Discussion on the semantics of $rdfs$:*Resource*, $rdfs$:*Class*, rdf:*Property*

Giorgos Flouris, Irini Fundulaki

July 10, 2008

Introduction

This document attempts to describe the semantics and intended use of the most basic RDF constructs ($rdfs$:*Resource*, $rdfs$:*Class*, rdf:*Property*), in order to enforce a uniform treatment of those constructs throughout the entire system.

The term class will not be used as it is ambiguous. We'll use the terms "schema class" (for a class at the schema level), "metaclass" (for a class at the meta level, whose instances are schema classes) and "metaproperty" (for a class at meta level, whose instances are properties at the schema level). The term "individual" will refer to schema class instances (i.e., to individuals at the data level, which are instances of schema classes). The term "property" will refer to properties at the schema level and the term "property instances" to pairs of individuals which are instances of a property.

Data Types

We introduce *SWKM data types* in our framework in order to be able to capture the intricacies of RDF Schema. We provide three sets of typing rules: *i)* set *RDFS* consists of typing rules that allow one to attribute an SWKM data type to one of the RDFS constructs ($rdfs$:*Class*, $rdfs$:*Resource*, etc.); *ii)* set O consists of typing rules that allow one to associate an RDF resource specified in a user ontology to one of the RDFS data types, usually used when the type of either the subject or the object of the triple is known; and *iii* set D consists of typing rules specifying some "default" behavior to be used when the other typing rules don't allow a type inference to be made for some RDF resource.

More specifically, we introduce the following data types that we consider in our model (we follow the idea from [1]): MetaClass, SchemaClass, MetaProperty, SchemaProperty, Individual. A metaclass is of type MetaClass, a schema class is of type SchemaClass, a metaproperty is of type MetaProperty, a schema property is of type SchemaProperty and finally an instance of a schema class is of type Individual. Following the RDF Schema specification document (and the proposal from [1]), we note the following:

Rules $RDFS_1$ to $RDFS_8$ associate to each of the RDF Schema constructs (that is, $rdfs$:*Class*, rdf:*Property*, $rdfs$:*Resource*, rdf:*type*, etc.) one of the data types in our framework.

1. $rdfs$:*Class* is a metaclass. All metaclasses should be subclasses of $rdfs$:*Class*, i.e., $rdfs$:*Class* is the root of all metaclasses. All schema classes should be instances of $rdfs$:*Class*. The triple [A rdf:*type* $rdfs$:*Class*] should exist for all schema classes, metaclasses and metaproperties A.

2. $rdfs$:*Resource* is a schema class. All schema classes should be subclasses of $rdfs$:*Resource*, i.e., $rdfs$:*Resource* is the root of all schema classes. All individuals should be instances of $rdfs$:*Resource*. $rdfs$:*Resource* itself is an instance of $rdfs$:*Class*.

3. rdf:*Property*, $rdfs$:*subPropertyOf*, $rdfs$:*subClassOf*, rdf:*type*, $rdfs$:*domain*, $rdfs$:*range* are all metaproperties. All metaproperties (except $rdfs$:*subPropertyOf*, $rdfs$:*subClassOf*, rdf:*type*, $rdfs$:*domain*, $rdfs$:*range*) should be subclasses of rdf:*Property*, i.e., rdf:*Property* is the root of all metaproperties. All properties should be instances of rdf:*Property*. rdf:*Property* itself is an instance of $rdfs$:*Class*.

$$RDFS_1: \quad \frac{}{rdfs{:}Class\ is\ of\ type\ \mathsf{MetaClass}}$$

$$RDFS_2: \quad \frac{}{rdf{:}Property\ is\ of\ type\ \mathsf{MetaProperty}}$$

$$RDFS_3: \quad \frac{}{rdfs{:}Resource\ is\ of\ type\ \mathsf{SchemaClass}}$$

$$RDFS_4: \quad \frac{}{rdfs{:}domain\ is\ of\ type\ \mathsf{MetaProperty}}$$

$$RDFS_5: \quad \frac{}{rdfs{:}range\ is\ of\ type\ \mathsf{MetaProperty}}$$

$$RDFS_6: \quad \frac{}{rdfs{:}subPropertyOf\ is\ of\ type\ \mathsf{MetaProperty}}$$

$$RDFS_7: \quad \frac{}{rdfs{:}subClassOf\ is\ of\ type\ \mathsf{MetaProperty}}$$

$$RDFS_8: \quad \frac{}{rdf{:}type\ is\ of\ type\ \mathsf{MetaProperty}}$$

Our objective is to define *typing rules* that allow one to associate an RDF resource (recall that according to RDF Vocabulary Description Language 1.0: RDF Schema)

