A Logical Interpretation of RDF

Wolfram Conen		Reinhold Klapsing
XONAR IT	*and*	Information Systems and Software Techniques
Velbert, Germany		U Essen, D–45117 Essen, Germany
Conen@gmx.de		Reinhold.Klapsing@uni-essen.de

`Discussion Paper -- Version: 1.2, 11-October-2000`

Preface – A Note to the Reader: This is a step towards a logic-based formalization of RDF. It grew out of the necessity to precisely capture the semantics of RDF schemata while developing an RDF-based modeling framework for web applications (XWMF). The rules and facts that are described in the following allow to validate RDF triple sets. This in itself is not very exciting (VRP could be used instead). It becomes, however, relevant, if schema constructs are required that restrict, modify or extend the initial semantics, as, for example, monotone inheritance or typed containers. In such cases, rules that allow a precise interpretation of introduced schema constructs can be used to **extend** the basic rule set provided here. If the rules are themself given as XML/RDF, this can be a natural extension of the preciseness of the semantic expressibility of RDF schema definitions, ie. each RDF schema definition (syntax) may be accompanied by a document defining its semantics "logically". Here, this starts to become pretty interesting. It would, however, require to extend parsers with a logic inference engine (e.g., as has been done with SiLRI[1]). Some more remarks on our philosophy: we tried to retain as much of the RDF "spirit" as possible. We avoided, for example, to introduce new meta constructs etc. We designed the rules in a way such that violations of constraints are explicitly *detected* (this avoids to leave the knowledge base in an inconsistent state as it may happen if negated facts would be asserted). In this way, the violation predicates can be queried and appropriate actions can be determined by the interpreting application. We did not "resolve any ambiguities" (someone said that we should have made more decisions), simply because we haven't found ambiguities in the strict sense. Instead, we tried to point out where problems in applying the rules may occur. The rules make use of negations – but, on the knowledge level (that is: we interpret the triples by making certain relationships explicit, such as the instanceOf predicate), everything is **stratified** (in contrary to the statement that due to the triple nature of RDF statements, no reasonable stratification can be found – this is true only on the representation level). This is nice, because a natural model-theoretic interpretation of the rules exist. The rules and facts can easily be fed into a *datalog* interpreter (e.g. SiLRI). Inference engines based on SLD resolution may have some problems with the subPropertyOf rules (not a problem of the rules but of top-down/depth-first inference mechanisms). One more goodie needs to be mentioned: it is easy to query the triples (no specific query language is needed), because it's logic (datalog $\rightarrow$ relational algebra $\rightarrow$ SQL 3). However, the "content" of an RDF instance is, in this simple approach, embedded into the triple predicate *statement*. This may render querying a little bit tedious. It may turn out to be "nicer" (in what sense soever) to transform the RDF predicates into knowledge base predicates (see, for example, the F-Logic encoding of triples in SiLRi). We hope, however, that the rule set below may already be a step towards improved semantic interoperability (it is explicit, machine-readable, uses a well-known formalism and the rules can be transmitted to any client that can make use of it).

We hope for lively discussions – thank you in advance!

[1]Which is a nice tool that is - in the version that we have available - unfortunately not completely free from bugs

A Logical Interpretation of RDF

Abstract: The Resource Description Framework (RDF) is intended to be used to capture and express the conceptual structure of information offered in the Web. Interoperability is considered to be an important enabler of future web applications. While XML supports syntactic interoperability, RDF is aimed at semantic interoperability. Interoperability is only given if different users/agents interpret an RDF data model in the same way. Important aspects of the RDF model are, however, expressed in prose which may lead to misunderstandings. To avoid this, capturing the intended semantics of RDF in first-order logic might be a valuable contribution and may provide RDF with a formalization allowing its full exploitation as a key ingredient of the evolving Semantic Web. This paper seeks to express the concepts and constraints of the RDF model in first-order logic.

Keywords: RDF, First-Order Logic, semantic networks, semantic interoperability

1 Introduction

The Resource Description Framework [6, 2] is a basis for processing Web metadata. The used metadata format should allow to reason about data. The World Wide Web Consortium (W3C) intends RDF to be used in a variety of application areas; for example: to describe the content and content relationships available at a particular Web site or page, to facilitate knowledge sharing and exchange by intelligent software agents, to describe collections of pages that represent a single logical "document", to describe intellectual property rights of Web pages, or to express the privacy preferences of an user as well as the privacy policies of a Web site.

The Resource Description Framework in one of its encodings is represented as a semantic network. Neither the semantics of the network representation nor the semantics of the underlying RDF model are formally defined. This may lead to different interpretations of the same semantic network by different users/agents and thus, the interoperability claimed does not seem to be justifiable with respect to semantics.

In [9], Reimer points out that semantic networks are expressible in first-order logic. First-order logic is well studied and enables automatic inferencing based on syntactical processing of a knowledge base. In this paper we present a formalization of RDF concepts and constraints expressed in first-order logic. This intends to enable a further step towards semantic interoperability.

