Negation & recursion validating RDF data

Jose Emilio Labra Gayo

Departamento de Informática
Universidad de Oviedo
Validating RDF Data
Jose E. Labra, E. Prud’hommeaux, I. Boneva, D. Kontokostas
Morgan & Claypool, 2017

http://book.validatingrdf.com/
Validating RDF data

RDF = *lingua franca* of semantic web

Simple Graph model
Query language (SPARQL)
Basis of knowledge representation (RDFS, OWL)
Lots of applications based on RDF

But...

Too much flexibility?
Most RDF applications have latent schemas
Tools to describe and validate those schemas?
RDF data validation history

Early initiatives

2000 – RDF Schema: lacks validation (only inference)
2004 – SPARQL: Queries can be used to validate
2007 – ICV: Modify OWL semantics with CWA
2010 – SPIN (by TopQuadrant): SPARQL templates

2013 – RDF Validation workshop (MIT, Boston)
ShEx (Shape Expressions) proposed

2014 – W3C Data Shapes working group created
Several inputs: ShEx, SPIN, …
Decision: create new language called SHACL (Shapes Constraint Language)
Difficulties to combine recursion, negation and other features

2017 – SHACL accepted as recommendation
Decision: Leave recursion undefined in SHACL
RDF data model

RDF data model is based on triples (statements)

Subject – predicate – object

A set of triples forms an RDF graph

Predicates are IRIs
Subjects can be IRIs or blank nodes
Objects can be IRIs, blank nodes or literals

RDF playground: http://rdfshape.weso.es
ShEx (Shape Expressions)

Compact syntax
Regular expressions declare shapes of nodes

```xml
<Person> IRI {
  :name xsd:string ;
  :age xsd:integer ? ;
  :knows @<Person>* ;
}
```

RDF playground: [http://rdfshape.weso.es](http://rdfshape.weso.es)
Some ShEx features...

Recursion: shapes can refer to themselves
  Cyclic data models
Node constraints (constraints about a node)
  XML Schema facets
Triple constraints (neighbourhood of nodes)
Logical operations on shapes:
  AND, OR and NOT

<Teacher> @<Person> AND {
  :age MinInclusive 18
  :teaches @<Course>+ 
}

<Course> IRI {
  :code xsd:string ;
  ^:teaches @<Teacher>
}

<Student> @<Person> AND {
  :enrolledIn @<Course>+ 
} AND NOT {
  :knows @<Teacher>
}
Recursion and negation in ShEx

ShEx approach [Boneva et al, ISWC17]

Stratified negation

Avoid schemas with negative cyclic dependencies

According to the ShEx spec:

“A schema must not contain any shapeExpr SE which has a shape which contains negated references, either directly or transitively, to SE.”

Note: Not all implementations force that constraint

SHACL (Shapes Constraint Language)

Syntax based on RDF
Shapes = conjunctions of constraints
Common features between ShEx/SHACL
Several differences:
  Validation triggering mechanism
  Repeated properties
  Built-in support for inferencing
  Paths
  ….

```
<Person> IRI {
  :name xsd:string ;
  :age xsd:integer ? ;
  :knows @<Person>* ;
}
```

```
:Person a sh:NodeShape ;
  sh:nodeKind sh:IRI ;
  sh:property [
    sh:path :name ;
    sh:minCount 1;
    sh:maxCount 1;
    sh:datatype xsd:string ;
  ] ;
  sh:property [
    sh:path :age ;
    sh:maxCount 1;
    sh:datatype xsd:integer ;
  ] ;
  sh:property [
    sh:path :knows ;
    sh:node :Person ;
  ] .
```
Recursion and negation in SHACL

Recursion left out of SHACL recommendation

Recursive shapes have no standard semantics

Implementers can choose

Either ignore or use your own semantics

Undesired incompatibilities

Good as a research challenge

Recent work [Corman et al’18] proposes partial assignments

3-valued logics

Current status of ShEx/SHACL

ShEx/SHACL are increasingly being adopted
   Some examples: EU multi-stakeholder, Wikidata, Clinical records,…

Driven by practical applications & use cases
   Although more theoretical work is needed
   Negation/recursion may be defined in SHACL 2.0
   Relating ShEx and SHACL languages

Other challenges
   Inheritance (a shape extends another shape)
   Scalability and RDF streams validation
   Visualization
   ...

