

# SHACL

## Shapes Constraint Language

Catherine Faron  
slides from Olivier Corby



# Semantic Web Languages

1. RDF: Resource Description Framework
2. RDFS: RDF Schema
3. SPARQL: RDF Query Language
4. OWL: Web Ontology Language
5. SKOS: Simple Knowledge Organization System (Thesaurus)

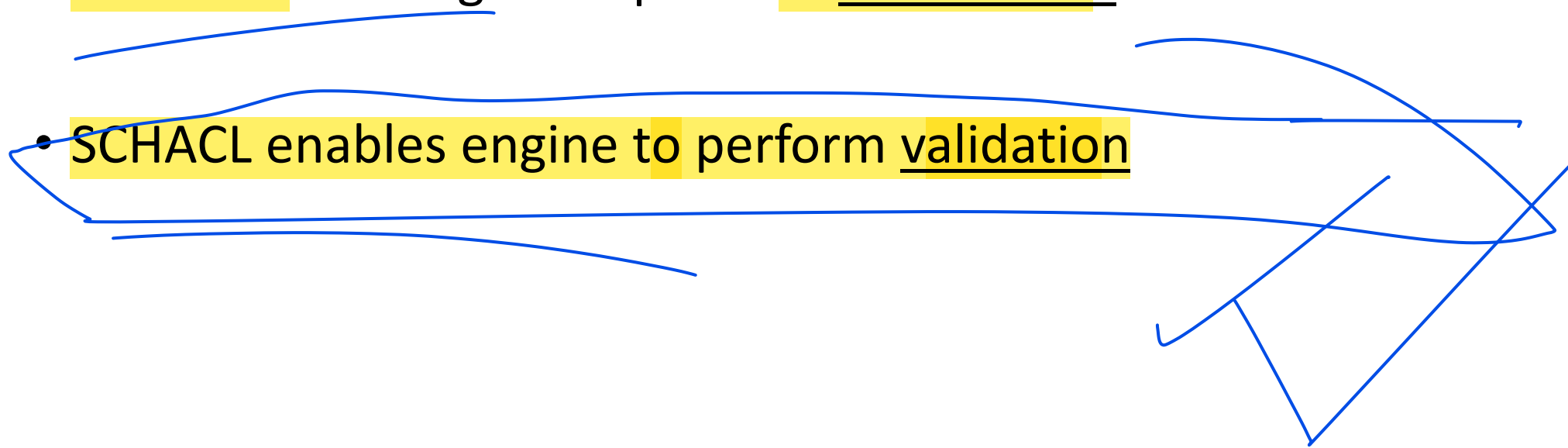
# Semantic Web Languages

1. RDF: Resource Description Framework
2. RDFS: RDF Schema
3. SPARQL: RDF Query Language
4. OWL: Web Ontology Language
5. SKOS: Simple Knowledge Organization System (Thesaurus)
6. SHACL: Shapes Constraint Language

# SHACL

- Language for **validating** RDF graphs against « constraints »
- W3C Recommendation, 2017
- SHACL syntax is RDF

# SHACL vs RDFS

- RDFS enables engine to perform entailments
  - SCHACL enables engine to perform validation
- 
- Hand-drawn blue scribbles and a checkmark. A horizontal line with a wavy underline is drawn under the first bullet point. A horizontal line with a wavy underline is drawn under the second bullet point. A checkmark is drawn to the right of the second bullet point.

# Example



instances of **class Person** must have  
**one property name**  
with **datatype string**