The remainder of this paper is structured as follows. Section 2 presents a brief discussion of related work. In Section 3, a set of rules and facts will be determined allowing to "materialize" the intended semantic constraints presented in the documents defining RDF [6, 2]. In the final section, possible consequences and applications of our approach will be presented.

2 Related Work

The relevance of knowledge representation for a number of research and application areas is well known. For a nice (technical) overview with an emphasis on logic, semantic nets and frames, see Reimer [9].

An example for an approach to relate logic-based formalisms and RDF is given in [8]. Because difficulties in representing the supposed self-referentiality are expected, the classic distinction between model and meta-models is made by introducing epistemological primitives (compare [9]) which are not further explainable in the formalism itself (such as special $type\_of$ predicates). To our opinion, this does not precisely capture what RDF was meant to be. After inspecting the set of rules that we provide below, we hope that it might be clear that the structural and semantic simplicity and clarity of RDF is based on the ability to make statements about instances, concepts, meta-concepts, meta-meta-concepts and so forth without being forced to introduce an explicit level of (meta-)constructs that are not explainable within the original model.

To be a little more precise: there is no self-referentiality in RDF, it is *self-expressibility*. Ultimately, RDF relies on a very small set of basic elements (namely the ordered sequences (triples)) and basic "axioms" (the rules and facts from below). What seems to be self-referentiality (in [8], for instance, the properties subClassOf and type are mentioned) becomes clear, if the context of their usage is regarded, that is their position in the triple. What remains in the end is syntactical manipulation to determine properties of the model – and the (semantic) interpretation of the result depends on the position of "entities" in a triple and not (only) on the fact, that the entities are "somewhere" in the triple. As can be seen from implementing the rules and facts given below, the concepts and constraints of RDF are nicely and *logically* expressible.

A useful approach to explain RDF with the help of logic can be found in the RDF tutorial of Champin [3]. Our approach is more complete, tries to seperate clearly between the level of representation (triples) and the level of knowledge (instanceOf, Resource, statements, predicates to express constraint violations etc.), avoids asserting negated facts, and discusses a way to utilize the rules in applications (see below). Nevertheless, we have profited from reading Champin's paper.

Earlier work on processing RDF triples with logic have been described in [4]. The "Simple Logic-based RDF Interpreter (SiLRI)" is presented. However, the intention had not been to clarify the semantics of RDF and to express them in suitable logic rules but to directly use "logified" triples as a fact base. Our approach is more general in the sense that we try to provide (in extension to the simple transformation of a RDF triple into a fact in F-Logic) a basic set of facts and rules that capture the RDF concepts and constraints and can be loaded into a logic-based inference engine to represent the intended semantics of RDF documents (be it schemata or schema instances). We used the inference engine in SiLRI to test our rule set. A SiLRI-conform datalog formulation of the rules and facts of our paper is given in the appendix. We also used SWI-Prolog [10] to test our facts and rules. A Prolog-conform formulation of the rules and facts of our paper can be obtained from [7]. The SGML package of SWI-Prolog provides an RDF parser. The SWI-Prolog RDF parser together with our rules and facts is used to shift an RDF model expressed in XML-Syntax from the representation level to the knowledge level. A knowledge database which is build up after reading an RDF model can be queried with Prolog predicates according to our facts and rules. An online test suite of this application can be found at [7].

The discussion of numerous aspects of a Semantic Web is ongoing and lively. We think, that RDF-based representations of semantics may play a crucial role in further developments. We think that capturing the intended semantics in first-order logic might be a valuable contribution to this and may provide RDF with a formalization allowing its full exploitation as an key ingredient to an evolving Semantic Web.

3 RDF Concepts Expressed in First Order Logic

An RDF graph consists of nodes and arcs. Nodes are labeled either with an URI (concept Resource) or an atomic value (concept Literal). We will assume the following definitions for an RDF graph, which are adapted and extended from the definitions stated in [9] for semantic networks. (1) In an RDF graph two nodes with different labels are considered to be different. (2) Two nodes with the same label are not allowed. (3) A Literal node is represented by a rectangle. (4) A Resource (concept) node is represented by an oval. (5) The representation of a Resource node or a Literal node implicitly states its existence. (6) An arc links a Resource node either with a Literal node or another Resource node and is labeled with an URI. Two nodes linked with an arc are called a statement. (7) All statements in an RDF Graph are considered to be implicitly and conjunctively concatenated.

Assume that an RDF model is given as a semantic network. The predicate symbol `arc(s,p,o)` be true if there is a directed arc from node `s` to node `o` and the arc is labeled with `p`.[2] The predicate symbol rfc2396_conform(x) is true if the string x is built according to RFC 2396 [1]. The following rule shifts the perspective from representation level (*arc*) to knowledge level (*statement*[3]).

$$\forall s, p, o : arc(s, p, o) \wedge rfc2396\_conform(p) \Rightarrow statement(s, p, o) \tag{1}$$

In the following subsections, we will present most of the RDF concepts and constraints and their formulation as logic rules. All text presented as Definitions have been taken directly from [6] (RDF-Definition) or [2] (RDFS-Definition).