"All things described by RDF are called resources, ..."

to one of the data types in our framework. These rules take as input RDF triples and associate with each RDF resource one of MetaClass, SchemaClass, MetaProperty, SchemaProperty, Individual. We consider that those typing rules *suffice* to derive the data type of the RDF resource.

We have to note here that the triples may be there either explicitly or implicitly. We say that a triple is an implicit triple if this is derived by applying the known inference rules on $rdfs{:}subClassOf$ $rdfs{:}subPropertyOf$ and $rdf{:}type$ RDF properties. The rules are assumed to be taken over the entire set of triples constituting an RDF KB, i.e., the order does not count.

$$O_1 : \quad \frac{A\ rdfs{:}subClassOf\ B\ and\ (B\ is\ of\ type\ \textsf{MetaClass}\ or\ A\ is\ of\ type\ \textsf{MetaClass})}{A\ is\ of\ type\ \textsf{MetaClass}\ and\ B\ is\ of\ type\ \textsf{MetaClass}}$$

$$O_2 : \quad \frac{A\ rdfs{:}subClassOf\ B\ and\ (B\ is\ of\ type\ \textsf{MetaProperty}\ or\ A\ is\ of\ type\ \textsf{MetaProperty})}{A\ is\ of\ type\ \textsf{MetaProperty}\ and\ B\ is\ of\ type\ \textsf{MetaProperty}}$$

$$O_3 : \quad \frac{A\ rdfs{:}subClassOf\ B\ and\ (B\ is\ of\ type\ \textsf{SchemaClass}\ or\ A\ is\ of\ type\ \textsf{SchemaClass})}{A\ is\ of\ type\ \textsf{SchemaClass}\ and\ B\ is\ of\ type\ \textsf{SchemaClass}}$$

$$O_4 : \quad \frac{A\ rdfs{:}subPropertyOf\ B}{A\ is\ of\ type\ \textsf{SchemaProperty}\ and\ B\ is\ of\ type\ \textsf{SchemaProperty}}$$

$$O_5 : \quad \frac{A\ rdf{:}type\ B\ and\ B\ is\ of\ type\ \textsf{MetaClass}\ and\ B \neq rdfs{:}Class}{A\ is\ of\ type\ \textsf{SchemaClass}}$$

$$O_6 : \quad \frac{A\ rdf{:}type\ B\ and\ B\ is\ of\ type\ \textsf{MetaProperty}}{A\ is\ of\ type\ \textsf{SchemaProperty}}$$

$$O_7 : \quad \frac{A\ rdf{:}type\ B\ and\ B\ is\ of\ type\ \textsf{SchemaClass}}{A\ is\ of\ type\ \textsf{Individual}}$$

$$O_8 : \quad \frac{A\ rdf{:}type\ B\ and\ A\ is\ of\ type\ \textsf{SchemaClass}}{A\ is\ of\ type\ \textsf{MetaClass}}$$

$$O_9 : \quad \frac{A\ rdf{:}type\ B\ and\ A\ is\ of\ type\ \textsf{SchemaProperty}}{A\ is\ of\ type\ \textsf{MetaProperty}}$$

$$O_{10} : \quad \frac{A\ rdf{:}type\ B\ and\ A\ is\ of\ type\ \textsf{Individual}}{A\ is\ of\ type\ \textsf{SchemaClass}}$$

$$O_{11} : \quad \frac{A\ rdfs{:}domain\ B}{A\ is\ of\ type\ \textsf{SchemaProperty}}$$

$$O_{12} : \quad \frac{A\ rdfs{:}range\ B}{A\ is\ of\ type\ \textsf{SchemaProperty}}$$

where A is a user defined resource in an RDF ontology $\mathcal{O}$ and B can be either a user defined resource in an RDF Ontology $\mathcal{O}$ or one of $rdfs{:}Class$, $rdfs{:}Resource$, $rdf{:}Property$.

