Building the Semantic Web Tower from RDF Straw

Peter F. Patel-Schneider
Bell Labs Research
600 Mountain Ave, Murray Hill, NJ 07974, U. S. A.
pfps@research.bell-labs.com

Abstract

A same-syntax extension of RDF to first-order logic results in a collapse of the model theory due to logical paradoxes resulting from diagonalization. RDF is thus the wrong material for building the Semantic Web tower.

1 The Semantic Web

> The Semantic Web is an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation. [Berners-Lee *et al.*, 2001]

Thus the goals of the Semantic Web are very similar to the goals of Knowledge Representation—to provide means for representing information in a way that can be processed by machines.

The base language of the Semantic Web is RDF [Manola and Miller, 2004], a simple representation language based on a graph structure.

> The Semantic Web provides a common framework that allows data to be shared and reused across application, enterprise, and community boundaries. [...] It is based on the Resource Description Framework (RDF), which integrates a variety of applications using XML for syntax and URIs for naming. [http://www.w3.org/2001/sw/]

RDF is based on labelled graphs—an RDF knowledge base is (roughly) a graph where some of the nodes have labels (in the form of URI references and literal data) and the edges have labels (which are also URI references). Node labels are unique, i.e., a URI reference can be the label of at most one node.

The normative surface syntax for RDF is XML. This syntax is called RDF/XML [Dave Beckett, 2004]. There are other surface syntaxes for RDF, including N-triples [Jan Grant and Dave Beckett, 2004]. What counts in RDF is the graph that is conveyed, and not any other aspect of the surface syntax.

RDF, however, is more than just a graph language. It is actually a logic, with a model theoretic semantics and an entailment relationship. The details of all this have been specified in an W3C Recommendation [Hayes, 2004]. In essence, an RDF graph can be read as a collection of facts corresponding to the edges in the graph (which are generally referred to as triples) where an edge with label R (the property) from node n_1 (the subject) to node n_2 (the object) corresponds to the fact $\mathsf{R}(n_1, n_2)$. If n_1 (n_2) has a label then it is treated as a constant and its label is the name of the constant, if n_1 (n_2) doesn't have a label then it is treated as an existentially quantified variable with some arbitrary name. The model theory for RDF is slightly non-standard, to provide an integrated meaning for edge labels that are also node labels.

So in RDF the only way of providing information is via triples (graph edges). These triples are often written informally in a form very similar to N-triples. An edge with label R from a node labelled with the URI reference A to node labelled with the URI reference B is written as

```
A R B .
```

As URI references can be very long, names of the form N:L are very often used where the URI reference is constructed by concatenating the expansion of N to L. (The expansion is generally specified informally or left unspecified.) Nodes without labels are given "local tags" of the form _:l. These tags are not URI references or constant names; they instead correspond to existentially quantified variables.

Because RDF has only triples, there is no way of providing syntactic constructs beyond what is specified above (i.e., binary facts, implicit conjunction, and existential quantification). Any other syntactic constructs have to be encoded as triples and their special meaning conveyed by stating semantic conditions in the model theory.

RDF has only a few of these semantic conditions. There is a particular property, rdf:type, which is considered to be the typing construct, i.e., the triple

```
A rdf:type B .
```

is interpreted as A has type B (or A belongs to the class B). There is a semantic condition that any name used as an edge label, i.e., as a property, must have type rdf:Property.

RDF is thus a very simple representation language. However, triples are a very powerful construct in that any ground or existentially quantified atomic fact can be encoded in RDF triples by by using a new unnamed node with triples that link it to each of its arguments. This simple trick has been used in many other knowledge representation formalisms.

2 Extending the Semantic Web

The power of triples has given rise to a vision that RDF should be the universal language for the Semantic Web. This means that the syntax of all Semantic Web languages should be (RDF) triples and their meaning should be specified as semantic conditions that add to the RDF model theory. (This is often referred to as a same-syntax extension of RDF.)

This process was successfully carried out with RDF Schema [Brinkley and Guha, 2004], an extension to RDF that provides simple vocabulary structuring facilities. RDF Schema interprets nodes labelled with particular URI references as providing these vocabulary structuring facilities. In particular, rdfs:Class is interpreted as the type of all types, so that if something is used as a type its types must include the interpretation of rdfs:Class (and objects of type rdfs:Class are called classes); and rdfs:subClassOf is interpreted as requiring a subtyping relationship between its arguments. These interpretations are provided by appropriate semantic conditions that extend the RDF model theory to the RDF Schema model theory [Hayes, 2004].