3.1 Basic RDF Concepts and Predicates

3.1.1 Resource

RDFS-Definition: All things being described by RDF expressions are called resources, and are considered to be instances of the class `rdfs:`[4] `Resource`. The RDF class `rdfs:Resource` represents the set called 'Resources' in the formal model for RDF presented in section 5 of the Model and Syntax specification [6].

Members of the set Resource (see definition (1) of the RDF definitions summary in Appendix 5.4) are the concepts in an RDF knowledge base. According to Reimer [9], concepts are all concrete and abstract things which should be described. "Named" Resources[5] are identified by URIs. For the definition of URIs see [1][6]. The first and second entity of a statement, respectively of a triple, are Resources, the later is identified by an URI[7].

$$\forall s, p, o : statement(s, p, o) \Rightarrow res(s) \wedge uri(p) \wedge obj(o) \tag{2}$$

$$\forall r : uri(r) \Rightarrow res(r) \tag{3}$$

$$\forall r : res(r) \wedge rfc2396\_conform(r) \Rightarrow named\_res(r) \tag{4}$$

[2] This directly corresponds to a triple $\{pred, sub, obj\}$ in the representation of an RDF model as a set of triple statements.

[3] In an earlier version of the paper, the following rule required that subjects are RFC2396 conform. In the specification, this is only necessary for (explicitly) named resources. Anonymous resources, which are present in the graphical and the XML representation of the model, lead to generated names in the triple model. These names are not necessarily URIs (to be more precise: no parser generates valid URIs for anonymous resources). To allow the inclusion of generated non-URI names in triples, we weakened the constraint in the following rule. It is interesting to note that the formal model does not constrain the name space for the identification of resources - however, in a complete set theoretic model, the alphabet to denote that certain entities are members of the set of resources would have to be present. Implicitly, the name space is partitioned into URIs, Literals and "generated" resource names. The specification does not provide a mechanism to prevent someone who is modelling in triples to use a literal that looks like an URI or to generate a URI-conform string as a name for anonymous resources. It is also hard to argue that any restriction on the identifier for a resource is necessary. Relevant is the meaning of the identifier – and this meaning will be a consequence of its use as an argument to predicates.

[4] The XML-namespace *rdf:* refers to the namespace-URI *http://www.w3.org/1999/02/22-rdf-syntax-ns#* and the *rdfs:* refers to *http://www.w3.org/2000/01/rdf-schema#*.

[5] In the underlying set theoretic model, each resources requires an identifier to "be something", that is, each resource has an identifier. Because there are anonymous resources in the graphical/XML representation, there is a necessity to distinguish among resources that are named (with an URI identifier) and anonymous resources (with an unconstrained identifier), see also the previous footnote.

[6] Note, that [1] defines an URI as a conceptual mapping to an entity. The referenced entity can be changed over time and according to the requested MIME-Type different entities are referenced. Thus URIs are not necessarily unique.

[7] Which is an interpretation of the specs.

3.1.2 Literal

RDF-Definition: The most primitive value type represented in RDF, typically a string of characters. The content of a literal is not interpreted by RDF itself and may contain additional XML markup. Literals are distinguished from Resources in that the RDF model does not permit literals to be the subject of a statement.

Members of the set Literal (see definition (2) of the RDF model) are atomic values such as textual strings and are always the object of a statement. In a semantic RDF network it is not allowed to draw an arc outgoing from a node of type Literal. Thus no statement can be made about a concept of type Literal. A node in a semantic RDF network representing a Literal is labeled with the corresponding atomic value. A Literal node it graphically represented as a rectangle.

$$\forall o : obj(o) \wedge \neg res(o) \Rightarrow lit(o) \tag{5}$$

$$\forall o : lit(o) \Rightarrow instanceOf(o, rdfs\_Literal) \tag{6}$$

3.1.3 Property

RDF-Definition: A specific attribute with defined meaning that may be used to describe other resources. A property plus the value of that property for a specific resource is a statement about that resource. A property may define its permitted values as well as the types of resources that may be described with this property.

No rule is required, properties are a projection of the second argument of the logical predicate `statement`. Note that a member of the set Property (see definition (3) of the RDF model) is also a Resource.

3.1.4 Statement

RDF-Definition: A specific resource together with a named property plus the value of that property for that resource is an RDF statement. These three individual parts of a statement are called, respectively, the subject, the predicate, and the object. The object of a statement (i.e., the property value) can be another resource or it can be a literal; i.e., a resource (specified by an URI) or a simple string or other primitive datatype defined by XML. In RDF terms, a literal may have content that is XML markup but is not further evaluated by the RDF processor.