Some intuitions...

Recursion/negation has been thoroughly studied in the ASP community

Maybe current ShEx/SHACL proposals could be improved

Can ASP be applied to define ShEx/SHACL semantics?
Can Converting ShEx/SHACL validators to ASP, is it feasible?

I started an exercise validating ShEx/SHACL in ASP (clingo)
More details?
Representing ShEx/SHACL in ASP

Possible ASP encoding of a simple shape

```asp
<User> { 
:name xsd:string {1,1}; 
}
```

```asp
hasShape(X,user) :- node(X), string(X,name,1,1) .
not hasShape(X,user):- node(X), not string(X,name,1,1) .

string(X,P,MIN,MAX):- integer(MIN), integer(MAX),
countStringProperty(X,P,C), C >= MIN, C <= MAX .
not string(X,P,MIN,MAX):- integer(MIN), integer(MAX),
countProperty(X,P,C), countNoStringProperty(X,P,NS),
C - NS < MIN .

countStringProperty(X,P,C):- node(X), property(P),
   C = #count { V: arc(X,P,V), string(V) } .
countNoStringProperty(X,P,C):- node(X), property(P),
   C = #count { V: arc(X,P,V), not string(V) } .
countProperty(X,P,C):- node(X), property(P),
   C = #count { V: arc(X,P,V) } .

hasShape(X,A) | not hasShape(X,A) :- node(X), shape(A) .
```

clingo version 5.3.0
Solving...
Answer: 1
hasShape(alice,user) hasShape(carol,user)

https://goo.gl/sYcLqL
https://goo.gl/54kwZZ
Representing ShEx/SHACL in ASP

Adding recursion and conjunctions

```prolog
<User> { 
  name xsd:string {1,1};
} AND { 
  knows @<User> {1,*};
}

hasShape(X,user) :- string(X,name,1,1),
                 nodeShape(X,knows,user,1,star).

not hasShape(X,user):- node(X), not string(X,name,1,1).
not hasShape(X,user):- node(X),
                      not nodeShape(X,knows,user,1,star).

nodeShape(X,P,S,MIN,MAX):- integer(MIN), integer(MAX), shape(S),
                          countShapeProperty(X,P,S,C),
                          C >= MIN, C <= MAX.
not nodeShape(X,P,S,MIN,MAX):-
                          integer(MIN), integer(MAX), shape(S),
                          countProperty(X,P,C),
                          countNoShapeProperty(X,P,S,NS),
                          C - NS < MIN.

countShapeProperty(X,P,S,C):- node(X), property(P), shape(S),
                              C = #count { V: arc(X,P,V), hasShape(V,S) }.

countNoShapeProperty(X,P,S,C):-
                               node(X), property(P), shape(S),
                               C = #count { V: arc(X,P,V), not hasShape(V,S) }.
```

clingo version 5.3.0
Solving...
Answer: 1
Answer: 2
hasShape(alice,user) hasShape(carol,user)
SATISFIABLE
Models       : 2
Representing ShEx/SHACL in ASP

Adding negation

```prolog
<User> {  
  :name xsd:string {1,1};  
} AND {  
  :knows @<User> {1,*};  
}

<Teacher> {  
  :name xsd:string {1,1};  
} AND NOT {  
  :knows @<User> {1,*};  
}
```

```prolog
hasShape(X,user) :- node(X),  
  string(X,name,1,1),  
  nodeShape(X,knows,user,1,star).

hasShape(X,teacher) :- node(X),  
  string(X,name,1,1),  
  not nodeShape(X,knows,user,1,star).

not hasShape(X,user) :- node(X),  
  not string(X,name,1,1).

not hasShape(X,teacher) :- node(X),  
  not nodeShape(X,knows,user,1,star).

not hasShape(X,teacher) :- node(X),  
  not string(X,name,1,1).

not hasShape(X,teacher) :- node(X),  
  nodeShape(X,knows,user,1,star).
```

