On the Model Theory of RDF*

Klaus Schild
Freie Universität Berlin
schild@inf.fu-berlin.de

Abstract

Recently, the base language of the Semantic Web, RDF, has been given a model theoretic semantics, thus defining its meaning in a rigorous way. It is a rather involved and unconventional model theory, however. In this paper, we are going to enhance the model theory of RDF and investigate its expressive power. In particular, we show that RDF can be reduced to its first-order fragment and can be given a comprehensible, conventional model theory with its inferential power unchanged.

1 Introduction

Today's Web succeeded because it is *useful*, *simple*, and *scalable*. If the Web violated only one of these three criteria, it certainly would not have been such a success. The Semantic Web has yet to deliver a similar success story. It is thus reasonable to evaluate whether the Semantic Web does meet these necessary criteria for success.

According to Gartner, the semantic Web is currently at a peak of inflated expectations [10]. There are at least two reasons to interpret this positively rather than negatively. First, not many technologies make it to the top of a hype curve, least of all Gartner's. Second, the trough of disillusionment ahead should deliver a mainstream applications and, thus, should prove the Semantic Web's utility.

*To appear in S. Bab und T. Noll (eds.): *Models and Human Reasoning – Eine Festschrift fr Bernd Mahr*, Wissenschaft & Technik Verlag, Berlin, 2005.

The scalability of basic Semantic Web standards has recently been investigated in [7]. The results are mixed. Deciding the equivalence of two RDF graphs is isomorphism complete,[1] while simple entailment between two RDF graphs is NP-complete (cf. Theorem 4 of [7]). Basic reasoning in RDF is therefore intractable, so that one cannot expect the Semantic Web to scale smoothly. The picture is, however, quite different if we consider RDF's data complexity. Data complexity is based on the assumption that even if the knowledge base was continuously getting larger, the size of the queries would certainly not [9]. The requirements on scalability are thus considerably softened. When considering this weaker notion of data complexity, RDF turns out to be tractable (cf. Theorem 14 of [7]).

The third criterion is simplicity. RDF certainly enjoys a simple, almost trivial syntax – at least this is true for the triple notation [6]. It just consists of triples of the form `s p o`, called *RDF statements*. From the point of view of linguistics, `s` is a subject, `p` a predicate, and `o` an object, while in terms of predicate logic, `s p o` is to be read as the statement $p(s, o)$. A more practical reading is that there is a link of type `p` leading from resource `s` to resource `o`, so that a set of such triples induce a directed graph, called an *RDF graph*.

The predicate `p` of an RDF statement is always a URI reference, so that any software program which is aware of the meaning of this very URI is able to interpret the statement `s p o` properly. This is the way meaning is given to RDF graphs. The following simple RDF graph exemplifies this.

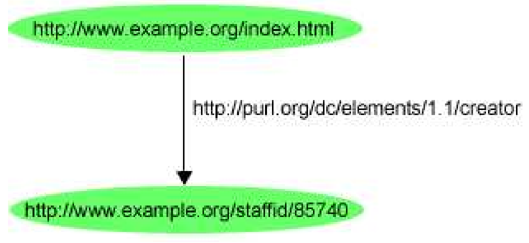
http://www.example.org/index.html
http://purl.org/dc/elements/1.1/creator
http://www.example.org/staffid/85740

Figure 1: A simple RDF statement (Source: [5])

[1]Thus this problem is as hard to determine as it is to decide whether two arbitrary graphs are isomorphic or not. There is no *known* deterministic polynomial time algorithm for deciding graph isomorphism, although the problem has not been shown to be NP-complete either.

This RDF statement signifies that a resource uniquely identified by `http://www.example.org/staffid/85740` is the creator of the resource `http://www.example.org/index.html`. Of course, one can interpret this RDF statement correctly only if one knows that

```
http://purl.org/dc/elements/1.1./creator
```

actually stands for the creator relationship. For a human being, this string is probably somewhat enigmatic; for a software program, however, this URI can uniquely identify the relationship commonly associated with the word creator.

This already describes the core of RDF; only literals and blank nodes are yet to be introduced. Literals have been included into RDF because sometimes identifying resources by a full URI is not convenient. This is certainly true for simple strings like “John Smith” or “27.” In RDF, it is thus admissible to use simple literals in lieu of URI references. Here is a sample RDF graph with two literals:

