClassOf zzz .	xxx rdfs:subClassOf zzz .
RDFS12	xxx rdf:type rdfs:CMP* .	xxx rdfs:subPropertyOf rdfs:member .
RDFS13	xxx rdf:type rdfs:Datatype .	xxx rdfs:subClassOf rdfs:Literal .

Table 3. RDF(S) Inference Rules. *The rules describe the entailment operations allowed under the RDF specification. In this table, aaa, xxx, yyy, zzz are URIs, blank nodes, or literals with the within-triple location restrictions defined earlier in the paper.* *CMP=ContainerMemberShipProperty

An important note on the entailment specification of RDF is that it is defined as recursive. That is, given a set of triples, a reasoning iteration that produces new triples mandates a new reasoning iteration, until a fixed point is reached when no other entailment can be derived. This means that reasoning trees can be arbitrarily deep, as well as wide.

3 Problem Statement

Given the full sets of RDF(S) axioms and inference rules, one can derive new statements about RDF concepts. In particular, we can derive the set of all pos-

sible asserted and inferred statements, called the *closure*. However, it might be that a smaller set of axioms leads to the same closure. In fact, the RDF Working Group does state rather ambiguously that there is some redundancy in the axioms [2]. We want to find what those redundancies are and remove them to obtain the minimal set of axioms that leads to the same closure, called the *reduction*. So our main research question can be stated as:

Q.1 What is the reduction of the RDF(S) axioms, in relation to the RDF(S) rules?

A natural way to empirically investigate this problem is to remove each RDF axiom one at a time and compute the closure. If the closure is the same, we can infer that the one axiom is redundant, meaning it can be inferred from the other axioms, given the rules of inference. Note, however, that we are not interested only in which axioms are individually redundant in relation to the rest of the axioms set, but rather what set of axioms is redundant. To this end, we may define 2 sub questions relating to our first one.

Q.1.1 Which axioms from the RDF(S) set are individually redundant, with respect to the rest of the RDF(S) axioms, given the RDF(S) rules?

Q.1.2 Which set of axioms from RDF(S) is redundant, with respect to the remaining RDF(S) axioms, given the RDF(S) rules?

In the empirical approach to this problem, we will make use of Stardog's RDF(S) reasoning capabilities. Because we are relying on this external tool for reasoning, we first need to evaluate the completeness and soundness of Stardog's RDF(S) reasoning with respect to the specification reasoning rules. As such, a secondary research question that will be studied before any others can be stated as:

Q.2 How does the Stardog RDF(S) reasoner perform in terms of *soundness* and *completeness* of results with respect to the RDF(S) reasoning rules?

4 Experimental Setup and Approach

Our research method follows 2 steps. Firstly, we derive the closure of the RDF(S) axioms using Stardog's reasoning capabilities. During this time, we validate the completeness and soundness of Stardog's RDF(S) entailment process. In the second step, we provide a formal proof of the derived redundant axioms using RDF(S), in order to further investigate the soundness of the RDF(S) reasoner and for completeness.

4.1 Empirical Setup

We import the 10 RDF axioms and the 40 RDFS axiomatic triples defined in the RDF(S) Semantics specification in Stardog [2], resulting in a dataset of 50 triples. Enabling RDFS reasoning in Stardog, we then retrieve the entire set of axioms (asserted and inferred) using a simple SPARQL query. One of the first things during initial trials is that although we explicitly disable any non-RDFS reasoning, the entailment returned by Stardog contains OWL statements as well. These are very general, such as $owl : Thing$, $owl : TopDataProperty$, $owl : BottomDataProperty$. Strictly speaking, this means that the entailment procedure used by Stardog is not sound, since the *owl* statements cannot be determined to be true based only on the RDF(S) axioms and reasoning rules defined in the specification. Thus, we have a first observation relating to research question *Q.2*.

O.2.1 Stardog RDFS reasoning is not *sound*, since *owl* entailments are derived even though no *owl* vocabulary has been loaded and no *owl* reasoning profile has been enabled.

