

SHACL by example

RDF Validation tutorial

Jose Emilio Labra Gayo

WESO Research group
University of Oviedo, Spain

Eric Prud'hommeaux

World Wide Web Consortium
MIT, Cambridge, MA, USA

Harold Solbrig

Mayo Clinic, USA

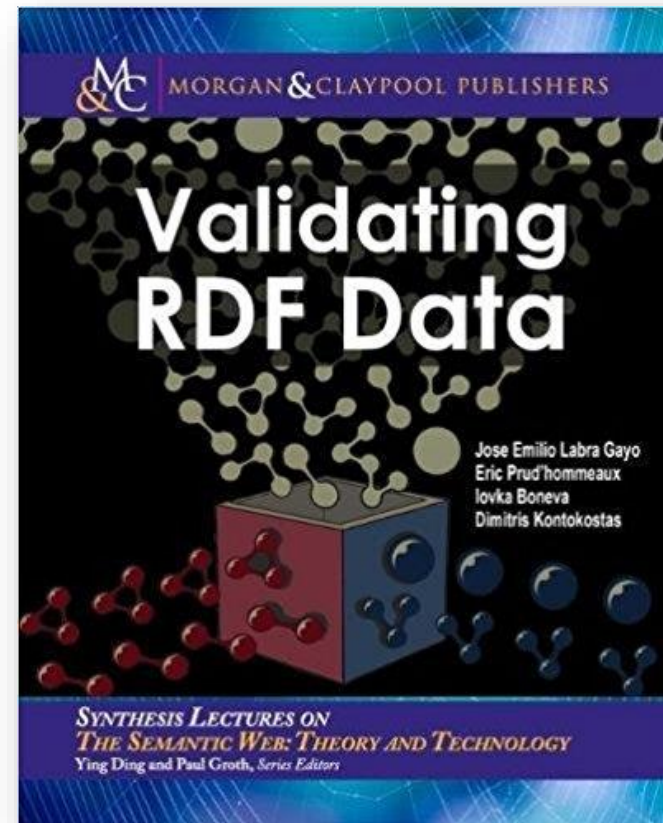
Iovka Boneva

LINKS, INRIA & CNRS
University of Lille, France

More info

Chapter 5 of Validating RDF Data book

[Online HTML version](#)



SHACL

W3C recommendation: <https://www.w3.org/TR/shacl/> (July 2017)

Inspired by SPIN, OSLC & bits of ShEx

2 parts: SHACL-Core, SHACL-SPARQL

RDF vocabulary

SHACL implementations

Name	Parts	Language - Library	Comments
Topbraid SHACL API	SHACL Core, SPARQL	Java (Jena)	Used by TopBraid composer
SHACL playground	SHACL Core	Javascript (rdflib.js)	http://shacl.org/playground/
SHACL-S Part of SHaclEX	SHACL Core	Scala (Jena, RDF4j)	http://rdfshape.weso.es
pySHACL	SHACL Core, SPARQL	Python (rdflib)	https://github.com/RDFLib/pySHACL
Corese SHACL	SHACL Core, SPARQL	Java (STTL)	http://wimmics.inria.fr/corese
RDFUnit	SHACL Core, SPARQL	Java (Jena)	https://github.com/AKSW/RDFUnit
Jena SHACL	SHACL Core, SPARQL	Java (Jena)	https://jena.apache.org/
RDF4j SHACL	SHACL Core	Java (RDF4J)	https://rdf4j.org
Stardog	SHACL Core, SPARQL	Java	https://www.stardog.com
Zazuko SHACL	SHACL Core	Javascript	https://github.com/zazuko/rdf-validate-shacl

In this tutorial we will use RDFShape online demo which supports:

- SHaclEX (SHACL-s)
- JenaSHACL
- SHACL TQ (SHACL TopBraid API)

Basic example

```
prefix :      <http://example.org/>
prefix sh:    <http://www.w3.org/ns/shacl#>
prefix xsd:   <http://www.w3.org/2001/XMLSchema#>
prefix schema: <http://schema.org/>
```

```
:UserShape a sh:NodeShape ;
  sh:targetNode :alice, :bob, :carol ;
  sh:nodeKind sh:IRI ;
  sh:property :hasName,
             :hasEmail .
:hasName sh:path schema:name ;
  sh:minCount 1;
  sh:maxCount 1;
  sh:datatype xsd:string .
:hasEmail sh:path schema:email ;
  sh:minCount 1;
  sh:maxCount 1;
  sh:nodeKind sh:IRI .
```

Shapes graph

S :UserShape
schema:name : xsd:string
schema:email : IRI
IRI

```
:alice schema:name "Alice Cooper" ;
       schema:email <mailto:alice@mail.org> .

:bob   schema:firstName "Bob" ;
       schema:email <mailto:bob@mail.org> . ☹️

:carol schema:name "Carol" ;
       schema:email "carol@mail.org" . ☹️
```

Data graph

Try it. RDFShape <https://tinyurl.com/y46b2f8q>

Same example with blank nodes

```
prefix :      <http://example.org/>
prefix sh:    <http://www.w3.org/ns/shacl#>
prefix xsd:   <http://www.w3.org/2001/XMLSchema#>
prefix schema: <http://schema.org/>
```

```
:UserShape a sh:NodeShape ;
  sh:targetNode :alice, :bob, :carol ;
  sh:nodeKind sh:IRI ;
  sh:property [
    sh:path      schema:name ;
    sh:minCount  1; sh:maxCount 1;
    sh:datatype  xsd:string ;
  ] ;
  sh:property [
    sh:path      schema:email ;
    sh:minCount  1; sh:maxCount 1;
    sh:nodeKind  sh:IRI ;
  ] .
```

```
:alice schema:name "Alice Cooper" ;
       schema:email <mailto:alice@mail.org> .

:bob   schema:firstName "Bob" ;
       schema:email <mailto:bob@mail.org> . ☹️

:carol schema:name "Carol" ;
       schema:email "carol@mail.org" . ☹️
```

