A Declarative Semantics for the ShEx grammar

August 18, 2015

Contents

1 The RDF Data Model

We begin by defining a formal model of the RDF triple and corresponding graph. Much of the work in this section derives directly or indirectly from the author's previous work on *A Formal Model for RDF Dataset Constraints*[**?**], *Validating RDF with Shape Expressions*[**?**] as well as from various submissions and contributions to the 2013 RDF Validation Workshop[**?**] where similar models were used to develop requirements for a language to specify pre and post conditions for RDF dataset updates as well as to describe a basic set of predicates which would allow invariants to be asserted about the state of RDF Graphs.

$$[String]$$

1.1 Basic RDF Data Structures

An **RDF graph** is a set of RDF Triples:

$$Graph == \mathbb{P}\ Triple$$

An **RDF triple** consists of three components:

- the subject, which is an IRI or a blank node
- the predicate, which is an IRI
- the object, which is an IRI, a literal or a blank node

Triple

$$s, p, o : RDFTerm$$

$$iri^{\sim} s \in IRI \lor bnode^{\sim} s \in BNODE$$
$$iri^{\sim} p \in IRI$$

IRIs, literals and blank nodes are collectively known as **RDF terms**. IRIs, literals and blank nodes are distinct and distinguishable.

$$[IRI, BNODE]$$
$$RDFTerm ::= iri\langle\!\langle IRI\rangle\!\rangle \mid literal\langle\!\langle RDFLiteral\rangle\!\rangle \mid bnode\langle\!\langle BNODE\rangle\!\rangle$$

A **literal** in an RDF graph consists of two or three elements:

- a **lexical form**, ...
- a **datatype IRI**, ...
- if there is a language tag the datatype IRI must be `http://www.w3.org/1999/02/22-rdf-syntax-ns#langString`

$[LANGTAG]$

$$rdf\_langString : IRI$$

$$
\begin{array}{|l}
\textbf{RDFLiteral} \\
v : String \\
dataType : IRI[0\,..\,1] \\
langTag : LANGTAG[0\,..\,1] \\
\hline
\#langTag > 0 \Leftrightarrow (\#dataType > 0 \land head\ dataType = rdf\_langString)
\end{array}
$$

1.2 Well Known RDF IRIs

$$
\begin{array}{|l}
RDFGraph : Graph \\
rdf\_type, rdfs\_subClassOf, rdfs\_subPropertyOf, rdf\_first, rdf\_rest, rdf\_Nil : IRI \\
rdf\_List, rdf\_Seq, rdf\_Bag, xsd\_string, xsd\_integer, xsd\_double, xsd\_boolean : IRI \\
rdf\_Alt : IRI
\end{array}
$$

Define a set of synonyms to represent the above IRIs as their corresponding RDF terms.

$$
\begin{array}{l}
rdf\_type\_t == iri\ rdf\_type \\
rdfs\_subClassOf\_t == iri\ rdfs\_subClassOf \\
rdfs\_subPropertyOf\_t == iri\ rdfs\_subPropertyOf \\
rdf\_first\_t == iri\ rdf\_first \\
rdf\_rest\_t == iri\ rdf\_rest \\
rdf\_Nil\_t == iri\ rdf\_Nil \\
rdf\_Seq\_t == iri\ rdf\_Seq \\
rdf\_List\_t == iri\ rdf\_List \\
rdf\_Bag\_t == iri\ rdf\_Bag \\
rdf\_Alt\_t == iri\ rdf\_Alt \\
xsd\_string\_t == iri\ xsd\_string \\
xsd\_integer\_t == iri\ xsd\_integer \\
xsd\_double\_t == iri\ xsd\_double \\
xsd\_boolean\_t == iri\ xsd\_boolean
\end{array}
$$

2 Graph Operations

This section defines a set of “low level” functions providing a foundation for making assertions about the members of an RDF Graph.