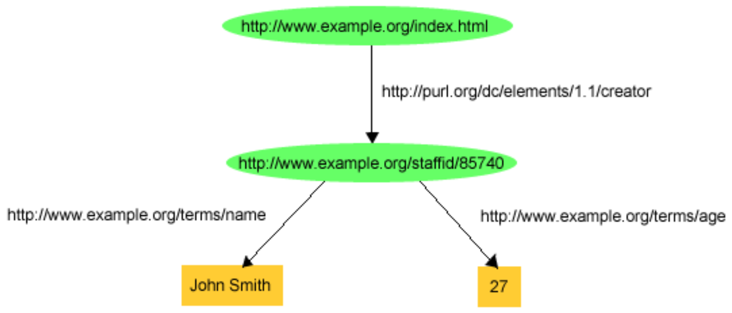
http://www.example.org/index.html
http://purl.org/dc/elements/1.1/creator
http://www.example.org/staffid/85740
http://www.example.org/terms/name
http://www.example.org/terms/age
John Smith
27

Figure 2: Three RDF Statements with two literals (Source: [5])

In RDF, literals may be used only as objects; it is not permitted to use them as subjects or predicates. Literals are thus a convenient way to state simple properties of resources. In some cases, however, identifying any resource *at all* may even be inconvenient. Perhaps we would like to make statements about a creator of a specific resource, although we do not know the URI of this particular person. In such a case, we may just want to state that there is a creator whose name is “John Smith” without referring to the URI of “John Smith.” For this purpose, we can use so-called *blank* *nodes*, which are essentially quantified variables. Blank nodes may be used not only as objects, but also as subjects, while it is not permitted to use them as predicates.

In a nutshell, this is the syntax of RDF. For a detailed description the reader is referred to [5]. With its graph-based info set, RDF even appears to be simpler than XML, at the same time being more flexible. Another argument in favor of RDF's simplicity is the fact that it tightly integrates content and vocabulary language: vocabulary information such as subclass relationship is encoded as RDF statements with the predicate being a special URI reference standing for the subclass relationship [5].

On the other hand, RDF suffers from a quite complex formal semantics [8]. This complexity is mainly due to an unconventional distinction between properties and their extensions; a distinction which in effect eliminates RDF's capability to make statements about predicates, which syntactically is perfectly admissible. Certainly, this approach has deliberately been adopted in order to avoid the implications of higher-order statements such as `s s s`. An incomprehensible semantics, however, can be a main stumbling block for the proliferation of a Semantic Web. This is because a formal semantics is a pivotal part of the specification of the Semantic Web. Moreover, this unconventional model theory prevents us from re-using a substantial amount of prior research done in mathematical logic, database theory, and Knowledge Representation, all typically based on a conventional model theory. We will show now that RDF can be given a comprehensible, conventional model theory, while leaving its inferential power unchanged. Before we delve into details, we have to define the semantics of RDF as recently specified by the W3C [8].

2 A Model Theory for RDF

This section introduces the semantics for RDF as given in [8], with only some minor changes. The somewhat lengthy introduction is partly due to the complexity of the matter itself; however, the introduction also contains some discussions which will be instrumental in understanding the remainder of the paper.

To begin with, let U be the set of all URI references, B the set of blank nodes, and L the set of literals. Let UBL be the union of U, B, and L.

A model theory is usually based on interpretations. An interpretation is a tuple ⟨D,I⟩, where D is the universe of discourse and I is a special function mapping predicates to n-ary relations over D. In the case of RDF, the universe of discourse is a set of resources denoted by IR, and an interpretation consists of four rather than just two components. The two additional components are needed because binary predicates are mapped by a two-staged process to binary relations over IR, first mapping them to so-called properties, which are then mapped to binary relations over IR.

Definition 1 An *interpretation* $\mathcal{I}$ is a tuple ⟨IR,IP,I,IEXT⟩ where:

1. IR is a non-empty set, the elements of which are called *resources*.
2. IP is a set, the elements of which are called of *properties*.
3. I is a mapping from UBL into IR union IP such that
   (a) literals and blank nodes are mapped into IR and
   (b) every literal `l` is mapped to itself, that is, $\mathrm{I}(\mathtt{l}) = \mathtt{l}$.
4. IEXT is a mapping from IP into the powerset of IR $\times$ IR.

This means that URI references are mapped either to a resource or a property, while literals and blank nodes are always mapped to a resource. This is due to the fact that neither a blank node nor a literal may occur as a predicate, so that they are always mapped to resources, while a URI reference can also occur as a predicate, which is always mapped to a property, as we shall see below. IEXT then maps properties to binary relations over IR. Here, the above definition deviates from conventional model theory in that predicates are mapped by a two-staged process to binary relations over IR (see Fig. 3).