Problems first arose in this vision when the Semantic Web was extended to include an ontology language. The charter for the group charged with doing this extension included the requirement that the language "will be designed for maximum compatibility with XML and RDF language conventions" [http://www.w3.org/2002/11/swv2/charters/WebOntologyCharter]. This was interpreted for the working group as requiring that the resultant language be a same-syntax extension of RDF.

After considerable work, and some compromises, the resultant language, OWL [Dean *et al.*, 2004], was finally specified as a same-syntax extension of RDF [Patel-Schneider *et al.*, 2004]. Because the only syntax allowed is triples, complex OWL constructs (such as Description Logic [Baader *et al.*, 2003] restrictions) have to be encoded as a number of triples. For example, the Description Logic restriction that represents everything whose friends are lawyers ($\forall$ex:friend.ex:lawyer) is encoded as the triples

_:x rdf:type owl:Restriction .
_:x owl:onProperty ex:friend .
_:x owl:allValuesFrom ex:lawyer .

Because all triples are *facts*, entailment would not work correctly in OWL unless OWL interpretations required that there be facts corresponding to every possible piece of complex OWL syntax. (These requirements are often called *comprehension principles*.) Otherwise, it would not be the case that ex:John belonging to $\forall$ex:friend.ex:lawyer entailed ex:John belonging to $\forall$ex:friend.(ex:lawyer $\sqcup$ ex:doctor), the restriction that represents everything that whose friends are lawyers or doctors. (Why? Because there might not be enough facts to encode the union of lawyers and doctors.)

The success of RDF Schema and OWL as same-syntax extensions of RDF has lead to several calls to make another extension of the Semantic Web in a similar style to a first-order logic language.

What do I mean here by a "similar style"? Well, "style" is in the eye of the beholder, but some ground rules can be laid down. First, the syntactic constructs of the extension have to be encoded as RDF triples. This means that, for example, encoding first order sentences as strings or Goedel numbers is not permitted. Second, the extension has to give meaning to every RDF graph. This means that the extension can't identify a subset of RDF graphs (such as tree-like RDF graphs) as well-behaved, and only provide meaning for these graphs.

What then would an "RDF Logic" look like? It would provide enough constructs to nicely provide facilities for all of first-order logic (including predicates, constants, variables, conjunction, disjunction, negation, quantification, equality, functions, etc.). Its entailment relationship would, when restricted to the constructs that are used to provide the facilities of first-order logic, mirror entailment in first-order logic.

A language suitable for this purpose is binary predicate calculus with equality. (Recall that predicate calculus is function-free first-order logic without function symbols and equality [Barr and Feigenbaum, 1981, Volume 1, p. 163].) As indicated above, binary predicates can be used to encode predicates of arbitrary arity, also equality can be used to encode functions thus obtaining all the facilities of first-order logic.

3 RDF Logic

Our task is thus to design a same-syntax extension of RDF to binary predicate calculus with equality. We are defining "logical" interpretations, special RDF interpretations that provide a mechanism for logical constructs. We both add some new vocabulary and some new semantic conditions, in the same style as used in the RDF model theory [Hayes, 2004].

3.1 Equality

Equality is the easiest part of our task. All we need to do is to "anoint" a particular URI reference as the equality predicate by providing its semantic conditions. As OWL already has a notion of equality, we will use the URI reference that it uses for this purpose, which we will write, using the standard abbreviation, as owl:sameAs.

The semantic condition for equality is

Equality Condition:
$\langle x,y\rangle \in IEXT(I(\text{owl:sameAs}))$ iff $x = y$

This condition says that in any logical interpretation two domain elements are related by the relationship that is the property interpretation (I followed by $IEXT$)[1] of owl:sameAs precisely if they are the same.

3.2 Encoding Formulae

Because of the need to use only RDF syntax we have to have an encoding facility for predicate calculus formulae. There are many possible schemes to do so, but let's pick one that uses RDF as much as possible.[2] As in OWL, we are also

[1] Here we see the slightly unusual nature of the RDF semantics. In a standard semantics for first-order logic we would go directly from the syntactic token owl:sameAs to its property interpretation. Here, instead, there is a preliminary mapping to an object in the domain of discourse (I) followed by a mapping to the property interpretation ($IEXT$). This means that properties are objects of the domain of discourse.

