A note on the history of the document: RDF Formalization, by Draltan Marin

Claudio Gutierrez
Computer Science Department,
Universidad de Chile

Santiago de Chile, May 2006

Draltan Marin, in 2004 a student of the *Programmes Masters of the Ecole Polytechnique, France*, spent the (austral) winter 2004 at the Computer Science Department of Universidad de Chile doing a Research Internship (Training Agreement 16/03/2004).

The topic of this research was: *Applications de la Logique à la sémantique du web*, and was carried out by Draltan during the months of May to July 2004, under the supervision of Claudio Gutiérrez (Universidad de Chile) and Gilles Dowek (Ecole Polytechnique).

The output of this research was the Report on the Formalization of RDF that follows, which was presented at the Ecole Polytechnique by Draltan Marin in September 2004. It contains a formalization in mathematical (logic) language of the W3C Recommendation of 10 February 2004, *RDF Semantics*, edited by P. Hayes, including suggestions and corrections.

The report circulated among some researchers, but was never published. Visiting Bolzano in May 2006, and discussing missing rules of the RDF Semantics document of the W3C based on this report, Enrico Franconi suggested me to make it public. Thus, following his suggestion, I am making it public as a Technical Report of the Year 2006.

RDF formalization

Draltan Marin
draltan.marin@polytechnique.org

August 25, 2004

Contents

1 Introduction

RDF stands for Resource Description Framework, and it is an assertional language intended to provide the necessary means to express propositions about almost any matter.

This framework was first created in 1998 and it was originally designed to achieve goals such as describing properties and relationship between resources in a machine-understandable way, thus permitting to make automated inferences about these resources.

A resource can be practically anything, it can be an url as well as an object such as a table, or as abstract as an idea or thought.

A proposition in RDF has a graph like structure. The meaning of one such a proposition will be given by a formal semantic which will tell us what can be inferred from it. RDF has a simple semantic which can be extended in order to adapt this framework to a wide variety of uses.

This document pretends to present as formal as possible the syntactic and semantic aspects of RDF, leaving apart the serialization or possible implementations of it. We have based this work on the documents presenting the formal syntax [RDF-Concepts] and abstract semantics [RDF-Semantics] which are two of the six documents which specify RDF. The last version at the time of writing this document of these specifications are recommendations of the W3C of 10th February 2004.

We will focus on section 2 on the basic definitions and structure of a proposition in RDF. On sections 3, 4 and 5 we will define its semantics from two different points of view, the first based on a model theory for RDF and some of its possible extensions, the notion of entailment among rdf graphs will then be introduced. The second based on a deductive system of rules which we will prove to be equivalent to the previous definitions, in the sense of entailment.

On section 6 we will show why we think that RDF is a fragment of first order logic.

During the creation of this document we have detected some problems in the definitions and we think that there are some concepts which might be reviewed in a next version of RDF specifications, these comments will be presented in the last section.

2 Formal syntax

In this section we will present the formal syntax of an rdf graph.

Informally the nodes of an rdf graph can be either an URI (Uniform Resource Identifier), a literal or a blank node. The arcs can only be URIs. Note that since an URI can be both a node and an arc the structure of an rdf graph is not exactly this of a normal graph. Intuitively an uri will represent a resource,

a blank node will represent an unidentified resource, and a literal will be a character string or some other type of data.

The basic unit of a graph is a triple, which consists of a subject, a predicate and an object. For example, a triple (s, p, o) can be seen as an oriented node arc node structure in which s and o are the nodes and p is an arc going from s to o. The intended meaning of one such a triple is that the binary property p holds for the couple (s, o). A graph will be a set of triples, where the sets of nodes and arcs will be the union of the nodes and arcs of each triple respectively.

However, this intended meaning is not important for the moment, and will be formalized when defining the semantic of one such a graph. All the following definitions are to be seen as formal objects only.

2.1 Graph definitions

Definition 1 (RDF URI reference) *An* RDF URI reference *consist of a character string which produces a valid URI character sequence (according to [URI]). Two RDF URI references are equal if and only if they compare as equal, character by character. We will note $\mathcal{U}$ the set of all RDF URI references.*

Definition 2 (Datatype) *A* datatype *d is defined by a set L_d of character strings, called* lexical space*, an arbitrary set V_d called the* value space *and a map $\phi_d : L_d \to V_d$ called the* lexical-to-value mapping*.*

Definition 3 (Literal) *A* lexical form *is an Unicode character string. A* literal *consist of either a* lexical form *ω combined with an optional language tag*[1] *τ in which case it is called a* plain literal *and we will note $< \omega, \tau >$ (or simply ω when there is no language tag); or a* lexical form *ω combined with an RDF URI reference u in which case it is called a* typed literal *and we will note $\omega + u$. The set of all literals will be called $\mathcal{L}$.*

Literals are distinct and distinguishable from RDF URI references ($\mathcal{U} \cap \mathcal{L} = \varnothing$).

Let $\mathcal{B}$ be an arbitrary infinite set disjoint from $\mathcal{U} \cup \mathcal{L}$.

Definition 4 (Blank node) *An element of $\mathcal{B}$ is called a* blank node*.*

Definition 5 (RDF triple) *An* RDF triple *is an element of $\mathcal{U} \cup \mathcal{B} \times \mathcal{U} \times \mathcal{U} \cup \mathcal{B} \cup \mathcal{L}$. Its first component is called the* subject *of the triple, the second the* predicate *and the third the* object*.*

Definition 6 (RDF graph) *An* RDF graph *or simply* graph *is a set of RDF triples. A* subgraph *(resp.* proper subgraph*) of a graph is a subset (resp. proper subset) of triples in the graph.*

Informally a datatype is identified by at least one RDF URI reference. This identification will be better explained when defining the datatype interpretations of an RDF graph.

RDF includes a built-in datatype *XMLLiteral* whose RDF URI reference is:

[1] A language tag may be used for identifying the language of a plain text when it's a natural language. Language tags are defined in [RFC-3066]

`http://www.w3.org/1999/02/22-rdf-syntax-ns#XMLLiteral`

and is fully[2] described in [RDF-Concepts].

The RDF URI reference in a typed literal is informally meant to be a datatype RDF URI identification. Then a typed literal can be seen as a couple (string, datatype), and will be well-typed if the string belongs to the lexical space of the corresponding datatype, otherwise it will be called ill-typed.

2.2 Working with graphs

The following definitions are useful for stating syntactic relationship between graphs, such as isomorphic graphs and instances of a graph. We will not treat in detail every one of these notions and the importance they might have.

Definition 7 (Map) *A* map *is a function* $\mu : \mathcal{U} \cup \mathcal{B} \cup \mathcal{L} \to \mathcal{U} \cup \mathcal{B} \cup \mathcal{L}$ *such that* $\mu|_{\mathcal{U}} = Id_{\mathcal{U}}$ *and* $\mu|_{\mathcal{L}} = Id_{\mathcal{L}}$*. For any triple* (s, p, o) *define* $\mu(s, p, o)$ *as the triple* $(\mu(s), \mu(p), \mu(o))$ *and for a graph* G*,* $\mu(G)$ *the image by* μ *of the set of triples* G*. A map* μ *is* consistent *with* G *if* $\mu(G)$ *is an RDF graph, that is, if for any subject* s *in a triple of* G *we have* $\mu(s) \in \mathcal{U} \cup \mathcal{B}$*.*

Definition 8 (Isomorphism) *Two graphs* G_1*,* G_2 *are* isomorphic *(*$G_1 \cong G_2$*) if and only if there exists a map* μ *such that* $\mu(G_1) = G_2$ *and such that* $\mu|_{\mathcal{B}_{G_1}}$ *is bijective where* $\mathcal{B}_{G_1}$ *is the set of blank nodes that occur in* G_1*.*

Definition 9 (Ground graph) *A triple with no blank nodes is a* ground triple*. A graph whose triples are all ground is called a* ground graph*.*

Definition 10 (Occurence) *We say that* $x \in \mathcal{U} \cup \mathcal{B} \cup \mathcal{L}$ occur *in a graph* G *if* x *is the subject, the object or the predicate of any triple in* G*.*

Definition 11 (Name, Vocabulary) *A* name *is a literal or a RDF URI reference. A typed literal comprises two names: itself and the RDF URI reference which is part of it.*
A vocabulary *is a set of names. The* vocabulary of a graph *is the set of all names which occur in the graph.*

Note that the RDF URI reference which is part of a typed literal does not necessarily belong to the vocabulary of a graph G.

Definition 12 (Instance) G_1 *is an* instance *of* G_2 *if there exist a map* μ *consistent with* G_2 *such that* $G_1 = \mu(G_2)$*. We say that* G_1 *is an* instance with respect to a vocabulary V *if* $\mu(\mathcal{B}) \subseteq V \cup \mathcal{B}$*, and* G_1 *is a* proper instance *if* $\mu^{-1}(\mathcal{B}_{G_1}) \subsetneq \mathcal{B}_{G_2}$*, where* $\mathcal{B}_{G_1}$ *and* $\mathcal{B}_{G_2}$ *are the sets of blank nodes which occur in* G_1 *and* G_2 *respectively.*

Note that any graph is an instance of itself, an instance of an instance of G is an instance of G, and if H is an instance of G then every triple in H is an instance of some triple in G.

[2]See comments on last section

Definition 13 (Lean graph) *A graph is* lean *if it has no instance which is a proper subgraph of it.*

3 Interpretations

In this section we will add meaning to rdf graphs. In order to do so we will follow the same construction of interpretations used in [RDF-Semantic] except for the notation.

Depending on the meaning we want to give to a certain graph we will consider different kinds of interpretations (e.g. simple, rdf, rdfs, xsd, etc.). For each one of them there will be some special semantic conditions.

Intuitively, an interpretation will represent a possible configuration of the world, such that we can verify wether or not what is said on a graph G is true on this world. This leads us to think of a graph as something which restricts the possible worlds we are looking at (those in which G is true), thus providing some information.

Now, given two graphs, G and H, if all the worlds in which G is true are such that H is also true we will say that H is entailed by G. In terms of information we can think of this notion as: All the information provided by H is already provided by G. Another way of thinking of this is that H is a semantic consequence of G, in that case we can say that from G we can "infer" H.

The same reasoning applies to any kind of interpretation, and this will be the departure point to describe semantic relationship among rdf graphs.

As described in [RDF-Semantic] any interpretation is relative to a certain vocabulary, so we will in general speak of an interpretation of the vocabulary V. However, at the end of this section, we will show that in terms of entailment this is not relevant, only the semantic conditions which describe the kind of interpretation are.

As we announced earlier, a triple (s, p, o) can be thought of as stating that a certain binary predicate associated to p holds for the couple (s, o). An interpretation will give us this association, and given a graph, it will be true if none of its triples state something false.