triplesFor - given a graph and an RDF term, return the set of triples where the term appears as a subject

$$
\begin{array}{l}
triplesFor : Graph \rightarrow\!\!\!\!\!+ RDFTerm \rightarrow\!\!\!\!\!+ \mathbb{P}\ Triple \\
\hline
\forall g : Graph;\ s : RDFTerm \bullet triplesFor\ g\ s = \{t : g \mid t.s = s\}
\end{array}
$$

objectsOf - given a graph, a subject RDF term and a predicate URI, return the set of objects occurring in triples with the given subject and predicate

$$
\begin{array}{l}
objectsOf : Graph \rightarrow\!\!\!\!\!+ RDFTerm \rightarrow\!\!\!\!\!+ RDFTerm \rightarrow\!\!\!\!\!+ \mathbb{P}\ RDFTerm \\
\hline
\forall g : Graph;\ s, p : RDFTerm \bullet objectsOf\ g\ s\ p = \{t : triplesFor\ g\ s \mid t.p = p \bullet t.o\}
\end{array}
$$

objectOf - return the unique target of a subject and predicate in a graph or rdf:Nil if there are no or more than one targets

$$
\begin{array}{l}
objectOf : Graph \rightarrow\!\!\!\!\!+ RDFTerm \rightarrow\!\!\!\!\!+ RDFTerm \rightarrow\!\!\!\!\!+ RDFTerm \\
\hline
\forall g : Graph;\ s, p : RDFTerm \bullet objectOf\ g\ s\ p = \\
\quad \textbf{if}\ \#(objectsOf\ g\ s\ p) \neq 1\ \textbf{then}\ rdf\_Nil\_t\ \textbf{else} \\
\quad (\mu\, o : RDFTerm \mid o \in objectsOf\ g\ s\ p)
\end{array}
$$

collection - the set of predicates that are collections of the given subject in the context of the graph. A predicate represents a collection if it (a) its object is the subject with a `rdf:type` of `rdf:List`, `rdf:Bag`, `rdf:Seq` or `rdf:Alt` or (b) it has an `rdf:first` or `rdf:rest` predicate. Note this is still an approximation of a list because the definition of lists in RDF is surprisingly lax: " RDFS does not require there be only one first element of a list-like structure, or even for a list-like structure to have a first element. "[**?**]

$$
\begin{array}{l}
collection : Graph \rightarrow\!\!\!\!\!+ RDFTerm \rightarrow\!\!\!\!\!+ RDFTerm \\
\hline
\forall g : Graph;\ s, p : RDFTerm \bullet collection\ g\ s = p \Leftrightarrow \\
\quad \#(objectsOf\ g\ s\ p) = 1 \land \\
\quad (\exists\, o : objectsOf\ g\ (objectOf\ g\ s\ p)\ rdf\_type\_t \bullet \\
\qquad o \in \{rdf\_List\_t, rdf\_Bag\_t, rdf\_Seq\_t, rdf\_Alt\_t\}) \lor \\
\quad (\exists\, p : triplesFor\ g\ (objectOf\ g\ s\ p) \bullet \\
\qquad p.o \in \{rdf\_first\_t, rdf\_rest\_t\})
\end{array}
$$

toSeq - return the sequence representing whose subject is the supplied RDF term. If the subject is not declared to be a list type (`rdf:List`, `rdf:Seq`, `rdf:Bag` or `rdf:Alt` return the set of all objects. Otherwise, unwind the list, ignoring extraneous content as "poorly structured"[1]

[1] Strict validation should probably declare the entire subject as invalid, but our goal is to validate UML, not the internal structure of RDF. Besides, anyone *can* say anything anywhere,

In order to return a sequence of the direct targets of the predicate or more than one `rdf:first` and/or `rdf:next` assertion we define a function that takes a set of RDF Terms and returns them as a sequence. Order is not important, which means we cannot compare the results of any expression utilizing this function(!).

$$
\mathit{asSeq} : \mathbb{P}\, \mathit{RDFTerm} \rightarrow \mathrm{seq}\, \mathit{RDFTerm}
$$

Unwind combines a sequence constructed of the targets of all of the `rdf:first` predicates not `rdf:Nil` with the sequence resulting from unwinding all of the non-`rdf:Nil` `rdf:rest` targets.