Data graph

Shapes graph

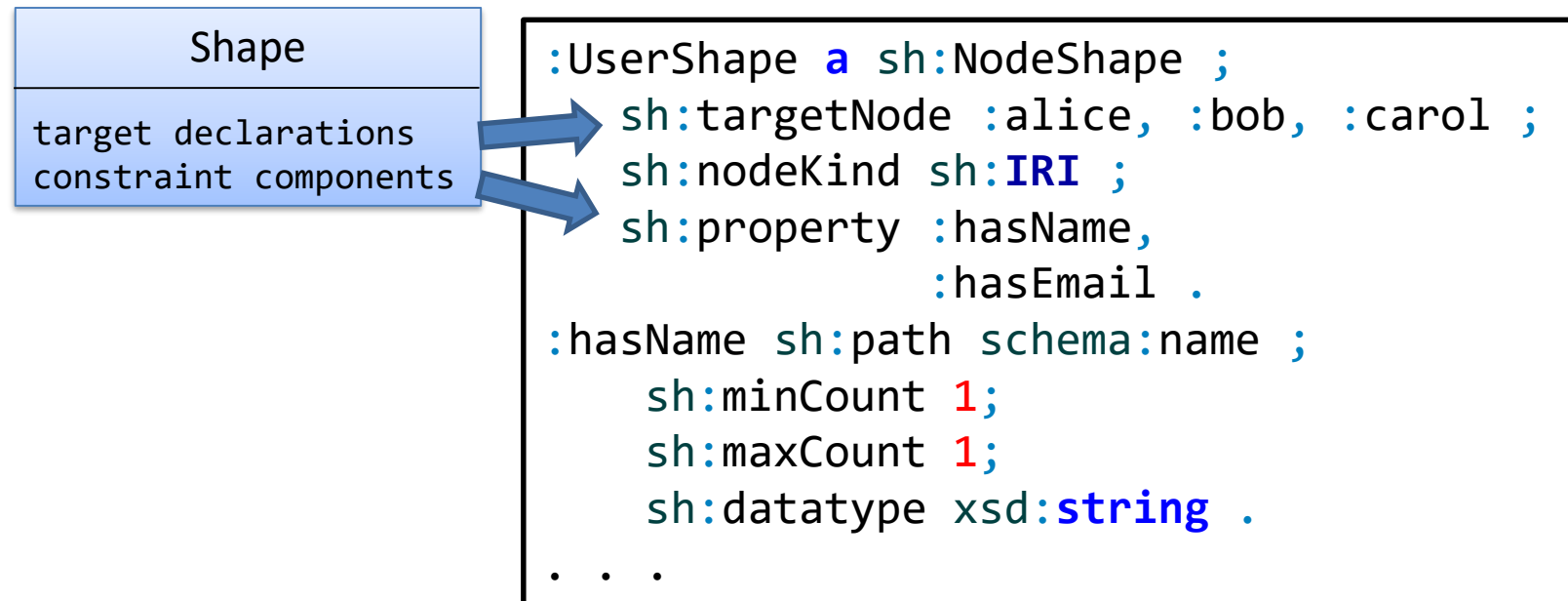
Try it. RDFShape <https://tinyurl.com/y4ycv2vn>

Some definitions about SHACL

Shape: collection of targets and constraints components

Targets: specify which nodes in the data graph must conform to a shape

Constraint components: Determine how to validate a node



Shapes graph and data graph

Conceptually: 2 graphs

Shapes graph: an RDF graph that contains shapes

Data graph: an RDF graph that contains data to be validated

Note: They could be the same

```
:UserShape a sh:NodeShape ;
  sh:targetNode :alice, :bob, :carol ;
  sh:nodeKind sh:IRI ;
  sh:property :hasName,
              :hasEmail .
:hasName sh:path schema:name ;
  sh:minCount 1;
  sh:maxCount 1;
  sh:datatype xsd:string .
. . .
```

Shapes graph

```
:alice schema:name "Alice Cooper" ;
        schema:email <mailto:alice@mail.org> .

:bob    schema:firstName "Bob" ;
        schema:email <mailto:bob@mail.org> .

:carol  schema:name "Carol" ;
        schema:email "carol@mail.org" .
```

Data graph

Validation Report

The output of the validation process is a list of violation errors

No errors \Rightarrow RDF conforms to shapes graph

```
[ a          sh:ValidationReport ;  
  sh:conforms true  
].
```

```
[ a          sh:ValidationReport ;  
  sh:conforms false ;  
  sh:result  [  
    a          sh:ValidationResult ;  
    sh:focusNode :bob ;  
    sh:message  
      "MinCount violation. Expected 1, obtained: 0" ;  
    sh:resultPath schema:name ;  
    sh:resultSeverity sh:Violation ;  
    sh:sourceConstraintComponent  
      sh:MinCountConstraintComponent ;  
    sh:sourceShape :hasName  
  ] ;  
  ...
```

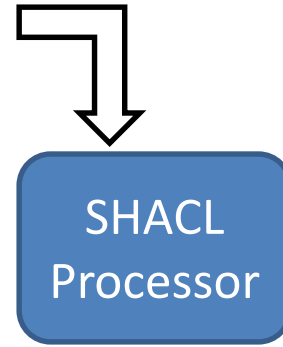
SHACL processor

Shapes graph
with target
declarations

```
:UserShape a sh:NodeShape ;  
  sh:targetNode :alice, :bob, :carol ;  
  sh:nodeKind sh:IRI ;  
  sh:property :hasName,  
             :hasEmail .  
:hasName sh:path schema:name ;  
  sh:minCount 1;  
  sh:maxCount 1;  
  sh:datatype xsd:string .  
. . .
```

Data
Graph

```
:alice schema:name "Alice Cooper" ;  
       schema:email <mailto:alice@mail.org> .  
  
:bob   schema:name "Bob" ;  
       schema:email <mailto:bob@mail.org> .  
  
:carol schema:name "Carol" ;  
       schema:email <mailto:carol@mail.org> .
```