Blank nodes will play a special role when evaluating the value of truth of a given graph, because every one of them represents *a priori* some unknown element of the world. To interpret a blank node we will have the freedom of choosing an arbitrary element in the domain of the interpretation as long as it is the same element for all the occurrences of the same blank node.

3.1 Simple interpretation

Definition 14 *A* simple interpretation I of a vocabulary V *is defined by:*
$\mathcal{I}_R \neq \varnothing$*: the set of resources, called the* domain *or* universe *of* I*.*
$\mathcal{I}_P$*: the set of* properties *of* I*.*
$\mathcal{L}_V \subseteq \mathcal{I}_R$*: the set of literal values which contains all plain literals in* V*.*

$\Gamma : \mathcal{I}_P \mapsto \mathcal{P}(\mathcal{I}_R \times \mathcal{I}_R)$
$|[\cdot]|_I : (\mathcal{U} \cup \mathcal{L}) \cap V \to \mathcal{I}_R \cup \mathcal{I}_P$ *such that if* x *is a plain literal then* $|[x]|_I = x$*, and if it is a typed literal* $|[x]|_I \in \mathcal{I}_R$*.*

Given an interpretation $I = \{\mathcal{I}_R, \mathcal{I}_P, \mathcal{L}_V, \Gamma, |[\cdot]|_I\}$ consider the function $|[\cdot]|^I$ defined by:

If t is a ground triple (s, p, o):

$$|[t]|^I = \begin{cases} true \text{ if } s, p, o \in V, |[p]|_I \in \mathcal{I}_P \text{ and } (|[s]|_I, |[o]|_I) \in \Gamma(|[p]|_I) \\ false \text{ otherwise} \end{cases}$$

If E is a ground graph:

$$|[E]|^I = \begin{cases} false \text{ if } \exists t \in E \text{ such that } |[t]|^I = false \\ true \text{ otherwise} \end{cases}$$

Let $A : \mathcal{B}' \to \mathcal{I}_R$ be a mapping from a set of blank nodes ($\mathcal{B}' \subseteq \mathcal{B}$) to the domain of I. Consider the extension $|[\cdot]|_{I+A}$ of $|[\cdot]|_I$ defined by $|[b]|_{I+A} = A(b)$ when b is a blank node in $\mathcal{B}'$. Then we can naturally define $|[\cdot]|^{I+A}$ for triples and graphs as we did before. Now we can extend the definition of $|[\cdot]|^I$ for non-ground graphs as:

If E is an RDF graph let $\mathcal{B}_E$ be the set of blank nodes in E:

$$|[E]|^I = \begin{cases} true \text{ if there exists } A : \mathcal{B}_E \to \mathcal{I}_R \text{ such that } |[E]|^{I+A} = true \\ false \text{ otherwise} \end{cases}$$

Note that blank nodes work as if they were existential variables, and properties can be assigned both a binary predicate and an element of the domain of the interpretation.

Definition 15 (Entailment) *I* satisfies *G if $|[G]|^I = true$, in that case we will note $I \models G$ otherwise $I \not\models G$. Let S be a set of graphs, and G a graph, then S* simply entails *G if and only if for every interpretation[3] I we have: $(\forall H \in S, I \models H) \Rightarrow I \models G$.*
In that case we note $S \models G$.

Note that if $I \models E$ then $I \models E'$ for every $E' \cong E$. This is, if we rename the blank nodes of E to obtain E', since there exists $A : \mathcal{B}_E \mapsto \mathcal{I}_R$ such that $|[E]|^{I+A} = true$ we can find $A' : \mathcal{B}_{E'} \to \mathcal{I}_R$ satisfying $|[E']|^{I+A'} = true$, in fact we can choose $A' = A \circ \mu^{-1}$, where μ is a bijective map which makes $E' = \mu(E)$. We can see also that the restriction of the blank nodes mapping A to a certain subset $\mathcal{B}'$ of $\mathcal{B}$ is not necessary, because since $\mathcal{I}_R \neq \varnothing$ we can always extend A to the whole set of blank nodes $\mathcal{B}$.

Definition 16 (Validity) *Any process which constructs a graph G from some other graph(s) S is said to be (simply)* valid *if S entails G in every case, otherwise* invalid*.*

[3] In fact, for this definition to be correct we should say: for every vocabulary V, for every interpretation of V ...

3.2 Interpreting RDF Vocabulary

In the rest of this document the prefix `rdf:` is an abbreviation for:
`http://www.w3.org/1999/02/22-rdf-syntax-ns#`

The RDF vocabulary *rdfV* is the following set of URI references:

- `rdf:type rdf:Property rdf:XMLLiteral rdf:value`
- (Reification vocabulary:) `rdf:Statement rdf:subject rdf:predicate rdf:object`
- (Container vocabulary:) `rdf:Seq rdf:Bag rdf:Alt rdf:_1 rdf:_2 ...`
- (Collection vocabulary:) `rdf:nil rdf:List rdf:first rdf:rest`

3.2.1 RDF Axiomatic triples

The following set of triples is called *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`)

3.2.2 RDF Interpretation

Let L_{XMLlit},V_{XMLlit} and ϕ_{XMLlit} be the lexical space, value space and lexical to value mapping of the `XMLLiteral` datatype.

Definition 17 (RDF interpretation) *An* rdf-interpretation of V *is a simple interpretation* I *of* $(V \cup rdfV)$ *which satisfies the extra conditions:*

- $x \in \mathcal{I}_P \Leftrightarrow (x, |[\texttt{rdf:Property}]|_I) \in \Gamma(|[\texttt{rdf:type}]|_I)$

- If l is the typed literal: ω+`rdf:XMLLiteral` $\in V$ then:
  if $\omega \in L_{XMLlit}$ then: $|[l]|_I = \phi_{XMLlit}(\omega)$, $|[l]|_I \in \mathcal{L}_V$ and $(|[l]|_I, |[\texttt{rdf:XMLLiteral}]|_I) \in \Gamma(|[\texttt{rdf:type}]|_I)$
  otherwise $|[l]|_I \notin \mathcal{L}_V$ and $(|[l]|_I, |[\texttt{rdf:XMLLiteral}]|_I) \notin \Gamma(|[\texttt{rdf:type}]|_I)$
- I satisfies the RDF axiomatic triples.

Definition 18 (RDF-entailment) *Let S be a set of graphs, and G a graph, then S* rdf-entails *G if and only if for every rdf-interpretation I we have: $(\forall H \in S, I \models H) \Rightarrow I \models G$.*
In that case we note $S \models_{rdf} G$.

3.2.3 Reification, Containers and Collections

Note that there are no special semantic conditions to restrict the meaning of this vocabulary. However there is an intended meaning for those names. We will not treat in detail this subject because it is not relevant for the purposes of this document. In fact finding an appropriate way to give semantic conditions for this vocabulary is not easy and may be a subject of further research. Instead we would like to point out the following aspects:

Reification vocabulary is intended to be used when talking about rdf triple like statements. For example, the statement which has a subject u a predicate x and an object v can be represented by the following graph ($\_b$ is a blank node):

$G = \{(\_b$,`rdf:type`,`rdf:Statement`$)$,
$(\_b$,`rdf:subject`$, u)$,
$(\_b$,`rdf:predicate`$, x)$,
$(\_b$,`rdf:object`$, v)\}$

It is important to note that in general neither does this graph entail the graph:

$H = \{(u, x, v)\}$

nor does H entails G.

In fact G only states that there is an element $\_b$ which is of type statement and has u, x and o as subject, predicate and object respectively, but H states that a certain property associated to x holds for the couple (u, v).

About container vocabulary we can say that its intended use is to represent sets of elements. There are different types of containers: `rdf:Seq` is considered to be an ordered set, `rdf:Alt` is considered to represent a set of alternatives, and `rdf:Bag` is considered to represent an unordered set which may have repeated elements.

Here there is an example to illustrate the intended use of this vocabulary:

$G = \{(\_b$,`rdf:type`,`rdf:Alt`$)$,
$(\_b$,`rdf:_1`$, u)$,
$(\_b$,`rdf:_2`$, v)$,
$(\_b$,`rdf:_3`$, w)\}$

In this example $\_b$ represents a container of type `rdf:Alt` whose first element is u, its second v and its third w. However nothing prevents us to have the following graph:

$G' = \{(\_b$,rdf:type,rdf:Alt),
$(\_b$,rdf:_1,$u)$,
$(\_b$,rdf:_1,$v)$,
$(\_b$,rdf:_4,$w)\}$

Note that the first element of $\_b$ is both u and v. Note also that this second graph does not entails that there exist a second or third element of this container. In fact there is no way of stating how many elements a container may have. Similar problems arise when using rdf:Bag or rdf:Seq.

Collection vocabulary is used to describe list structures. An example of this is:

$G' = \{(x$,rdf:type,rdf:List),
$(x$,rdf:first,$u)$,
$(x$,rdf:rest,$v)$,
$(v$,rdf:first,$w)$,
$(v$,rdf:rest,rdf:nil)$\}$

Even thought rdf:nil is meant to represent the empty list and is used to mark the end of the list, nothing prevents us to have a list without an end, or a list with multiple endings, etc. For example:

$G' = \{(x$,rdf:type,rdf:List),
$(x$,rdf:first,$u)$,
$(x$,rdf:rest,$v)$,
$(v$,rdf:first,$w)$,
$(v$,rdf:rest,rdf:nil)
$(v$,rdf:rest,$z)$,
$(z$,rdf:first,$u)$
$(z$,rdf:rest,rdf:nil)$\}$

All these pathological cases we have just seen are perfectly possible and the lack of semantic conditions for these vocabularies requires us to be careful when using it.

3.3 Interpreting RDFS Vocabulary

In the rest of this document the prefix rdfs: is an abbreviation for:
http://www.w3.org/2000/01/rdf-schema#

The RDFS vocabulary *rdfsV* is the following set of URI references:

- rdfs:domain rdfs:range
- rdfs:Resource rdfs:Literal rdfs:Datatype
- rdfs:Class rdfs:subClassOf rdfs:subPropertyOf
- rdfs:member rdfs:Container rdfs:ContainerMembershipProperty
- rdfs:comment rdfs:seeAlso rdfs:isDefinedBy rdfs:label

3.3.1 RDFS Axiomatic triples

The following set of triples is called *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)
- ...

