IN3060/4060 – Semantic Technologies – Spring 2021

Lecture 13: RDF Validation

Jieying Chen

16th April 2021

Department of Informatics

University of Oslo

Today's Plan

1. What is Validation
2. Validation for RDF
3. Different Approaches to Validation
4. SHACL – the Shapes Constraint Language
5. SHACL systematically

Outline

An XML document

```xml
<?xml version="1.0"?>
<note>
  <to>Thomas</to>
  <from>Jieying</from>
  <heading>Reminder</heading>
  <body>Don't forget to publish mandatory 6!</body>
</note>
```

A "wrong" XML document

```
<?xml version="1.0"?>
<note>
  <from><theboss/></from>
  <subject>Reminder</subject>
  <body>Don't forget to do what I told you!</body>
</note>
```

- No <to> element
- Not text in <from> element
- No <header> element
- unknown <subject> element

Software reading such an XML document will have difficulties!

An XML Schema for notes

```
<?xml version="1.0"?>
<xs:schema xmlns:xs=...>
  <xs:element name="note">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="to" type="xs:string"/>
        <xs:element name="from" type="xs:string"/>
        <xs:element name="heading" type="xs:string"/>
        <xs:element name="body" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

XML Schema Validation

XML Schema Validation takes

- An XML Schema (.XSD) document S
- An XML 'instance document' X

and checks that X conforms to the rules given by S

Another example: Regular expressions

- Some floating point literals: `-12.3`, `+.7E-3`, `12e12`
- *not* floating point literals: `7.5.2020`, `1E2E3`

A regexp describing all admissible floating point literals:

```
[-+]?[0-9]*\.?[0-9]+([eE][-+]?[0-9]+)?
```

Regular Expression matching: finding out whether a string conforms to a regexp

Another example: Database Constraints

```
CREATE TABLE employees (
  id int NOT NULL,
  department int NOT NULL,
  CONSTRAINT emp_pk PRIMARY KEY (id),
  CONSTRAINT emp_dept_fk
     FOREIGN KEY department
     REFERENCES departments
);
```

- *Check* that all employees have an id and department
- *Check* that any two employees have different IDs
- *Check* that the department of any employee occurs in the departments table

Note: only does something if, and when, data is added. OK to have no emps and depts

Outline

RDF *Schema?*

- RDF "Schema":

```
:worksInDepartment rdfs:range :Department
```

- RDF "Database":

```
:martin :worksInDepartment :ifi
:maths a :Department
:physics a :Department
```

What about RDF Schema Validation? `:ifi` not listed as department!

Rule rdfs3 allows us to *infer* that `:ifi a Department`

RDF Schema cannot be used for validation!

In this sense, it is not a schema language like XML schema.

What about OWL?

- Ontology:

  $\text{Person} \sqsubseteq \exists \text{hasFather}.\text{Person}$

- ABox:

  $\text{Person}(\text{haakon})$
  $\text{Person}(\text{harald})$
  $\text{hasFather}(\text{haakon}, \text{harald})$

Does this "validate"? No information about Harald's father!

We can infer that $\exists\text{hasFather}.\text{Person}(\textit{harald})$, i.e. he has a father

OWL cannot be used for validation!

OWL and RDFS are good for *adding* missing fats, not detecting that they are missing

What is needed?

- In applications, often need info about *available* information
- E.g. queries become a lot easier to write if we know the data!
- Ontology: Every person has a name
- Needed: For every person in the dataset, we know the name
- Ontology: Every employee works in some department
- Needed: For every employee, we know which department he/she works in, and it is a department we know about.

Need a Constraint language to describe RDF graphs

- Ontology — describes persons, employees, cars,. . .
- Constraints — describe **data about** persons, employees, cars,. . .

Ontology vs. Constraints

Ontology

- Knowledge about domain
- *Can do:* infer *new* knowledge
- Reuse across applications

Constraints

- Knowledge about our knowledge of the domain
- *Can do:* check completness of existing information: Validation
- Specific to use (one system or exchange)

Outline

OWL as Constraint Language (Stardog ICV)

`https://docs.stardog.com/data-quality-constraints/`

- Idea: Allow some OWL Axioms to be interpreted as constraints
- E.g.: Supervisor $\sqsubseteq$ $\exists$supervises.Employee...
- ...interpreted as constraint means:

  For every triple x `a :Supervisor`

  There must be at least one triple x `:supervises` y

  and a triple y `a :Employee` for some resource y
- Advantages:
  - easy to define mathematically (Take RDF graph as DL interpretation)
  - parsers, APIs, etc. already there
  - "constraints" can be translated to SPARQL queries that check them
- Disadvantages:
  - not everything in OWL has a sensible constraint interpretation
  - not every useful constraint can be expressed in OWL

Epistemic Description Logics

E.g. `https://dl.acm.org/citation.cfm?id=505373`

- “epistemic” logics add a knowledge operator $\mathcal{K}$
- $\mathcal{K}C$ contains things *known* to belong to C; $\mathcal{K}R$ relates things *known* to be related by R
- Every known supervisor is known to supervise someone known to be an Employee

  $\mathcal{K}\text{Supervisor} \sqsubseteq \exists\mathcal{K}\text{supervises}.\mathcal{K}\text{Employee}$
- Every employee is employee in the database:

  $\text{Employee} \sqsubseteq \mathcal{K}\text{Employee}$
- Advantages:
  - Expressive
  - Describes knowledge not triples
- Disadvantages:
  - Mathematical details are hairy... require different knowledge operators...
  - Without restrictions, high computational complexity
  - For applications, describing data may be more important than describing knowledge

Why not simply SPARQL?

https://www.topquadrant.com/technology/sparql-rules-spin/spin-constraints/

- Idea: write SPARQL queries that detect constraint violations
- E.g. Every superviser must supervise some employee:

```
SELECT ?p WHERE {
   ?p a :Supervisor.
   FILTER NOT EXISTS {?p :supervises ?q. ?q a :Employee.}
}
```

- Every query answer is a constraint violation!
- Advantages:
  - Low tech, all required tool support already there
  - Full expressivity of SPARQL
- Disadvantages:
  - Hard to write and read for complex constraints
  - Like OWL, SPARQL is not a language *made for* the purpose

W3C RDF Data Shapes Working Group

- Goal: "produce a language for defining structural constraints on RDF graphs"
- Originally people with many different ideas.
- Eventuallt two main directions:
- Shapes
  - Describe what must be in the graph
  - Similar to XML Schema, regular expressions, grammars
  - Outcome: Shape Expressions (ShEx)
- Constraints
  - Describe which violations to check for
  - Similar to DB constraints
  - Outcome: Shape Constraint Language (SHACL)
- SHACL became W3C recommendation June 2017
- ShEx and SHACL now incorporate many of each others ideas.

Book

- "Validating RDF Data" by Jose Emilio Labra Gayo, Eric Prud'hommeaux, Iovka Boneva, Dimitris Kontokostas
- Complete text of book online: `https://book.validatingrdf.com/`
- By the group behind ShEx
- Covers both ShEx and SHACL
- (source of many of the examples here)

Outline

SHACL Example

SHACL constraints are RDF graphs using the SHACL vocabulary.

```
@prefix sh: <http://www.w3.org/ns/shacl#> .

:UserShape a sh:NodeShape;          # declare a shape :UserShape
  sh:targetClass :User ;            # apply to all resources of type :User
  sh:property [                     # the property...
    sh:path      schema:name ;      # ... schema:name ...
    sh:minCount 1;                  # ... must be given at least once ...
    sh:maxCount 1;                  # ... and at most once ...
    sh:datatype xsd:string ;        # ... and the object must be a string
  ] .
```

- Applies to all resources x of type `:User`
- These must have exactly one triple x `schema:name` y for each x
- y must have datatype `xsd:string` (so it must be a literal)

SHACL Example, continued

```
:UserShape a sh:NodeShape;
  sh:targetClass :User ;
  sh:property [
    sh:path schema:knows ;
    sh:nodeKind sh:IRI ;
    sh:class :User ;
  ] .
```

- There can be 0, 1, or several `schema:knows` triples for a `User`
- But for each, the object has to be a resource y (not a literal)
- And there must be a triple typing y as a `:User`

SHACL Example, continued

```
:UserShape a sh:NodeShape;
  sh:targetClass :User ;
  sh:property [
    sh:path schema:gender ;
    sh:minCount 1;
    sh:maxCount 1;
    sh:or (
      [ sh:in (schema:Male schema:Female) ]
      [ sh:datatype xsd:string]
    )
] .
```

- There must be exaclty one `schema:gender` triple for a `User`
- The object can be `schema:Male` or `schema:Female` or a string.

Putting it together