To be able to make a statement about a statement it is possible to reify a statement (see definitions (9) and (9a)-(9d) of the RDF model given in Appendix 5.4). Reifying a statement is achieved by creating a concept node of the type Statement representing a statement. The values of the Property instances `rdf:subject`, `rdf:predicate` and `rdf:object` refer to the parts of a statement.

$$\begin{gathered} \forall s, s', p, o : res(s) \wedge uri(p) \wedge statement(s', TYPE^{8}, STATEMENT) \wedge \\ statement(s', SUBJECT, s) \wedge statement(s', PREDICATE, p) \wedge statement(s', OBJECT, o) \\ \Rightarrow reifies(s', s, p, o) \end{gathered} \tag{7}$$

[8]Abbreviation for a complete URI. Abbreviations will be expanded using either the rdf- or rdfs-namespace. We decided to not emphasize this distinction more than necessary.

$$\forall s, s', p, o : reifies(s', s, p, o) \Rightarrow reifyingStatement(s') \tag{8}$$

RDF-Definition: A statement and its corresponding reified statement exist independently in an RDF graph and either may be present without the other. The RDF graph is said to contain the fact given in the statement if and only if the statement is present in the graph, irrespective of whether the corresponding reified statement is present.

$$\forall s, s', p, o : reifies(s', s, p, o) \land statement(s, p, o) \Rightarrow reifies\_fact(s', s, p, o) \tag{9}$$

3.2 Schema-Definition Concepts and Predicates

3.2.1 type

RDFS-Definition: This indicates that a resource is a member of a class, and thus has all the characteristics that are to be expected of a member of that class. When a resource has an `rdf:type` property whose value is some specific class, we say that the resource is an instance of the specified class. The value of an `rdf:type` property for some resource is another resource which must be an instance of `rdfs:Class`. The resource known as `rdfs:Class` is itself a resource of `rdf:type` `rdfs:Class`. Individual classes (for example, 'Dog') will always have an `rdf:type` property whose value is `rdfs:Class` (or some subclass of `rdfs:Class` ...). A resource may be an instance of more than one class.

See also definition (5) given in Appendix 5.4.

$$\forall i, c : statement(i, TYPE, c) \Rightarrow instanceOf(i, c) \tag{10}$$

Additional instanceOf relations will be derived from the rules capturing the subClassOf property. The constraints mentioned above are expressed as facts, such as $statement(CLASS, TYPE, CLASS)$ (listed in the Appendix), and constraint violations will be detected by applying the range and domain constraint rules (see below).

3.2.2 Class

RDFS-Definition: This corresponds to the generic concept of a Type or Category, similar to the notion of a Class in object-oriented programming languages such as Java. When a schema defines a new class, the resource representing that class must have an `rdf:type` property whose value is the resource `rdfs:Class`. RDF classes can be defined to represent almost anything, such as Web pages, people, document types, databases or abstract concepts.

We feel that the sentence *"...similar to the notion of a Class in object-oriented programming languages such as Java."* is misleading, as no monotonic inheritance of properties is given in the RDF model. However, with additional constraint rules monotonic inheritance can be expressed.

RDFS-Definition: The resource known as `rdfs:Class` is itself a resource of `rdf:type` `rdfs:Class`.

This, again, translates to `statement(CLASS,TYPE,CLASS)`. We included this kind of factual knowledge in the fact base that can be found in the appendix of this paper. It also includes (most of) the other facts that can be deduced from [6, 2], especially the `rdf:type`, `rdfs:subClassOf` properties and the `rdfs:range` and `rdfs:domain` constraints.

3.2.3 subClassOf

RDFS-Definition: This property specifies a subset/superset relation between classes. The `rdfs:subClassOf` property is transitive. If class `A` is a subclass of some broader class `B`, and `B` is a subclass of `C`, then `A` is also implicitly a subclass of `C`. Consequently, resources that are instances of class `A` will also be instances of `C`, since `A` is a sub-set of both `B` and `C`.

$$\forall c1, c2 : statement(c1, SUBCLASSOF, c2) \Rightarrow subClassOf(c1, c2) \tag{11}$$

$$\forall c1, c2, c3 : subClassOf(c1, c2) \wedge subClassOf(c2, c3) \Rightarrow subClassOf(c1, c3) \tag{12}$$

$$\forall i, c1, c2 : instanceOf(i, c1) \wedge subClassOf(c1, c2) \Rightarrow instanceOf(i, c2) \tag{13}$$

RDFS-Definition: A class may be a subclass of more than one class.

No rule required.

RDFS-Definition: A class can never be declared to be a subclass of itself, nor of any of its own subclasses.

So, cycles in the SUBCLASSOF chain are forbidden. As this is disputable (it expresses equality, compare [3]) and as we decided to *detect* constraint violations but *not* to *enforce* certain reactions[9], the following rule is introduced:

$$\forall c1, c2 : subClassOf(c1, c2) \wedge c1 = c2 \Rightarrow subclass\_cycle\_violation(c1). \tag{14}$$

The other constraints translate to range and domain constraints.