3.3.2 RDFS Interpretation

Definition 19 (RDFS interpretation) *An* rdfs-interpretation of V *is an rdf-interpretation* I *of* $(V \cup rdfV \cup rdfsV)$ *plus a set* $\mathcal{I}_C \subset \mathcal{I}_R$ *and a map* $\Gamma_C : \mathcal{I}_C \rightarrow \mathcal{P}(\mathcal{I}_R)$ *which satisfy the extra conditions:*

- $x \in \Gamma_C(y) \Leftrightarrow (x, y) \in \Gamma(|[\texttt{rdf:type}]|_I)$
- $\mathcal{I}_C = \Gamma_C(|[\texttt{rdfs:Class}]|_I)$
- $\mathcal{I}_R = \Gamma_C(|[\texttt{rdfs:Resource}]|_I)$

- $\mathcal{L}_V = \Gamma_C(|[\texttt{rdfs:Literal}]|_I)$
- $(x, y) \in \Gamma(|[\texttt{rdfs:domain}]|_I)$ and $(u, v) \in \Gamma(x) \Rightarrow u \in \Gamma_C(y)$
- $(x, y) \in \Gamma(|[\texttt{rdfs:range}]|_I)$ and $(u, v) \in \Gamma(x) \Rightarrow v \in \Gamma_C(y)$
- $\Gamma(|[\texttt{rdfs:subPropertyOf}]|_I)$ is reflexive and transitive on $\mathcal{I}_P$
- $(x, y) \in \Gamma(|[\texttt{rdfs:subPropertyOf}]|_I) \Rightarrow x, y \in \mathcal{I}_P$ and $\Gamma(x) \subset \Gamma(y)$
- $x \in \mathcal{I}_C \Rightarrow (x, |[\texttt{rdfs:Resource}]|_I) \in \Gamma(|[\texttt{rdfs:subClassOf}]|_I)$
- $(x, y) \in \Gamma(|[\texttt{rdfs:subClassOf}]|_I) \Rightarrow x, y \in \mathcal{I}_C$ and $\Gamma_C(x) \subset \Gamma_C(y)$
- $\Gamma(|[\texttt{rdfs:subClassOf}]|_I)$ is reflexive and transitive on $\mathcal{I}_C$
- $x \in \Gamma_C(|[\texttt{rdfs:ContainerMembershipProperty}]|_I) \Rightarrow (x, |[\texttt{rdfs:member}]|_I) \in \Gamma(|[\texttt{rdfs:subPropertyOf}]|_I)$
- $x \in \Gamma_C(|[\texttt{rdfs:Datatype}]|_I) \Rightarrow (x, |[\texttt{rdfs:Literal}]|_I) \in \Gamma(|[\texttt{rdfs:subClassOf}]|_I)$
- I satisfies the RDFS axiomatic triples.

Note that since an rdfs interpretation is an rdf interpretation then for an ill typed XML literal l, this is a typed literal whose uri is `rdf:XMLLiteral` and such that its lexical form does not belong to the lexical space of the XMLLiteral datatype, we must have that $|[l]|_I \notin \mathcal{L}_V$.

Now by the first, fourth and sixth conditions of an rdfs interpretation we have that a graph such as:

$G = \{(a,\texttt{rdfs:range},\texttt{rdfs:Literal}), (x, a, l)\}$

has no interpretation if l is an ill typed XML literal.

In that case, this is when there is no rdfs interpretation which satisfies a graph, we will say that it is an XML clash. Note that since this is the only negative condition all the XML clashes are product of stating in a graph that an ill typed literal is of type literal.

For rdf interpretations this problem does not arise because there is no possibility of stating one such a thing.

Definition 20 (RDFS-entailment) *Let S be a set of graphs, and G a graph, then S* rdfs-entails *G if and only if for every rdfs-interpretation I we have:* $(\forall H \in S, I \models H) \Rightarrow I \models G$.
In that case we note $S \models_{rdfs} G$.

3.4 Interpreting Datatypes

As we said earlier, every datatype is "identified" by at least one URI, and a typed literal is intended to represent an element of the value space of the datatype identified by the URI given in it. In order to give a formal definition for this "identification" we need to have a function which associates URIs to datatypes. One such a function will be called a datatype map. Then, by means

of this map we will be able to interpret typed literals. There will be then the notion of well and ill typed literals. A well typed literal will correspond to a typed literal l whose character string belongs to the lexical space of the datatype corresponding to the URI included in l, otherwise it will be an ill typed literal. However this notion will change if we change the map we are considering.

So, when interpreting datatypes, we will need to specify which is the map we are considering. There will be only one restriction to one such a datatype map: the datatype XML Literal, which is intended to be the built in datatype of rdf will have to be always identified by `rdf:XMLLiteral`. So for the special case of a typed literal whose URI is `rdf:XMLLiteral` there will be no confusion possible to determine wether it is or not a well typed literal. In general we will speak of a well/ill typed XML Literal when this is the case.

Formally:

Definition 21 (Datatype map) *Let $\mathfrak{D}$ be a set of datatypes containing the XML Literal datatype (which we will note d_{xmlLit}) and $\mathfrak{U}$ a set of RDF URI references containing `rdf:XMLLiteral`, then a* datatype map *D is a map $D : \mathfrak{U} \rightarrow .\mathfrak{D}$ such that $D(\texttt{rdf:XMLLiteral}) = d_{xmlLit}$.*
For each datatype d we will note respectively L_d, V_d and ϕ_d the lexical space, the value space and the lexical to value mapping of d.

3.4.1 D-Interpretation

Definition 22 (D-interpretation) *Let D be a datatype map, then a* D-interpretation *of a vocabulary V is an rdfs-interpretation I of $(V \cup \mathfrak{U})$ which satisfies the following extra conditions:*

- $|[u]|_I = D(u)$
- If $D(u) = d$ then $\mathcal{I}_C(d) = V_d$ and $\mathcal{I}_C(d) \subseteq \mathcal{L}_V$
- If $D(u) = d$ then for any typed literal $\omega + v$, if $|[v]|_I = d$: if $\omega \in L_d$ then $|[\omega + v]|_I = \phi_d(\omega)$ otherwise $|[\omega + v]|_I \notin \mathcal{L}_V$
- If $D(u) = d$ then $|[u]|_I \in \Gamma_C(|[\texttt{rdfs:Datatype}]|_I)$

Definition 23 (Well / ill -typed literal) *A typed literal $\omega + u$ is called a* well-typed literal *in a D interpretation I if $u \in \mathfrak{U}$ and $\omega \in L_{D(u)}$. If $u \in \mathfrak{U}$ and $\omega \notin L_{D(u)}$ it is called an* ill-typed literal.

The first condition ensures that I interprets the URI reference according to the datatype map provided. Note that this does not prevent other URI references from also denoting the datatype.

The second condition ensures that the datatype URI reference, when used as a class name, refers to the value space of the datatype, and that all elements of a value space must be literal values.

The third condition ensures that typed literals in the vocabulary respect the datatype lexical-to-value mapping. The condition also requires that an ill-typed

literal, where the literal string is not in the lexical space of the datatype, not denote any literal value. Intuitively, such a name does not denote any value, but in order to avoid the semantic complexities which arise from empty names, the semantics requires such a typed literal to denote an 'arbitrary' non-literal value. An ill-typed literal does not in itself constitute an inconsistency, but a graph which entails that an ill-typed literal has `rdf:type rdfs:Literal`, or that an ill-typed XML literal has `rdf:type rdf:XMLLiteral`, would be inconsistent.

Note that this third condition applies only to datatypes in the range of D. Typed literals whose type is not in the datatype map of the interpretation are treated as before, i.e. as denoting some unknown thing. The condition does not require that the URI reference in the typed literal be the same as the associated URI reference of the datatype; this allows semantic extensions which can express identity conditions on URI references to draw appropriate conclusions.

The fourth condition ensures that the class `rdfs:Datatype` contains the datatypes used in any satisfying D-interpretation. Notice that this is a necessary, but not a sufficient, condition; it allows the class $|[\texttt{rdfs:Datatype}]|_I$ to contain other datatypes.

Definition 24 (D-entailment) *Let S be a set of graphs, and G a graph, then S* D-entails *G if and only if for every D-interpretation I we have:* $(\forall H \in S, I \models H) \Rightarrow I \models G$.
In that case we note $S \models_D G$.

3.4.2 XSD Interpretation

In the rest of this document the prefix `xsd:` is an abbreviation for:
`http://www.w3.org/2001/XMLSchema#`

The following are the XSD datatypes defined in XML Schema Part 2: Datatypes [XML-Schema2] and are referred to as XSD Datatypes:

`xsd:string, xsd:boolean, xsd:decimal, xsd:float, xsd:double, xsd:dateTime, xsd:time, xsd:date, xsd:gYearMonth, xsd:gYear, xsd:gMonthDay, xsd:gDay, xsd:gMonth, xsd:hexBinary, xsd:base64Binary, xsd:anyURI, xsd:normalizedString, xsd:token, xsd:language, xsd:NMTOKEN, xsd:Name, xsd:NCName, xsd:integer, xsd:nonPositiveInteger, xsd:negativeInteger, xsd:long, xsd:int, xsd:short, xsd:byte, xsd:nonNegativeInteger, xsd:unsignedLong, xsd:unsignedInt, xsd:unsignedShort, xsd:unsignedByte, xsd:positiveInteger`

Note that here we have denoted each datatype with its respective RDF URI reference.

Definition 25 *A datatype map which maps each one of the above RDF URI references as the respective XSD datatype is called an* XSD datatype map. *A D interpretation where D is an XSD datatype map is called an* XSD interpretation.

3.5 Alternative definition for a simple interpretation

In terms of entailment we can see that the restriction of an interpretation to a certain vocabulary is not necessary. In fact, we could consider a definition of interpretation independent of the vocabulary and even when the notion of satisfaction would change, the notion of entailment remains the same.

For sake of clarity we will note, during this paragraph, a simple interpretation of a vocabulary a *v-interpretation*. We will also note the satisfaction and entailment of a graph with the $\models_v$ symbol.

Definition 26 *A* β-interpretation I *is defined by:*
$\mathcal{I}_R \neq \varnothing$*: the set of resources, called the* domain *or* universe *of* I*.*
$\mathcal{I}_P$*: the set of* properties *of* I*.*
$\mathcal{L}_V \subseteq \mathcal{I}_R$*: the set of literal values which contains all plain literals in* $\mathcal{L}$*.*
$\Gamma : \mathcal{I}_P \mapsto \mathcal{P}(\mathcal{I}_R \times \mathcal{I}_R)$
$|[\cdot]|_I : \mathcal{U} \cup \mathcal{L} \to \mathcal{I}_R \cup \mathcal{I}_P$ *such that if* x *is a plain literal then* $|[x]|_I = x$*, and if it is a typed literal* $|[x]|_I \in \mathcal{I}_R$*.*

Here we have changed the previous restriction of $|[\cdot]|_I$ to the whole space of URI references and Literals ($\mathcal{U} \cup \mathcal{L}$). Note that a β-interpretation is always a v-interpretation where the vocabulary is $\mathcal{U} \cup \mathcal{L}$.

The definition of $|[\cdot]|^I$ for ground triples and graphs is the same, except that now the condition "$s, p, o \in V$" for a ground triple is not necessary and can be omitted.