```
:UserShape a sh:NodeShape;
  sh:targetClass :User ;
  sh:property [ sh:path schema:name ; ...] ;
  sh:property [ sh:path schema:gender ; ...] ;
  sh:property [ sh:path schema:birthDate ; ...] ;
  sh:property [ sh:path schema:knows ; ...] .
```

- UserShape is a “node shape”
- This node shape includes four “property shapes”
- Each property shape adds constraints that are checked individually
- All are checked, conjunction of constraints.

Validation

- Results of validation are given as a “Validation Report” in RDF.
- Everything OK:

```
:report a sh:ValidationReport ; sh:conforms true .
```

- Problems:

```
:report a sh:ValidationReport ;
  sh:conforms false ;
  sh:result [ a sh:ValidationResult ;
              sh:resultSeverity sh:Violation ;
              sh:sourceConstraintComponent sh:DatatypeConstraintComponent ;
              sh:sourceShape ... ;
              sh:focusNode :dave ;
              sh:value 1980 ;
              sh:resultPath schema:birthDate ;
              sh:resultMessage "Value does not have datatype xsd:date" ],...
```

Outline

Node Shapes and Targets

- SHACL constraints apply to “focus nodes”
- A node shape specifies which are the focus nodes it applies to
  - Known as the *targets* of the node shape
- And the constraints that should apply
- Target declarations:

Property	Description
sh:targetNode	Directly point to a node
sh:targetClass	All nodes that are instances of some class
sh:targetSubjectsOf	All nodes that are subjects of some predicate
sh:targetObjectsOf	All nodes that are objects of some predicate

- All selected targets become focus nodes after each other, and are checked for comformance

SHACL Instances

- A node x is a *SHACL instance* of a SHACL class C if x `rdf:type/rdfs:subClassOf*` C.
- I.e. if there are triples

  x `rdf:type` C_0 .

  C_0 `rdfs:subClassOf` C_1 .

  ...

  C_k `rdfs:subClassOf` C .
- `sh:targetClass` uses SHACL instances
- Built-in RDFS-style subclass reasoning
- But nothing else, no range/domain/subproperty reasoning

Implicit Class Target

```
:User a sh:NodeShape, rdfs:Class ;
  sh:property [
     sh:path schema:name ;
     sh:minCount 1;
     sh:maxCount 1;
     sh:datatype xsd:string ;
  ] .
```

- `:User` is an `rdfs:Class`
- but also a `sh:NodeShape`
- with the *implicit* `sh:targetClass :User`
- Confusing, but sometimes convenient

Constraint Components for Node Shapes

`:UserShape a sh:NodeShape ;`

- sh:nodeKind sh:IRI — node must be resource (not literal or blank node)
    - Other node kinds: `sh:BlankNode`, `sh:Literal`, `sh:BlankNodeOrIRI`, `sh:BlankNodeOrLiteral`, `sh:IRIOrLiteral`
- sh:class :Person — has to be SHACL instance of some type
- sh:datatype xsd:int — has to be literal with given datatype
- sh:hasValue :Norway — has to be a specific value (IRI or literal)
- sh:in (:Cat :Dog) — has to be one of the given values (IRIs or literals)
- sh:minInclusive 1 ; sh:maxInclusive 5 — range of admitted values
- sh:minLength 4; sh:maxLength 20 — range of admitted string lengths
- sh:pattern "^a(bc)*d" — string must match regexp
- . . . and a few more. . .

Logical Constraint Components

Constraints can be combined:

`:aShape a sh:NodeShape;`

- `sh:and` $(S_1 \ldots S_k)$ — must conform to all shapes
- `sh:or` $(S_1 \ldots S_k)$ — must conform to at least one of the shapes
- `sh:not` S — must not conform to S
- `sh:xone` $(S_1 \ldots S_k)$ — must conform to exactly one of the shapes

Property Shapes

- Given a focus node. . .
- . . . a property shape constrains nodes that can be reached via some path.
- Paths can be just properties, or something similar to SPARQL property paths

SHACL path	**SPARQL path**
`schema:name`	`schema:name`
`[sh:inversePath schema:knows]`	`^schema:knows`
`(schema:knows schema:name)`	`schema:knows/schema:name`
`[sh:alternativePath (schema:knows schema:follows)]`	`schema:knows|schema:follows`
`[sh:zeroOrOnePath schema:knows]`	`schema:knows?`
`[sh:oneOrMorePath schema:knows]`	`schema:knows+`
`([sh:zeroOrMorePath schema:knows] schema:name)`	`schema:knows*/schema:name`

Cardinality Constraint Components

- Given a property shape

  `... sh:property [ sh:path` p `; ...]`

- And a focus node x
- Gather the set of all *value nodes* $v \in V$, that can be reached from x by p.
- `... sh:property [ sh:path` p `; sh:minCount 3 ... ]` — check that $|V| \geq 3$
- `... sh:property [ sh:path` p `; sh:maxCount 5 ... ]` — check that $|V| \leq 5$
- What about: `[ sh:path` p `; sh:maxCount 5; sh:datatype xsd:int ... ]` ?
  - There must be at most 5 value nodes
  - *All* of them must have type `xsd:int`
- "Max 5 of `xsd:int` but possibly others" $\rightarrow$ Qualified Value Constraints

Diverse Constraints

- `sh:name` — human readable label
- `sh:description` — human readable description
- `sh:message` — human readable message for validation report
- `sh:severity` — `sh:Info`, `sh:Warning`, or `sh:Violation`

Property Shape Example

Users have to know someone who has an email address, which matches a regexp