Although no clear specification is provided in the Stardog documentation, we postulate that Stardog works over an inherent vocabulary, which is embedded into all datasets. Naturally, Since these *owl* entailments are of no interest to our analysis, we filter them out in the SPARQL query using the base URI of *owl* to identify them.

Moving on, a second and more troublesome issue with Stardog RDFS reasoning is that it does not handle entailment recursively. That is, if a particular entailment requires a proof tree that is more that 1 level deep, it will not be recognised by the Stardog reasoner. This can happen, for example, because the set of axioms together with one or more entailed triples result in other valid entailments. This is a known issue according to the Stardog documentation [5] and is problematic, as it means that the reasoning results retrieved by Stardog will generally not be *complete*.

Rule	Entailment
RDF1	ex:x ex:p _:n _:n rdf:type xsd:string
RDFS1	xsd:string rdf:type rdfs:DataType
RDFS1	any IRI aaa in $\mathcal{D}$

Table 4. Entailment based on statement containing literal.

Finally, we notice that some simple entailments based on RDFS rules are also not derived. In particular, Stardog does not recognise the normative set

of data types $\{rdf\,langString,\ xsd\,string\}$ and also does not implement rules $RDF1$, $RDFS1$, $RDFS12$, and $RDFS13$. The procedure for testing this was straightforward. We start by inserting a dummy triple.

ex:x ex:p "hello"^^xsd:string

If this were recognised as a data type by Stardog, we should now see the entailment described in Table 4 based on the rules in Table 3, which is not the case. Furthermore, we would expect to see the entailment:

rdf:_1 rdfs:SubPropertyOf rdfs:member

per rule $RDFS12$, which is also not in the Stardog closure. Note that by no means is our analysis of the Stardog reasoner complete, but it provides a starting point for a more detailed analysis, which we leave for future initiatives. We can now state a second observation relating to research question *Q.2* as follows:

O.2.2 Stardog RDFS reasoning is generally not *complete*, since entailment is not derived recursively and rules $RDF1$, $RDFS1$, $RDFS12$ are not implemented.

We proceed by manually retrieving the first entailment from Stardog using a SPARQL query. We load the derived axioms into the dataset and repeat the procedure until we reach a fixed point. In total, it seems that the full closure computed by Stardog contains 117 triples.

4.2 Algorithmic Approach

In order to derive redundant axioms, we make use of the following lemma. If there is a redundant axiom in the set of 50 axioms collected, then removing one (or conversely importing only 49 triples in Stardog) should recursively lead to the same closure with 117 triples according to RDFS reasoning.

The algorithm we implement loops over each one of the 50 axioms, removing every time one of them. Because Stardog does not handle entailment recursively, we implement this ourselves by getting the Stardog closure, and checking against the previous closure. If the closures are identical, we have reached a fixed point where all possible triples have been inferred. If not, we import the retrieved closure into Stardog and rerun the algorithm. If the closure meets the same amount of triples (117) then the axiom removed is referred to be redundant. The Figures 1 and 2 illustrate the main steps of the algorithm.

5 Empirical Results and Discussion

After running the algorithm, the CSV file is created indicating whether an axiom is redundant and the total closure number. An example of this file is illustrated in Table 5. There were a number of other axioms that our algorithm identified

Algorithm 1 generate_closure_report
Input: A: the set of 50 axioms, Γ: total closure (117)
Output: E: the report of the full closure for all axioms to .csv
for a in A **do**
  reset_dataset(A) — put the database in initial state
  $B := A \setminus \{a\}$ — remove axiom a from the set
  $C :=$ get_recursive_closure_count(B) — get the count of the full closure
  if $C = \Gamma$ **then**
    mark_redundant(a) — mark axiom as redundant in the report
  end if
  generate_closure_report(C) — write axiom report to a .csv file
end for
return rdfs_closure_report.csv