In order to interpret blank nodes we will consider maps of the form $A : \mathcal{B} \to \mathcal{I}_R$ and the same construction used earlier to define $|[\cdot]|_{I+A}$ and $|[\cdot]|^{I+A}$, which leads to the following definition of $|[\cdot]|^I$ for a graph E:

Given a map $A : \mathcal{B} \to \mathcal{I}_R$:

$$|[x]|_{I+A} = \begin{cases} |[x]|_I \text{ if } x \in \mathcal{U} \cup \mathcal{L} \\ A(x) \text{ if } x \in \mathcal{B} \end{cases}$$

If t is a triple (s, p, o):

$$|[t]|^{I+A} = \begin{cases} true \text{ if } |[p]|_{I+A} \in \mathcal{I}_P \text{ and } (|[s]|_{I+A}, |[o]|_{I+A}) \in \Gamma(|[p]|_{I+A}) \\ false \text{ otherwise} \end{cases}$$

If E is a graph:

$$|[E]|^{I+A} = \begin{cases} false \text{ if } \exists t \in E \text{ such that } |[t]|^{I+A} = false \\ true \text{ otherwise} \end{cases}$$

$$|[E]|^I_\beta = \begin{cases} true \text{ if there exists } A : \mathcal{B} \to \mathcal{I}_R \text{ such that } |[E]|^{I+A} = true \\ false \text{ otherwise} \end{cases}$$

Definition 27 *Given a* β*-interpretation* I*, we say that* I β-satisfies G *if* $|[G]|^I_\beta = true$*, in that case we will note* $I \models_\beta G$*, otherwise* $I \not\models_\beta G$*.*

Definition 28 *Let* S *be a set of graphs, and* G *a graph, then* S β-entails G *if and only if for every* β*-interpretation* I *we have:* $(\forall H \in S, I \models_\beta H) \Rightarrow I \models_\beta G$*. In that case we note* $S \models_\beta G$*.*

Proposition 1 *Let* S *be a set of graphs, and* G *a single graph, then* $(S \models_\beta G) \Leftrightarrow (S \models_v G)$*.*

Proof. It is easy to see that $(S \models_v G) \Rightarrow (S \models_\beta G)$, because we know that a β-interpretation can be seen as a v-interpretation of the vocabulary $\mathcal{U} \cup \mathcal{L}$.

We will now prove $\neg(S \models_v G) \Rightarrow \neg(S \models_\beta G)$. This means that if there exist a v-interpretation $I = \{\mathcal{I}_R, \mathcal{I}_P, \mathcal{L}_V, \Gamma, |[\cdot]|_I\}$ such that $\forall H \in S, I \models_v H$ and $I \not\models_v G$ then there exists a β-interpretation I' such that $\forall H \in S, I' \models_\beta H$ and $I' \not\models_\beta G$. If the vocabulary V of I is $\mathcal{U} \cup \mathcal{L}$, then I is also a β-interpretation and we can consider $I' = I$.

Assume now that $V \subsetneq \mathcal{U} \cup \mathcal{L}$, and consider the case when the vocabulary of G is a subset of V ($V_G \subseteq V$). Then the β-interpretation I' defined by:
$\mathcal{I}'_R = \mathcal{I}_R \cup \mathcal{L}_{plain}$ where $\mathcal{L}_{plain}$ is the set of all plain literals in $\mathcal{L}$.
$\mathcal{I}'_P = \mathcal{I}_P$
$\mathcal{L}'_V = \mathcal{L}_{plain} \cup \mathcal{L}_V$
$\Gamma' = \Gamma$

$$|[\cdot]|_{I'}(x) = \begin{cases} |[x]|_I \text{ if } x \in V \\ x \text{ if } x \text{ is a plain literal} \quad \text{where } x_0 \text{ is a fixed element in } \mathcal{I}_R. \\ x_0 \text{ otherwise.} \end{cases}$$

trivially satisfies $\forall H \in S, I' \models_\beta H$. If we had $I' \models_\beta G$ then there would be a map $A' : \mathcal{B} \to \mathcal{I}'_R$ such that: $|[G]|^{I'+A'} = true$, this means that for every triple $t \in G : |[t]|^{I'+A'} = true$, but then we would have that since the interpretations I and I' give the same value of truth for every triple in G then I would also satisfy G as a v-interpretation which is a contradiction.

In the case when $V_G \nsubseteq V$ we have that $V_G \neq \varnothing$ and therefore $G \neq \varnothing$. Moreover, since there exist x' in V_G not in V there is a triple t' in which x' occur. One such a triple can't be in any $H \in S$. We will construct a β-interpretation in which the triple t' will always be $false$. Let I' be a β-interpretation satisfying $\forall H \in S, I' \models_\beta H$. Then redefine I' as follows:

If $t' = (a, x', b)$ then $|[x']|_{I'} \in \mathcal{I}'_P$ and $\Gamma'(|[x']|_{I'}) = \varnothing$.

If $t' = (x', a, b)$ then $|[a]|_{I'} \in \mathcal{I}'_P$ and $\forall c \in \mathcal{I}_R, (|[x']|_{I'}, c) \notin \Gamma'(|[a]|_{I'})$

If $t' = (b, a, x')$ then $|[a]|_{I'} \in \mathcal{I}'_P$ and $\forall c \in \mathcal{I}_R, (c, |[x']|_{I'}) \notin \Gamma'(|[a]|_{I'})$

None of these conditions can affect the value of truth given to any $H \in S$ by $|[\cdot]|^I_\beta$ and they guarantee that the interpretation obtained doesn't β-satisfy G. ■

4 Entailment rules

In this section we will introduce a set of rules with which we will be able to prove entailment between graphs. Each one of these rules consists of two parts: $\frac{A}{B}$. The upper part is a set (possibly infinite) of clauses of the form $\Sigma : K \vdash G$ in which K and G are rdf graphs, and Σ is a mapping from a subset of $\mathcal{B}$ to $\mathcal{B}_K \cup V_K \cup rdfsV \cup rdfV$. The lower part is a clause of the same form.

Informally a proof will be a tree whose nodes are labeled by a set of clauses and for each one of these clauses we will apply a rule. The out-arcs from a node will then be the rules applied to these clauses. We will show that if a proof tree whose root is the clause $\varnothing : K \vdash G$ where K and G are graphs with no blank nodes in common and whose leafs are all an empty inference, then $K \models G$.

Inversely, if $K \models G$ we will show that it is possible to construct one such a proof tree.

Note that in order to "apply" a rule certain conditions must be satisfied, which will be exposed with each rule.

Definition 29 (Proof tree) *Is a rooted labeled tree. Each node of the tree is a set of clauses A, and for each clause C of this set there is exactly one out-arc labeled by one of the rules of the form* $\frac{X}{C}$*. For each node A different from the root there is exactly one in-arc labeled by a rule of the form* $\frac{A}{X}$*.*

Note that a triple (s, p, o) represents the graph containing only this triple and *Axioms* means the set of rdf and rdfs axiomatic triples.

The rules are the following:

4.1 General:

1. $\frac{}{\Sigma : K \vdash (s, p, o)}$ *init* only if $(s, p, o) \in K$
2. $\frac{\{\Sigma : K \vdash t; t \in E\}}{\Sigma : K \vdash E}$ *break* only if $\mathcal{B}_E \subset dom(\Sigma)$

4.2 Instance:

1. $\frac{\Sigma : K \vdash E}{\varnothing : K \vdash E} alloc$ only if $\mathcal{B}_E \subset dom(\Sigma)$
2. $\frac{\Sigma : K \vdash (s, p, x)}{\Sigma : K \vdash (s, p, \_b)} ins1$ only if $\Sigma(\_b) = x$
3. $\frac{\Sigma : K \vdash (x, p, o)}{\Sigma : K \vdash (\_b, p, o)} ins2$ only if $\Sigma(\_b) = x$
4. $\frac{\Sigma : K \vdash (s, p, \_b)}{\Sigma : K \vdash (s, p, x)} gl$ only if $\Sigma(\_b) = x \in \mathcal{L}$

4.3 Rdf/Rdfs:

4.3.1 Type:

1. $\frac{\Sigma : K \vdash (s, p, o)}{\Sigma : K \vdash (p, \texttt{rdf:type}, \texttt{rdf:Prop})} rdf1$
2. $\frac{}{\Sigma : K \vdash (x, \texttt{rdf:type}, \texttt{rdf:Res})} rdfs4$
   only if $x \in \mathcal{B}_K \cup V_K \cup rdfsV \cup rdfV$
3. $\frac{}{\Sigma : K \vdash (\_b, \texttt{rdf:type}, \texttt{rdf:Res})} rdfs4l$
   only if $\Sigma(\_b) = x \in \mathcal{L} \cap V_K$

4. $\dfrac{}{\Sigma : K \vdash (\_b, \texttt{rdf:type}, \texttt{rdf:XMLLit})} rdf2$
only if $\Sigma(\_b) = l$ is a well-typed XMLLiteral and $l \in V_K$.

5. $\dfrac{}{\Sigma : K \vdash (\_b, \texttt{rdf:type}, \texttt{rdfs:Literal})} rdfs1$
only if $\Sigma(\_b) = l$ is a plain literal and $l \in V_K$.

6. $\dfrac{\Sigma : K \vdash (a, \texttt{rdf:domain}, b), \Sigma : K \vdash (x, c, y), \Sigma : K \vdash (c, \texttt{rdfs:sp}, a)}{\Sigma : K \vdash (x, \texttt{rdf:type}, b)} rdfs2$

7. $\dfrac{\Sigma : K \vdash (a, \texttt{rdf:range}, b), \Sigma : K \vdash (x, c, y), \Sigma : K \vdash (c, \texttt{rdfs:sp}, a)}{\Sigma : K \vdash (y, \texttt{rdf:type}, b)} rdfs3$

8. $\dfrac{\Sigma : K \vdash (a, \texttt{rdfs:sc}, b), \Sigma : K \vdash (x, \texttt{rdf:type}, a)}{\Sigma : K \vdash (x, \texttt{rdf:type}, b)} rdfs9$

4.3.2 Properties:

1. $\dfrac{\Sigma : K \vdash (a, \texttt{rdfs:sp}, b), \Sigma : K \vdash (x, a, y)}{\Sigma : K \vdash (x, b, y)} rdfs7$

2. $\dfrac{\Sigma : K \vdash (a, \texttt{rdfs:sp}, b), \Sigma : K \vdash (b, \texttt{rdfs:sp}, c)}{\Sigma : K \vdash (a, \texttt{rdfs:sp}, c)} rdfs5$

3. $\dfrac{\Sigma : K \vdash (a, \texttt{rdf:type}, \texttt{rdf:Prop})}{\Sigma : K \vdash (a, \texttt{rdfs:sp}, a)} rdfs6$

4. $\dfrac{\Sigma : K \vdash (a, \texttt{rdf:type}, \texttt{rdfs:ContMemProp})}{\Sigma : K \vdash (a, \texttt{rdfs:sp}, \texttt{rdf:Member})} rdfs12$