# Example

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

---

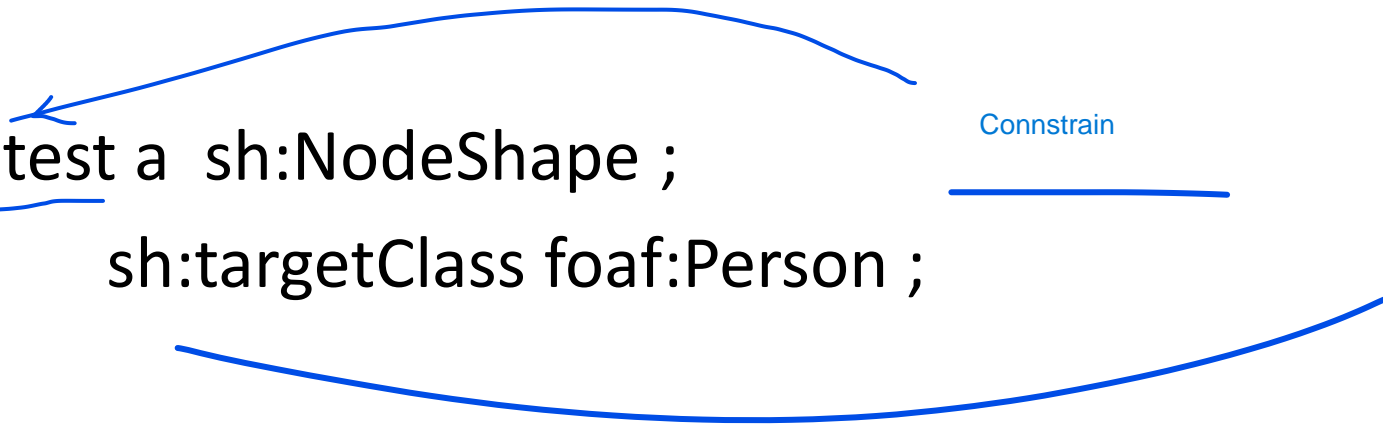
ex:test a sh:NodeShape ;



# Example

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

```
ex:test a sh:NodeShape ;  
    sh:targetClass foaf:Person ;
```



Connstrain



# Example

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

```
ex:test a sh:NodeShape ;  
    sh:targetClass foaf:Person ;  
    sh:property [  
        sh:path foaf:name ;  
  
    ] .
```

# Example

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

```
ex:test a sh:NodeShape ;  
    sh:targetClass foaf:Person ;  
    sh:property [  
        sh:path foaf:name ;  
        sh:datatype xsd:string ;  
    ] .
```

# Example

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

```
ex:test a sh:NodeShape ;
```

```
    sh:targetClass foaf:Person ;
```

```
    sh:property [
```

```
        sh:path foaf:name ;
```

```
        sh:datatype xsd:string ;
```

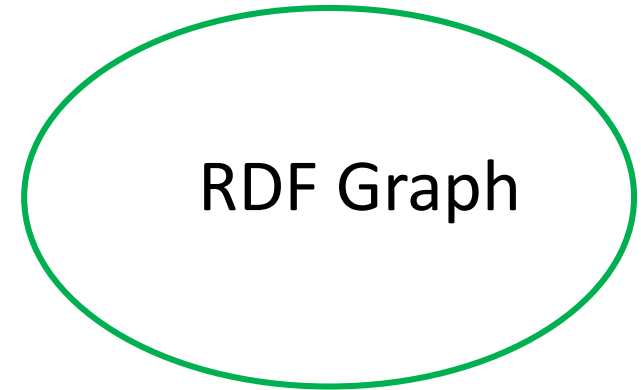
```
        sh:minCount 1
```

```
    ] .
```

# Example

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

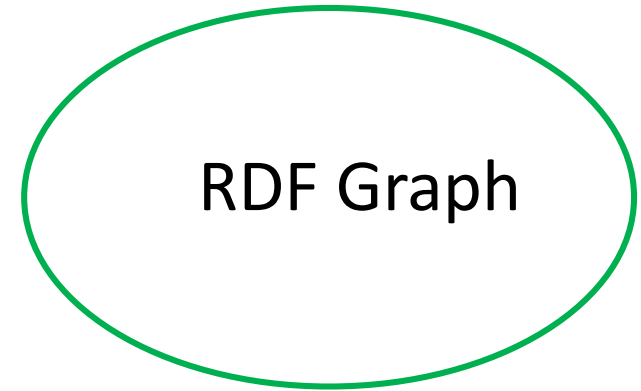
```
ex:test a sh:NodeShape ;  
    sh:targetClass foaf:Person ;  
    sh:property [  
        sh:path foaf:name ;  
        sh:datatype xsd:string ;  
        sh:minCount 1  
    ] .
```



# Example

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

```
ex:test a sh:NodeShape ;  
    sh:targetClass foaf:Person ;  
    sh:property [  
        sh:path foaf:name ;  
        sh:datatype xsd:string ;  
        sh:minCount 1  
    ] .
```



# Example

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

```
ex:test a sh:NodeShape ;
```

```
  sh:targetClass foaf:Person ;
```

```
  sh:property [
```

```
    sh:path foaf:name ;
```