$$
\begin{array}{l}
\mathit{unwind} : \mathit{Graph} \nrightarrow \mathit{RDFTerm} \nrightarrow \mathrm{seq}\, \mathit{RDFTerm} \\
\mathit{rest} : \mathit{Graph} \nrightarrow \mathrm{seq}\, \mathit{RDFTerm} \nrightarrow \mathrm{seq}\, \mathit{RDFTerm} \\
\hline
\forall\, g : \mathit{Graph};\ s : \mathit{RDFTerm} \bullet \mathit{unwind}\ g\ s = \\
\quad \mathit{asSeq}(\{f : \mathit{objectsOf}\ g\ s\ \mathit{rdf\_first\_t} \mid f \neq \mathit{rdf\_Nil\_t}\}) ^\frown \\
\qquad \mathit{rest}\ g\ (\mathit{asSeq}\ \{\mathit{rst} : \mathit{objectsOf}\ g\ s\ \mathit{rdf\_rest\_t}\}) \\
\forall\, g : \mathit{Graph};\ s : \mathrm{seq}\, \mathit{RDFTerm} \bullet \mathit{rest}\ g\ s = \\
\quad \textbf{if}\ s = \langle\rangle\ \textbf{then}\ \langle\rangle \\
\quad \textbf{else}\ \mathit{unwind}\ g\ (\mathit{head}\ s) ^\frown \mathit{rest}\ g\ (\mathit{tail}\ s)
\end{array}
$$

The actual *toSeq* function:

$$
\begin{array}{l}
\mathit{toSeq} : \mathit{Graph} \nrightarrow \mathit{RDFTerm} \nrightarrow \mathit{RDFTerm} \nrightarrow \mathrm{seq}\, \mathit{RDFTerm} \\
\hline
\forall\, g : \mathit{Graph};\ s, p : \mathit{RDFTerm} \bullet \mathit{toSeq}\ g\ s\ p = \textbf{if}\ \mathit{collection}\ g\ s = p \\
\textbf{then}\ \mathit{unwind}\ g\ s\ \textbf{else}\ \mathit{asSeq}(\mathit{objectsOf}\ g\ s\ p)
\end{array}
$$

isOrdered - determine whether a property is an ordered list.

This test reflects some assumptions about RDF possibly deserving further examination – the presence of the RDF List Collection type (see: 5.1 Container Classes and Properties in RDF Schema[?])[2] is *the* way ordering is represented. We also make some assumptions about lists that, while valid in practice, aren't guaranteed. One example is we assume a graph of the form below can never occur.

```
@prefix s: <http://sample.org/sample/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

s:subj a s:Narwahl;
       foaf:firstName "Jim";
       rdf_first s:n1;
       rdf_rest rdf\_Nil.
```

right?

[2] Also note this section is *not* normative - an indication the community may be moving away from the LISP like representation of RDF collections

or, more formally that:

$$
\begin{array}{|l}
isOrdered : Graph \rightarrow\!\!\!\!\!+ RDFTerm \rightarrow\!\!\!\!\!+ RDFTerm \\
\hline
\forall g : Graph;\ s, p : RDFTerm \bullet isOrdered\ g\ s = p \Leftrightarrow \\
\quad \#(objectsOf\ g\ s\ rdf\_first\_t) > 0 \wedge \\
\quad (\#(objectsOf\ g\ s\ rdf\_rest\_t) > 0 \wedge \\
\qquad (\#(triplesFor\ g\ s) = 2) \vee (\#(triplesFor\ g\ s) = 3 \wedge \\
\qquad objectOf\ g\ s\ rdf\_type\_t = rdf\_Seq\_t))
\end{array}
$$

This isn't strictly true because it states "RDFS does not require that there be only one first element of a list-like structure, or even that a list-like structure have a first element."[**?**] This may be a place where it would be useful to create some RDF "meta-schema" profiles to reduce the amount of optionality.