4.3.3 Classes:

1. $\dfrac{\Sigma : K \vdash (a, \texttt{rdf:type}, \texttt{rdfs:Class})}{\Sigma : K \vdash (a, \texttt{rdfs:sc}, \texttt{rdf:Res})} rdfs8$

2. $\dfrac{\Sigma : K \vdash (a, \texttt{rdfs:sc}, b), \Sigma : K \vdash (b, \texttt{rdfs:sc}, c)}{\Sigma : K \vdash (a, \texttt{rdfs:sc}, c)} rdfs11$

3. $\dfrac{\Sigma : K \vdash (a, \texttt{rdf:type}, \texttt{rdfs:Class})}{\Sigma : K \vdash (a, \texttt{rdfs:sc}, a)} rdfs10$

4. $\dfrac{\Sigma : K \vdash (a, \texttt{rdf:type}, \texttt{rdfs:Datatype})}{\Sigma : K \vdash (a, \texttt{rdfs:sc}, \texttt{rdfs:Literal})} rdfs13$

4.4 Validity of the entailment rules

We will now show that if there exists a finite height proof tree whose root is $\Sigma : K \vdash E$ then $K \models_{rdfs} E$. Moreover, we will show that for every interpretation I such that $I \models K$, for every A such that $|[K]|^{I+A} = true$ there exist a mapping

$\sigma : \mathcal{B} \to \mathcal{I}_R$ satisfying $\sigma(x) = |[\Sigma(x)]|_{I+A}$ for every blank node x in $dom(\Sigma)$ such that $|[G]|^{I+\sigma} = true$.

In more simple words this means that every blank node in G can be identified to a node in K, which can be an uri or literal as well as a blank node and the function Σ is the mean to manage this identification during the proof.

For sake of clarity we will note $\Sigma : K \models E$ the following proposition: $\forall I, \forall A : |[K]|^{I+A} = true \Rightarrow \exists \sigma : \mathcal{B} \to \mathcal{I}_R$ such that $\sigma(x) = |[\Sigma(x)]|_{I+A}$ for every x in $dom(\Sigma)$ and $|[E]|^{I+\sigma} = true$.

We can restrict this notation for rdf or rdfs interpretations only to have $\Sigma : K \models_{rdf} E$ and $\Sigma : K \models_{rdfs} E$ respectively. Note that $\Sigma : K \models E \Rightarrow K \models E$.

Proposition 2 *Given two rdf graphs K and E such that and there exist a proof tree of finite height and whose root is $\Sigma : K \vdash E$, where Σ is such that $dom(\Sigma) \cap img(\Sigma) = \varnothing$ and $dom(\Sigma) \cap \mathcal{B}_K = \varnothing$ then $\Sigma : K \models_{rdfs} E$.*

Proof. *We will prove this by induction over the height of the tree. There are however many cases for which the proof is practically the same, so we will only present the most significant cases. Note that we will only consider rdfs interpretations.*

For every case when the root of the tree is $\Sigma : K \vdash E$ we will consider, given I and A such that $|[K]|^{I+A} = true$, the following definition of σ:

$\sigma(x) = \begin{cases} |[\Sigma(x)]|_{I+A} \text{ if } x \in dom(\Sigma) \\ A(x) \text{ otherwise} \end{cases}$

If the height of the tree is 1 we have the following cases:

- $\dfrac{}{\Sigma : K \vdash (s,p,o)}$ *init:*
  Since to apply this rule we must have that $(s,p,o) \in K$, we know that neither s nor o belong to $dom(\Sigma)$. Then $|[(s,p,o)]|^{I+\sigma} = |[(s,p,o)]|^{I+A} = true$.

- $\dfrac{}{\Sigma : K \vdash (x, \texttt{rdf:type}, \texttt{rdf:Res})} rdfs4$*:*
  In order to apply this rule x must belong to $\mathcal{B}_K \cup V_K \cup rdfsV \cup rdfV$ so for every rdfs interpretation I for every A we know that $|[x]|_{I+\sigma} = |[x]|_{I+A} \in \mathcal{I}_R$ and by the semantic conditions of an rdfs interpretation $\mathcal{I}_R = \Gamma_C(|[\texttt{rdf:Res}]|_I)$ which means that $(|[x]|_{I+A}, |[\texttt{rdf:Res}]|_I) \in \Gamma(|[\texttt{rdf:type}]|_I)$.

- $\dfrac{}{\Sigma, : K \vdash (\_b, \texttt{rdf:type}, \texttt{rdf:Res})} rdfs4l$*:*
  Since $x \in \mathcal{L} \cap V_K$ and we have that $|[\_b]|_{I+\sigma} = |[x]|_I$ the proof is the same as in the previous case.

- $\dfrac{}{\Sigma : K \vdash (\_b, \texttt{rdf:type}, \texttt{rdf:XMLLit})} rdf2$
  We know that l is a well typed XML literal and $l \in V_K$ so for any I an rdf interpretation we have that $|[l]|_I \in \Gamma_C(|[\texttt{rdf:XMLLit}]|_I)$, so $(|[l]|_I, |[\texttt{rdf:XMLLit}]|_I) \in \Gamma(|[\texttt{rdf:type}]|_I)$. Besides we know that $|[l]|_I = |[\_b]|_{I+\sigma}$ which concludes the proof.

- $\frac{}{\Sigma : K \vdash (\_b, \texttt{rdf:type}, \texttt{rdfs:Literal})} rdfs1$:
  The proof is the same as in the previous case except that we use the semantic conditions for rdfs interpretations.

For the rest of the cases we will only prove the following (because the proof is practically the same for the others):

- $\frac{\Sigma : K \vdash E}{\varnothing : K \vdash E} alloc$:
  We know that there exists a proof tree whose root is $\Sigma : K \vdash E$ by hypothesis of induction we know that $\forall I, \forall A$ such that $|[K]|^{I+A} = true$ there exists σ such that $|[E]|^{I+\sigma} = true$

- $\frac{\Sigma : K \vdash (s, p, x)}{\Sigma : K \vdash (s, p, \_b)} ins1$:
  We know that $\Sigma(\_b) = x$ so $\forall I, \forall A$ we have that $|[\_b]|_{I+\sigma} = |[x]|_{I+A}$ and then, since $\Sigma : K \models (s, p, x)$ we conclude $\Sigma : K \models (s, p, \_b)$. (The proof is almost the same for the cases $ins2$ and gl)

- $\frac{\{\Sigma : K \vdash t; t \in E\}}{\Sigma : K \vdash E}$ *break*:
  This means that for every $t \in E$ we have $\Sigma : K \models t$. In other words
  $\forall t \in E, \forall I, \forall A, \exists \sigma_t$ *such that* $\forall x \in dom(\Sigma) : \sigma_t(x) = |[\Sigma(x)]|_{I+A}$ *and* $|[K]|^{I+A} \Rightarrow |[t]|^{I+\sigma_t}$.
  Since $dom(\Sigma)$ contains every blank node in E then for every triple t the restriction of σ_t to the set of blank nodes $\mathcal{B}_E$ is the same function, name it σ_E. Then we can choose for every triple t the same σ_E which leads to the following:
  $\forall I, \forall A, \exists \sigma_E$ *such that* $\forall x \in dom(\Sigma) : \sigma_E(x) = |[\Sigma(x)]|_{I+A}$ *and* $|[K]|^{I+A} \Rightarrow$
  $\forall t \in E : |[t]|^{I+\sigma_E}$
  Which is exactly $\Sigma : K \models E$.

For all the rest of the cases the proof is straightforward from the semantic conditions of an rdf or rdfs interpretation. ■

5 Soundness and Completeness

The main result of this section is that the deductive system of the previous entailment rules is complete for rdfs entailment. It is easy to prove the soundness of these rules, in fact, in order to do so we must only prove for each one of these rules that they are valid. This follows directly from the definitions.

If we are in the case of simple and rdf entailment not all the rules are valid.

Theorem 3 (Simple entailment) *Given K and E two finite rdf graphs where $\mathcal{B}_K \cap \mathcal{B}_E = \varnothing$ then $K \models E$ if and only if there exist a proof tree whose root is $\varnothing : K \vdash E$ and the arcs can only be labeled by one of the following rules: init, break, alloc, $ins1$, $ins2$.*

Proof. Since E has an instance which is a subgraph of K there exist a mapping μ satisfying $\mu(E) \subseteq K$. We apply then the *alloc* rule to get $\mu : K \vdash E$ followed by the *break* rule and then for each one of the branches the $ins1$ and $ins2$ rules when necessary. Then the *init* rule to finish the proof tree.
The proof of the "if" we have to prove that all these rules are valid, which is the same proof we did earlier. ■

Theorem 4 (RDF Entailment) *Given K and E two finite rdf graphs where $\mathcal{B}_K \cap \mathcal{B}_E = \varnothing$ then $K \models_{rdf} E$ if and only if there exist a proof tree whose root is $\varnothing : K \vdash E$ and the arcs can only be labeled by one of the following rules: init, break, alloc, $ins1$, $ins2$, $rdf1$, $rdf2$.*

We will not prove this theorem since the proof follows the same principle of the proof for rdfs entailment.

The following theorem says that this deductive system is sound and complete for rdfs entailment.