```
ex:UsersKnowSomeoneWithMailShape
    a sh:NodeShape ;
    sh:targetClass :User ;
    sh:property [
       sh:path (ex:knows ex:email) ;
       sh:name "Friend's e-mail" ;
       sh:description "We need at least one email for everyone you know" ;
       sh:minCount 1 ;
       sh:pattern "^[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,4}$" ;
    ] .
```

Property Pair Constraints – sh:equals

The set of Bob's `foaf:givenName` values is the same as that of `foaf:firstName`

```
ex:EqualExampleShap a sh:NodeShape ;
    sh:targetNode ex:Bob ;
    sh:property [
            sh:path ex:firstName ;
            sh:equals ex:givenName ;
    ] .
```

The country a city lies in is the same as the country of the district it lies in

```
:CityShape a sh:NodeShape;
sh:targetClass :City;
sh:property [
  sh:path (:isCityInDistrict :isDistrictInCountry) ;
  sh:equals :isCityInCountry ;
] .
```

Property Pair Constraints – sh:disjoint

None of of Bob's ancestors is also one of his children

```
ex:DisjointExampleShape
    a sh:NodeShape ;
    sh:targetNode ex:Bob ;
    sh:property [
            sh:path [ sh:zeroOrMorePath ex:hasParent ] ;
            sh:disjoint ex:hasChild ;
    ] .
```

Note how transitive closure using `sh:zeroOrMorePath` reaches all ancestors.

Property Pair Constraints – Value Comparison

Every screening in the dataset starts before it ends.

```
ex:DisjointExampleShape
    a sh:ScreeningShape ;
    sh:property [
            sh:path movie:screeningStart ] ;
            sh:lessThan ex:screeningEnd ;
    ] .
```

Can also use `sh:lessThanOrEquals`

References

Require that the address of a person has the address shape

```
ex:PersonShape
    a sh:NodeShape ;
    sh:targetClass ex:Person ;
    sh:property [
            sh:path ex:address ;
            sh:minCount 1 ;
            sh:node ex:AddressShape ;
    ] .
```

- Note: cyclic references are not supported by the standard.
- E.g. AddressShape can't refer back to PersonShape, has to go via `sh:class`
- Often stated as advantage of ShEx

SHACL-SPARQL

SHACL-SPARQL: express restrictions based on a SPARQL SELECT query.

```
ex:LanguageExampleShape
  a sh:NodeShape ;
  sh:targetClass ex:Country ;
  sh:sparql [
    sh:message "Values are literals with German language tag." ;
    sh:prefixes ex: ;
    sh:select """
      SELECT $this (ex:germanLabel AS ?path) ?value
      WHERE {
        $this ex:germanLabel ?value .
        FILTER (!isLiteral(?value) || !langMatches(lang(?value), "de"))
      }
      """ ;
  ] .
```

Takeaways

- Ontologies are no good for validation
  - Ontologies express facts about the domain
  - Constraints, data models, etc., express facts about the data
- Several different approaches have been explored
- One of them, SHACL, has become a W3C recommendation
- Built around constraints that must be checked

Outlook

Lecture 14: Guest Lecture

- Christian M. Hansen, Ontology Specialist at Aibel
- Dirk Walther, Principal Consultant at DNV

Lecture 15: OTTR Templates: Basics

Lecture 16: OTTR Templates: Template libraries and practical applications (Oblig 7)

Lecture 17: Open Data

Lecture 18: Repetition