This departure from conventional model theory has deliberately been chosen in order to distinguish between a property $\mathrm{I}(\mathtt{p})$ and its extension $\mathrm{I}(\mathrm{IEXT}(\mathtt{p}))$. The binary relation $\mathrm{IEXT}(\mathrm{I}(\mathtt{p}))$ is the actual meaning of `p`, while $\mathrm{I}(\mathtt{p})$ can be thought of as a proxy for this binary relation. Nothing prevents two properties $\mathrm{I}(\mathtt{p})$ and $\mathrm{I}(\mathtt{p}')$ with $\mathrm{I}(\mathtt{p}) \neq \mathrm{I}(\mathtt{p}')$ from having the same extension $\mathrm{IEXT}(\mathrm{I}(\mathtt{p})) = \mathrm{IEXT}(\mathrm{I}(\mathtt{p}'))$. Philosophically, this means that the identity of sets (that is, property extensions) is not uniquely determined by the elements (that is, tuples) which they contain, but by a

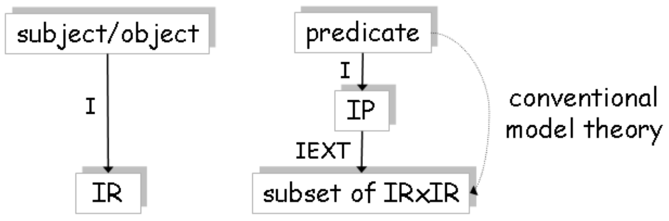

Figure 3: Two-staged interpretations

special identifier, called property. Hence, there is only a loose relationship between a property and its extension.

The question arises why there has been made this unconventional distinction between a property I(`p`) and its extension IEXT(I(`p`)). The answer can be found in RDF's unconstrained syntax: Without this distinction it would be possible to make statements about a predicate, that is, a statement about a binary relation. Consider the RDF statement `s p o`. Nothing prevents us from formulating a second RDF statement of the form `p p′ o′`, thus making a statement about the predicate `p`. With no clear distinction between a property and its extension, the RDF triple `p p′ o′` would indeed make a statement about a binary relation. From the perspective of the Semantic Web such a kind of higher-order statement should be avoided. Assume, for instance, that `p′` and `o′` is defined as a special vocabulary such that `p p′ o′` forces `p` to be symmetric. At the end of this chapter we shall see that RDF actually provides us with the possibility to define `p′` and `o′` in this way. But then, a local RDF statement `p p′ o′` would have a global consequence, *i.e.*, `p` being symmetric – a situation which would contradict the openness of the Semantic Web. This is why RDF draws a clear distinction between properties and their extensions. This very distinction manifests itself in the interpretation of RDF statements:

Definition 2 An interpretation $\mathcal{I} = \langle \text{IR},\text{IP},\text{I},\text{IEXT}\rangle$ *satisfies* an RDF statement `s p o` iff I(`p`) $\in$ IP and $\langle$I(`s`),I(`o`)$\rangle \in$ IEXT(I(`p`)).

This explains how a single RDF statement is to be interpreted. The above definition, however, does not take into account that blank nodes are actually existentially quantified variables. The scope of the existential quantification is actually the whole RDF graph (that is, a set of RDF statements) rather than a single RDF statement: If a blank node occurs in two distinct

RDF statements of an RDF graph, it falls within the scope of the same existential quantifier. The following definition realizes this kind of global existential quantification.

Definition 3 A *variable assignment* is a mapping from B into IR. If $\mathcal{I} = \langle$IR,IP,I,IEXT$\rangle$ is an interpretation and A: B $\rightarrow$ IR is a variable assignment, then $\mathcal{I}_{\mathrm{A}}$ denotes the interpretation which is identical with $\mathcal{I}$ save for the fact that for every blank node `b`, I(`b`) = A(`b`). An interpretation $\mathcal{I}$ *satisfies* a set S of RDF statements iff there is a variable assignment A: B $\rightarrow$ IR such that $\mathcal{I}_{\mathrm{A}}$ satisfies every RDF statement of S.

Now, suppose that $\mathcal{I}$ satisfies S, where a URI reference `u` occurs in S as a predicate. According to Definition 2, I(`u`) then yields a property of IP. If, at the same time, `u` occurs in S as a subject or object, say, the RDF statement `u p o` is in S, then I(`u`) must also be a resource of IR. This is because $\langle$I(`u`),I(`o`)$\rangle$ must be in IEXT(I(`p`)), which is a binary relation over IR. As I(`u`) is a member of IP, too, IR and IP obviously are not disjoint, a situation which is perfectly admissible according to Definition 1. As a matter of fact, whenever a URI reference occurs in S both as a predicate and as a subject or object, an interpretation I cannot satisfy S without the intersection of IR and IP being non-empty.

Example 1 Consider the RDF statement `s s s`. A simple interpretation which satisfies `s s s` is as follows:

- IR = 1
- IP = 1
- I(`s`) = 1
- IEXT(1) = $\{\langle 1, 1\rangle\}$

According to Definition 2, $\mathcal{I}$ satisfies `s s s` iff I(`s`) is in IP and $\langle$I(`s`),I(`s`)$\rangle$ is in IEXT(I(`s`)). When substituting 1 for I(`s`) and $\{\langle 1, 1\rangle\}$ for IEXT(1), we immediately see that $\mathcal{I}$ indeed satisfies `s s s`.