Theorem 5 (RDFS Entailment) *Given K and E two rdf graphs where $\mathcal{B}_K \cap \mathcal{B}_E = \varnothing$ and such that neither K nor E are XML clash, then $K \models_{rdfs} E$ if and only if there exist a proof tree whose root is $\varnothing : K \vdash E$ and the arcs can only be labeled by any of the following rules: general rules, instance rules, or rdf/rdfs rules.*

To prove this theorem we will need to define an rdfs Herbrand interpretation, in order to do so, let's consider the following construction:

Let K be an rdf graph such that there exist at least one rdfs interpretation satisfying K. This means that it is not an XML clash. Then we define I_0 as:

- $\mathcal{I}_R =$
  $\{x; x \in U \cup \mathcal{B}_K$ and x occur in $K\} \cup$
  $\{x; x$ is a plain literal or ill typed XML literal$\} \cup$
  $\{\phi_{XML}(x); x$ is a well typed XML literal$\} \cup$
  $rdfV \cup rdfsV$
- $|[x]| = \begin{cases} \phi_{XML}(x) \text{ if } x \text{ is a well typed XML literal} \\ x \text{ otherwise} \end{cases}$
- $\mathcal{I}_P = \{|[p]|; (s,p,o) \in K \cup Axioms\} \cup \{p|(p,$`rdf:type`,`rdf:Prop`$) \in K \cup Axioms\}$
- $\mathcal{L}_v =$
  $\{|[x]|; x$ is a plain literal$\} \cup$
  $\{|[x]|; x$ is a well typed XML literal$\} \cup$
  $\{|[x]|; (x,$ `rdf:type`, `rdf:Literal`$) \in K\} \cup$
  $\{|[x]|; (x,$ `rdf:type`, `rdf:XMLLit`$) \in K\}$

- $\Gamma_0(|[\texttt{rdf:type}]|) =$
  $\{(|[x]|, |[\texttt{rdf:Literal}]|); x \text{ is a plain literal}\} \cup$
  $\{(|[x]|, |[\texttt{rdf:XMLLit}]|); x \text{ is a well typed XML literal}\} \cup$
  $\{(|[x]|, |[\texttt{rdf:Prop}]|); (u, x, v) \in K\} \cup$
  $\{(|[x]|, |[\texttt{rdf:Res}]|; |[x]| \in \mathcal{I}_R\} \cup$
  $\{(|[x]|, |[y]|); (x, \texttt{rdf:type}, y) \in K\}$

- if p is different from $\texttt{rdf:type}$:
  $\Gamma_0(|[p]|) = \{(|[x]|, |[y]|); (x, p, y) \in K\}$

Now for each $i \in \mathbb{N}^+$ consider I_i defined inductively as:

- $\mathcal{I}_{R,i} = \mathcal{I}_R$
- $|[x]|_i = |[x]|$
- $\mathcal{I}_{P,i} = \mathcal{I}_{P,i-1} \cup \{x; (x, |[\texttt{rdf:Prop}]|) \in \Gamma_{i-1}(|[\texttt{rdf:type}]|)\}$
- $\mathcal{L}_{v,i} = \mathcal{L}_{v,i-1} \cup \{x; (x, |[\texttt{rdf:Literal}]|) \in \Gamma_{i-1}(|[\texttt{rdf:type}]|)\}$
- $\Gamma_i(|[\texttt{rdf:type}]|) =$
  $\Gamma_{i-1}(|[\texttt{rdf:type}]|) \cup$
  $\{(u, v); (u, v) \in \Gamma_{i-1}(x) \wedge (x, |[\texttt{rdf:type}]|) \in \Gamma_{i-1}(|[\texttt{rdfs:sp}]|)\}$
  $\{(u, y); (u, v) \in \Gamma_{i-1}(x) \wedge (x, y) \in \Gamma_{i-1}(|[\texttt{rdfs:domain}]|)\}$
  $\{(v, y); (u, v) \in \Gamma_{i-1}(x) \wedge (x, y) \in \Gamma_{i-1}(|[\texttt{rdfs:range}]|)\}$
  $\{(u, y); (u, x) \in \Gamma_{i-1}(|[\texttt{rdf:type}]|) \wedge (x, y) \in \Gamma_{i-1}(|[\texttt{rdfs:sc}]|)\}$
- $\Gamma_i(|[\texttt{rdfs:sc}]|) =$
  $\Gamma_{i-1}(|[\texttt{rdfs:sc}]|) \cup$
  $\{(u, v); (u, v) \in \Gamma_{i-1}(x) \wedge (x, |[\texttt{rdf:sc}]|) \in \Gamma_{i-1}(|[\texttt{rdfs:sp}]|)\} \cup$
  $\{(x, |[\texttt{rdfs:Res}]|); (x, |[\texttt{rdfs:Class}]|) \in \Gamma_{i-1}(|[\texttt{rdf:type}]|)\} \cup$
  $\{(x, x); (x, |[\texttt{rdfs:Class}]|) \in \Gamma_{i-1}(|[\texttt{rdf:type}]|)\} \cup$
  $\{(x, y); (x, z) \in \Gamma_{i-1}(|[\texttt{rdfs:sc}]|) \wedge (z, y) \in \Gamma_{i-1}(|[\texttt{rdfs:sc}]|)\} \cup$
  $\{(x, |[\texttt{rdf:Literal}]|); (x, |[\texttt{rdfs:Datatype}]|) \in \Gamma_{i-1}(|[\texttt{rdf:type}]|)\}$
- $\Gamma_i(|[\texttt{rdfs:sp}]|) =$
  $\Gamma_{i-1}(|[\texttt{rdfs:sp}]|) \cup$
  $\{(u, v); (u, v) \in \Gamma_{i-1}(x) \wedge (x, |[\texttt{rdf:sp}]|) \in \Gamma_{i-1}(|[\texttt{rdfs:sp}]|)\} \cup$
  $\{(x, x); (x, |[\texttt{rdf:Prop}]|) \in \Gamma_{i-1}(|[\texttt{rdf:type}]|)\} \cup$
  $\{(x, y); (x, z) \in \Gamma_{i-1}(|[\texttt{rdfs:sp}]|) \wedge (z, y) \in \Gamma_{i-1}(|[\texttt{rdfs:sp}]|)\} \cup$
  $\{(x, |[\texttt{rdf:Member}]|); (x, |[\texttt{rdfs:ContMemProp}]|) \in \Gamma_{i-1}(|[\texttt{rdf:type}]|)\}$
- $\Gamma_i(p) =$
  $\Gamma_{i-1}(p) \cup$
  $\{(u, v); (u, v) \in \Gamma_{i-1}(x) \wedge (x, p) \in \Gamma_{i-1}(|[\texttt{rdfs:sp}]|)\} \cup$

Clearly I_0 simply entails K, and so for all the other simple interpretations I_i.

Now, given a set of interpretations M such that $\forall I, I' \in M$ we have: $\mathcal{I}_{R,I} = \mathcal{I}_{R,I'} = \mathcal{I}_R$ and $|[\cdot]|_I = |[\cdot]|_{I'} = |[\cdot]|$ we can define:

$I_* = \bigcup_{I \in M} I$ as:
$\mathcal{I}_{R,*} = \mathcal{I}_R$
$|[x]|_* = |[x]|$
$\mathcal{I}_{P,*} = \bigcup_{I \in M} \mathcal{I}_{P,I}$
$\mathcal{L}_{v,*} = \bigcup_{I \in M} \mathcal{L}_{v,I}$
$\Gamma(p) = \bigcup_{I \in M} \Gamma_I(p)$
All this leads us to the following definition:

Definition 30 (RDFS Herbrand Interpretation) *The rdfs Herbrand interpretation I_H of K is defined by:*
$\bigcup_{i \in \mathbb{N}} I_i$
and $\Gamma_C(y) = \{x; (x, y) \in \Gamma(|[\texttt{rdf:type}]|_{I_H})\}$

It is easy to verify that I_H is an rdfs interpretation of K. In fact I_H is such that $|[K]|^{I_H + id} = true$ where id is the identity function over blank nodes. And it satisfies all the rdfs semantic conditions.

Moreover, given a triple t and a function over blank nodes A if $|[t]|^{I_H + A} = true$ there exists $i \in \mathbb{N}$ such that $|[t]|^{I_i + A} = true$. And $K \models_{rdfs} E \Leftrightarrow I_H \models E$.

Lemma 6 *Let E, K be rdf graphs such that $\mathcal{B}_E \cap \mathcal{B}_K = \varnothing$ and such that neither K nor E are XML clash, then there exist Σ such that $dom(\Sigma) = \mathcal{B}_E$ and $\Sigma : K \vdash_{rdfs} E$ if and only if $K \models_{rdfs} E$.*

Proof (of the lemma). As we said earlier if there exist one such Σ, satisfying $\Sigma : K \vdash E$ then clearly we have $K \models E$. The opposite way requires a more delicate treatment:

The idea is to identify by means of the Herbrand model of K all the blanks in E to nodes of K. Then, with this identification Σ we will see that for every triple in E the condition $\Sigma : K \vdash (s, p, o)$ holds. Since for every triple in E which doesn't contain any blank this Σ function won't change anything the proof is over, for the rest of the triples we show that the value of truth of the triple will be given by the Herbrand model because it satisfies only those triples which are true in every interpretation of K.

Assume $K \models E$ then we know that for every interpretation I which makes $|[K]|^I$ true also makes $|[E]|$ true. Let's take the rdfs Herbrand interpretation of K.

We know that $I_H \models_{rdfs} E$ which means that there exist $A : \mathcal{B}_E \to \mathcal{I}_R$ such that $|[E]|^{I_H + A} = true$. But since there exist a bijection between $\mathcal{I}_R$ and the set of nodes of K (the identity except for well typed XML literals) we can construct a function $\Sigma : \mathcal{B}_E \to \mathcal{B}_K \cup V_K \cup rdfsV \cup rdfV$ as follows:
$\Sigma(x) = \begin{cases} \phi^{-1}_{XML}(A(x)) \text{ if } A(x) \in V_{XMLlit} \\ A(x) \text{ otherwise} \end{cases}$
We must now prove that this Σ satisfies $\Sigma : K \vdash E$.

Let I be an rdfs interpretation of K and A' be such that $|[K]|^{I+A'} = true$. Consider σ defined as $\sigma(x) = |[\Sigma(x)]|_{I+A'}$ then, for every triple $(s,p,o) \in E$ we have that exactly one of the following cases occur:

1. neither s nor o are blank nodes
2. only s is a blank node:
3. only o is a blank node:
4. both s and o are blank nodes:

Clearly for the first case we have $|[(s,p,o)]|^{I+\sigma} = |[(s,p,o)]|^{I} = true$, which is exactly what we have to prove.

For the rest of the cases the proof is quite similar so we will only prove this for case 2:

$|[(s,p,o)]|^{I+\sigma} = true \Leftrightarrow (\sigma(s), |[o]|_I) \in \Gamma(|[p]|_I) \Leftrightarrow (|[\Sigma(x)]|_{I+A'}, |[o]|_I) \in \Gamma(|[p]|_I)$

if $b = \Sigma(x) \in \mathcal{B}_K$ we have to prove: $(A'(b), |[o]|_I) \in \Gamma(|[p]|_I)$ which is true because $(b, |[o]|_{I_H}) \in \Gamma_{I_H}(p)$ in the Herbrand model. In fact this assertion means that the pair formed by the interpretation of the blank node b and the interpretation of o belongs to the set which Γ_{I_H} associates to the interpretation of p and the only way for this to be true is that this is true in every model of K because I_H only satisfies the conditions that every interpretation must satisfy (and identifying every interpretation uniquely to the node it represents).

On the other hand if $y = \Sigma(x) \notin \mathcal{B}_K$ we have by doing a similar reasoning that $(|[y]|_I, |[o]|_I) \in \Gamma(|[p]|_I)$.

Finally we have that for every triple $(s,p,o) \in E$ the evaluation $|[(s,p,o)]|^{I+\sigma} = true$ then $|[E]|^{I+\sigma} = true$ which concludes the demonstration. ■

Proof (of the theorem). Assume that $K \models E$, where $\mathcal{B}_K \cap \mathcal{B}_E = \varnothing$ and such that neither K nor E are XML clash, then by the previous lemma we know that there exists Σ such that $\Sigma : K \vdash E$. For this proof we will extend this Σ function in order to have at least one blank node associated to each literal.