3.2.4 subPropertyOf

RDFS-Definition: The property `rdfs:subPropertyOf` is an instance of `rdf:Property` that is used to specify that one property is a specialization of another. A property may be a specialization of zero, one or more properties. If some property `P2` is a `subPropertyOf` another more general property `P1`, and if a resource `A` has a `P2` property with a value `B`, this implies that the resource `A` also has a `P1` property with value `B`.

$$\forall p1, p2 : statement(p1, SUBPROPERTYOF, p2) \Rightarrow subPropertyOf(p1, p2) \tag{15}$$

$$\forall p1, p2, p3 : subPropertyOf(p1, p2) \wedge subPropertyOf(p2, p3) \Rightarrow subPropertyOf(p1, p3) \tag{16}$$

This establishes the transitivity of `subPropertyOf`. It is also used to attach "derived" properties to resources:

$$\forall s, p1, p2, o : statement(s, p1, o) \wedge subPropertyOf(p1, p2) \Rightarrow statement(s, p2, o) \tag{17}$$

[9] With respect to this, asserting negated facts (as is suggested in [3]) does not seem to be a reasonable way to express constraint violations.

RDFS-Definition: A property can never be declared to be a subproperty of itself, nor of any of its own subproperties.

With a reasoning analogous to the SUBCLASS case, the following rule is introduced:

$$\forall p1, p2 : subPropertyOf(p1, p2) \wedge p1 = p2 \Rightarrow subproperty\_cycle\_violation(p1). \quad (18)$$

This concludes the presentation of rules for `subPropertyOf`. Only this property requires the generation[10] of new statements. The rules above simply add all derivable properties. This mutual dependency between the predicates subPropertyOf and statement may lead to problems if the rule set is used with standard SLD-resolution. In this case, the rules can be rewritten (and reduced) to check, if every statement that should be present is indeed modeled.

3.2.5 domain

RDFS-Definition: An instance of ConstraintProperty that is used to indicate the class(es) on whose members a property can be used. A property may have zero, one, or more than one class as its domain. If there is no domain property, it may be used with any resource. If there is exactly one domain property, it may only be used on instances of that class (which is the value of the domain property). If there is more than one domain property, the constrained property can be used with instances of any of the classes (that are values of those domain properties). ...The `rdfs:domain` of `rdfs:domain` is the class `rdf:Property`. This indicates that the domain property is used on resources that are properties. The `rdfs:range` of `rdfs:domain` is the class `rdfs:Class`. This indicates that any resource that is the value of a domain property will be a class. Note: This specification does not constraint the number of `rdfs:domain` properties that a property may have. If there is no domain property, we know nothing about the classes with which the property is used. If there is more than one `rdfs:domain` property, the constrained property can be used with resources that are members of any of the indicated classes. Note that unlike range this is a very weak constraint.

Rules for determining domain constraint violations[11]:

$$\forall p, i, c : statement(p, DOMAIN, c) \wedge instanceOf(i, c) \Rightarrow domain(i, p) \quad (19)$$

$$\forall p, c : statement(p, DOMAIN, c) \Rightarrow domain\_constrained\_prop(p) \quad (20)$$

$$\forall s, p, o : statement(s, p, o) \wedge domain\_constrained\_prop(p) \wedge \neg domain(s, p) \quad (21)$$
$$\Rightarrow domain\_violation(s, p, o)$$

Note: Rules for determining a violation of the only-classes-as-object-of-domain-property-statements constraint are not necessary, because, with the given facts, this is a subcase of the general case above.

[10]The inherently transitive rule $\forall s, p1, p2, o : statement(s, p1, o) \wedge statement(p1, SUBPROPERTYOF, p2) \Rightarrow statement(s, p2, o)$ would suffice to generate all derivable new "statements". However, the no-cycle constraint requires to make the transitivity of the subProperty relation explicit, so we decided to generate the new statements using the knowledge level predicate subPropertyOf, which is present anyway.

[11]With respect to potential extensions or additional rules describing new or further restricted constraints for specific schemata, a more general approach might be to introduce a *violation* predicate and to register the violating statements and a violation identifier in this predicate. This has been done for the datalog rules in the appendix and is a simple extension of the rules given here.

3.2.6 range

RDFS-Definition: An instance of ConstraintProperty that is used to indicate the class(es) that the values of a property must be members of. The value of a range property is always a Class. Range constraints are only applied to properties. A property can have at most one range property. It is possible for it to have no range, in which case the class of the property value is unconstrained.... The `rdfs:domain` of `rdfs:range` is the class `rdf:Property`. This indicates that the range property applies to resources that are themselves properties. The `rdfs:range` of `rdfs:range` is the class `rdfs:Class`. This indicates that any resource that is the value of a range property will be a class.

Rules for determining if at most one range constraint is present:

$$\forall s, o : statement(p, RANGE, o) \Rightarrow is\_range(o, p) \tag{22}$$

$$\forall p, r1, r2 : is\_range(r1, p) \wedge is\_range(r2, p) \wedge (r1 \neq r2) \Rightarrow range\_cardinality\_violation(p) \tag{23}$$

Remark: It might be reasonable to allow multiple range constraints. For pointers to a discussion, see the next footnote.