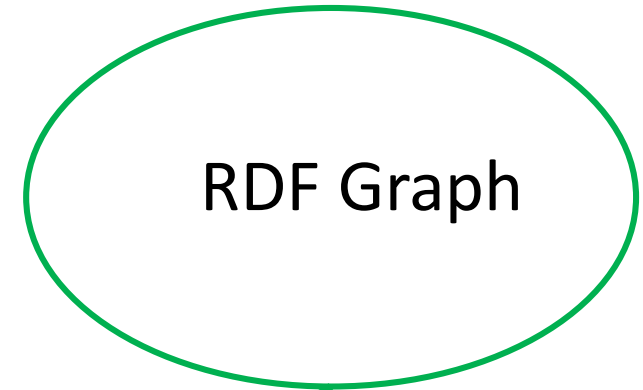
```
    sh:datatype xsd:string ;
```

```
    sh:minCount 1
```

```
  ] .
```



$O_1 \dots O_n$



# Example

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

```
ex:test a sh:NodeShape ;
```

```
  sh:targetClass foaf:Person ;
```

```
  sh:property [
```

```
    sh:path foaf:name ;
```

```
    sh:datatype xsd:string ;
```

```
    sh:minCount 1
```

```
  ] .
```



$O_1 \dots O_n$



```
ex:John a foaf:Person ;  
        foaf:name "John" .
```

# Example

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

```
ex:test a sh:NodeShape ;
```

```
  sh:targetClass foaf:Person ;
```

```
  sh:property [
```

```
    sh:path foaf:name ;
```

```
    sh:datatype xsd:string ;
```

```
    sh:minCount 1
```

```
  ] .
```



$O_1 \dots O_n$



```
ex:Jack a foaf:Person ;
```

```
  foaf:name "Jack"@en .
```



# Example

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

ex:test a sh:NodeShape ;

sh:targetClass foaf:Person ;

sh:property [

sh:path foaf:name ;

sh:datatype xsd:string ;

sh:minCount 1

].



$O_1 \dots O_n$



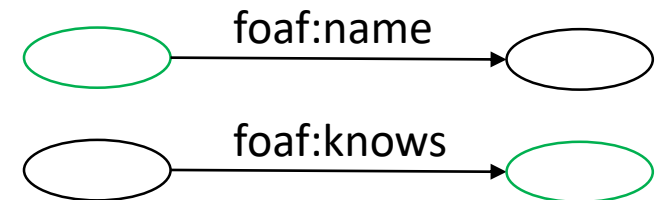
ex:James a foaf:Person ;

rdfs:label "James".

# Target

Determine the set of nodes relevant for a shape

1. sh:targetClass foaf:Person
2. sh:targetNode us:John, us:Jack for specific resources
3. sh:targetSubjectsOf foaf:name
4. sh:targetObjectsOf foaf:knows



# Target

Determine the set of nodes relevant for a shape

```
sh:targetClass foaf:Person, foaf:Organization
```

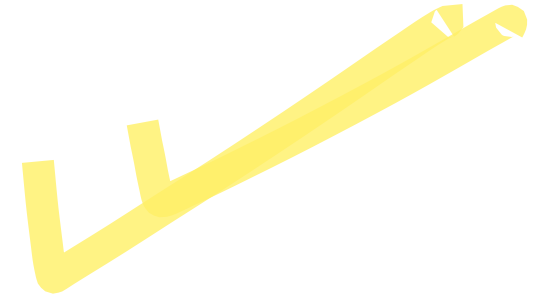
->

```
foaf:Person U foaf:Organization
```

# SHACL Shapes

## 1. Property shape

```
sh:property [ sh:path foaf:name ; sh:minLength 10 ]
```



## 2. Node shape

```
sh:minLength 10
```

# Property Path

✓ sh:property [ sh:path foaf:name ; sh:minLength 10]

✓ sh:property [ sh:path EXP ; sh:minLength 10]



SPARQL Property Path

# Property Path Expression



SPARQL: `rdf:type/rdfs:subClassOf*`

SHACL: `(rdf:type [sh:zeroOrMorePath rdfs:subClassOf])`

```
sh:property [  
    sh:path (rdf:type [sh:zeroOrMorePath rdfs:subClassOf]) ;  
    ...  
]
```

# Property Path

## SHACL

URI

(EXP<sub>1</sub> EXP<sub>2</sub> ...)