We are now in a position to define the basic notion of entailment.

Definition 4 Let S and S$'$ be two sets of RDF statements. S is said to entail S$'$ iff every interpretation which satisfies S also satisfies S$'$.

This definition looks exactly as the definition of entailment in conventional first-order logic; however, the underlying notion of an interpretation does divert from conventional logic. This deviation might appear minor. As a matter of fact, it prevents us from re-using the substantial amount of work on mathematical logic, Knowledge Representation, and database theory, which are all typically based on a conventional model theory. It is exactly this deviation from conventional logic which also considerably complicates the overall Semantic Web stack. As proclaimed by Berners-Lee, any upper layer of this stack (like a powerful ontology language) is to be expressed *within* RDF, the base language of the Semantic Web [1]. In a nutshell, this means that any extension of RDF is to be encoded within RDF itself by defining a special vocabulary obeying a special built-in meaning. This is exactly the way RDF Schema is defined [8]. This approach, however, forces each extension not only to use RDF's syntax, but also its semantics: The meaning of an extension's special vocabulary must be given in terms of the two-staged interpretation of Definition 1, on which only additional constraints can be imposed. This may cause severe problems as the case OWL has demonstrated clearly [3].

RDF has its own special vocabulary which is treated in exactly the same way as the vocabulary of any extension of RDF would be handled. We consider here only `ref:type`, `rdf:Property`, `rdf:subject`, `rdf:predicate`, and `rdf:object`. For a complete treatment the reader is referred to Chapter 3.1 of [8].

We begin with `rdf:type`, which is a shorthand for

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

This URI can be thought of as denoting the instance or membership relationship between a resource and a class. Although an interpretation as introduced in Definition 1 does not involve the notion of a class, a resource of IR can be anything, including a class or a set. This is why an RDF statement of the following type does make sense:

(1) `http://.../object123  rdf:type  http://.../MotorVehicle`.

A natural interpretation of `rdf:type` would be as follows:

$$\mathrm{IEXT}(\mathrm{I}(\texttt{rdf:type})) = \{\langle \mathrm{e,S}\rangle \in \mathrm{IR} \times \mathrm{IR}\text{: S is a set and } \mathrm{e} \in \mathrm{S}\}.$$

However, this is actually *not* the way `rdf:type` is interpreted. This is because of RDF statements of the type

(2) `rdf:type  rdf:type  rdf:type`.

According to Definition 2, an interpretation $\mathcal{I}$ satisfies the RDF statement (2) iff ⟨I(`rdf:type`),I(`rdf:type`)⟩ is in IEXT(I(`rdf:type`)). With the above interpretation of IEXT(I(`rdf:type`)) put into effect, this would imply that I(`rdf:type`) is a set which contains itself. In order to avoid the implications of such expressions, a more indirect interpretation of `rdf:type` is chosen instead. Before we show how this is done, we first introduce `rdf:Property`, which is a shorthand for

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

This URI can be thought of as denoting the set of all binary relations over IR. One can phrase, e.g.,

(3) `http://.../has-color  rdf:type  rdf:Property`.

For the sake of brevity, we do not use a full URI here. Anyway, this triple states that `http://.../has-color` is an instance of `rdf:Property` and thus denotes a binary relation. In [8], `rdf:type` and `rdf:Property` are simultaneously given meaning by imposing the following constraint:

(C1) x is in IP iff ⟨x,I(`rdf:Property`)⟩ $\in$ IEXT(I(`rdf:type`)).

This defines each property x of IP as a special kind of a resource, namely one whose value of the I(`rdf:type`) property is I(`rdf:Property`). The if-direction of (C1) means that from the RDF statement (3) we can infer that I(`http://.../has-color`) is a property of IP. The only-if-direction of (C1) means that from an RDF statement of the type

(4) `http://.../object123  http://.../has-color  yellow`

we can infer (3).

An immediate consequence of (C1) is that IP is a subset of IR. According to Definition 1, IEXT maps each property of IP to a binary relation over IR, so that IEXT(I(`rdf:type`)) is a binary relation over IR as well. Therefore, we can deduce from (C1) that x is in IR whenever x is in IP, so that IP is in fact a subset of IR. Note also that, when writing IEXT(I(`rdf:type`)), we implicitly state that I(`rdf:type`) is in IP because IEXT maps properties of IP to binary relations over IR. Hence, from (C1) we can infer that ⟨I(`rdf:type`),I(`rdf:Property`)⟩ is a member of IEXT(I(`rdf:type`)) and that $\mathcal{I}$ also satisfies the following RDF statement:

(A1) `rdf:type  rdf:type  rdf:Property`.