Rules for determining range violations:

$$\forall p, r : is\_range(r, p) \Rightarrow has\_range(p) \tag{24}$$

$$\forall s, p, o, c : statement(s, p, o) \wedge instanceOf(o, c) \wedge is\_range(c, p) \Rightarrow range(o, p) \tag{25}$$

$$\forall s, p, o : statement(s, p, o) \wedge has\_range(p) \wedge \neg range(o, p) \Rightarrow range\_violation(s, p, o) \tag{26}$$

This parallels the determination of domain constraint violations, it has the consequence that also multiple range constraints are checked. We've chosen to model in this way because it is not immediately clear how the reaction on a violation of the one-range-at-most constraint should look like. If the one-range constraint should be enforced, an application should react appropriately if the $range\_cardinality\_violation$ predicate is not empty before utilizing the $range\_violation$ information. If the general behaviour should be to ignore multiple range constraints, the negated $range\_cardinality\_violation$ predicate can be added to the $range\_violation$ subgoals. We think, however, that the least common denominator in interpreting the semantics should be to *detect* violations and to leave further reactions to applications.[12]

3.3 Utility Concepts

3.3.1 Seq

RDFS-Definition: This corresponds to the class called 'Sequence' in the formal model for RDF presented in section 5 of the Model and Syntax specification [6]. It is an instance of `rdfs:Class` and `rdfs:subClassOf` `rdfs:Container`.

[12] Further comments on the usefulness of range constraints and the limitation imposed by the one-range-at-most constraint can be found in the discussion related to a draft of this paper that took place in the RDF interest group mailing list, see http://lists.w3.org/Archives/Public/www-rdf-interest/2000Sep/0107.html (discussion starting point). For an overview of the whole discussion related to the draft paper see http://nestroy.wi-inf.uni-essen.de/rdf/logical_interpretation/discussion.html.

See also definition (10) and (11) given in the Appendix.

$$\forall s, o : statement(s, TYPE, SEQ) \wedge statement(s, \_1, o) \Rightarrow seq(s, o) \tag{27}$$

$$\forall s, o1, o2 : seq(s, o1) \wedge statement(s, \_2, o2) \Rightarrow seq(s, o1, o2) \tag{28}$$

$$\vdots$$

Remark: Consider these rules as a proposal only. They could more easily be formulated with lists or other "builtin" devices in specific implementations of inference engines. Some problems on the semantic level remain: it is not clear, how a missing $\_X$ in a sequence (e.g.,$\_1$, $\_3$, $\_4$) should be treated – or, for instance, two $\_X$ properties that are attached to a resource (which value is relevant?). More problems may occur if the evolution of a model is considered: removing or adding an element may require excessive renumbering (most problems are "automatically " avoided, if the arcs have been generated from RDF/XML serialization syntax). The rules (and problems) for BAG and ALT are similar.

4 Discussion

We tried to design the presented logic-based formulation of RDF concepts and constraints in the RDF-spirit of simplicity, universality, and extensibility. It gives an opportunity to more precisely capture the intended semantic constraints that underly schemata developed on top of RDF. The semantics of RDF provide an open and relatively unconstrained framework of basic concepts and constraints. It is possible to exploit and restrict these opportunities by introducing rules into our simple knowledge base that capture the intended semantics of newly introduced concepts and are build upon the "fundamental" rules and facts of RDF. We plan to describe the logical formalization of XWMF (see [5, 11]), a framework and toolset for RDF-based web engineering as an example. In XWMF, for instance, strictly monotonous inheritance is used to constrain the typing of components. This can easily be encoded (analogous to the rule for subPropertyOf) as a logic rule, which is then added to the basic knowledge base. It is straightforward to provide an XML vocabulary that allows the specification of the knowledge base in XML. An extended RDF parser could use the logic rules, the RDF Schema definition and the RDF document to determine the validity of the document. For example, the SiLRI inference engine could be extended to provide this capability (we provide the basic set of facts and rules in datalog syntax in the appendix). With a suitable API or via the exchange of XML messages in a suitable protocol, the inference engine could and should be made part of the ongoing processing of RDF statements in the supported application. In a naive implementation, this would only require that queries could be directed to the engine and answers to the query would be delivered to the application (the possibility to retract facts - that is, to treat non-monotonous dynamic behaviour of the encoded knowledge would require some more effort). In XWMF or other modelling tools that represent the modelling knowledge in RDF, this could, for example, be used to offer a set of possibilities for attaching properties to resources (by exploiting the DOMAIN constraint), to select values for objects (RANGE), for generating inherited properties (if inheritance is monotonous as in XWMF) etc.

While this alone is already useful for "standalone" applications, the true power of an unified logic representation of the RDF concepts (which are, in the RDF documents, only syntactically formalized) and the intended constraints (which are only informally described in the RDF documents) lies in the possibility to improve interoperability by providing a level of precise and validable *basic (or minimal) common semantics*. Without this, the (semantic) interpretation of RDF documents (be it schemata or schema instances) will be subject to interpretation and discussion and this will ultimately lead to ambiguity – very much like the textual description of presentation "semantics" in HTML lead to different visual interpretations – but this time with even more unwanted effects, because RDF wants to specify (some) semantics, and thus the meaning of the basic set of semantic constructs in RDF should be made as precise as possible or an universe of interpretations will arise where clarity and common understanding were intended. In the present state, it is difficult for anyone to verify if his understanding of RDF's intentions is "correct" – the approach described herein allows to do so under the sufficiently safe assumption that the scientific community has a well-developed common understanding of how to interpret first-order logic.