[sh:alternativePath (EXP<sub>1</sub> EXP<sub>2</sub> ...)]

[sh:inversePath EXP]

[sh:zeroOrMorePath EXP]

[sh:oneOrMorePath EXP]

[sh:zeroOrOnePath EXP]

## SPARQL

URI

EXP<sub>1</sub>/EXP<sub>2</sub>/...

EXP<sub>1</sub>|EXP<sub>2</sub>|...

^EXP

EXP\*

EXP+

EXP?

# Property Path

foaf:knows / foaf:name

(foaf:knows foaf:name)



sparql

shacl





# Property Path

(h:hasFriend | h:hasChild)+

[sh:oneOrMorePath

[sh:alternativePath (h:hasFriend h:hasChild)]]

# Type Constraint

- sh:class
- sh:datatype
- sh:nodeKind
  - URI, Blank Node, Literal

# Class

```
sh:property [  
    sh:path foaf:knows ;  
    sh:class foaf:Person  
]
```

Semantics: `rdf:type/rdfs:subClassOf* foaf:Person`

# Class

```
sh:property [  
    sh:path foaf:knows ;  
    sh:class ex:Professor, ex:Researcher  
]
```

intersection

->

ex:Professor  $\sqcap$  ex:Researcher

# Datatype

```
sh:property [  
    sh:path foaf:name ;  
    sh:datatype xsd:string  
]
```

XSD datatypes: xsd:integer, xsd:boolean, xsd:date, etc.

RDF datatypes: rdf:langString, rdf:XMLLiteral, etc.

            
    ↓  
@en @fr

# Node Kind

## sh:nodeKind

- sh:BlankNode, sh:IRI, sh:Literal
  - sh:BlankNodeOrIRI, sh:BlankNodeOrLiteral, sh:IRIOrLiteral
- 
- A hand-drawn blue checkmark is positioned to the right of the first list item. Below the second list item, there are three horizontal blue underlines, one under each term: 'sh:BlankNodeOrIRI', 'sh:BlankNodeOrLiteral', and 'sh:IRIOrLiteral'. Additionally, a long, slightly wavy blue underline is drawn beneath the entire second list item.

# Cardinality Constraint

```
sh:property [  
    sh:path foaf:age ;  
    sh:minCount 1 ;  
    sh:maxCount 1  
]
```

# Value Range Constraint

- sh:minExclusive 0
- sh:maxExclusive 10
- sh:minInclusive “a”
- sh:maxInclusive “z”




# String Value Range Constraint

- `sh:minLength 1` -- string length
- `sh:maxLength 10`
- `sh:pattern "uca"` -- regex
- ✓ `sh:languageIn ("fr" "it" "en")` -- `@en @ru` "test" `ex:value`
- `sh:uniqueLang true` -- `(@fr @it) (@en @fr @en)`

# Exercise

Write a shape:

target class Person  
property name  
datatype string  
string length > 0  
cardinality > 0



```
ex:test a sh:NodeShape ;  
sh:targetClass foaf:Person  
sh:property [  
sh:path ex:name ;  
sh:datatype xsd:string ;  
sh:minLength 1;  
sh:minCount 1.  
].
```

# Exercise

```
ex:test a sh:NodeShape ;  
sh:targetClass ex:Person ;  
sh:property [  
    sh:path ex:name ;  
    sh:datatype xsd:string ;  
    sh:minLength 1 ;  
    sh:minCount 1  
] .
```

# Exercise

Write SHACL Property Path for:

`rdf:rest*/rdf:first`

# Exercise

Write SHACL Property Path for:

rdf:rest\*/rdf:first

([sh:zeroOrMorePath rdf:rest] rdf:first)

# Exercise

Write SHACL Property Path for:

foaf:knows | ^foaf:knows

# Exercise

Write SHACL Property Path for:

foaf:knows | ^foaf:knows

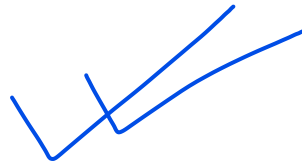


```
[sh:alternativePath (foaf:knows [sh:inversePath foaf:knows])]
```