Validation report

```
[ a sh:ValidationReport ;  
  sh:conforms true  
].
```

Importing shapes graphs

SHACL processors follow `owl:imports` declarations

It extends the current shapes graph with the imported shapes

```
:UserShape a sh:NodeShape ;
  sh:targetNode :alice, :bob, :carol ;
  sh:nodeKind sh:IRI ;
  sh:property :hasName .
:hasName sh:path schema:name ;
  sh:minCount 1;
  sh:maxCount 1;
  sh:datatype xsd:string .
```

```
<> owl:imports <http://example.org/UserShapes> .

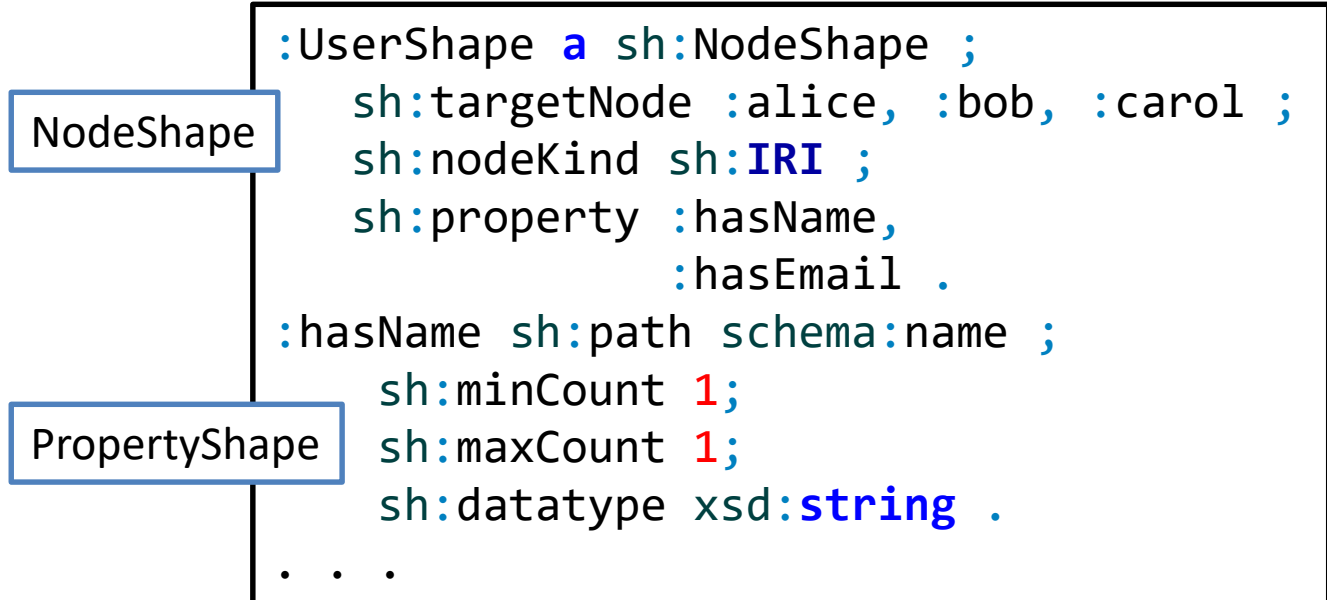
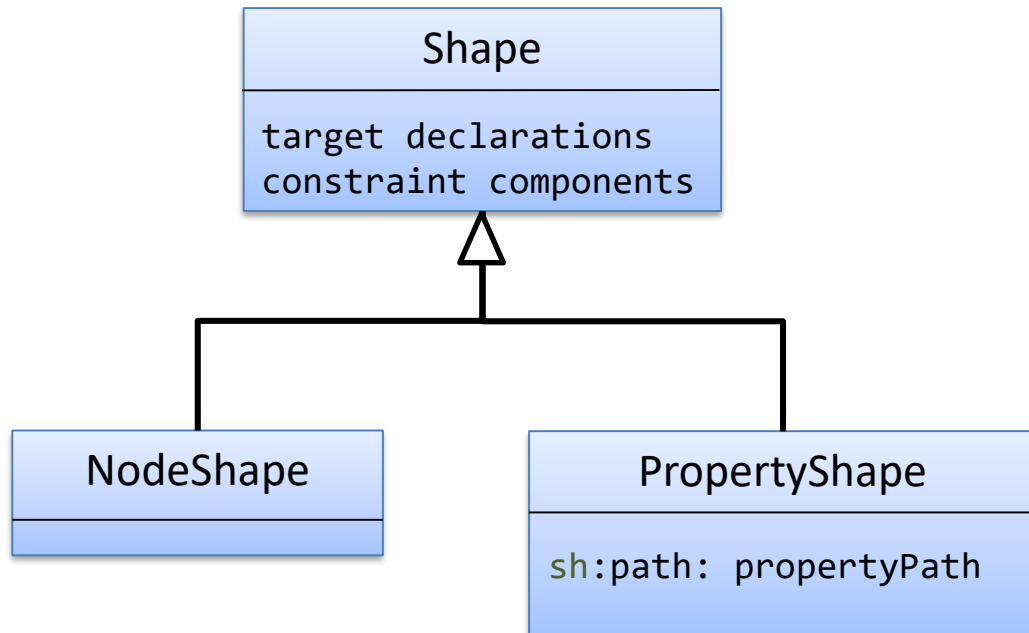
:TeacherShape a sh:NodeShape;
  sh:targetClass :Teacher ;
  sh:node :UserShape ;
  sh:property [
    sh:path :teaches ;
    sh:minCount 1;
    sh:datatype xsd:string;
  ] .
```

Node and property shapes

2 types of shapes:

NodeShape: constraints about shapes of nodes

PropertyShapes: constraints on property path values of a node



Node Shapes

Constraints about a focus node

```
:UserShape a sh:NodeShape ;  
  sh:nodeKind      sh:IRI ;  
  sh:targetClass   :User .
```

```
:alice a :User .  
<http://example.org/bob> a :User .  
_:1 a :User . ☹️
```

Try it: <https://tinyurl.com/y6qyqo5g>

Property shapes

Constraints about a given property and its values for the focus node

`sh:property` associates a shape with a property shape

`sh:path` identifies the path

```
:User a sh:NodeShape ;  
  sh:property [  
    sh:path      schema:email ;  
    sh:nodeKind  sh:IRI  
  ] .
```

```
:alice a :User ;  
       schema:email <mailto:alice@mail.org> .  
  
:bob   a :User;  
       schema:email <mailto:bob@mail.org> . ☹️  
  
:carol a :User;  
       schema:email "carol@mail.org" . ☹️
```

Paths in property shapes

Subset of SPARQL property paths using the following names:

inversePath

alternativePath

zeroOrMorePath

oneOrMorePath

zeroOrOnePath


```
:User a sh:NodeShape, rdfs:Class ;
  sh:property [
    sh:path [sh:inversePath schema:follows ];
    sh:nodeKind sh:IRI ;
  ] .
```

```
:alice a :User;
        schema:follows :bob .

:bob    a :User .

:carol  a :User;
        schema:follows :alice .

_:1 schema:follows :bob .
```



Constraint components

Nodes that declare constraints associated with shapes

They have parameters whose values specify the constraints

SHACL-core provides a list of predefined constraint components

Most of them have one parameter which identifies them

Convention:
 Parameter: `sh:xx`
 C. Component: `sh:xxConstraintComponent`

```
:UserShape a          sh:NodeShape ;
           sh:nodeKind sh:IRI .
```