3 Interpretation of ShEx Schema

3.1 Schema

```
<xs:element name="Schema" type="shex:Schema"/>
<xs:complexType name="Schema">
```

$$ShapeLabel == IRI$$

$$
\begin{array}{|l}
\textit{Schema} \\
\hline
startActions : SemanticActions[0\mathinner{\ldotp\ldotp}1] \\
shape : ShapeLabel \rightarrow\!\!\!\!\!+ Shape \\
start : ShapeLabel[0\mathinner{\ldotp\ldotp}1] \\
\hline
start \subseteq \operatorname{dom} shape \\
\#start > 0 \Rightarrow shape\,(head\ start).virtual = implementable
\end{array}
$$

3.2 Shape

$$
\begin{array}{|l}
\textit{Shape} \\
\hline
ShapeConstraintGroup \\
include : \mathbb{P}\, ShapeRef \\
extra : \mathbb{P}\, IRIRef \\
is\_virtual : Virtual[0\mathinner{\ldotp\ldotp}1] \\
is\_closed : Closed[0\mathinner{\ldotp\ldotp}1] \\
\\
virtual : Virtual \\
closed : Closed \\
\hline
virtual = \textbf{if}\ \#is\_virtual > 0\ \textbf{then}\ head\ is\_virtual\ \textbf{else}\ implementable \\
closed = \textbf{if}\ \#is\_closed > 0\ \textbf{then}\ head\ is\_closed\ \textbf{else}\ open
\end{array}
$$

$$
\begin{aligned}
&\mathit{ShapeConstraintGroupChoice} ::= \\
&\quad \mathit{scg\_tripleConstraint}\langle\!\langle \mathit{TripleConstraint} \rangle\!\rangle \mid \\
&\quad \mathit{scg\_oneOf}\langle\!\langle \mathit{ShapeConstraint} \rangle\!\rangle \mid \\
&\quad \mathit{scg\_someOf}\langle\!\langle \mathit{ShapeConstraint} \rangle\!\rangle \mid \\
&\quad \mathit{scg\_group}\langle\!\langle \mathit{ShapeConstraint} \rangle\!\rangle \mid \\
&\quad \mathit{scg\_include}\langle\!\langle \mathit{ShapeRef} \rangle\!\rangle
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{Virtual} ::= \mathit{abstract} \mid \mathit{implementable} \\
&\mathit{Closed} ::= \mathit{fixed} \mid \mathit{open}
\end{aligned}
$$

ShapeConstraintGroup

$$
\begin{aligned}
&\mathit{entry} : \mathbb{P}\,\mathit{ShapeConstraintGroupChoice} \\
&\mathit{semanticActions} : \mathit{SemanticActions}[0\,..\,1]
\end{aligned}
$$

3.3 TripleConstraint

TripleConstraint

$$
\begin{aligned}
&\mathit{Cardinality} \\
&\mathit{subjectConstrant} : \mathit{ValueClass}[0\,..\,1] \\
&\mathit{predicate} : \mathit{IRI} \\
&\mathit{objectConstraint} : \mathit{ValueClass}[0\,..\,1] \\
&\mathit{annotation} : \mathbb{P}\,\mathit{Annotation} \\
&\mathit{semanticActions} : \mathit{SemanticActions}[0\,..\,1] \\
&\mathit{label} : \mathit{ShapeLabel}[0\,..\,1] \\
\hline
&\#\mathit{subjectConstraint} + \#\mathit{objectConstraint} \leq 1
\end{aligned}
$$

NodeType comes from the triple model

$$
\begin{aligned}
&\mathit{AnnotationChoice} ::= \\
&\quad \mathit{annot\_iriref}\langle\!\langle \mathit{IRI} \rangle\!\rangle \mid \\
&\quad \mathit{annot\_literal}\langle\!\langle \mathit{RDFLiteral} \rangle\!\rangle
\end{aligned}
$$

Annotation

$$
\begin{aligned}
&\mathit{iri} : \mathit{IRI} \\
&\mathit{annot} : \mathit{AnnotationChoice}
\end{aligned}
$$