# Property Pair Constraint

Compare two sets of values

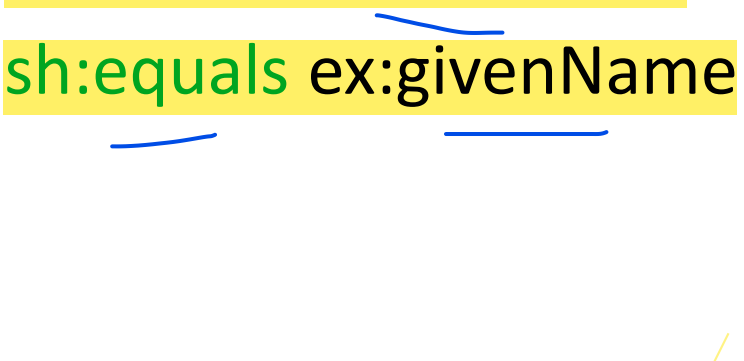
- sh:equals
- sh:disjoint
- sh:lessThan
- sh:lessThanOrEquals





# Property Pair Constraint

```
sh:property [  
    sh:path ex:firstName ;  
    sh:equals ex:givenName  
]
```



# Property Pair Constraint

```
sh:property [  
    sh:path    ex:child ;  
    sh:disjoint ex:parent  
]
```

# Property Pair Constraint


```
sh:property [  
    sh:path      ex:startDate ;  
    sh:lessThan ex:endDate  
]
```

# Boolean Constraint

- sh:not
- sh:and
- sh:or
- sh:xone – each node conforms to exactly one constraint

# Boolean Constraint

```
sh:property [  
    sh:path rdfs:label ;  
    sh:or ([sh:datatype xsd:string] [sh:datatype rdf:langString])  
]
```



# Boolean Constraint

```
sh:property [  
    sh:path foaf:knows ;  
    sh:not [sh:path foaf:name ; sh:hasValue ex:JohnDoe]  
]
```

# Boolean Constraint Example

Every class in namespace <http://example.org/ns> must be declared by `rdf:type`

# Boolean Constraint Example

Every class in namespace <http://example.org/ns> must be declared by `rdf:type`

Not (And (namespace=<http://example.org/ns>)  
(cardinality(`rdf:type`) = 0)



# Boolean Constraint Example

Every class in namespace <http://example.org/ns> must be declared by `rdf:type`

Not (And (namespace=<http://example.org/ns>)  
(cardinality(`rdf:type`) = 0)

```
sh:property [  
  sh:path rdf:type ; # this is a class  
  sh:not [ sh:and (  
    [sh:pattern 'http://example.org/ns']  
    [sh:property [sh:path rdf:type ; sh:maxCount 0]]]  
  ]  
]
```

# Qualified Value Shape

Instance of Person have exactly:

- one parent of type Man
- one parent of type Woman

# Qualified Value Shape

```
sh:property [ sh:path ex:parent ;  
              sh:qualifiedMinCount 1 ;  
              sh:qualifiedMaxCount 1 ;  
              sh:qualifiedValueShape [ sh:class ex:Man ] ]
```


# Qualified Value Shape

```
sh:property [ sh:path ex:parent ;  
              sh:qualifiedMinCount 1 ;  
              sh:qualifiedMaxCount 1 ;  
              sh:qualifiedValueShape [ sh:class ex:Man ] ] ;
```

```
sh:property [ sh:path ex:parent ;  
              sh:qualifiedMinCount 1 ;  
              sh:qualifiedMaxCount 1 ;  
              sh:qualifiedValueShape [ sh:class ex:Woman ] ]
```


# Qualified Value Shapes Disjoint

```
sh:property [ sh:path ex:parent ;  
              sh:qualifiedMinCount 1 ;  
              sh:qualifiedMaxCount 1 ;  
              sh:qualifiedValueShapesDisjoint true;  
              sh:qualifiedValueShape [ sh:class ex:Man ] ] ;  
sh:property [ sh:path ex:parent ;  
              sh:qualifiedMinCount 1 ;  
              sh:qualifiedMaxCount 1 ;  
              sh:qualifiedValueShapesDisjoint true;  
              sh:qualifiedValueShape [ sh:class ex:Woman ] ]
```