NOTE: Custom constraint components can be defined in SHACL-SPARQL

Constraint component
`sh:nodeKindConstraintComponent`

Parameter
`sh:nodeKind`

Value of Parameter
`sh:IRI ;`

Repeated parameter

Each value of the parameter declares a different constraint

```
:UserShape a sh:NodeShape;  
sh:class foaf:Person ;  
sh:class schema:Person .
```

```
:alice a schema:Person, foaf:Person .  
:bob a schema:Person .
```



SHACL Core constraint components

Type	Constraints
Cardinality	minCount, maxCount
Types of values	class, datatype, nodeKind
Values	node, in, hasValue, property
Range of values	minInclusive, maxInclusive minExclusive, maxExclusive
String based	minLength, maxLength, pattern
Language based	languageIn, uniqueLang
Logical constraints	not, and, or, xone
Closed shapes	closed, ignoredProperties
Property pair constraints	equals, disjoint, lessThan, lessThanOrEquals
Non-validating constraints	name, description, order, group
Qualified shapes	qualifiedValueShape, qualifiedValueShapesDisjoint qualifiedMinCount, qualifiedMaxCount


See later



Human friendly messages

Message declares the message that will appear in the validation report in case of violation

```
:UserShape a sh:NodeShape ;  
  sh:targetClass :User ;  
  sh:property [  
    sh:path      schema:name ;  
    sh:minCount  1 ;  
    sh:message   "Where is the name?"  
  ] .
```



```
:bob a :User ;  
  schema:alias "Bob" . ☹️
```




```
:report a :ValidationReport ;  
sh:conforms false ;  
sh:result [ a sh:ValidationResult ;  
  sh:resultSeverity      sh:Violation ;  
  sh:sourceConstraintComponent sh:MinCountConstraintComponent ;  
  sh:sourceShape         ... ;  
  sh:focusNode           :bob ;  
  sh:resultPath          schema:name ;  
  sh:resultMessage       "Where is the name?" ;  
].
```

Severities


Declare the level of the violation

3 predefined levels: Violation (default), Warning, Info

```
:UserShape a sh:NodeShape ;  
sh:targetClass :User ;  
sh:property [  
  sh:path      schema:name ;  
  sh:datatype  xsd:string ;  
  sh:severity  sh:Warning  
] .
```



```
:bob a :User ;  
schema:alias "Bob" .
```



```
:report a :ValidationReport ;  
sh:conforms false ;  
sh:result [ a sh:ValidationResult ;  
  sh:resultSeverity      sh:Warning ;  
  sh:sourceConstraintComponent sh:MinCountConstraintComponent ;  
  sh:sourceShape         ... ;  
  sh:focusNode           :bob ;  
  sh:resultPath          schema:name ;  
  sh:resultMessage       "MinCount Error" ;  
] .
```

Deactivating shapes

Deactivate a shape

Useful when importing shapes

UserShapes

```
:UserShape a sh:NodeShape;  
  sh:targetClass :User ;  
  sh:property :HasName ;  
  sh:property :HasEmail .  
  
:HasName sh:path schema:name ;  
  sh:datatype xsd:string .  
  
:HasEmail sh:path schema:email ;  
  sh:minCount 1;  
  sh:nodeKind sh:IRI .
```

```
<> owl:imports <UserShapes> .  
  
:TeacherShape a sh:NodeShape;  
  sh:targetClass :Teacher ;  
  sh:node :UserShape ;  
  sh:property [ sh:path :teaches ;  
    sh:minCount 1;  
    sh:datatype xsd:string;  
  ] .  
  
:HasEmail sh:deactivated true .
```

Target declarations

Targets specify nodes that must be validated against the shape
Several types

Value	Description
targetNode	Directly point to a node
targetClass	All nodes that have a given type
targetSubjectsOf	All nodes that are subjects of some predicate
targetObjectsOf	All nodes that are objects of some predicate

Target node

Directly declare which nodes must validate the against the shape

```
:UserShape a sh:NodeShape ;
  sh:targetNode :alice, :bob, :carol ;
  sh:property [
    sh:path schema:name ;
    sh:minCount 1;
    sh:maxCount 1;
    sh:datatype xsd:string ;
  ] ;
  sh:property [
    sh:path schema:email ;
    sh:minCount 1;
    sh:maxCount 1;
    sh:nodeKind sh:IRI ;
  ] .
```

```
:alice schema:name "Alice Cooper" ;
      schema:email <mailto:alice@mail.org> .

:bob  schema:givenName "Bob" ;
      schema:email <mailto:bob@mail.org> .

:carol schema:name "Carol" ;
       schema:email "carol@mail.org" .
```

Target class

Selects all nodes that have a given class

Looks for `rdf:type` declarations*

```
:UserShape a sh:NodeShape ;
sh:targetClass :User ;
sh:property [
  sh:path schema:name ;
  sh:minCount 1;
  sh:maxCount 1;
  sh:datatype xsd:string ;
] ;
sh:property [
  sh:path schema:email ;
  sh:minCount 1;
  sh:maxCount 1;
  sh:nodeKind sh:IRI ;
] .
```

```
:alice a :User;
       schema:name "Alice Cooper" ;
       schema:email <mailto:alice@mail.org> .

:bob   a :User;
       schema:givenName "Bob" ;
       schema:email <mailto:bob@mail.org> .

:carol a :User;
       schema:name "Carol" ;
       schema:email "carol@mail.org" .
```

* Also looks for `rdfs:subClassOf*/rdf:type` declarations

Implicit class target

A shape with type `sh:Shape` and `rdfs:Class` is a scope class of itself
The `targetClass` declaration is implicit

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

```
:alice a :User;  
  schema:name "Alice Cooper" ;  
  schema:email <mailto:alice@mail.org> .  
  
:bob a :User;  
  schema:givenName "Bob" ;  
  schema:email <mailto:bob@mail.org> .  
  
:carol a :User;  
  schema:name "Carol" ;  
  schema:email "carol@mail.org" .
```

targetSubjectsOf

```
:UserShape a sh:NodeShape;  
sh:targetSubjectsOf :teaches ;  
sh:property [  
  sh:path schema:name ;  
  sh:minCount 1;  
  sh:maxCount 1;  
  sh:datatype xsd:string ;  
] .
```

```
:alice :teaches :Algebra ;      #Passes as :UserShape  
      schema:name "Alice" .  
  
:bob   :teaches :Logic ;       #Fails as :UserShape  
      foaf:name "Robert" .  
  
:carol foaf:name 23 .          # Ignored
```

targetObjectsOf

```
:UserShape a          sh:NodeShape;  
sh:targetObjectsOf :isTaughtBy ;  
sh:property [  
  sh:path          schema:name ;  
  sh:minCount      1;  
  sh:maxCount      1;  
  sh:datatype      xsd:string ;  
] .
```

```
:alice schema:name "Alice" . #Passes as :UserShape  
:bob   foaf:name "Robert" . #Fails as :UserShape  
:carol foaf:name 23 .        # Ignored  
:algebra :isTaughtBy :alice, :bob .
```