Observation 1 *If an interpretation $\mathcal{I}$ meets (C1), then $\mathcal{I}$ satisfies (A1) as well.*

This conflicts with the fact that [8] introduces (A1) as a basic axiom, at the same time imposing (C1). It is also worthwhile to mention that the opposite direction of Observation 1 does not hold: An interpretation which satisfies (A1) does not necessarily meet (C1). Roughly speaking, (C1) means that each resource used in some RDF statement as a predicate is an instance of `rdf:Property`, while (A1) forces only `rdf:type` to be an instance of `rdf:Property`.

Finally, we introduce the remainder of the RDF vocabulary, `rdf:subject`, `rdf:predicate`, and `rdf:object`, which are to be expanded similarly as `rdf:type` and `rdf:Property`. This special RDF vocabulary is mainly used for reification. Reification means that we can encode an RDF statement such as (1) also as follows:

```
X  rdf : subject    http : //.../object123
X  rdf : predicate  rdf : type
X  rdf : object     http : //.../MotorVehicle
```

Here, `X` is a blank node. Together these three RDF statements are to be read as follows: There is a resource (that is, a statement) whose subject, predicate, and object is `http://.../object123`, `rdf:type`, and `http://.../MotorVehicle`, respectively. We require that any interpretation satisfies the following three axioms:

(A2) `rdf:subject  rdf:type  rdf:Property`

(A3) `rdf:predicate  rdf:type  rdf:Property`

(A4) `rdf:object  rdf:type  rdf:Property`

Note that (A2)-(A4) are actually redundant whenever `rdf:subject`, `rdf:predicate`, and `rdf:object` are used in some RDF statements as predicates. To be more accurate, we state the following proposition:

Observation 2 *Let S be a set of RDF statements in which* `rdf:subject`*,* `rdf:predicate`*, and* `rdf:object` *each occur as a predicate. Then every interpretation which satisfies S and for which (C1) holds, satisfies (A2)-(A4) as well.*

The proof is analogous to the argument given above supporting Observation 1. The presupposition that `rdf:subject`, `rdf:predicate` as well as `rdf:object` occur as a predicate is actually needed here. If one of these, say, `rdf:subject`, were not used as a predicate, then I(`rdf:subject`) would not necessarily be a property of IP, preventing (C1) from being applied to I(`rdf:subject`). In other words, the only reason why, in addition to constraint (C1), axioms (A2)-(A4) are needed is the possibility to infer that `rdf:subject`, `rdf:predicate`, and `rdf:object` are properties, even if they are not used as predicates.

It should be noted that this semantics of `rdf:subject`, `rdf:predicate`, and `rdf:object` is actually very weak. If r(`s p o`) denotes the reification of an RDF statement `s p o` analogous to the reification given above, then neither r(`s p o`) rdf-entails `s p o` nor, vice versa, `s p o` rdf-entails r(`s p o`). In fact, r(`s p o`) does not have any logical relationship with `s p o` (see also Chapter 3.3.1 of [8]).

We have already used the notion of rdf-entailment, which remains to be formally defined . The definition comes in two parts:

Definition 5 An interpretation which meets (C1) and satisfies (A2)-(A4) is called an *rdf-interpretation*.

Definition 6 Let S and S′ be two sets of RDF statements. S is said to *rdf-entail* S′ iff every rdf-interpretation which satisfies S, satisfies S′ as well.

This is the semantics of RDF. Some details have been omitted, in particular typed literals and some more special RDF vocabulary; however, this does not affect any of the results given in this paper.

3 Reducing RDF to its First-Order Fragment

We have seen that the distinction between properties I(`p`) and their extension I(IEXT(`p`)) has deliberately been drawn in order to avoid the implications of higher-order statements. This means that the semantics of RDF has been defined in such a way that higher-order capabilities are effectively eliminated. In this section, we are going to show that this approach indeed achieves its goal: despite of its appearance, RDF is essentially first order in nature. We begin with defining what we mean by RDF's higher-order capabilities.

Definition 7 Let S be a set of RDF statements. S is said to be *higher order* if there is a URI reference which appears in S both as a predicate and as subject or object; otherwise S is *first order.*

Example 2 The following sets of RDF statements are higher order:

1. $\{$ `s s o` $\}$ and
2. $\{$ `s p o`, `p` $\mathtt{p}'$ `o` $\}$.

Let P be a distinguished subset of U. This set of URI references is to be used only as predicates; URI references from P must not occur in an RDF statement as a subject or object. We can think of P as special URI references exclusively reserved for predicates. One possibility might be that the URI references of P are distinguished by a unique suffix such as #predicate.