# Has Value

```
sh:property [  
    sh:path foaf:knows ;  
    sh:hasValue ex:JohnDoe  
]
```



- At least one node must be equal to the value
- 

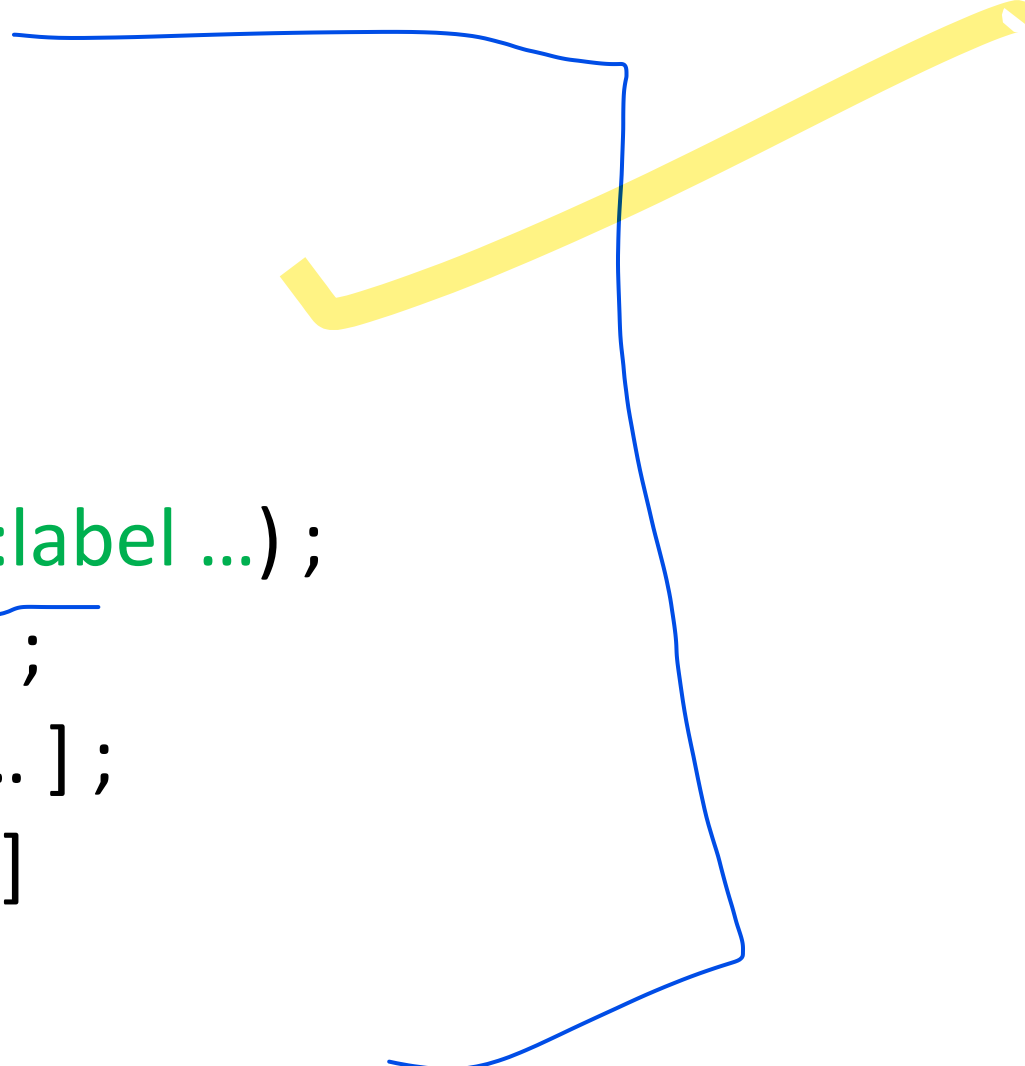
# Value In

```
sh:property [  
    sh:path ex:value ;  
    sh:in (ex:Good ex:Bad ex:Ugly)  
]
```

- Every node must be member of the value list

# Other Constraints

```
ex:test a sh:NodeShape ;  
sh:targetClass ex:Person ;  
sh:closed true ;  
sh:ignoredProperties (rdf:type rdfs:label ...) ;  
sh:property [ sh:path ex:name ; ... ] ;  
sh:property [ sh:path ex:address ; ... ] ;  
sh:property [ sh:path ex:phone ; ... ]  
.
```



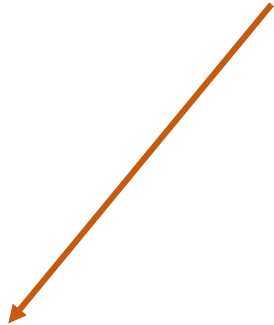


# Node Shape

```
ex:test1 a sh:NodeShape ;  
sh:targetClass foaf:Person ;
```

sh:node ex:test2

.



```
ex:test2 a sh:NodeShape ;
```

...