The following set of rules provide a “default” behavior, when the types of the subject and object of the triple under question is unknown. Note that the term “unknown” in this respect refers to the types being unknown *after* the entire set of triples has been read, not at some intermediate point.

$$D_1 : \quad \frac{A\ rdf{:}type\ rdfs{:}Class\ and\ not\ (A\ is\ of\ type\ \mathsf{MetaClass}\ or\ A\ is\ of\ type\ \mathsf{MetaProperty}\ )}{A\ is\ of\ type\ \mathsf{SchemaClass}}$$

$$D_2 : \quad \frac{A\ rdf{:}type\ B\ and\ not\ (A\ is\ of\ type\ \mathsf{SchemaClass}\ orA\ is\ of\ type\ \mathsf{SchemaProperty}\ orB\ is\ of\ type\ \mathsf{MetaClass}\ orB\ i}{A\ is\ of\ type\ \mathsf{Individual}\ and\ B\ is\ of\ type\ \mathsf{SchemaClass}}$$

$$D_3 : \quad \frac{A\ rdfs{:}subClassOf\ B\ and\ not\ (A\ is\ of\ type\ \mathsf{MetaClass}\ orA\ is\ of\ type\ \mathsf{MetaProperty}\ orB\ is\ of\ type\ \mathsf{MetaClass}\ o}{A\ is\ of\ type\ \mathsf{SchemaClass}\ and\ B\ is\ of\ type\ \mathsf{SchemaClass}}$$

$$D_4 : \quad \frac{A\ rdfs{:}domain\ B\ and\ not\ (B\ is\ of\ type\ \mathsf{MetaClass}\ or\ B\ is\ of\ type\ \mathsf{MetaProperty}\ )}{B\ is\ of\ type\ \mathsf{SchemaClass}}$$

$$D_5 : \quad \frac{A\ rdfs{:}range\ B\ and\ not\ (B\ is\ of\ type\ \mathsf{MetaClass}\ or\ B\ is\ of\ type\ \mathsf{MetaProperty}\ )}{B\ is\ of\ type\ \mathsf{SchemaClass}}$$

Type Errors

We are going to use the SWKM data types introduced in the previous section to define type mismatch errors. In short, we consider that all SWKM types introduced in the previous section are incomparable.

$$E_1 : \frac{A\ is\ of\ type\ \textsf{MetaClass}\ and\ A\ is\ of\ type\ \textsf{SchemaClass}}{type\ mismatch\ error}$$

$$E_2 : \frac{A\ is\ of\ type\ \textsf{MetaClass}\ and\ A\ is\ of\ type\ \textsf{MetaProperty}}{type\ mismatch\ error}$$

$$E_3 : \frac{A\ is\ of\ type\ \textsf{MetaClass}\ and\ A\ is\ of\ type\ \textsf{SchemaProperty}}{type\ mismatch\ error}$$

$$E_4 : \frac{A\ is\ of\ type\ \textsf{MetaClass}\ and\ A\ is\ of\ type\ \textsf{Individual}}{type\ mismatch\ error}$$

$$E_5 : \frac{A\ is\ of\ type\ \textsf{SchemaClass}\ and\ A\ is\ of\ type\ \textsf{MetaProperty}}{type\ mismatch\ error}$$

$$E_6 : \frac{A\ is\ of\ type\ \textsf{SchemaClass}\ and\ A\ is\ of\ type\ \textsf{SchemaProperty}}{type\ mismatch\ error}$$

$$E_7 : \frac{A\ is\ of\ type\ \textsf{SchemaClass}\ and\ A\ is\ of\ type\ \textsf{Individual}}{type\ mismatch\ error}$$

$$E_8 : \frac{A\ is\ of\ type\ \textsf{MetaProperty}\ and\ A\ is\ of\ type\ \textsf{SchemaProperty}}{type\ mismatch\ error}$$

$$E_9 : \frac{A\ is\ of\ type\ \textsf{MetaProperty}\ and\ A\ is\ of\ type\ \textsf{Individual}}{type\ mismatch\ error}$$

$$E_{10} : \frac{A\ is\ of\ type\ \textsf{SchemaProperty}\ and\ A\ is\ of\ type\ \textsf{Individual}}{type\ mismatch\ error}$$