Core constraint components

Type	Constraints
Cardinality	minCount, maxCount
Types of values	datatype, class, nodeKind
Values	node, in, hasValue
Range of values	minInclusive, maxInclusive minExclusive, maxExclusive
String based	minLength, maxLength, pattern, stem, uniqueLang
Logical constraints	not, and, or, xone
Closed shapes	closed, ignoredProperties
Property pair constraints	equals, disjoint, lessThan, lessThanOrEquals
Non-validating constraints	name, value, defaultValue
Qualified shapes	qualifiedValueShape, qualifiedMinCount, qualifiedMaxCount

Cardinality constraints

Constraint	Description
minCount	Restricts minimum number of triples involving the focus node and a given predicate. Default value: 0
maxCount	Restricts maximum number of triples involving the focus node and a given predicate. If not defined = unbounded

```

:User a sh:NodeShape ;
  sh:property [
    sh:path      schema:follows ;
    sh:minCount  2 ;
    sh:maxCount  3 ;
  ] .

```

```

:alice schema:follows :bob,
                               :carol .

:bob   schema:follows :alice . ☹️

:carol schema:follows :alice,
                               :bob,
                               :carol,
                               :dave . ☹️

```

Datatypes of values

Constraint	Description
datatype	Restrict the datatype of all value nodes to a given value

```
:User a sh:NodeShape ;
  sh:property [
    sh:path      schema:birthDate ;
    sh:datatype  xsd:date ;
  ] .
```

```
:alice schema:birthDate "1985-08-20"^^xsd:date .
:bob   schema:birthDate "Unknown"^^xsd:date .
:carol schema:birthDate 1990 .
```



Class of values

Constraint	Description
class	Verify that each node in an instance of some class It also allows instances of subclasses*

(*) The notion of SHACL instance is different from RDFS
It is defined as `rdfs:subClassOf*/rdf:type`

```
:User a sh:NodeShape, rdfs:Class ;
  sh:property [
    sh:path schema:follows ;
    sh:class :User
  ] .
```

```
:Manager rdfs:subClassOf :User .

:alice a :User;
  schema:follows :bob .
:bob a :Manager ;
  schema:follows :alice .
:carol a :User;
  schema:follows :alice, :dave . ☹️
:dave a :Employee .
```

Kind of values

Constraint	Description
nodeKind	Possible values: BlankNode, IRI, Literal, BlankNodeOrIRI, BlankNodeOrLiteral, IRIOrLiteral

```

:User a sh:NodeShape, rdfs:Class ;
  sh:property [
    sh:path      schema:name ;
    sh:nodeKind  sh:Literal ;
  ];
sh:property [
  sh:path      schema:follows ;
  sh:nodeKind  sh:BlankNodeOrIRI
];
sh:nodeKind  sh:IRI .

```

```

:alice a :User;
  schema:name      _:1 ;
  schema:follows  :bob .

```

☹️

```

:bob  a :User;
  schema:name      "Robert";
  schema:follows  [ schema:name "Dave" ] .

```

☹️

```

:carol a :User;
  schema:name      "Carol" ;
  schema:follows  "Dave" .

```

☹️

```

_:1 a :User .

```

☹️

Constraints on values

Constraint	Description
hasValue	Verifies that the focus node has a given value
in	Enumerates the value nodes that a property may have

```

:User a sh:NodeShape, rdfs:Class ;
  sh:property [
    sh:path      schema:affiliation ;
    sh:hasValue :OurCompany ;
  ];
sh:property [
  sh:path schema:gender ;
  sh:in   (schema:Male schema:Female)
] .

```

```

:alice a :User;
       schema:affiliation :OurCompany ;
       schema:gender schema:Female .

:bob   a :User;
       schema:affiliation :AnotherCompany ; ☹️
       schema:gender schema:Male .

:carol a :User;
       schema:affiliation :OurCompany ;
       schema:gender schema:Unknown . ☹️

```

Constraints on values with another shape

Constraint	Description
node	All values of a given property must have a given shape Recursion is not allowed in current SHACL

```

:User a sh:NodeShape, rdfs:Class ;
  sh:property [
    sh:path schema:worksFor ;
    sh:node :Company ;
  ] .

```

```

:Company a sh:Shape ;
  sh:property [
    sh:path schema:name ;
    sh:datatype xsd:string ;
  ] .

```

```

:alice a :User;
       schema:worksFor :OurCompany .

```

```

:bob   a :User;
       schema:worksFor :Another .

```



```

:OurCompany
  schema:name "OurCompany" .

```

```

:Another
  schema:name 23 .

```

Value shapes and recursion

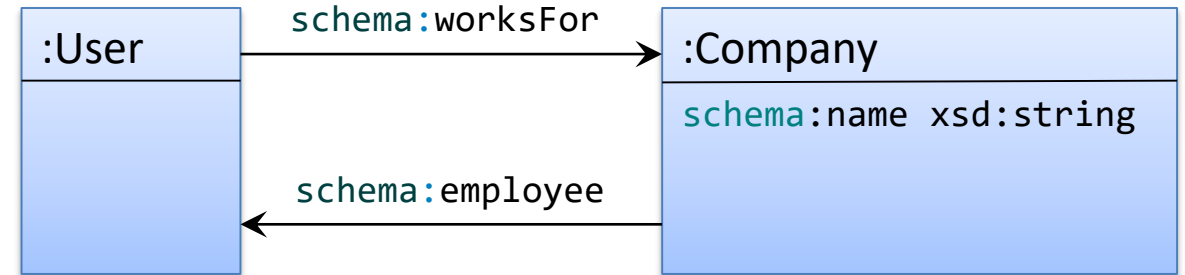
Can we define cyclic data models as the following?

```

:User a sh:NodeShape ;
  sh:property [
    sh:path schema:worksFor ;
    sh:node :Company ;
  ] .

:Company a sh:Shape ;
  sh:property [
    sh:path schema:name ;
    sh:datatype xsd:string ;
  ] ;
  sh:property [
    sh:path schema:employee ;
    sh:node :User ;
  ] .

```



```

:alice schema:worksFor :OneCompany .
:bob schema:worksFor :OneCompany .
:carol schema:worksFor :OneCompany .

:OneCompany schema:name "One" ;
  schema:employee :alice, :bob, :carol .