[2] It really doesn't matter which one as long as formulae can be "manipulated" within the formalism itself.

going to need comprehension principles so that the formulae exist in all interpretations, so we might as well introduce them along with the syntax encoding.

We will use lx:Formula as the class of formulae and lx:Atomic as the class of atomic formulae. We can't just use RDF triples directly as atomic formulae because these formulae might not be true and they might have variables in them.

We require that atomic formulae be formulae, i.e.,

Atomic Formulae Condition:
$ICEXT(I(\text{lx:Atomic})) \subseteq ICEXT(I(\text{lx:Formula}))$

We are here using some terminology from the RDF Schema semantic conditions. $ICEXT(x)$, the class extension of x, is defined as those domain elements that have x as one of their types.

We will use lx:predicate, lx:subject, lx:object, *and* lx:subjectv and lx:objectv to construct atomic formulae. In our syntax anything can be a variable, so we need to distinguish when something is being used as a variable or not and thus we need two variants of the subject and object links. We do not allow predicates to be variables, i.e., our logic is syntactically first-order, not higher-order.

Our first comprehension principles are concerned with atomic formulae, requiring that all atomic formulae exist in all logical interpretations. There are four different comprehension principles corresponding to the different ways variables can be used in atomic formulae.

Atomic Formulae Comprehension Principles:
$\forall r \in IP \;\; \forall x,y \in IR$[3]

$$\begin{array}{ll}
\exists f_1 \in IR & f_1 \in ICEXT(I(\text{lx:Atomic})) \;\&\; \\
& \langle f_1,x\rangle \in IEXT(I(\text{lx:subject})) \;\&\; \\
& \langle f_1,r\rangle \in IEXT(I(\text{lx:predicate})) \;\&\; \\
& \langle f_1,y\rangle \in IEXT(I(\text{lx:object})); \\
\exists f_2 \in IR & f_2 \in ICEXT(I(\text{lx:Atomic})) \;\&\; \\
& \langle f_2,x\rangle \in IEXT(I(\text{lx:subjectv})) \;\&\; \\
& \langle f_2,r\rangle \in IEXT(I(\text{lx:predicate})) \;\&\; \\
& \langle f_2,y\rangle \in IEXT(I(\text{lx:object})); \\
\exists f_3 \in IR & f_3 \in ICEXT(I(\text{lx:Atomic})) \;\&\; \\
& \langle f_3,x\rangle \in IEXT(I(\text{lx:subject})) \;\&\; \\
& \langle f_3,r\rangle \in IEXT(I(\text{lx:predicate})) \;\&\; \\
& \langle f_3,y\rangle \in IEXT(I(\text{lx:objectv})); \\
\exists f_4 \in IR & f_4 \in ICEXT(I(\text{lx:Atomic})) \;\&\; \\
& \langle f_4,x\rangle \in IEXT(I(\text{lx:subjectv})) \;\&\; \\
& \langle f_4,r\rangle \in IEXT(I(\text{lx:predicate})) \;\&\; \\
& \langle f_4,y\rangle \in IEXT(I(\text{lx:objectv})).
\end{array}$$

We will use a limited but complete set of connectives to construct composite formulae, so lx:Negation, lx:Conjunction, and lx:Exists are our only classes of composite formulae. We will use lx:left and lx:right as our structural connectives for all composite formulae.

We require that composite formulae be formulae, i.e.,

Composite Formulae Conditions:
$ICEXT(I(\text{lx:Negation})) \subseteq ICEXT(I(\text{lx:Formula}))$
$ICEXT(I(\text{lx:Conjunction})) \subseteq ICEXT(I(\text{lx:Formula}))$
$ICEXT(I(\text{lx:Exists})) \subseteq ICEXT(I(\text{lx:Formula}))$

[3]IP is the set of all RDF properties. IR is the set of all RDF domain elements, including properties.

We also need the composite formulae to exist.