We also consider that certain constructs must have specific positions in a triple; if they don't have the "expected" position, a *not allowed position* error should be raised. For example, any triple that has as subject one of $rdfs$-:$Resource$, $rdfs$:$Class$, rdf:$Property$should raise such an error. That is, we forbid one to include *explicit* triples that modify or refer to as subject one of $rdfs$:$Class$, $rdfs$:$Resource$, rdf:$Property$.

$$E_{11} : \frac{rdfs{:}Resource\ p\ o}{not\ allowed\ position\ error}$$

$$E_{12} : \frac{rdfs{:}Class\ p\ o}{not\ allowed\ position\ error}$$

$$E_{13} : \frac{rdf{:}Property\ p\ o}{not\ allowed\ position\ error}$$

$$E_{14} : \frac{s\ rdfs{:}Resource\ o}{not\ allowed\ position\ error}$$

$$E_{15} : \frac{s\ rdfs{:}Class\ o}{not\ allowed\ position\ error}$$

$$E_{16} : \frac{s\ rdf{:}Property\ o}{not\ allowed\ position\ error}$$

$$E_{17} : \frac{rdfs{:}range\ p\ o}{not\ allowed\ position\ error}$$

$$E_{18} : \frac{s\ p\ rdfs{:}range}{not\ allowed\ position\ error}$$

$$E_{19} : \frac{rdfs{:}domain\ p\ o}{not\ allowed\ position\ error}$$

$$E_{20} : \frac{s\ p\ rdfs{:}domain}{not\ allowed\ position\ error}$$

$$E_{21} : \frac{rdfs{:}subPropertyOf\ p\ o}{not\ allowed\ position\ error}$$

$$E_{22} : \frac{s\ p\ rdfs{:}subPropertyOf}{not\ allowed\ position\ error}$$

$$E_{23} : \frac{rdfs{:}subClassOf\ p\ o}{not\ allowed\ position\ error}$$

$$E_{24} : \frac{s\ p\ rdfs{:}subClassOf}{not\ allowed\ position\ error}$$

$$E_{25} : \frac{rdf{:}type\ p\ o}{not\ allowed\ position\ error}$$

$$E_{26} : \frac{s\ p\ rdf{:}type}{not\ allowed\ position\ error}$$

with s, p, o are free variables that range over resources.

Typing Inference

In this section we describe how typing information can be used to generate new triples.

As was made clear by the typing rules above (especially the default ones – D_i), we can always infer the type of a resource by the triples it participates in; however, the type is not always explicit, or is not always resulting by the application of a single rule. For this purpose, we propose the introduction of typing inference: whenever the type of a certain resource becomes known, we add all the triples that are associated with this type. For example, a metaclass A should be involved in the triples [A *rdf:type rdfs:Class*] and [A *rdfs:subClassOf rdfs:Class*].

The following type inference rules apply:

$$I_1 : \quad \frac{A\ is\ of\ type\ \textsf{MetaClass}}{A\ rdf{:}type\ rdfs{:}Class\ and\ A\ rdfs{:}subClassOf\ rdfs{:}Class}$$

$$I_2 : \quad \frac{A\ is\ of\ type\ \textsf{MetaProperty}}{A\ rdf{:}type\ rdfs{:}Class\ and\ A\ rdfs{:}subClassOf\ rdf{:}Property}$$

$$I_3 : \quad \frac{A\ is\ of\ type\ \textsf{SchemaClass}}{A\ rdf{:}type\ rdfs{:}Class\ and\ A\ rdfs{:}subClassOf\ rdfs{:}Resource}$$

$$I_4 : \quad \frac{A\ is\ of\ type\ \textsf{SchemaProperty}}{A\ rdf{:}type\ rdf{:}Property}$$

$$I_5 : \quad \frac{A\ is\ of\ type\ \textsf{Individual}}{A\ rdf{:}type\ rdfs{:}Resource}$$

Standard Inference

In this section we describe how standard inference can be used to generate new triples.