We will now show how to construct a proof tree. To do so we will first apply the *alloc* rule with Σ and then the *break* rule in order to construct a proof tree from clauses of the form $\Sigma : K \vdash (s,p,o)$ where $(s,p,o) \in E$. Then, by induction we will show how to construct the proof tree corresponding to each of these clauses.

We know that $|[K]|^{I_H+id} = true$ so for each triple t in E we have $|[t]|^{I_H+\Sigma} = true$. By construction of the Herbrand model we also know that for each one of these triples there exists $i \in \mathbb{N}$ such that $|[t]|^{I_i+\Sigma} = true$. Now, we will prove that if there exists $i \in \mathbb{N}$ such that $|[(s,p,o)]|^{I_i+\Sigma} = true$ holds then we can construct a finite proof tree for the following clause: $\Sigma : K \vdash (s,p,o)$.

From now on $s' = \Sigma(s)$ if s is a blank node or $s' = s$ otherwise and $o' = \Sigma(o)$ if o is a blank node or $o' = o$ otherwise

If $|[(s,p,o)]|^{I_0+\Sigma} = true$ then $(|[s']|_{I_0+id}, |[o']|_{I_0+id}) \in \Gamma_0(|[p]|_{I_0})$ This can only be possible in one of the following cases:

- either $(s', p, o') \in K$:
  Then we apply the $ins1$ and $ins2$ rules if necessary and finally the $init$ rule to finish the proof tree.

- or $p =$`rdf:type` and:

  - s' is a plain literal and $o' =$`rdf:Literal`:
    Then we apply the $ins1$ rule if necessary and then we finish the proof tree by the $rdfs1$ rule.
  - or s' is a well typed XML literal and $o' =$`rdf:XMLLit`:
    Then again we apply the $ins1$ rule if necessary and then we finish the proof tree by the $rdf2$ rule.
  - or $o' =$`rdf:Prop` and there exist u, v such that $(u, s', v) \in K$:
    Then we apply the $ins1$ and $ins2$ rule if necessary, then the $rdf1$ rule to get $\Sigma : K \vdash (u, s', v)$ (note that s' must be an uri because $(u, s', v) \in K$) and the proof tree finishes by the $init$ rule.
  - or $o' =$`rdfs:Res` and $s' \in \mathcal{B}_K \cup V_K \cup rdfsV \cup rdfV$:
    Then we apply the $ins1$ rule if necessary, if s' isn't a literal we apply the $ins2$ rule and then the $rdfs4$ rule to finish the proof three. Otherwise we apply the $rdfs4l$ rule.

We have just seen the base case, now by induction we will see that given $i \in \mathbb{N}$ if $\forall j < i, |[(s, p, o)]|^{I_j + \Sigma} = true \Rightarrow$ there is a finite proof tree whose root is $\Sigma : K \vdash (s, p, o)$ then so this holds for i.

Let's assume that $|[(s, p, o)]|^{I_i + \Sigma} = true$. In other words $(|[s']|, |[o']|) \in \Gamma_i(|[p]|)$ (note that since there is no possible confusion we have omitted the index). Again there are several different cases for this to occur:

- either $p =$`rdfs:sp` and:

  - either $s' = o'$ and $(|[s']|, |[$`rdf:Prop`$]|) \in \Gamma_{i-1}(|[$`rdf:type`$]|)$:
    The idea is to get $\Sigma : K \vdash (s',$`rdfs:sp`$, s')$, but this is not always possible since s' might be a literal. So we will see in detail each one of the possible cases:
    In the case s' is not a literal or s and o are not blank nodes we will have (after eventually applying the $ins1$ or $ins2$ rules) $\Sigma : K \vdash (s',$`rdfs:sp`$, s')$. Now, by applying the $rdfs6$ rule we get $\Sigma : K \vdash (s',$`rdf:type`$,$`rdf:Prop`$)$.
    On the other hand, if s' is the literal l: if o is a blank node we apply the $ins1$ rule to get $\Sigma : K \vdash (s,$`rdfs:sp`$, l)$ otherwise we do have already this. Note that s must be a blank node because there are no literals in the subject. We can now apply the gl rule to get $\Sigma : K \vdash (s,$`rdfs:sp`$, s)$. And here again, by applying the $rdfs6$ rule we get $\Sigma : K \vdash (s,$`rdf:type`$,$`rdf:Prop`$)$.
    In both cases we can finish the demonstration with the proof tree for $\Sigma : K \vdash (x,$`rdf:type`$,$`rdf:Prop`$)$, where x is either s or s' depending

on the case. This tree is finite because $|[(x,\texttt{rdf:type},\texttt{rdf:Prop})]|^{I_{i-1}+\Sigma} = true$.

- or there exists z such that $(|[s']|, |[z]|) \in \Gamma_{i-1}(|[\texttt{rdfs:sp}]|)$ and $(|[z]|, |[o']|) \in \Gamma_{i-1}(|[\texttt{rdfs:sp}]|)$:
  We apply the $rdfs5$ rule to get $\Sigma : K \vdash (s,\texttt{rdfs:sp}, z), \Sigma : K \vdash (z,\texttt{rdfs:sp}, o)$ then we finish with the proof tree of each one of the branches. Note that we can always choose z as to be either a blank node or an uri because if it was a literal l, we know that there is at least one blank node which satisfies $\Sigma(\_b) = l$.
- or $o' =$`rdf:Member` and $(|[s']|, |[\texttt{rdfs:ContMemProp}]|) \in \Gamma_{i-1}(|[\texttt{rdf:type}]|)$:
  First we apply the $ins1$ rule to get $\Sigma : K \vdash (s,\texttt{rdfs:sp},\texttt{rdfs:member})$ (if o is a blank node) and then the $rdfs12$ rule to get $\Sigma : K \vdash (s,\texttt{rdf:type},\texttt{rdfs:ContMemProp})$ and we finish with the proof tree for this clause.

- or $p =$`rdfs:sc` and:
  - either $o' =$`rdfs:Res` and $(|[s']|, |[\texttt{rdfs:Class}]|) \in \Gamma_{i-1}(|[\texttt{rdf:type}]|)$:
    Again we first apply the $ins1$ rule to get $\Sigma : K \vdash (s,\texttt{rdfs:sc},\texttt{rdfs:Res})$ (if o is a blank node) and then the $rdfs8$ rule to get $\Sigma : K \vdash (s,\texttt{rdf:type},\texttt{rdfs:Class})$. And the proof finish with the proof tree for this clause.
  - or $s' = o'$ and $(|[s']|, |[\texttt{rdfs:Class}]|) \in \Gamma_{i-1}(|[\texttt{rdf:type}]|)$:
    The idea is to get $\Sigma : K \vdash (s',\texttt{rdfs:sc}, s')$, but again this is not always possible since s' might be a literal. So as we did before:
    In the case s' is not a literal or s and o are not blank nodes we will have (after eventually applying the $ins1$ or $ins2$ rules) $\Sigma : K \vdash (s',\texttt{rdfs:sc}, s')$. Now, by applying the $rdfs10$ rule we get $\Sigma : K \vdash (s',\texttt{rdf:type},\texttt{rdfs:Class})$.
    On the other hand, if s' is the literal l: if o is a blank node we apply the $ins1$ rule to get $\Sigma : K \vdash (s,\texttt{rdfs:sc}, l)$ otherwise we do have already this. Note that s must be a blank node because there are no literals in the subject. We can now apply the gl rule to get $\Sigma : K \vdash (s,\texttt{rdfs:sc}, s)$. And here again, by applying the $rdfs10$ rule we get $\Sigma : K \vdash (s,\texttt{rdf:type},\texttt{rdfs:Class})$.
    In both cases we can finish the demonstration with the proof tree for $\Sigma : K \vdash (x,\texttt{rdf:type},\texttt{rdfs:Class})$, where x is either s or s' depending on the case.
  - or there exists z such that $(|[s']|, |[z]|) \in \Gamma_{i-1}(|[\texttt{rdfs:sc}]|) \wedge (|[z]|, |[o']|) \in \Gamma_{i-1}(|[\texttt{rdfs:sc}]|)$:
    We apply the $rdfs11$ rule to get $\Sigma : K \vdash (s,\texttt{rdfs:sc}, z), \Sigma : K \vdash (z,\texttt{rdfs:sc}, o)$ then we finish with the proof tree of each one of the branches. Note that we can always choose z as to be either a blank node or an uri because if it was a literal l, we know that there is at least one blank node which satisfies $\Sigma(\_b) = l$.

– or o' =`rdf:Literal` and $(|[s']|, |[\texttt{rdfs:Datatype}]|) \in \Gamma_{i-1}(|[\texttt{rdf:type}]|)$:
First we apply the $ins1$ rule to get $\Sigma : K \vdash (s,\texttt{rdfs:sc},\texttt{rdfs:Literal})$ (if o is a blank node) and then the $rdfs13$ rule to get $\Sigma : K \vdash (s,\texttt{rdf:type},\texttt{rdfs:Datatype})$ and we finish with the proof tree for this clause.

• or p =`rdf:type` and:

– either there exist x, y such that $(|[s']|, |[y]|) \in \Gamma_{i-1}(|[x]|)$ and $(|[x]|, |[o']|) \in \Gamma_{i-1}(|[\texttt{rdfs:domain}]|)$:
We will consider two cases: either x is an uri or it's a blank node.
In the first case we have that $|[(s, x, y)]|^{I_{i-1}+\Sigma} = true$, and then $|[(x,\texttt{rdfs:sp}, x)]|^{I_i+\Sigma} = true$ which is already proved to have a proof tree. Then we can apply the $rdfs2$ rule to get $\Sigma : K \vdash (x,\texttt{rdfs:domain}, o), \Sigma : K \vdash (s, x, y), \Sigma : K \vdash (x,\texttt{rdfs:sp}, x)$ and we finish by the proof tree for each one of these branches.
On the second case we know that x is a blank node, so we must prove that there exist at least one uri c which satisfies both $(|[s']|, |[y]|) \in \Gamma_{i-1}(|[c]|)$ and $(|[x]|, |[c]|) \in \Gamma_{i-1}(|[\texttt{rdfs:sp}]|)$. We will do so by induction:
if $i = 2$ then the only way to have $(|[s']|, |[y]|) \in \Gamma_1(|[x]|)$ is that there exists one such c.
Now if $i > 2$ we have two cases: either $(|[s']|, |[y]|) \in \Gamma_{i-2}(|[x]|)$ or there exists c' such that $(|[s']|, |[y]|) \in \Gamma_{i-2}(|[c']|)$ and $(|[c']|, |[x]|) \in \Gamma_{i-2}(|[\texttt{rdfs:sp}]|)$. On the first case we know by induction that there exists one such c. For the second suppose that c' isn't an uri then we can assume that it is a blank node. By hypothesis of induction we know that there exists an uri c such that $(|[s']|, |[y]|) \in \Gamma_{i-3}(|[c]|)$ and $(|[c]|, |[c']|) \in \Gamma_{i-2}(|[\texttt{rdfs:sp}]|)$. But then we have that $(|[c]|, |[x]|) \in \Gamma_{i-1}(|[\texttt{rdfs:sp}]|)$ and so we have an uri which satisfies what we need.
Then we can apply the $rdfs2$ rule to get $\Sigma : K \vdash (x,\texttt{rdfs:domain}, o), \Sigma : K \vdash (s, c, y), \Sigma : K \vdash (c,\texttt{rdfs:sp}, x)$ and we finish by the proof tree for each one of these branches.