References

[1] Tim Berners-Lee, Roy Fielding, and Larry Masinter. Uniform Resource Identifiers (URI): Generic Syntax. RFC, category: Standards track, IETF, August 1998. http://www.ietf.org/rfc/rfc2396.txt.

[2] Dan Brickley and R.V. Guha. Resource Description Framework (RDF) Schema Specification 1.0. Candidate Recommendation, W3C, March 2000. http://www.w3.org/TR/2000/CR-rdf-schema-20000327.

[3] Pierre-Antoine Champin. RDF Tutorial, March 2000. http://www710.univ-lyon1.fr/ champin/rdf-tutorial/rdf-tutorial.ps.gz.

[4] Stefan Decker, Dan Brickley, Janne Saarela, and Jürgen Angele. A Query and Inference Service for RDF. In *Online Proceedings of the QL'98 - The Query Languages Workshop*. http://www.w3.org/TandS/QL/QL98/pp/queryservice.html.

[5] Reinhold Klapsing and Gustaf Neumann. Applying the Resource Description Framwork to Web Engineering. In *Proceedings of the 1st International Conference on Electronic Commerce and Web Technologies: EC-Web 2000*, LNCS. Springer, September 2000. http://nestroy.wi-inf.uni-essen.de/xwmf/paper/xwmf_EcWeb/.

[6] Ora Lassila and Ralph R. Swick. Resource Description Framework (RDF) Model and Syntax Specification. Recommendation, W3C, February 1999. http://www.w3.org/TR/1999/REC-rdf-syntax-19990222.

[7] Logcial Interpretation of RDF – Online Test Suite. http://wonkituck.wi-inf.uni-essen.de/rdfs.html.

[8] Wolfgang Nejdl, Martin Wolpers, and Christian Capelle. The RDF Schema Specification Revisited. In *Modelle und Modellierungssprachen in Informatik und Wirtschaftsinformatik: Beiträge des Workshops Modellierung 2000*, volume 15 of *Koblenzer Schriften zur Informatik*, 2000. http://www.kbs.uni-hannover.de/Arbeiten/Publikationen/2000/modeling2000/wolpers.pdf.

[9] Ulrich Reimer. *Einführung in die Wissensrepräsentation*. Leitfäden der angewandten Informatik. Teubner, 1991.

[10] SWI-Prolog. http://www.swi.psy.uva.nl/projects/SWI-Prolog/.

[11] XWMF-Home-Page. http://nestroy.wi-inf.uni-essen.de/xwmf/.

5 Appendix

5.1 Datalog Rules

Assume that the arcs have already been transformed to statements (predicate s below). The rules and facts can be fed to the datalog parser (with option -simple) of SiLRI. The rules can also be requested as files from the authors.

```
res(S) & uri(P) & obj(O) <- s(S,P,O).
res(R) <- uri(R).
named_res(R) <- res(R) & rfc2396_conform(R).
lit(O) <- obj(O) & -res(O).
instanceOf(O,rdfs_Literal) <- lit(O).

reifies(R,S,P,O) <- res(S) & uri(P) & s(R,rdf_type,rdf_statement) &
        s(R,rdf_subject,S) & s(R,rdf_predicate,P) & s(R,rdf_object,O).
reifyingStatement(R) <- reifies(R,S,P,O).
reifies_fact(R,S,P,O) <- reifies(R,S,P,O) &  s(S,P,O).

instanceOf(I,C) <- s(I,rdf_type,C).
subClassOf(C,D) <- s(C,rdfs_subClassOf,D).
subClassOf(C,E) <- subClassOf(C,D) & subClassOf(D,E).
instanceOf(I,D) <- instanceOf(I,C) & subClassOf(C,D).
subClass_cycle_violation(C1) <- subClassOf(C1,C2) & unify(C1,C2).

subPropertyOf(A,B) <- s(A,rdfs_subPropertyOf,B).
subPropertyOf(A,C) <- subPropertyOf(A,B) & subPropertyOf(B,C).
s(S,P2,O) <- s(S,P1,O) & subPropertyOf(P1,P2).
subProperty_cycle_violation(P1) <- subPropertyOf(P1,P2) & unify(P1,P2).

domain_constrained_prop(P) <- s(P,rdfs_domain,X).
domain(X,P) <- s(P,rdfs_domain,C) & instanceOf(X,C).
domain_violation(S,P,O) <- s(S,P,O) & domain_constrained_prop(P) & -domain(S,P).

is_range(X,P) <- s(P,rdfs_range,X).
range_cardinality_violation(P) <- is_range(X,P) & is_range(Y,P) & -unify(X,Y).
has_range(P) <- is_range(X,P).
range(X,P) <- is_range(C,P) & instanceOf(X,C).
range_violation(S,P,O) <- s(S,P,O) & has_range(P) & -range(O,P).

violation("Domain violation",S,P,O) <- domain_violation(S,P,O).
violation("Range cardinality",S,rdfs_range,O) <-
           range_cardinality_violation(S) & s(S,rdfs_range,O).
violation("Range violation",S,P,O) <- range_violationy(S,P,O).
```