Negation Comprehension Principle:
$\forall f \in ICEXT(I(\text{lx:Formula}))$
$\exists n \in ICEXT(I(\text{lx:Negation}))$
$\langle n,f\rangle \in IEXT(I(\text{lx:left}))$

Conjunction Comprehension Principle:
$\forall f,g \in ICEXT(I(\text{lx:Formula}))$
$\exists j \in ICEXT(I(\text{lx:Conjunction}))$
$\langle j,f\rangle \in IEXT(I(\text{lx:left})) \;\&\; \langle j,g\rangle \in IEXT(I(\text{lx:right}))$

Quantification Comprehension Principle:
$\forall f \in ICEXT(I(\text{lx:Formula})),\; \forall v \in IR$
$\exists q \in ICEXT(I(\text{lx:Exists}))$
$\langle q,v\rangle \in IEXT(I(\text{lx:left})) \;\&\; \langle q,f\rangle \in IEXT(I(\text{lx:right}))$

We have now guaranteed that all finite non-circular formulae exist in all interpretations. Lots of other formulae or quasi-formulae might exist as well, of course, but (in keeping with the permissive nature of RDF) we won't rule them out.

3.3 Truth

So far all we have is the existence of formulae, not whether they are true or not. We now need to provide a mechanism for determining the truth of formulae in an interpretation. We start out doing this meta-theoretically and only later provide syntax for determining true formulae.

Because this is a first-order logic, i.e., it has variables, we need some mapping from variables to their values. Our mechanism needs to be slightly different from that in standard first-order logic because our variables are just elements of the domain of discourse.

Definition 1 (Variable mapping) *Let $\mathcal{V}$ be the set of total mappings from IR to IR.*

Now we can provide a meta-theoretic mapping that, given a formula and a variable mapping, assigns a truth value to the formula.

Definition 2 (Formula truth) *Let Bool be a total mapping from $IR \times \mathcal{V}$ into boolean values (*true *and* false*) such that $\forall V \in \mathcal{V}$*

1. $\forall f \in ICEXT(I(\text{lx:Atomic})),\; \forall x \in IR,\; \forall r \in IP,\; \forall y \in IR$
   $(\langle f,x\rangle \in IEXT(I(\text{lx:subject})) \;\&\;$
   $\langle f,r\rangle \in IEXT(I(\text{lx:predicate})) \;\&\;$
   $\langle f,y\rangle \in IEXT(I(\text{lx:object}))) \rightarrow$
   $Bool(f,V) = \textit{true} \;\; \textit{iff} \;\; \langle x,y\rangle \in IEXT(r)$
2. $\forall f \in ICEXT(I(\text{lx:Atomic})),\; \forall x \in IR,\; \forall r \in IP,\; \forall y \in IR$
   $(\langle f,x\rangle \in IEXT(I(\text{lx:subjectv})) \;\&\;$
   $\langle f,r\rangle \in IEXT(I(\text{lx:predicate})) \;\&\;$
   $\langle f,y\rangle \in IEXT(I(\text{lx:object}))) \rightarrow$
   $Bool(f,V) = \textit{true} \;\; \textit{iff} \;\; \langle V(x),y\rangle \in IEXT(r)$
3. $\forall f \in ICEXT(I(\text{lx:Atomic})),\; \forall x \in IR,\; \forall r \in IP,\; \forall y \in IR$
   $(\langle f,x\rangle \in IEXT(I(\text{lx:subject})) \;\&\;$
   $\langle f,r\rangle \in IEXT(I(\text{lx:predicate})) \;\&\;$
   $\langle f,y\rangle \in IEXT(I(\text{lx:objectv}))) \rightarrow$
   $Bool(f,V) = \textit{true} \;\; \textit{iff} \;\; \langle x,V(y)\rangle \in IEXT(r)$
4. $\forall f \in ICEXT(I(\text{lx:Atomic})),\; \forall x \in IR,\; \forall r \in IP,\; \forall y \in IR$
   $(\langle f,x\rangle \in IEXT(I(\text{lx:subjectv})) \;\&\;$
   $\langle f,r\rangle \in IEXT(I(\text{lx:predicate})) \;\&\;$
   $\langle f,y\rangle \in IEXT(I(\text{lx:objectv}))) \rightarrow$
   $Bool(f,V) = \textit{true} \;\; \textit{iff} \;\; \langle V(x),V(y)\rangle \in IEXT(r)$

5. $\forall f \in ICEXT(I(\mathsf{lx{:}Negation})), \forall x \in IR$
$\langle f,x\rangle \in IEXT(I(\mathsf{lx{:}left})) \rightarrow$
$Bool(f,V) = true \ \textit{iff} \ Bool(x,V) = false$