Algorithm 2 get_recursive_closure_count
Input: R: the new set of axioms, *old_count*: set 0 by default
Output: F: the total closure after the recursion meets a fix point.
count := get_closure_count(R) — get the Stardog entailment count
if *count* = *old_count* **then**
  return *count*
else
  closure = get_closure_turtle(R) — get the Stardog entailment triples
  remove_data(R) — remove current dataset to prevent duplicate entailment
  add_new_data(*closure*) — add the Stardog entailment
  return get_recursive_closure(*closure*,*count*) — run recursion until fix point
end if

as redundant. Table 6 holds all these axioms. We observe that there is a form of clustering in terms of grouped axioms that appear redundant. For example there are some axioms that carry the same object-predicate pair (*rdf:type rdf:property*).

Excluded Axiom	Closure Count	Data Count	Redundant
rdf:type rdf:type rdf:Property	117	49	**True**
rdf:type rdfs:domain rdfs:Resource	86	49	False

Table 5. Example rows of the exported csv file.

Sub	Obj	Pred
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 .
rdfs:seeAlso	rdfs:domain	rdfs:Resource .
rdfs:seeAlso	rdfs:range	rdfs:Resource .
rdfs:isDefinedBy	rdfs:domain	rdfs:Resource .
rdfs:isDefinedBy	rdfs:range	rdfs:Resource .
rdfs:isDefinedBy	rdfs:subPropertyOf	rdfs:seeAlso .
rdfs:comment	rdfs:domain	rdfs:Resource .
rdfs:comment	rdfs:range	rdfs:Literal .
rdfs:label	rdfs:domain	rdfs:Resource .
rdfs:label	rdfs:range	rdfs:Literal .

Table 6. Axioms indicated as redundant.

Lets take for instance the axiom:

rdf:subject rdf:type rdf:property

In order to prove that this axiom is redundant we will have to derive this axiom using other axioms applying to them the RDFS rules to find new entailment. The RDFS2 rule from Table 3 states:

Premises	Conclusion
aaa rdfs:domain xxx . yyy aaa zzz .	yyy rdf:type xxx .

So if we identify as premises the axioms

rdfs:domain rdfs:domain rdf:Property .
rdf:subject rdfs:domain rdf:Statement .

we can conclude:

rdf:subject rdf:type rdf:Property .

So we managed to derive the axiom as we set out to do. Thus we can conclude that the axiom *rdf:subject rdf:type rdf:property .* is redundant as it can be inferred through the other two axioms mentioned.

The same rule can be applied for the rest of the axioms:

rdf:type 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 .

The next redundant axiom to investigate is:

rdfs:isDefinedBy rdfs:subPropertyOf rdfs:seeAlso .

The RDFS5 rule states:

Premises	Conclusion
xxx rdfs:subPropertyOf yyy . yyy rdfs:subPropertyOf zzz .	xxx rdfs:subPropertyOf zzz .

That means that in order to infer this triple we should find premises where

rdfs:isDefinedBy rdfs:subPropertyOf yyy .
yyy rdfs:subPropertyOf rdfs:seeAlso .

But there are no axioms that hold these apart from the

rdfs:isDefinedBy rdfs:subPropertyOf rdfs:seeAlso .

That means that this axioms in not actually redundant.

For the rest of axioms indicated as redundant as well, further investigation should be done, as based on the RDFS rules these axioms are not redundant. We can explain this issue with the example axiom:

rdfs:seeAlso rdfs:domain rdfs:Resource .

In order for this triple to be redundant we should be able to find a rule that could produce this axiom. Specificall, at the very least, we would need a rule with a conclusion of the form:

xxx rdfs:domain yyy

but this does not exist in the specification. The same logic applies to the rest of the axioms that could not be proved as redundant. In order to understand deeper why these axioms appeared redundant we should look at the *RDFS Reasoning* algorithm that Stardog has integrated. However, at this moment we can make another observation relating to research question *Q.2*:

O.2.3 Stardog RDFS reasoning is not sound, since some rules are derived to be redundant when there is no achievable proof of this.