– or there exist x, y such that $(|[y]|, |[s']|) \in \Gamma_{i-1}(|[x]|)$ and $(|[x]|, |[o']|) \in \Gamma_{i-1}(|[\texttt{rdfs:range}]|)$:
The proof is the same as in the previous case.

– or there exist x, y such that $(|[s']|, |[x]|) \in \Gamma_{i-1}(|[\texttt{rdf:type}]|)$ and $(|[x]|, |[o']|) \in \Gamma_{i-1}(|[\texttt{rdfs:sc}]|)$:
We can apply the $rdfs9$ rule to get $\Sigma : K \vdash (x,\texttt{rdfs:sc}, o), \Sigma : K \vdash (s,\texttt{rdf:type}, x)$ and finally the proof tree for each clause.

• or p is any uri and:

– either $(|[s']|, |[o']|) \in \Gamma_{i-1}(p)$:
Then, by hypothesis of induction, we already have a proof tree.

– or there exists p' such that $(|[s']|, |[o']|) \in \Gamma_{i-1}(|[p']|)$ and $(|[p']|, |[p]|) \in \Gamma_{i-1}(|[\texttt{rdfs:sp}]|)$:
We apply the $rdfs7$ rule to get $\Sigma : K \vdash (s, p', o), \Sigma : K \vdash (p', \texttt{rdfs:sp}, p)$.
Now by induction we know that for each one of these branches there is a proof tree which finish the demonstration in both cases. Note that p' can be chosen as to be an uri, (Why?).

We have shown how to construct a proof tree if we have $|[E]|^{I_H + \Sigma} = true$. Otherwise, this is when $|[E]|^{I_H + \Sigma} = false$ we know that $K \models E$ is false because the Herbrand model doesn't satisfy E (see the construction of Σ in the previous lemma), and since the rules are all valid we can't construct a proof tree. This finish the demonstration of the theorem. ■

6 RDF and First order logic

In this section we will try to show a parallel between RDF and First order logic. We will do so by showing how to translate the an rdf graph into a first order logic formula, and we will try to expose some results on interpretations and entailment. We will focus this discussion on rdfs graphs and on how to prove rdfs entailment.

Since formulas in FOL are finite we will restrict this analysis to finite rdf graphs.

The language over which we will write our formulas will be formed by an infinite set of constants, containing one symbol for each uri, literal and blank node, a 3-ary predicate symbol st and two unary predicates Lit and $XMLLit$.

Definition 31 *The translation $\tilde{t}$ of a triple $t = (s, p, o)$ will be defined as:*
if o is a plain literal then: $\tilde{t} = st(s, p, o) \wedge Lit(o)$
if o is a well typed literal then: $\tilde{t} = st(s, p, o) \wedge XMLLit(o)$
otherwise: $\tilde{t} = st(s, p, o)$
And given an rdf graph K, we will define the translation of K as:
$\tilde{K} = \bigwedge_{t \in K \cup Axioms} \tilde{t}$

In order to provide this formulas with a semantic meaning we will consider the following set of formulas which we will call $rdfsA$:

- $\forall s, \forall p, \forall o, st(s, p, o) \Rightarrow st(p, type, prop) \wedge st(s, type, res) \wedge st(p, type, res) \wedge st(o, type, res)$
- $\forall l, Lit(l) \Leftrightarrow st(l, type, literal)$
- $\forall l, XMLLit(l) \Leftrightarrow st(l, type, xmlliteral)$
- $\forall s, \forall p, \forall o, \forall p', st(s, p, o) \wedge st(p, sp, p') \Rightarrow st(s, p', o)$
- $\forall s, st(s, type, prop) \Rightarrow st(s, sp, s)$

- $\forall s, \forall x, \forall o, st(s, sp, x) \wedge st(x, sp, o) \Rightarrow st(s, sp, o)$
- $\forall s, st(s, type, ContMemP) \Rightarrow st(s, sp, member)$
- $\forall s, st(s, type, Class) \Rightarrow st(s, sc, s) \wedge st(s, sc, res)$
- $\forall s, \forall x, \forall o, st(s, sc, x) \wedge st(x, sc, o) \Rightarrow st(s, sc, o)$
- $\forall s, st(a, type, Datatype) \Rightarrow st(a, sc, literal)$
- $\forall u, \forall v, \forall x, \forall y, st(x, domain, y) \wedge st(u, x, v) \Rightarrow st(u, type, y)$
- $\forall u, \forall v, \forall x, \forall y, st(x, range, y) \wedge st(u, x, v) \Rightarrow st(v, type, y)$

Instead of considering only the translation of an rdf graph K we will consider the theory of the set formed by this formula $\tilde{K}$ and $rdfsA$. Note that there are no restrictions in the theory as to prevent a blank node to be in the position of a predicate, nor to a literal to be in the position of a subject, so there are in this theory many formulas which can't be translated back into rdfs graphs, however we will see that if a translation of another graph G is a formula of this theory, then $K \models_{rdfs} G$.

We will announce without giving a proof the following proposition:

Proposition 7 (translation) *Given two rdfs graphs K and E such that neither of them is an XML clash, $K \models_{rdfs} E \Leftrightarrow \exists \sigma$ such that $\tilde{K} \models_{1st} \tilde{E}[\sigma(x)/x]$, where σ is a map from the set of blank nodes of E to blank nodes, uris and literals, and $\tilde{E}[\sigma(x)/x]$ is the translation of E in which we have replaced all the occurrences of x for $\sigma(x)$, for all $x \in dom(\sigma)$.*

7 Conclusion and proposed corrections

We have presented the formal aspects of syntax and semantics of RDF following as much as possible the specifications of this language. However, there are some points that might be reviewed. Some concerning the definitions, other concerning the rules of entailment.

1. First of all we would like to point out that the definition of instance of an rdf graph given in the specifications is quite ambiguous. The proposed definition for an instance is:
   "Suppose that M is a mapping from a set of blank nodes to some set of literals, blank nodes and URI references; then any graph obtained from a graph G by replacing some or all of the blank nodes N in G by $M(N)$ is an instance of G".
   Note that G_2 is an instance of G_2 depending on the map which is being considered. For example:
   If M is the map defined by $M(\_b_1) = x$ and $M(b_2) = y$,G_1 and G_2 defined as:
   $G_1 = \{(x, u, \_b_1), (\_b_2, v, x)\}$

$G_2 = \{(x, u, x), (y, v, x)\}$
then G_2 is an instance of G_1. Now, if M is defined as $M(\_b_2) = x$ and $M(b_1) = y$ then, is G_2 an instance of G_1?
Moreover, in the first example, there is no guarantee that G_2 was "obtained" by replacing the blank nodes of G_1.
Clearly this is not a serious problem, but we think that this concept should be well defined.

2. The concept of equivalence defined in the specifications is the following:
"Two RDF graphs G and G' are equivalent if there is a bijection M between the sets of nodes of the two graphs, such that:

   - M maps blank nodes to blank nodes.
   - $M(lit) = lit$ for all RDF literals lit which are nodes of G.
   - $M(uri) = uri$ for all RDF URI references uri which are nodes of G.
   - The triple (s, p, o) is in G if and only if the triple $(M(s), p, M(o))$ is in G'."

   We think that a more suitable world for that is "isomorphism", and "equivalence" should be reserved for semantic equivalence only, this is when given two graphs, each one entails the other.

   Some other notions which may be useful and frequently used would be formalized, such as maps of rdf graphs, map consistent with a graph, morphisms, or nodes of a graph (in order to extend the vocabulary to the whole set of nodes occurring on a graph).

3. The definition of well/ill typed literals is not correct unless a datatype map is defined (see comment on section 3.4). The only definition which could be proposed without a previous definition of a datatype map is the well/ill typed XML literal. However the XML literal datatype is not completely defined in the specifications, its lexical-to-value mapping is not precised. The only thing it is said about it is that it is a bijective map. We think that the scope of the possible consequences this may have should be treated more carefully, in fact, depending on the lexical-to-value map we are considering, a graph can be either true or false in a given interpretation.

4. When defining entailment among graphs, the vocabulary should be quantified. In fact, since one should speak of an interpretation *of a vocabulary* V, talking about every interpretation is meaningless unless we specify of which vocabulary. Entailment can then be defined either as:

   - $G \models E \Leftrightarrow \forall V$ vocabulary, $\forall I$ interpretation of $V, (I \models G \Rightarrow I \models E)$ or:
   - $G \models E \Leftrightarrow \forall I$ interpretation of $V_G, (I \models G \Rightarrow I \models E)$, where V_G is the vocabulary of G.

     Note that both definitions are equivalent.

5. Finally, and most important of all, we have noticed that the rdfs entailment lemma is not true. In fact if we consider the following graph:
   $G = \{(a,\texttt{rdfs:sp}, \_b), (\_b,\texttt{rdfs:range}, c), (x, a, y)\}$
   where $\_b$ is a blank node and the rest are all URIs, we can see that G entails the triple $(y,\texttt{rdf:type}, c)$. However there are no rules which permit to deduce this triple.
   When looking at the demonstration we can see that the definition of the rdfs Herbrand interpretation is far from being complete. Note the following:

   If x is in IPRH then IEXTRH(x) = {¡s,o¿: D contains the triple sur(s) x sur(o) . }

   The semantic conditions would force that:

   IEXTRH(x) = {¡s,o¿: D contains the triple sur(s) x sur(o) or there exists y such that IEXTRH(y) contains ¡s,o¿ }

   This semantic condition is supposed to be guaranteed by applying rule $rdfs7$, but it can only be applied if the resulting triple is well formed, this is, its predicate cannot be a blank node.

   We have proposed a set of rules which corrects this problem by changing rules $rdfs2$ and $rdfs3$. These rules also formalize the notion of blank node "allocated" to x.

References

[URI] http://www.isi.edu/in-notes/rfc2396.txt

[RDF-Concepts] http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/

[RDF-Semantics] http://www.w3.org/TR/2004/REC-rdf-mt-20040210/

[XML-Schema2] http://www.w3.org/TR/xmlschema-2/

[RFC-3066] http://www.isi.edu/in-notes/rfc3066.txt