```

No, current SHACL specification doesn't allow this

Depends on Implementation ☹️

Try it: <https://tinyurl.com/y3hkka6s>

An approach to avoid recursion

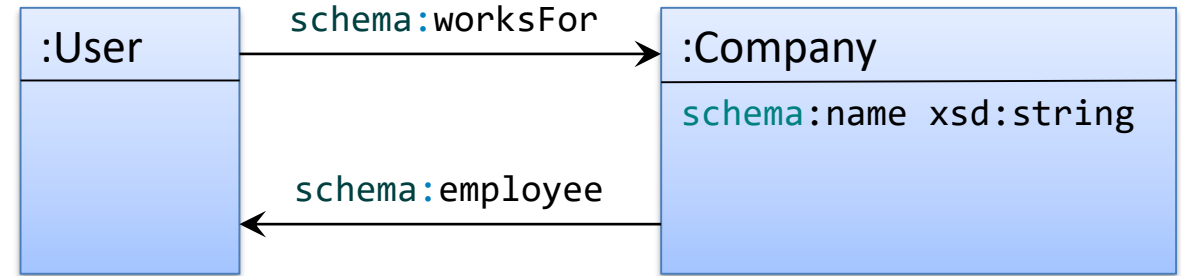
Add `rdf:type` arcs for every resource and use `sh:class`

```

:User a sh:NodeShape ;
  sh:property [
    sh:path schema:worksFor ;
    sh:class :Company ;
  ] .

:Company a sh:Shape ;
  sh:property [
    sh:path schema:name ;
    sh:datatype xsd:string ;
  ] ;
  sh:property [
    sh:path schema:employee ;
    sh:class :User ;
  ] .

```



```

:alice a :User ;
      schema:worksFor :OneCompany .
:bob a :User ;
     schema:worksFor :OneCompany .
:carol a :User ;
       schema:worksFor :Something . ☹️

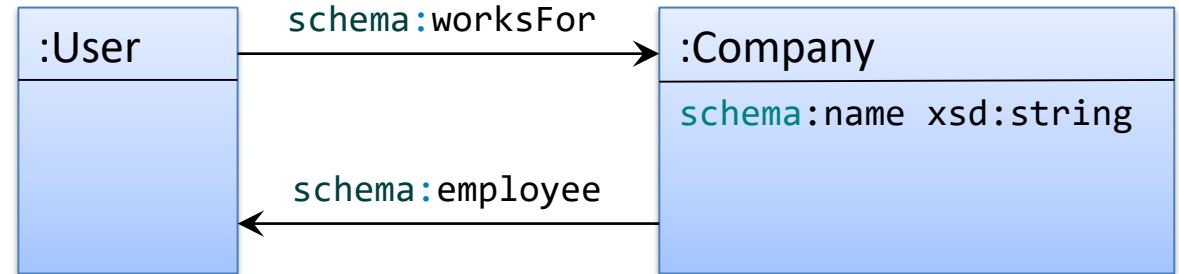
:OneCompany a :Company ;
            schema:name "One" ;
            schema:employee :alice, :bob, :carol .

```

Try it: <https://tinyurl.com/yynnts8o>

Exercise:

Represent the previous shapes without recursion using property paths



Logical Operators

Constraint	Description
and	Conjunction of a list of shapes
or	Disjunction of a list of shapes
not	Negation of a shape
xone	Exactly one (similar XOR for 2 arguments)

and

Default behavior

```
:User a sh:NodeShape ;
  sh:and (
    [ sh:property [
      sh:path      schema:name;
      sh:minCount 1;
    ]
    [ sh:property [
      sh:path      schema:affiliation;
      sh:minCount 1;
    ]
  ]
) .
```

≡

```
:User a sh:Shape ;
  [ sh:property [
    sh:path      schema:name;
    sh:minCount 1;
  ]
  [ sh:property [
    sh:path      schema:affiliation;
    sh:minCount 1;
  ]
] .
```

or

```
:User a sh:NodeShape ;
  sh:or (
    [ sh:property [
      sh:predicate foaf:name;
      sh:minCount 1;
    ]
  ]
  [ sh:property [
      sh:predicate schema:name;
      sh:minCount 1;
    ]
  ]
) .
```

```
:alice schema:name "Alice" .
:bob foaf:name "Robert" .
:carol rdfs:label "Carol" .
```



not

```
:NotFoaf a sh:NodeShape ;  
  sh:not [ a sh:Shape ;  
    sh:property [  
      sh:predicate foaf:name ;  
      sh:minCount 1 ;  
    ] ;  
  ] .
```

```
:alice schema:name "Alice" .  
:bob   foaf:name "Robert" . ☹️  
:carol rdfs:label "Carol" .
```

Exactly one

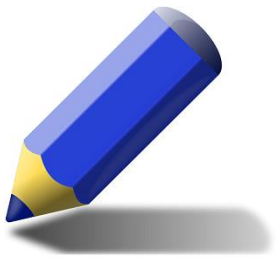
```
:UserShape a sh:NodeShape ;
sh:targetClass :User ;
sh:xone (
  [ sh:property [
    sh:path foaf:name;
    sh:minCount 1;
  ]
  [ sh:property [
    sh:path schema:name;
    sh:minCount 1;
  ]
]
) .
```

```
:alice a :User ; #Passes as :User
      schema:name "Alice" .

:bob a :User ; #Passes as :User
   foaf:name "Robert" .

:carol a :User ; #Fails as :User
     foaf:name "Carol";
     schema:name "Carol" .

:dave a :User ; #Fails as :User
     rdfs:label "Dave" .
```



Exercise

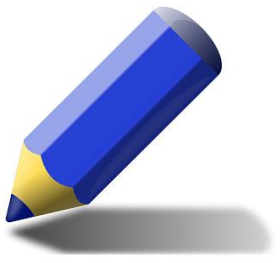
IF-THEN pattern

All products must have `:productID` and, if a product has `rdf:type schema:Vehicle` then it must have the properties `schema:vehicleEngine` and `schema:fuelType`

```
:p1 a :Book; # Conforms
  schema:productID "P1" .

:p2 a schema:Vehicle ; # Conforms
  schema:productID "P2" ;
  schema:fuelType "Gasoline" ;
  schema:vehicleEngine "X2" .

:p3 a schema:Vehicle ; # Fails
  schema:productID "P3" .
```



Exercise

IF-THEN-ELSE pattern

All products must have `:productID` and, if a product has `rdf:type schema:Vehicle` then it must have the properties `schema:vehicleEngine` and `schema:fuelType` else it must have `schema:category` with a string value.