As a final remark, one should notice that not only the individual triples of the form (*xxx rdf:type rdf:Property*) are redundant, but in fact the whole set is redundant. This can be easily seen, since each individual triple is redundant and no triple from this redundant set plays a role in deriving any other. As such, we can make 2 observations relating to our initial research question.

O.1.1 All axioms of the form (*xxx rdf:type rdf:Property*) are redundant given the set of RDF(S) axioms and rules, and can be proven using rules $RDFS2$.

O.1.2 The set of all axioms of the form (*xxx rdf:type rdf:Property*) is redundant given the set of RDF(S) axioms and rules, since no individual in the set plays a role in inferring any other from this set.

6 Summary and Conclusion

To conclude, the bulk of this research ended up being focused on the particularities of the Stardog RDFS reasoning profile. In relation to our first research question, we have found that all but one of the *rdf* axioms are redundant, with the non-redundant one being (*rdf:nil rdf:type rdf:List*) and we have provided a formal proof for this. This serves just to make explicit what the RDF Working Group has already stated vaguely in the semantics specification[2].

In relation to the second research question, we found that the implementation is neither *sound* nor *complete* since, among others, *owl* statements are inferred and entailment is not performed recursively. We also notice that some derived redundancies are not in fact redundant formally, adding to the lack of soundness of the reasoner. Overall, it seems that the Stardog RDFS reasoner profile cannot be trusted as a up-to-spec implementation.

We end with some ideas for future work. As mentioned, we have not entirely explored the Stardog reasoner. A more thorough analysis would be required to pinpoint all issues and edge cases, for example, by comparing the closure of Stardog with a validated RDFS reasoner. Moreover, although we suspect there are no other redundancies, we cannot be sure since some reasoning rules were not implemented in Stardog, meaning that other axioms might be derived as redundant once all these rules are taken into consideration.

References

1. 2014.
2. Patrick Hayes and Peter Patel-Schneider. Rdf semantics, 2014.
3. Carroll Jeremy nd McBride Brian Klyne, Graham. Resource description framework (rdf): Concepts and abstract syntax, 2014.
4. Miller Erik Manola, Frank and Brian McBride. Rdf primer, 2014.
5. Stardog Union. Stardog 5: The manual, 2018.

Vocabulary	Axioms
rdfs:domain	rdf:type rdfs:domain rdfs:Resource .
rdfs:range	rdfs:domain rdfs:domain rdf:Property .
rdfs:Resource	rdfs:range rdfs:domain rdf:Property .
rdfs:Literal	rdfs:subPropertyOf rdfs:domain rdf:Property .
rdfs:Datatype	rdfs:subClassOf rdfs:domain rdfs:Class .
rdfs:Class	rdf:subject rdfs:domain rdf:Statement .
rdfs:subClassOf	rdf:predicate rdfs:domain rdf:Statement .
rdfs:subPropertyOf	rdf:object rdfs:domain rdf:Statement .
rdfs:member	rdfs:member rdfs:domain rdfs:Resource .
rdfs:Container	rdf:first rdfs:domain rdf:List .
rdfs:CMP*	rdf:rest rdfs:domain rdf:List .
rdfs:comment	rdfs:seeAlso rdfs:domain rdfs:Resource .
rdfs:seeAlso	rdfs:isDefinedBy rdfs:domain rdfs:Resource .
rdfs:isDefinedBy	rdfs:comment rdfs:domain rdfs:Resource .
rdfs:label	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 .
	rdfs:Datatype rdfs:subClassOf rdfs:Class .
	rdf:_1 rdf:type rdfs:CMP* .
	rdf:_1 rdfs:domain rdfs:Resource .
	rdf:_1 rdfs:range rdfs:Resource .

Table 2. RDFS Vocabulary and Axioms *There is no row-wise relation between the vocabulary and the axioms. Note that the axioms are technically an infinite set, but for all practical purposes, a single 'underscore' triple can be included because all the rest are equivalent so any entailment for that one instance will hold in general.* *CMP=ContainerMembershipProperty