6. $\forall f \in ICEXT(I(\mathsf{lx{:}Conjunction})), \forall x \in IR, \ \forall y \in IR$
$(\langle f,x\rangle \in IEXT(I(\mathsf{lx{:}left})) \ \&$
$\langle f,y\rangle \in IEXT(I(\mathsf{lx{:}right}))) \rightarrow$
$Bool(f,V) = true \ \textit{iff}$
$(Bool(x,V) = true \ \& \ Bool(y,V) = true)$

7. $\forall f \in ICEXT(I(\mathsf{lx{:}Exists})), \forall x \in IR, \forall y \in IR$
$(\langle f,x\rangle \in IEXT(I(\mathsf{lx{:}left})) \ \&$
$\langle f,y\rangle \in IEXT(I(\mathsf{lx{:}right}))) \rightarrow$
$Bool(f,V) = true \ \textit{iff} \ \exists w \in IR \ Bool(y,V') = true$
where $V'(x) = w$, $V'(z) = V(z)$ *otherwise.*

Note that this gives truth values to all elements of the domain of discourse, including non-formulae, but does not give any specification of what these truth values are for non-formulae. Note also that this does provide conditions on the truth values for over-specified formulae like a formula that is both a conjunction and a disjunction. Finally, note that this is not a recursive definition of truth values. Instead we just require that *Bool* satisfies the above conditions and rule out interpretations for which no such *Bool* exists; which amounts to almost the same thing but with fewer opportunities for paradoxes. For example, this method will work for infinite and circular formulae.

Now we need some way of getting at the truth of formulae. We will use lx:True to do so, providing the appropriate semantic condition so that true formulae belong to its class extension:

Truth Condition:
$f \in ICEXT(I(\mathsf{lx{:}True}))$ iff
$\exists V \in \mathcal{V} \ \ Bool(f,V) = true$

This looks as if $ICEXT(I(\mathsf{lx{:}True}))$ is defined somehow after *Bool*, but this is not really the case, as *Bool* depends on $ICEXT(I(\mathsf{lx{:}True}))$ for formulae that use lx:True.

A *logical* interpretation is then simply an RDF interpretation that satisfies the above semantic conditions, including having a *Bool* mapping, and *logical* entailment is defined in the obvious way.

3.4 The Result

I claim that this is about as good a version of RDF Logic as one can get. I would have liked to make the following claim:

Claim 1 *For every logical interpretation, L, there is a (standard) predicate calculus with equality interpretation, P, such that for every predicate calculus sentence, S, i.e., a predicate calculus formula with no free variables, for every domain element, d, in L that corresponds to S (in the obvious way) S is true in P iff* $d \in ICEXT(I(\mathsf{lx{:}True}))$.

This could be then used to show a strong correspondence between RDF Logic and predicate calculus with equality.

Unfortunately, this claim is *not* true. Ian Horrocks [Horrocks and Patel-Schneider, 2003] has pointed out that there are sentences whose truth is affected by the unusual nature of the RDF semantics. (Actually Ian's observation was done in the context of SCL (http://www.ihmc.us/users/phayes/CL/), but SCL has similar semantic features to those of RDF.) For example, in an RDF interpretation that has only one domain element, there is only *one* predicate, no matter what name.

This doesn't matter much in RDF and RDF Schema because neither can place conditions on the size of the domain. However, such conditions can be easily stated in predicate calculus with equality. For example, the formula
$(\forall x \, \forall y \, x = y) \rightarrow (\mathsf{ex{:}P}(\mathsf{ex{:}a},\mathsf{ex{:}a}) \leftrightarrow \mathsf{ex{:}Q}(\mathsf{ex{:}a},\mathsf{ex{:}a}))$
is not a theorem of predicate calculus with equality but every encoding of it is true in every logical interpretation.

So I can only claim that this version of RDF Logic is as good as any other. Further, I claim that this version of RDF Logic is as paradox-free as any other.

4 Self-Reference and Diagonalization

Does the above encoding actually work? Well, in a word, *no*. It falls prey to one of the (many) paradoxes related to the truth predicate, first noticed by Tarski [Tarski, 1933].

As in OWL, we have only provided comprehension principles for tree-like formulae. This means we don't require the existence of self-referential sentences like the Liar's sentence ("I am lying" or "This sentence is false"), which would here be encoded as

```
_:x rdf:type lx:Negation .
_:x lx:left _:x .
```

If this construction had to exist in every logical interpretation then there would be no logical interpretations. (Try to determine whether $I(\mathsf{_{:}x}) \in ICEXT(I(\mathsf{lx{:}True}))$.) We have thus lost something of the free-form flavour of RDF, as some self-referential sentences are useful and we can't entail them, but otherwise our model theory have have easily been shown to break down.