3.4 Cardinality

```
<xs:attributeGroup name="Cardinality">
```

$$\mathit{UnboundedInt} ::= \mathit{num}\langle\!\langle \mathbb{N} \rangle\!\rangle \mid \mathit{unlimited}$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{Cardinality}} \\
\hline
min\_v : \mathbb{N}[0 \mathinner{\ldotp\ldotp} 1] \\
max\_v : UnboundedInt[0 \mathinner{\ldotp\ldotp} 1] \\
min : \mathbb{N} \\
max : UnboundedInt \\
\hline
min = \mathbf{if}\ \#min\_v > 0\ \mathbf{then}\ head\ min\_v\ \mathbf{else}\ 1 \\
max = \mathbf{if}\ \#max\_v > 0\ \mathbf{then}\ head\ max\_v\ \mathbf{else}\ num\ 1 \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{ShapeConstraint}} \\
\hline
ShapeConstraintGroup \\
Cardinality \\
label : ShapeLabel \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{GroupShapeConstr}} \\
\hline
ref : \mathbb{P}\ ShapeLabel \\
stringFacet : \mathbb{P}\ StringFacet \\
\hline
\end{array}
$$

$$endType ::= open \mid closed$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{Range}} \\
\hline
\mathbb{N} \\
is\_open : endType[0 \mathinner{\ldotp\ldotp} 1] \\
open : endType \\
\hline
open = \mathbf{if}\ \#is\_open > 0\ \mathbf{then}\ head\ is\_open\ \mathbf{else}\ closed \\
\hline
\end{array}
$$

3.5 Facets

$$
\begin{array}{l}
StringFacet ::= \\
\qquad sf\_pattern\langle\langle String\rangle\rangle \mid \\
\qquad sf\_not\langle\langle String\rangle\rangle \mid \\
\qquad sf\_minLength\langle\langle \mathbb{N}\rangle\rangle \mid \\
\qquad sf\_maxLength\langle\langle \mathbb{N}\rangle\rangle \mid \\
\qquad sf\_length\langle\langle \mathbb{N}\rangle\rangle
\end{array}
$$

$$
\begin{array}{l}
NumericFacet ::= \\
\qquad nf\_minValue\langle\langle Range\rangle\rangle \mid \\
\qquad nf\_maxValue\langle\langle Range\rangle\rangle \mid \\
\qquad nf\_totalDigits\langle\langle \mathbb{N}\rangle\rangle \mid \\
\qquad nf\_fractionDigits\langle\langle \mathbb{N}\rangle\rangle
\end{array}
$$

$$
\begin{aligned}
&XSFacet ::= \\
&\qquad xsf\_pattern\langle\langle String\rangle\rangle \mid \\
&\qquad xsf\_not\langle\langle String\rangle\rangle \mid \\
&\qquad xsf\_minLength\langle\langle \mathbb{N}\rangle\rangle \mid \\
&\qquad xsf\_maxLength\langle\langle \mathbb{N}\rangle\rangle \mid \\
&\qquad xsf\_length\langle\langle \mathbb{N}\rangle\rangle \mid \\
&\qquad xsf\_minValue\langle\langle Range\rangle\rangle \mid \\
&\qquad xsf\_maxValue\langle\langle Range\rangle\rangle \mid \\
&\qquad xsf\_totalDigits\langle\langle \mathbb{N}\rangle\rangle \mid \\
&\qquad xsf\_fractionDigits\langle\langle \mathbb{N}\rangle\rangle
\end{aligned}
$$

$$
\begin{aligned}
&ValueClassChoice ::= \\
&\qquad facet\langle\langle XSFacet\rangle\rangle \mid \\
&\qquad groupShapeConstr\langle\langle GroupShapeConstr\rangle\rangle \mid \\
&\qquad valueSet\langle\langle ValueSet\rangle\rangle
\end{aligned}
$$

3.6 ValueClass

ValueClass

$entry : \mathbb{P}_1\ ValueClassChoice$

3.7 ValueSet

ValueSet

3.8 SemanticActions

$$
\begin{aligned}
&[CODE\_LABEL, CODE\_BODY] \\
&CODE\_DECL \mathrel{\widehat=} [iri : IRI[0\,..\,1];\ body : CODE\_BODY] \\
&SemanticAction ::= codeLabel\langle\langle CODE\_LABEL\rangle\rangle \mid \\
&\quad codeDecl\langle\langle CODE\_DECL\rangle\rangle \\
&SemanticActions == \mathbb{P}\ SemanticAction
\end{aligned}
$$

4 Optional elements

Representing optional elements of type T. Representing it as a sequence allows us to determine absence by $\#T = 0$ and the value by $head\ T$.

$$T[0\,..\,1] == \{s : \mathrm{seq}\ T \mid \#s \leq 1\}$$