Definition 8 Let f be a function mapping U $\setminus$ P into P such that for every `u`, `v` in U $\setminus$ P with $\mathtt{u} \neq \mathtt{v}$, $\mathrm{f}(\mathtt{u}) \neq \mathrm{f}(\mathtt{v})$. For every set S of RDF statements, we define fo(S) to be the set of RDF statements obtained from S by replacing every predicate `p` in S by f(`p`).

Example 3

1. fo($\{$ `s s o` $\}$) = $\{$ `s` f(`s`) `o` $\}$
2. fo($\{$ `s p o`, `p` $\mathtt{p}'$ `o` $\}$) = $\{$ `s` f(`p`) `o`, `p` f($\mathtt{p}'$) `o` $\}$

If a set S of RDF statements does not involve any URI reference from P, then fo(S) is first order. This is because f replaces each predicate with special URI references from P. As S is free of any URI reference from P, URI references from P occur in fo(S) only as predicates. Thus no URI reference can occur in fo(S) as a predicate and at the same time as a subject or object, which means that fo(S) is actually first order.

Now, for the main theorem of this paper.

Reduction Theorem 1 *Let S and S′ be two sets of RDF statements which do not involve any URI reference from P. Then S rdf-entails S′ iff fo(S) rdf-entails fo(S′).*

Recall that fo(S) and fo(S′) are first order. The theorem thus states that RDF can be reduced to its first-order fragment while leaving its inferential power unchanged.

Example 4 Consider two sets of RDF statements S = { `s s o` } and S′ = { `s s o′` }. If neither S nor S′ involves any URI reference from P, then both fo(S) and fo(S′) are first order. By Reduction Theorem 1, S rdf-entails S′ iff { `s` f(`s`) `o` } rdf-entails { `s` f(`s`) `o′` }.

Proof of Reduction Theorem 1. We first show that S rdf-entails S′ if fo(S) rdf-entails fo(S′). Assume the contrary, that is, fo(S) rdf-entails fo(S′) but S does not rdf-entail S′. We will show that this assumption leads to a contradiction. As S does not rdf-entail S′, there must be an rdf-interpretation $\mathcal{I}$ = ⟨IR,IP,I,IEXT⟩ which satisfies S but not S′). By assumption, neither S nor S′ involve any URI reference from P. Without loss of generality, we can hence assume that I is not defined on P. Let us define an extension $\mathcal{I}'$ = ⟨IR,IP,I′,IEXT′⟩ of $\mathcal{I}$ as follows:

- For every x of UBL \ P, I′(x) = I(x).
- For every `p` of P, I′(f(`p`)) = I(`p`).
- For every `p` of P, IEXT′(I′(f(`p`))) = IEXT(I(`p`)).

Note that f(`p`) is a member of P and thus neither appears in S nor in S′. This means that the redefinition of I′(f(`p`)) and IEXT′(I′(f(`p`))) does not affect the interpretation of S and S′. Hence, it can be shown that $\mathcal{I}'$ satisfies fo(S) but not fo(S′). But then, fo(S) does not rdf-entail fo(S′). T This contradicts the assumption above.

We now show that S entails S′ only if fo(S) entails fo(S′). Again assume the contrary, that is, S entails S′ but fo(S) does not entail fo(S′). We will show that this assumption leads to a contradiction, too. As fo(S) does not entail fo(S′), there must be an interpretation $\mathcal{I}$ which satisfies fo(S) but not fo(S′)). Let $\mathcal{I}'$ be the interpretation $\mathcal{I}$ except that for every predicate `p` in S and S′ we have:

$$\mathrm{IEXT}'(\mathrm{I}'(\mathtt{p})) = \mathrm{IEXT}(\mathrm{I}(\mathrm{f}(\mathtt{p}))).$$

Note that only IEXT$'$(I$'$(`p`)) is redefined, while I$'$(`p`) is left unchanged, i.e., I$'$(`p`) = I(`p`). Observe also that `p` neither appears in fo(S) nor in fo(S$'$) as a predicate. This is because in fo(S) and fo(S$'$) every predicate `p` is replaced by f(`p`). Therefore, the redefinition of IEXT$'$(I$'$(`p`)) does not affect the interpretation of fo(S) and fo(S$'$). It can be shown that $\mathcal{I}'$ satisfies fo(S) but not fo(S$'$). But then, fo(S) does not rdf-entail fo(S$'$), contradicting the assumption above. □

4 A Conventional Model Theory for RDF

We have learned so far that RDF can be reduced to its first-order fragment without sacrificing any bit of inferential power. There is, however, another reading of Reduction Theorem 1. It can also be interpreted in such a way that the semantic distinction between properties and their extensions can be lifted to the level of syntax. Consider again

$$\mathrm{fo}(\{\ \mathtt{s\ p\ o},\ \mathtt{p\ p'\ o}\ \}) = \{\ \mathtt{s}\ \mathrm{f}(\mathtt{p})\ \mathtt{o},\ \mathtt{p}\ \mathrm{f}(\mathtt{p'})\ \mathtt{o}\ \}.$$

Here, `p` can be thought of as denoting the property I(`p`), whereas f(`p`) and f(`p`$'$) represent the property extensions I(IEXT(`p`)) and I(IEXT(`p`$'$)), respectively. The mapping f thus represents the distinction between properties and their extension on the level of syntax.