We have also been careful not to break down on self-referential sentences if they do occur. For example, if we had *defined Bool* in a recursive manner starting with atomic formulae, then we would have had problems with circular and infinite formulae.

We have thus been fairly careful not to get hung up on the obvious problems. However we still have enough power in our formalism for it to break down. This is actually not a surprising result—just about any encoding of first-order logic falls prey to the problem, but our minimalist RDF Logic is weaker in some ways than the standard first-order logics in which this result is usually shown.

What we are going to do next is to show how our encoding breaks down by building a self-referential sentence using a process of diagonalization. We are proceeding in a somewhat similar fashion as done by Goedel in his diagonalization lemma.

4.1 Building Up Formulae

Throughout this process we will need to get ahold of composite syntactic structures. The formulae for this turn out to be fairly long, so we will introduce some abbreviations. These are *only* abbreviations and thus don't add any power.

First, an abbreviation for a formula that checks for a single triple.

Definition 3 *Let* s, p, *and* o *be URI references, and let sv and ov be boolean values. Then* **require(**s,*sv*,p,o,*ov***)** *is a formula that looks like*

_:*r* rdf:type lx:Atomic .
_:*r* lx:predicate p .
_:*r* lx:subjectv s . *if sv = true*
_:*r* lx:subject s . *if sv = false*
_:*r* lx:objectv o . *if ov = true*
_:*r* lx:object o . *if ov = false*

Next, an abbreviation for a formula that checks for a collection of triples all with the same subject.

Definition 4 *Let* s *be a URI reference (used for the subject); let* t *be a URI reference (used for the type of the subject); let* p1, a1, p2, a2, ..., pn, an *be URI references (used for predicates and values); let* q1, v1, q2, v2, ..., qm, vm *be URI references (used for predicates and variables). Then* **construct**(s;t;p1,a1,...,pn,an;q1,v1,...,qm,vm) *is a formula that is a conjunction of*

require(s,*true*,rdf:type,t,*false*)
require(s,*true*,p1,a1,*false*) ...
require(s,*true*,pn,an,*false*)
require(s,*true*,q1,v1,*true*) ...
require(s,*true*,qm,vm,*true*)

This doesn't actually construct anything, of course. Neither does it require the existence of anything. However, for any domain element v such that

$v \in ICEXT(I(\mathsf{t}))$
$\langle v, I(\mathsf{a1})\rangle \in IEXT(I(\mathsf{p1}))$ ...
$\langle v, I(\mathsf{an})\rangle \in IEXT(I(\mathsf{pn}))$
$\langle v, v_1\rangle \in IEXT(I(\mathsf{q1}))$ ...
$\langle v, v_m\rangle \in IEXT(I(\mathsf{qm}))$

then **construct**(s;t;p1,a1,...,pn,an;q1,v1,...,qm,vm) is true under any variable mapping that maps s to v and vi to v_i, $1 \leq i \leq m$. Note especially that because all the non-circular formulae exist, from the formula comprehension principles, therefore any non-circular formula "built" in this way will exist.

4.2 Diagonalization

Diagonalization of a formula with one free variable is the application of that formula to itself, i.e., the diagonalization of A(x) is A('A(x)'), where the quotes are not string quotes, but instead represent some encoding process. Diagonalization is part of the process of creating a self-referential structure.

Definition 5 *Let* s *and* g *be URI references (used as variable names) Then* **diag**(s,g) *is a formula of the form*

_:*e1* rdf:type lx:Exists .
_:*e1* lx:left *s1* .
_:*e1* lx:right _:*e2* .
_:*e2* rdf:type lx:Exists .
_:*e2* lx:left *s2* .
_:*e2* lx:right _:*c1* .
_:*c1* rdf:type lx:Conjunction .
_:*c1* lx:left *construct*(s;lx:Exists;
lx:left,*x*;lx:right,*s1*) .
_:*c1* lx:right *c2* .
_:*c2* rdf:type lx:Conjunction .
_:*c2* lx:left *construct(s1;*lx:Conjunction*;*
*;*lx:left,*s2*,lx:right,*g*) .
_:*c2* lx:right *construct(s2;*lx:Atomic*;*
lx:subjectv,*x*,
lx:predicate,owl:sameAs*;*
lx:object,g*)* .