clingo version 5.3.0
Reading from userNameKnows.pl
Solving...
Answer: 1
hasShape(bob,teacher) hasShape(emily,teacher) hasShape(alice,teacher)  
hasShape(carol,teacher)
Answer: 2
hasShape(bob,teacher) hasShape(emily,teacher) hasShape(alice,user)  
hasShape(carol,user)  
SATISFIABLE
Models : 2
Language S

Minimal ShEx/SHACL

\[ \phi ::= T \mid \@s \mid \Delta \mid IRI \mid \phi_1 \land \phi_2 \mid \neg \phi \mid \xrightarrow{p} \phi\{min, max\} \]

- \( T \) \quad \text{true}
- \( \@s \) \quad \text{reference to shape } s
- \( \Delta \) \quad \text{node has datatype } \Delta
- \( IRI \) \quad \text{node is an IRI}
- \( \phi_1 \land \phi_2 \) \quad \text{conjunction}
- \( \neg \phi \) \quad \text{negation}
- \( \xrightarrow{p} \phi\{min, max\} \) \quad \text{between } min \text{ and } max \text{ triples with predicate } p \text{ whose values conform to } \phi

\[ S ::= \text{true} \mid \@S \mid \text{datatype} \mid IRI \mid S_1 \land S_2 \mid \neg S \mid p S \{\text{min, max}\} \]
Example encoding ShEx in S

```xml
<User> {  
  :name xsd:string {1,1};
} AND {  
  :knows @<User> {1,*};
}
<Teacher> {  
  :name xsd:string {1,1};
} AND NOT {  
  :knows @<User> {0,*};
}
```

```
user : ¬ name → string{1,1} ∧  
      know → @user{0,*}

teacher : ¬ name → ⊤{1,1} ∧  
         ¬ know → @user{0,*}
```
3-valued semantics for language $S$

The semantics is inspired by [Corman et al, 18] paper
The paper uses partial assignments
Which assign values $s$ or $\neg s$ to nodes during validation

\[
\begin{align*}
\llbracket \top \rrbracket^{n,g,\sigma} &= 2 \\
\llbracket \sigma(s) \rrbracket^{n,g,\sigma} &= \llbracket \sigma(n) \rrbracket^{n,g,\sigma} \\
\llbracket \Delta \rrbracket^{n,g,\sigma} &= \begin{cases} 2 & \text{if } n \in \Delta \\ 0 & \text{otherwise} \end{cases} \\
\llbracket IRI \rrbracket^{n,g,\sigma} &= \begin{cases} 2 & \text{if } n \text{ is an IRI} \\ 0 & \text{otherwise} \end{cases} \\
\llbracket \phi_1 \land \phi_2 \rrbracket^{n,g,\sigma} &= \min(\llbracket \phi_1 \rrbracket^{n,g,\sigma}, \llbracket \phi_2 \rrbracket^{n,g,\sigma}) \\
\llbracket \neg \phi \rrbracket^{n,g,\sigma} &= 2 - \llbracket \phi \rrbracket^{n,g,\sigma} \\
\llbracket \_ \rightarrow \phi\{\min, \max\} \rrbracket^{n,g,\sigma} &= \begin{cases} 2 & \text{if } \min \leq |(n, p, v) \in g \land \llbracket \phi \rrbracket^{v,g,\text{schema}} = 2| \leq \max \\ 0 & \text{if } |(n, p, v) \in g| - |(n, p, v) \in g \land \llbracket \phi \rrbracket^{v,g,\text{schema}} = 0| < \min \\ 1 & \text{otherwise} \end{cases}
\end{align*}
\]
Prototype implementation

Converts language S expressions to Clingo

Available at: http://labra.weso.es/shaclex/

Option --clingoFile

Example generated:
https://github.com/labra/shaclex/blob/master/examples/clingo/simple.pl