This raises the following question. We already tell a property and its extension apart on a syntactic level. So, do we still need a two-staged interpretation, which was solely introduced for the purpose of this very distinction? The answer is clearly no. To see why, let us recall how S = { `s p o`, `p p`$'$ `o` } is interpreted: According to Definition 2 and 3, an interpretation $\mathcal{I}$ = ⟨IR,IP,I,IEXT⟩ satisfies S iff

1. I(`p`) and I(`p`$'$) are both in IP,
2. ⟨I(`s`),I(`o`)⟩ is in IEXT(I(`p`)), and
3. ⟨I(`p`),I(`o`)⟩ is in IEXT(I(`p`$'$)).

Among other things, this means that, whenever used as a predicate, `p` denotes the property extension IEXT(I(`p`)). This is the way how `p` is interpreted in the first RDF statement `s p o`. Otherwise, if `p` is used as a subject or object, it denotes the property I(`p`). This happens to be the case in the second RDF statement `p` `p′` `o`. Now, $\mathcal{I}$ satisfies fo(S) = { `s` f(`p`) `o`, `p` f(`p′`) `o` } iff

1. I(f(`p`)) and I(f(`p′`)) are both in IP,
2. ⟨I(`s`),I(`o`)⟩ is in IEXT(I(f(`p`))), and
3. ⟨I(`p`),I(`o`)⟩ is in IEXT(I(f(`p′`))).

This means that, even if I maps URI resources directly to binary relations over IR, we nevertheless do not get involved in the complications of a set containing itself. As a matter of fact, if I maps URI resources directly to binary relations over IR, we have:

1. I(f(`p`)) and I(f(`p′`)) are both in IP,
2. ⟨I(`s`),I(`o`)⟩ is in I(f(`p`)), and
3. ⟨I(`p`),I(`o`)⟩ is in I(f(`p′`)).

But then, the first of these conditions become superfluous, too, in that in this case I(f(`p`)) and I(f(`p′`)) yield a binary relation over IR rather than properties of IP. As a result, we then have:

1. ⟨(`s`),I(`o`)⟩ is in I(f(`p`)) and
2. ⟨I(`p`),I(`o`)⟩ is in I(f(`p′`)).

This is exactly the way a conventional model theory would look like. So, let us replace the two-staged interpretation by a notion of an interpretation which is perfectly in line with conventional model theory.

Definition 9 An *interpretation* $\mathcal{I}$ is a tuple ⟨IR,I⟩ where:

1. IR is a non-empty set, the elements of which are called resources.
2. I is a mapping from UBL into IR such that
   (a) UBL \ P is mapped into IR,
   (b) every literal `l` is mapped to itself, that is, I(`l`) = `l`, and
   (c) P is mapped into the powerset of IR × IR.

Definition 10 An interpretation $\mathcal{I} = \langle \mathrm{IR}, \mathrm{I} \rangle$ *satisfies* an RDF statement `s p o` iff $\langle \mathrm{I}(\mathtt{s}), \mathrm{I}(\mathtt{o}) \rangle \in \mathrm{I}(\mathtt{p})$.

All other definitions remain unchanged. The constraint (C1), however, has to be modified slightly in that we do not have the set IP at our disposal any more. A proper reformulation of (C1) is as follows:

(C1′) If `p` is used as a predicate, then $\langle \mathrm{I}(\mathrm{p}), \mathrm{I}(\mathtt{rdf:Property}) \rangle \in \mathrm{I}(\mathtt{rdf:type})$.

Equivalently, we can impose the following axiom for every `p` used as a predicate:[2]

(A_{C1}) `p rdf:type rdf:Property`.

The version of rdf-entailment which is based on this variant of an interpretation is called *rdf-entailment with a conventional model theory.* If S and S′ are two sets of RDF statements, we write S $\models_{\mathrm{rdf}}$ S′ iff S rdf-entails S′ in the sense of this variant.

Reduction Theorem 2 *Let S and S′ be two sets of RDF statements which do not involve any URI reference from P. Then S rdf-entails S′ iff fo(S)* $\models_{\mathrm{rdf}}$ *fo(S′).*

This means that RDF can be given a conventional model while leaving its inferential power unchanged. The proof is similar to that of Reduction Theorem 1.