diag(s,g) is true under a variable mapping V precisely when there are some domain elements s_1 and s_2 such that the variable mapping of s, $V(\mathsf{s})$, is a domain element with the following characteristics

$V(\mathsf{s}) \in ICEXT(I(\mathsf{lx:Exists}))$
$\langle V(\mathsf{s}), I(x)\rangle \in IEXT(I(\mathsf{lx:left}))$
$\langle V(\mathsf{s}), s_1\rangle \in IEXT(I(\mathsf{lx:right}))$
$s_1 \in ICEXT(I(\mathsf{lx:Conjunction}))$
$\langle s_1, s2\rangle \in IEXT(I(\mathsf{lx:left}))$
$\langle s_1, V(g)\rangle \in IEXT(I(\mathsf{lx:right}))$
$s_2 \in ICEXT(I(\mathsf{lx:Atomic}))$
$\langle s_2, I(x)\rangle \in IEXT(I(\mathsf{lx:subjectv}))$
$\langle s_2, I(\mathsf{owl:sameAs})\rangle \in IEXT(I(\mathsf{lx:predicate}))$
$\langle s_2, V(\mathsf{g})\rangle \in IEXT(I(\mathsf{lx:object}))$

Informally, if g is a formula with only x free, which we could write $g(x)$, then if $diag(s,g)$ then s is the formula $g('g(x)')$.

4.3 A Paradoxical Formula

Now consider any logical interpretation. All non-circular finite formula exist in this interpretation, so, in particular there is a formula with the following characteristics. (Here I am being a bit sloppy, using triples instead of a metatheoretical formulation, but this does not introduce any problems.)

_:p rdf:type lx:Exists .
_:p lx:left y .
_:p lx:right _:p1 .
_:p1 rdf:type lx:Conjunction .
_:p1 lx:left _:p2 .
_:p1 lx:right _:d .
_:p2 rdf:type lx:Atomic .
_:p2 lx:subjectv y .
_:p2 lx:predicate owl:sameAs .
_:p2 lx:object _:d .

_:d rdf:type lx:Exists .
_:d lx:left x .
_:d lx:right _:d1 .
_:d1 rdf:type lx:Conjunction .
_:d1 lx:left *diag*(x,y) .[4]
_:d1 lx:right _:d2 .
_:d2 rdf:type lx:Negation .
_:d2 lx:left _:d3 .
_:d3 rdf:type lx:Atomic .
_:d3 lx:subjectv x .
_:d3 lx:predicate rdf:type .
_:d3 lx:object lx:True .

Informally this is the formula $\exists x\ (x = D)$ & D where D is $\exists y\ diag(y,x)$ & $\neg True(y)$.

Now what is the truth value of _:p, i.e., is _:p in $ICEXT(I(\mathsf{lx:True}))$? Well the truth value of _:p is precisely the truth value of _:d under a variable mapping that maps x to _:d, i.e., $Bool(\text{_:d}, x = \text{_:d})$.

[4] That is, _:d1's left conjunct is a formula that requires that x be a diagonalization of y.

But this is true exactly when there is some diagonalization of _:d whose truth value is $false$ and false exactly when every diagonalization of _:d has truth value $true$. As the diagonalization of _:d is a vanilla formula (finite, non-circular) thus at least one exists in any interpretation.

But the truth value of any diagonalization of _:d is precisely the truth value of _:d under a variable mapping that maps x to _:d, i.e., $Bool(_:d, x = _:d)$. So if $Bool(_:d, x = _:d) = true$ then it must be false, and if $Bool(_:d, x = _:d) = false$ then it must be true. Therefore no truth valuation for this formula can exist.

As a formula of this form exists in every interpretation, therefore there are no logical interpretations and our extension breaks down. Nothing in this argument depends on the details of the encoding and constructions so any extension falls prey to the same problem as long as it is possible to represent the diagonalization relationship.

5 Discussion

What happened?

Because the syntax of RDF is so impoverished we *had* to write formulae using facts—facts that have to be true everywhere. Then to state that a formula is true we *had* to use a truth predicate—no other solution is possible.[5] As the resultant logic has sufficient power to specify a form of diagonalization a paradox exists in the logic, resulting in its collapse.

Is is possible to wiggle out of this dilemma?