We consider the following standard inference rules:

$$C_1 : \quad \frac{Ardfs{:}subClassOf\ B\ and\ Brdfs{:}subClassOf\ C}{A\ rdfs{:}subClassOf\ C}$$

$$C_2 : \quad \frac{Ardfs{:}subPropertyOf\ B\ and\ Brdfs{:}subPropertyOf\ C}{A\ rdfs{:}subPropertyOf\ C}$$

$$C_3 : \quad \frac{Ardf{:}type\ B\ and\ Brdfs{:}subClassOf\ C}{A\ rdf{:}type\ C}$$

Redundancy Elimination

In this section, we will describe the circumstances under which redundancy elimination should "clear out" specific triples.

Redundancy elimination should consider only triples that contain $rdf{:}type$, $rdfs{:}subClassOf$, or $rdfs{:}subPropertyOf$ triples, i.e., triples that have in "predicate" position the resource $rdf{:}type$, $rdfs{:}subClassOf$, or $rdfs{:}subPropertyOf$. In effect, redundancy elimination should consider only the implicit instantiations and implicit (class or property) IsAs for removal.

As a result, the combination of triples

1. $[A\ rdf{:}type\ rdfs{:}Class]$,
2. $[A\ rdfs{:}subClassOf\ rdfs{:}Resource]$

is not redundant. On the other hand, the triples

1. $[A\ rdfs{:}subClassOf\ B]$,
2. $[A\ rdfs{:}subClassOf\ rdfs{:}Class]$,
3. $[B\ rdfs{:}subClassOf\ rdfs{:}Class]$

contain the redundant triple, namely $[A\ rdfs{:}subClassOf\ rdfs{:}Class]$ is redundant.

The above discussion implies that this document should not affect the redundancy elimination procedures. For example, the $[x\ rdf{:}type\ rdfs{:}Resource]$ triple is not redundant, unless there is a combination of triples of the form $[x\ rdf{:}type\ A]$, $[A\ rdfs{:}subClassOf\ rdfs{:}Resource]$, for some A, which implies $[x\ rdf{:}type\ rdfs{:}Resource]$.

Therefore, we have the following redundancy elimination rules, for A, B, C any three different resources:

$$R_1 : \frac{Ardfs{:}subClassOf\ B\ and\ Brdfs{:}subClassOf\ C}{Remove\ A\ rdfs{:}subClassOf\ C}$$

$$R_2 : \frac{Ardfs{:}subPropertyOf\ B\ and\ Brdfs{:}subPropertyOf\ C}{Remove\ A\ rdfs{:}subPropertyOf\ C}$$

$$R_3 : \frac{Ardf{:}type\ B\ and\ Brdfs{:}subClassOf\ C}{Remove\ A\ rdf{:}type\ C}$$

Note that redundancy elimination should be applied *after* all type inference rules have been applied.

Labeling

This section describes the consequences of our semantics upon the labeling.

Labeling should give labels to $rdfs{:}Class$, $rdf{:}Property$ and $rdfs{:}Resource$ normally. According to our semantics, the $rdfs{:}Class$ should be the root of a hierarchy containing all metaclasses (and only metaclasses). Similarly, the *rdf*:*Property* should be the root of a hierarchy containing all metaproperties (and only metaproperties) and the *rdfs*:*Resource* should be the root of a hierarchy containing all schema classes (and only schema classes).

All hierarchies should have as root one of *rdfs*:*Class*, *rdf*:*Property* or *rdfs*-:*Resource*, unless the hierarchy contains (only) schema properties, in which case it can have any schema property as a root. Thus:

- There are only three hierarchies involving classes (metaclasses, metaproperties and schema classes).
- All classes (metaclasses, metaproperties and schema classes) should belong to one of the three hierarchies mentioned above, depending on their type (metaclass, metaproperty or schema class).
- There should be no class belonging to more than one hierarchy.