5 Conclusions

Higher-order statements (such as statements about predicates) are perfectly admissible in RDF. These higher-order statements, however, do not have any logical consequences. As a matter of fact, we have shown that RDF can be reduced to its first-order fragment, which means that this fragment of RDF has the very same inferential power as full RDF. We have also seen that the first-order fragment of RDF can be given an equivalent semantics which is perfectly in line with the model theories known from standard first-order logic, relational databases, and Knowledge Representation. Thus the

[2] A reasonable (though stronger) alternative is imposing axiom (A_{C1}) for every `p` in P, no matter whether `p` actually appears in some RDF statement or not.

semantics of RDF can be considerably simplified, with its inferential power left unchanged.

A practical consequence of this observation might be the proliferation of a suitable first-order sublanguage of RDF. Let us call this sublanguage SimpleRDF. SimpleRDF is identical with RDF save for the fact that the set U of URI references is divided into two disjoint sets: $\mathrm{U} \setminus \mathrm{P}$ and P. The URI references of P are to be used as predicates only, whereas those of $\mathrm{U} \setminus \mathrm{P}$ may be used as subjects and objects only. These two disjoint subsets should have exactly the same cardinality, so that it is possible to define pairs of URI references, $\mathtt{u} \in \mathrm{U} \setminus \mathrm{P}$ and $\mathtt{u}' \in \mathrm{P}$, which uniquely correspond to each other. That is to say, if $\mathtt{u}'$ is used as a predicate, we immediately know that this URI reference corresponds to the URI reference `u` used as a subject or object. This might be realized by defining P as the subset of URI references which have a special suffix such as `#predicate` appended to them.

The resulting variant would obviously be first order in nature and thus can be given a simple, conventional model theory, while having the same inferential power as full RDF with its original two-staged semantics. This is an immediate consequence of Reduction Theorem 2.

It is this conventional model theory which puts us in a position to explore the expressive power of SimpleRDF. In fact, the expressive power of two logics can be compared only if they do have the same model theory (or at least the same semantics). In its original version, the expressive power of RDF thus simply cannot be compared with those of standard logics. For SimpleRDF, however, it can be done. It is not hard to verify that SimpleRDF is a strict sublanguage of the Schönfinkel-Bernays class with prefixes of the form $\exists ... \exists \forall ... \forall$ [4]. This holds even with axioms of the type (A_{C1}) and those of RDF Schema taken into account. One immediate consequence is the fact that a set S of RDF statements, possibly including RDF Schema expressions, too, is satisfiable iff S is satisfied by an interpretation $\langle \mathrm{IR}, \mathrm{I} \rangle$ such that IR contains at most n elements. Here, n is the number of URI references, blank nodes, and literals occurring in S [4]. Hence, we can safely ignore infinite models and consider only models of linear size. In other words, without loss of generality, we may assume a proper domain-closure axiom for RDF and RDF Schema. This is an example of re-using prior work on mathematical logic. Many other results are yet to be discovered, and not only in the realm of mathematical logic, but also in such areas as relational database theory and Knowledge Representation. We shall present some of these results elsewhere. Of course, this is approach is admissible only if a conventional model theory is employed.

References

[1] Berners-Lee, *Semantic Web – XML 2000, http://www.w3.org/2000/Talks/1206-xml2k-tbl*, 2000.

[2] Berners-Lee, Hendler, and Lassila, The Semantic Web: A new form of Web content that is meaningful to computers will unleash a revolution of new possibilities, *Scientific American*, 2001.

[3] Horrocks and Patel-Schneider, Three theses of representation in the Semantic Web, In *Proc. of the Twelfth International World Wide Web Conference*, 2003.

[4] Lewis, Complexity Results for Classes of Quantificational Formulas, *Journal of Computer and System Sciences*, 21: 317–353, 1980.

[5] Manola and Miller (Eds.), *RDF Primer*, http://www.w3.org/TR/rdf-primer, 2004.

[6] Klyne and Carrol (Eds.), Resource Description Framework (RDF): *Concepts and Abstract syntax*, http://www.w3.org/TR/rdf-concepts, 2004.

[7] Gutirrez, Hurtado, and Mendelzon, Foundations of Semantic Web Databases. In *Proc. of the Twenty-third ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, 95–106, 2004.

[8] Hayes (Ed.), *RDF Semantics*, http://www.w3.org/TR/rdf-mt, 2004.

[9] Vardi, The complexity of relational query languages (Extended Abstract), In *Proc. of the fourteenth annual ACM symposium on Theory of computing*, 137–146, 1982

[10] COMPUTERWOCHE, vol. 33, 2004.