5.2 Datalog Concept Facts

```
s(rdfs_Literal,rdf_type,rdfs_Class).
s(rdfs_Class,rdf_type,rdfs_Class).
s(rdfs_Resource,rdf_type,rdfs_Class).
s(rdf_Property,rdf_type,rdfs_Class).
s(rdfs_ConstraintResource,rdf_type,rdfs_Class).
s(rdfs_ConstraintProperty,rdf_type,rdfs_Class).
s(rdfs_ContainerMembershipProperty,rdf_type,rdfs_Class).
s(rdfs_range,rdf_type,rdfs_ConstraintProperty).
s(rdfs_domain,rdf_type,rdfs_ConstraintProperty).
s(rdf_type,rdf_type,rdf_Property).
s(rdfs_subPropertyOf,rdf_type,rdf_Property).
s(rdfs_subClassOf,rdf_type,rdf_Property).
s(rdfs_seeAlso,rdf_type,rdf_Property).
s(rdfs_isDefinedBy,rdf_type,rdf_Property).
s(rdfs_comment,rdf_type,rdf_Property).
s(rdfs_label,rdf_type,rdf_Property).

s(rdf_subject,rdf_type,rdf_Property).
s(rdf_predicate,rdf_type,rdf_Property).
s(rdf_object,rdf_type,rdf_Property).
s(rdf_Statement,rdf_type,rdfs_Class).

s(rdfs_Class,rdfs_subClassOf,rdfs_Resource).
```

```
s(rdfs_ConstraintResource,rdfs_subClassOf,rdfs_Resource).
s(rdf_Property,rdfs_subClassOf,rdfs_Resource).
s(rdfs_ConstraintProperty,rdfs_subClassOf,rdf_Property).
s(rdfs_ConstraintProperty,rdfs_subClassOf,rdfs_ConstraintResource).
s(rdfs_ContainerMembershipProperty,rdfs_subClassOf,rdf_Property).
```

5.3 Datalog Constraint Facts

```
s(rdfs_range,rdfs_range,rdfs_Class).
s(rdf_type,rdfs_range,rdfs_Class).
s(rdfs_subClassOf,rdfs_range,rdfs_Class).
s(rdfs_domain,rdfs_range,rdfs_Class).
s(rdfs_comment,rdfs_range,rdfs_Literal).
s(rdfs_label,rdfs_range,rdfs_Literal).
s(rdf_subject,rdfs_range,rdfs_Resource).
s(rdf_predicate,rdfs_range,rdf_Property).
s(rdfs_subPropertyOf,rdfs_range,rdf_Property).

s(rdfs_subPropertyOf,rdfs_domain,rdf_Property).
s(rdfs_range,rdfs_domain,rdf_Property).
s(rdfs_domain,rdfs_domain,rdf_Property).
s(rdf_subject,rdfs_domain,rdf_Statement).
s(rdf_predicate,rdfs_domain,rdf_Statement).
s(rdf_object,rdfs_domain,rdf_Statement).
s(rdf_type,rdfs_domain,rdfs_Resource).
s(rdfs_subClassOf,rdfs_domain,rdfs_Class).
s(rdfs_comment,rdfs_domain,rdfs_Resource).
s(rdfs_label,rdfs_domain,rdfs_Resource).
```

5.4 Basic Definitions of the RDF Model

The following definitions (1)-(11) of the RDF model have been taken from Section 5 (Formal Model for RDF) of the RDF Model & Syntax Specification [6].

(1) There is a set called Resources.
(2) There is a set called Literals.
(3) There is a subset of Resources called Properties.
(4) There is a set called Statements, each element of which is a triple of the form `{pred, sub, obj}` where `pred` is a property (member of Properties), `sub` is a resource (member of Resources), and `obj` is either a resource or a literal (member of Literals).
(5) There is an element of Properties known as `type`.
(6) Members of Statements of the form `{type, sub, obj}` must satisfy the following: `sub` and `obj` are members of Resources. [2] places additional restrictions on the use of `type`.
(7) There is an element of Resources, not contained in Properties, known as `RDF:Statement`.
(8) There are three elements in Properties known as `RDF:predicate`, `RDF:subject` and `RDF:object`.
(9) Reification of a triple `{pred, sub, obj}` of Statements is an element `r` of Resources representing the reified triple and the elements `s1, s2, s3,` and `s4` of Statements such that

(9a) `s1: {RDF:predicate, r, pred}`
(9b) `s2: {RDF:subject, r, subj}`
(9c) `s3: {RDF:object, r, obj}`
(9d) `s4: {RDF:type, r, [RDF:Statement]}`

(10) There are three elements of Resources, not contained in Properties, known as `RDF:Seq`, `RDF:Bag`, and `RDF:Alt`.
(11) There is a subset of Properties corresponding to the ordinals (1, 2, 3, ...) called `Ord`. We refer to elements of `Ord` as `RDF:_1, RDF:_2, RDF:_3, ...`