For schema properties, there is no explicitly defined root, so there could be one or more schema property hierarchies; each hierarchy should, of course, have exactly one root, and this root should be a schema property. Schema properties are not constrained by the above rules on classes. Finally, individuals are not involved in hierarchies.

Storage Level

In this section, we describe the consequences of the above discussion upon the storage.

At the storage level, we don't store triples explicitly, but store the relationships between the various names (which represent the various objects). As a result, we store, for each name describing an object, its direct subclasses, superclasses, subproperties, superproperties, types etc, depending on the type of the object.

Based on the above discussion, we conclude that *rdfs*:*Class*, *rdf*:*Property* and *rdfs*:*Resource* should be classified in the database as a metaclass, a metaproperty and a schema class respectively. They should have no superclasses. We should store all their direct subclasses (if any) as such.

For these particular classes, we have the "strange property" that we may be able to infer some direct subclass, without explicitly having a *rdfs*:*subClassOf* relationship. For example, the combination of triples [A *rdfs*:*subClassOf* B], [A *rdfs*:*subClassOf* *rdfs*:*Class*] implies that A and B are metaclasses (by the application of rule $RDFS_1$ and the double application of O_1); in addition, if there is no other metaclass that is a superclass of B, this combination should additionally imply that B is a direct subclass of *rdfs*:*Class*˙The storage should be able to identify these cases and react accordingly.

In order to do that, we should automatically add certain triples, in the manner described in the discussion on the "type inference". Once these triples have been added, the standard redundancy elimination process (see relevant section) would remove all unnecessary triples, leaving only the direct subclass triples including direct subclass triples of the form [A $rdfs$:*subClassOf* $rdfs$:*Resource*], [A $rdfs$:*subClassOf* $rdfs$:*Class*], [A $rdfs$:*subClassOf* rdf:*Property*], and direct typing triples of the form [A rdf:*type* $rdfs$:*Resource*], [A rdf:*type* $rdfs$:*Class*], [A rdf:*type* rdf:*Property*].

Main Memory Model

In this section, we describe the consequences of the above discussion upon the main memory model. In general, for the main memory model similar things hold as for the storage level, except that redundancy elimination is not performed.

One problem with the Main Memory Model is that it receives the triples in a sequence, and we can never be sure whether the end of the sequence has been reached. To avoid this problem, we propose the following procedure:

- At any point in the sequence, the main memory model will apply the rules $RDFS_i$ and O_i to determine the typing.
- At any point in the sequence, the main memory model will apply the rules E_i to identify any errors.
- The rules D_i (default rules), I_i (type inference), R_i (redundancy elimination) will not be applied by default.

In addition, there should exist two special function calls in the API:

- There should be a function call signaling the end of the triple sequence, at which point the default rules (D_i) and the type inference rules (I_i) should be applied; following the application of these rules, any errors should be identified (using rules E_i).
- There should be a function call requesting for redundancy elimination to take place. In order to do that, the rules R_i should be used.
- There should be a function call requesting for closure inference to take place. In order to do that, the rules D_i, I_i and C_i should be used.

Export Service

In this section we describe the expected behavior of the Export Service.

The export service reads the database and exports the contents of a namespace into a TRIG or RDF/XML file. There are no particular difficulties here: whatever is stored in the database should be exported.

The special triples [$Ardf$:*type* $rdfs$:*Class*] etc should be exported, per the type inference rules, I_i, provided that they are not redundant (see rules R_i). For example, for a schema class, the triples [$Ardf$:*type* $rdfs$:*Class*] and [$Ardfs$:*subClassOf* $rdfs$-:*Resource*] should be reported, provided that they are not redundant.

RUL/RQL

This section describes the expected behavior for the RUL/RQL. This behavior is basically dictated by the expected behavior of the labeling and the inference mechanism.