Value ranges

Constraint	Description
minInclusive	\leq
maxInclusive	\geq
minExclusive	$<$
maxExclusive	$>$

```

:Rating a sh:NodeShape ;
sh:property [
  sh:path          schema:ratingValue ;
  sh:minInclusive  1 ;
  sh:maxInclusive  5 ;
  sh:datatype      xsd:integer
] .

```

```

:bad          schema:ratingValue 1 .
:average     schema:ratingValue 3 .
:veryGood    schema:ratingValue 5 .
:zero        schema:ratingValue 0 . ☹️

```

String based constraints

Constraint	Description
minLength	Restricts the minimum string length on value nodes
maxLength	Restricts the maximum string length on value nodes
pattern	Checks if the string value matches a regular expression

minLength/maxLength

Checks the string representation of the value

This cannot be applied to blank nodes

If minLength = 0, no restriction on string length

```
:User a sh:NodeShape ;
  sh:property [
    sh:path      schema:name ;
    sh:minLength 4 ;
    sh:maxLength 10 ;
  ] .
```

```
:alice schema:name "Alice" .
:bob schema:name "Bob" .
:carol schema:name :Carol .
:strange schema:name _:strange .
```



pattern

Checks if the values matches a regular expression

It can be combined with sh:flags

```
:Product a sh:NodeShape ;  
  sh:property [  
    sh:path      schema:productID ;  
    sh:pattern   "^P\\d{3,4}" ;  
    sh:flags     "i" ;  
  ] .
```

```
:car      schema:productID "P2345" .  
:bus      schema:productID "p567" .  
:truck    schema:productID "P12" .  
:bike     schema:productID "B123" .
```



Language based constraints

Constraint	Description
languageIn	Declares the allowed languages of a literal
uniqueLang	Specifies that no pair of nodes can have the same language tag

languageIn

Specifies the allowed language that a literal can have

```
:ProductShape a sh:NodeShape;  
  sh:targetClass :Product ;  
  sh:property [  
    sh:path      rdfs:label ;  
    sh:languageIn ("es" "en" "fr")  
  ] .
```

```
:p234 a :Product ;  
  rdfs:label "jamón"@es, "ham"@en .  
  
:p235 a :Product ;  
  rdfs:label "milk"@en .  
  
:p236 a :Product ;  
  rdfs:label "Käse"@de .  
  
:p237 a :Product ;  
  rdfs:label "patatas"@es ,  
            "kartofeln"@de .
```



uniqueLang

Checks that no pair of nodes use the same language tag

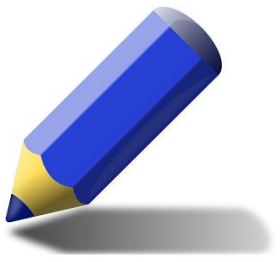
```
:CountryShape a sh:NodeShape ;  
  sh:targetClass :Country ;  
  sh:property [  
    sh:path      skos:prefLabel ;  
    sh:uniqueLang true  
  ] .
```

```
:spain a :Country;  
  skos:prefLabel "Spain"@en,  
                 "España"@es .  
  
:france a :Country;  
  skos:prefLabel "France",  
                 "France"@en,  
                 "Francia"@es .
```

```
:italy a :Country .
```

```
:usa a :Country;  
  skos:prefLabel "USA"@en,  
                 "United States"@en.
```





Exercise

Nodes must have exactly one literal per language in English and Spanish for property `skos:prefLabel`

Property pair constraints

Constraint	Description
<code>equals</code>	The sets of values of both properties at a given focus node must be equal
<code>disjoint</code>	The sets of values of both properties at a given focus node must be different
<code>lessThan</code>	The values must be smaller than the values of another property
<code>lessThanOrEquals</code>	The values must be smaller or equal than the values of another property

```

:User a sh:NodeShape ;
  sh:property [
    sh:path    schema:givenName ;
    sh>equals  foaf:firstName
  ];
  sh:property [
    sh:path    schema:givenName ;
    sh:disjoint schema:lastName
  ] .

```

```

:alice schema:givenName "Alice";
       schema:lastName  "Cooper";
       foaf:firstName    "Alice" .

:bob   schema:givenName "Bob";
       schema:lastName  "Smith" ;
       foaf:firstName    "Robert" .

:carol schema:givenName "Carol";
       schema:lastName  "Carol" ;
       foaf:firstName    "Carol" .

```



Closed shapes

Constraint	Description
closed	Valid resources must only have values for properties that appear in <code>sh:property</code>
ignoredProperties	Optional list of properties that are also permitted

```
:User a sh:NodeShape ;
  sh:closed true ;
  sh:ignoredProperties ( rdf:type ) ;
  sh:property [
    sh:path schema:givenName ;
  ];
  sh:property [
    sh:path schema:lastName ;
  ] .
```

```
:alice schema:givenName "Alice";
        schema:lastName "Cooper" .

:bob    a :Employee ;
        schema:givenName "Bob";
        schema:lastName "Smith" .

:carol  schema:givenName "Carol";
        schema:lastName "King" ;
        rdfs:label "Carol" .
```



Qualified value shapes

Problem with repeated properties

Example: Books have two IDs (an isbn and an internal code)

```
:Book a sh:NodeShape ;
  sh:property [
    sh:path      schema:productID ;
    sh:minCount  1;
    sh:datatype  xsd:string ;
    sh:pattern   "^isbn"
  ];
  sh:property [
    sh:path      schema:productID ;
    sh:minCount  1;
    sh:datatype  xsd:string ;
    sh:pattern   "^code"
  ] .
```

```
:b1 schema:productID "isbn:123-456-789" ;
    schema:productID "code234" .
```

It fails!!