Not really.

If strings were used for formulae and no string manipulation primitives were available, then there would be no problem, but I have already argued that such string encodings are not permissible. In any case, adding string manipulation (i.e., concatenation) to the language would bring back the paradox. Similar issues rule out using other encodings like Goedel numbers.

If our truth "predicate" was weakened, perhaps so that it treated itself specially, as in KIF (http://logic.stanford.edu/kif/kif.html), then it might be possible to wiggle out. However, our truth predicate forms the basis of our logic, and thus weakening it weakens our logic, which I claim is also not permissible.

This also rules out, for example, potential solutions related to overall incomplete reasoners like the CycL reasoner (http://www.cyc.com/doc/handbook/oe/oe-handbook-toc-opencyc.html). It is not that the reasoning process of the solution is broken. Instead the fundamental model theory and expressive power of RDF make it unsuitable as a foundation for a same-syntax extension to first-order logic.

I see no way out of this dilemma, and so I claim:

Claim 2 *A paradox-free same-syntax extension of RDF to first-order logic is* ***not*** *possible.*

Note that this is a much stronger claim than

Claim 3 *A same-syntax extension of RDF to first-order logic is* ***not*** *desirable.*

In conclusion, a puff of philosophical wind is sufficient to knock over a Semantic Web tower built solely from RDF. A stronger tower needs better building materials.

Acknowledgments

The work on this paper was partly supported by DARPA under contract HR0011-05-C-0094.

References

[Baader *et al.*, 2003] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, implementation, and applications*. Cambridge University Press, 2003.

[Barr and Feigenbaum, 1981] Avron Barr and Edward A. Feigenbaum, editors. *The Handbook of Artificial Intelligence*. William Kauffman, Los Altos, California, 1981.

[Berners-Lee *et al.*, 2001] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific American*, 284(5):34–43, May 2001.

[Brinkley and Guha, 2004] Dan Brinkley and R. V. Guha. RDF vocabulary description language 1.0: RDF schema. W3C Recommendation, `http://www.w3.org/TR/rdf-schema`, 2004.

[Corcoran, 1983] John Corcoran, editor. *Logic, Semantics, Metamathematics, papers from 1923 to 1938*. Hackett Publishing Company, Indianapolis, 1983.

[Dave Beckett, 2004] RDF/XML syntax specification (revised). W3C Recommendation, `http://www.w3.org/TR/rdf-syntax-grammar/`, 2004.

[Dean *et al.*, 2004] Mike Dean, Guus Schreiber, Sean Bechhofer, Frank van Harmelen, Jim Hendler, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, and Lynn Andrea Stein. OWL web ontology language: Reference. W3C Recommendation, `http://www.w3.org/TR/owl-ref/`, 2004.

[Hayes, 2004] Patrick Hayes. RDF semantics. W3C Recommendation, `http://www.w3.org/TR/rdf-mt/`, 2004.

[Horrocks and Patel-Schneider, 2003] Ian Horrocks and Peter F. Patel-Schneider. Three theses of knowledge representation in the semantic web. In *The Twelfth International World Wide Web Conference*, pages 39–47. ACM Press, May 2003.

[Jan Grant and Dave Beckett, 2004] RDF test cases. W3C Recommendation, `http://www.w3.org/TR/rdf-testcases/`, 2004.

[Manola and Miller, 2004] Frank Manola and Eric Miller. RDF primer. W3C Recommendation, `http://www.w3.org/TR/rdf-primer/`, 2004.

[Patel-Schneider *et al.*, 2004] Peter F. Patel-Schneider, Patrick Hayes, and Ian Horrocks. OWL web ontology language: Semantics and abstract syntax. W3C Recommendation, `http://www.w3.org/TR/owl-semantics/`, 2004.

[Tarski, 1933] A. Tarski. The concept of truth in the languages of the deductive sciences [in Polish]. *Prace Towarzystwa Naukowego Warszawskiego, Wydzial III Nauk Matematyczno-Fizycznych*, 34, 1933. Reprinted in Zygmunt 1995, pages 13-172; expanded English translation in [Corcoran, 1983], pages 152-278.

[5] OWL has a form of truth predicate as well—various OWL properties, including `owl:equivalentClass` and `rdf:type`, play the role of specialized truth predicates. OWL, however, does not have the same power as first-order logic, and thus might not fall prey to diagonalization paradoxes.