In particular, RQL should report subclasses/superclasses etc per the hierarchies defined in the labeling (see the relevant section). In all cases, the special classes $rdfs$:*Resource*, $rdfs$:*Class*, *rdf*:*Property* should be reported, if they are part of the output; for example, if we ask for the superclasses of a schema class A, then $rdfs$:*Resource* should be reported, among other things; similarly, if we ask the types of A (i.e., all x for which there is a triple of the form [A *rdf*:*type* x]), $rdfs$:*Class* should be one of them.

Similarly, RUL should consider the hierarchies and typing imposed by the standard inference mechanism (i.e., rules C_i) and the labeling when calculating the side-effects of a change.

Comparison Service (MM and PS)

The expected behavior of the comparison service should be the same regardless of whether it works on the main memory or the persistent storage. The effects of this discussion on the expected behavior is described below.

When, in a delta function, the explicit knowledge is considered (i.e., K or K'), then whatever is explicitly stored (in the memory or in the database) should be considered and/or reported (including the special triples). Therefore, typing inference or standard inference should not take place. Note that there are different actions taking place by default in the main memory and the secondary storage (e.g., typing inference or redundancy elimination) and this might affect the result; it should be emphasized however that this has nothing to do with the comparison service itself, but is related to our assumptions on the storage medium (see relevant sections), which are external to the comparison service. Thus, the comparison service should not perform, itself, any action like typing inference, standard inference, redundancy elimination or other kind of "completing" the input(s).

On the other hand, when, in a delta function, both the explicit and the implicit knowledge is considered (i.e., $C(K)$ or $C(K')$), then the standard inference and the typing inference should take place, and the results of these two types of inference should be considered and/or reported.

For example, if a schema class A exists in K but not in K', then the triples [A $rdfs$:$subClassOf$ $rdfs$:*Resource*] and [A *rdf*:*type* $rdfs$:*Class*] should be reported as deleted triples (among other things), regardless of whether they are redundant or not.

Change Impact Service

The expected behavior of the Change Impact service per our assumptions is described here.

The change impact service considers the closure of the RDF KB when calculating the side-effects. As a result, both the typing inference and the standard inference should take place upon the input RDF KB, in addition to all type inference and checking ($RDFS_i$, O_i, D_i, E_i – see the section on the model) as a pre-processing before performing any side-effects calculations.

When it comes to the update, things are quite different. No inference should be performed whatsoever (neither standard nor typing inference). The update should be considered as-is. When information in the update is incomplete, some standard default decisions should be taken (see rules D_i). Note that the side-effects should support those default decisions.

For example, if the update contains just the added triple [A rdf:*type* $rdfs$:$Class$], then change impact should automatically assume that the update dictates the addition of a new schema class, A. The side-effects of this addition should include, among other things, the triple [A $rdfs$:$subClassOf$ $rdfs$:*Resource*] (see also the discussion on the output in the next paragraph). Similarly, if the triple [A $rdfs$:$subClassOf$ $rdfs$:$Class$] is removed, then the metaclass A should be removed, and the side-effects should include the triple [A rdf:*type* $rdfs$:$Class$].

Concerning the output, the change impact should respect the typing inference rules. For example, if change impact chooses to add a new schema class (as a side-effect) then both triples [A $rdfs$:$subClassOf$ $rdfs$:*Resource*] and [A rdf:*type* $rdfs$:$Class$] should be reported as "added" side-effects. Similarly, if a metaclass A is deleted, then the triples [A $rdfs$:$subClassOf$ $rdfs$:$Class$] and [A rdf:*type* $rdfs$:$Class$] should be reported as "deleted" side-effects.

$\mathbf{U}_{ir}$

This section discusses the effects of the semantics described above upon the U_{ir} module.

The only real effect upon this module is that triples of the form [A $rdfs$:$subClassOf$ $rdfs$:$Class$], [A $rdfs$:$subClassOf$ rdf:*Property*] and [A $rdfs$:$subClassOf$ $rdfs$-:*Resource*] should be treated like all the other "$rdfs$:$subClassOf$" triples (e.g., [A $rdfs$:$subClassOf$ B]).

References

[1] Gregory Karvounarakis, Sofia Alexaki, Vassilis Christophides, Dimitris Plexousakis, and Michel Scholl. Rql: a declarative query language for rdf. In *WWW*, pages 592–603, 2002.