Qualified value shapes

Qualified value shapes verify that certain number of values of a given property have a given shape

```
:Book a sh:NodeShape;  
sh:property [  
  sh:path schema:productID ;  
  sh:minCount 2; sh:maxCount 2; ];  
sh:property [  
  sh:path schema:productID ;  
  sh:qualifiedMinCount 1 ;  
  sh:qualifiedValueShape [  
    sh:pattern "^isbn"  
  ]];  
sh:property [  
  sh:path schema:productID ;  
  sh:qualifiedMinCount 1 ;  
  sh:qualifiedValueShape [  
    sh:pattern "^code" ;  
  ] .
```

```
:b1 schema:productID "isbn:123-456-789" ;  
    schema:productID "code234" .
```


Non-validating constraints

Can be useful to annotate shapes or design UI forms

Constraint	Description
name	Provide human-readable labels for a property
description	Provide a description of a property
order	Relative order of the property
group	Group several constraints together

```
:User a sh:NodeShape ;
  sh:property [
    sh:path schema:url ;
    sh:name "URL";
    sh:description "User URL";
    sh:order 1
  ];
  sh:property [
    sh:path schema:name ;
    sh:name "Name";
    sh:description "User name";
    sh:order 2
  ] .
```

Non-validating constraints

```
:User a sh:NodeShape ;
  sh:property [ sh:path schema:url ;
    sh:name "URL";
    sh:group :userDetails
  ];
  sh:property [ sh:path schema:name ;
    sh:name "Name"; sh:group :userDetails
  ];
  sh:property [ sh:path schema:address ;
    sh:name "Address"; sh:group :location
  ];
  sh:property [ sh:path schema:country ;
    sh:name "Country"; sh:group :location
  ] .
```

```
:userDetails a sh:PropertyGroup ;
  sh:order 0 ;
  rdfs:label "User details" .

:location a sh:PropertyGroup ;
  sh:order 1 ;
  rdfs:label "Location" .
```

An agent could generate a form like:

User details

URL: _____

Name: _____

Location

Address: _____

Country: _____

SHACL-SPARQL

SPARQL constraints

Constraints based on SPARQL code.

When the SPARQL query return validation errors a violation is reported

SPARQL constraints have type `sh:SPARQLConstraint`

Constraint	Description
<code>message</code>	Message in case of error
<code>sparql</code>	SPARQL code that is run
<code>prefixes</code>	Points to namespace prefix declarations defined by <code>sh:declare:</code> Each one has: <code>sh:prefix:</code> Prefix alias <code>sh:namespace:</code> namespace IRI

SPARQL constraints

Special variables are pre-bound by the SHACL-SPARQL processor

Constraint	Description
<code>\$this</code>	Focus Node
<code>\$shapesGraph</code>	Can be used to query the shapes graph in named graphs Similar to: <code>GRAPH \$shapesGraph { ... }</code>
<code>\$currentShape</code>	Current shape

SPARQL constraints

Mappings between result rows and error validation information

Constraint	Description
sh:focusNode	Value of <code>\$this</code> variable
sh:subject	Value of <code>?subject</code> variable
sh:predicate	Value of <code>?predicate</code> variable
sh:object	Value of <code>?object</code> variable
sh:message	Value of <code>?message</code> variable
sh:sourceConstraint	The constraint that was validated against
sh:sourceShape	The shape that was validated against
sh:severity	sh:ViolationError by default or the value of sh:severity

SPARQL constraints

Example: Name must be the concatenation of singleName and familyName

```

:UserShape a sh:NodeShape ;
sh:targetClass :User ;
sh:sparql [ a sh:SPARQLConstraint ;
  sh:message "schema:name must equal schema:givenName+schema:familyName";
  sh:prefixes [ sh:declare [
    sh:prefix "schema" ;
    sh:namespace "http://schema.org/"^^xsd:anyURI ;
  ] ] ;
sh:select
  """SELECT $this (schema:name AS ?path) (?name as ?value)
  WHERE {
    $this schema:name ?name .
    $this schema:givenName ?givenName .
    $this schema:familyName ?familyName .
    FILTER (!isLiteral(?value) ||
      !isLiteral(?givenName) || !isLiteral(?familyName) ||
      concat(str(?givenName), ' ', str(?familyName))!=?name )
  }""" ;
] .

```

```

:alice a :User ;
  schema:givenName "Alice" ;
  schema:familyName "Cooper" ;
  schema:name "Alice Cooper" .

:bob a :User ;
  schema:givenName "Bob" ;
  schema:familyName "Smith" ;
  schema:name "Robert Smith" .

```



SPARQL constraint components

SHACL-SPARQL allows to declare custom constraint components
Once defined, they can be used like built-in constraint components

```
:ProductShape a sh:NodeShape ;  
sh:targetClass :Product ;  
sh:property [  
  sh:path      :color ;  
  :size        3 ;  
  sh:minCount  1 ;  
] .
```



```
:c1 :color (255 0 255) .
```

```
:c2 :color (255 0 210 345) . ☹️
```

```
:c3 :color (255 0) . ☹️
```


SPARQL constraint components

```

:FixedListConstraintComponent
  a sh:ConstraintComponent ;
  sh:parameter [
    sh:path      :size ;
    sh:name      "Size of list" ;
    sh:description "The size of the list" ;
  ] ;
  sh:labelTemplate "Size of values: \"{$size}\"" ;
  sh:propertyValidator :fixedLengthValidator .

```

Two types of validators:
 SPARQLSelectValidator
 SPARQLASKValidator

```

:fixedLengthValidator a sh:SPARQLSelectValidator ;
  sh:message
    "{$PATH} must have length {?size}, not {?count}" ;
  sh:prefixes [ sh:declare [
    sh:prefix "rdf" ;
    sh:namespace
      "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  ]
  ] ;
  sh:select """SELECT $this ?value $count WHERE {
    $this $PATH ?value .
    { SELECT $this ?value
      (COUNT(?member) AS ?count)
      $size WHERE {
        ?value rdf:rest*/rdf:first ?member
      } GROUP BY $this ?value $size
    }
    FILTER (!isBlank(?value) || ?count != $size)
  }"""
.

```

SPARQL constraint components

Property	Description
sh:parameter	Declares the parameters of the constraint component The values are subclasses of property shapes sh:path declares the parameter name sh:optional declares if the parameter is optional
sh:labelTemplate	Suggests how constraints are rendered. Can refer to parameter names using: \$varName
sh:nodeValidator	Associates a node shape validator
sh:propertyValidator	Associates a property shape validator

SPARQL based validators can be SELECT or ASK-based validators

SHACL and inference systems

SHACL uses a subset of RDFS for target declarations

rdfs:subClassOf, rdf:type, owl:imports

A shapes graph containing sh:entailment with value E indicates the SHACL processor the kind of entailment to apply to the data

Possible values:

RDFS: <http://www.w3.org/ns/entailment/RDFS>

OWL 2 RDF based: <http://www.w3.org/ns/entailment/OWL-RDF-Based>

...and more, see: <https://www.w3.org/TR/sparql11-entailment/>

Other features

SHACL accepted as Recommendation on July 2017

SHACL community group created:

<https://www.w3.org/community/shacl/>

Several features were postponed

Advanced features: <https://w3c.github.io/data-shapes/shacl-af/>

SHACL functions and rules

Compact syntax: <https://w3c.github.io/shacl/shacl-compact-syntax/>

End of presentation

Solutions to exercises

Simulate recursion with property paths