# Node Shape

ex:test a sh:NodeShape ;

sh:property [

sh:path foaf:knows ;

sh:node [

sh:property [ sh:path foaf:name ; sh:datatype xsd:string ]

]

].



# Exercise

Check that the values (the objects) of property `rdf:type` are either of class `rdfs:Class` or class `owl:Class`

# Exercise

Check that the values (the objects) of property `rdf:type` are either of class `rdfs:Class` or class `owl:Class`

```
sh:targetObjectsOf rdf:type ;
```

```
sh:or ( [sh:class rdfs:Class] [sh:class owl:Class] )
```



SHACL		
st:targetClass		
sh:targetNode		
sh:targetSubjectsOf		
sh:targetObjectsOf		
sh:property		
sh:path		
sh:node		

SHACL	OWL	RDFS
sh:class		rdfs:range
sh:datatype		rdfs:range
sh:minCount, sh:maxCount	owl:minCardinality, owl:maxCardinality	
sh:minInclusive, sh:maxInclusive		
sh:minExclusive, sh:maxExclusive,		
sh:minLength, sh:maxLength		
sh:pattern		
sh:languageIn		
sh:uniqueLang		

SHACL	OWL	RDFS
sh:equal		
sh:disjoint		
sh:lessThan, sh:lessThanOrEquals		
sh:or, sh:and, sh:not, sh:xone		
sh:hasValue		
sh:in		
sh:qualifiedValueShape		
sh:qualifiedMinCount, sh:qualifiedMaxCount		
sh:qualifiedValueShapesDisjoint		
sh:closed		
sh:ignoredProperties		

# Import Shape

```
[] owl:imports <http://www.w3.org/ns/shacl-shacl> .
```

---



# Shape Documentation



sh:message "This is a terrible mistake" ;


sh:severity sh:Info | sh:Warning | sh:Violation

+ user defined severity: ex:UltimateCriticalDamnedError

# Validation Report

RDF graph with a list of errors, if any

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



# Validation Report

```
[    a sh:ValidationReport ;  
    sh:conforms false ;  
    sh:result  
    [  
        a sh:ValidationResult ;  
        sh:resultSeverity sh:Warning ;  
        sh:focusNode ex:MyInstance ;  
        sh:resultPath ex:myProperty ;  
        sh:value "http://toomanycharacters"^^xsd:anyURI ;  
        sh:sourceConstraintComponent sh:DatatypeConstraintComponent ;  
        sh:sourceShape _:b1 ;  
    ]  
].
```

# SHACL SPARQL

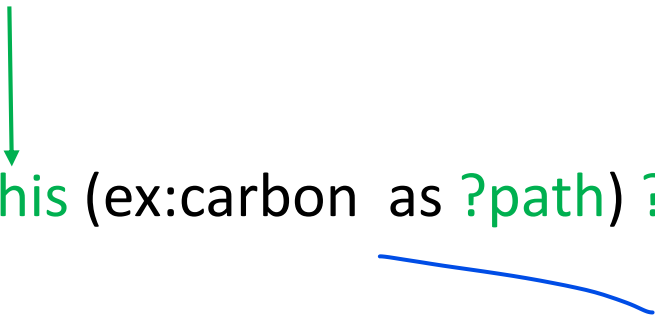
# SHACL SPARQL

Constraint defined by a SPARQL query

Query computes and returns nodes that fail, if any

# SHACL SPARQL: select nodes that fail

```
ex:sparql a sh:NodeShape ;  
sh:targetClass ex:Car ;  
sh:sparql [  
sh:select ""  
    select $this (ex:carbon as ?path) ?value  
    where {  
        $this ex:carbon ?value .  
        filter (?value > 95) # 95 g/km  
    }  
"" ;  
].
```



# Usage

Download Corese last version (Java 11)

<https://project.inria.fr/corese/download/>

# Usage

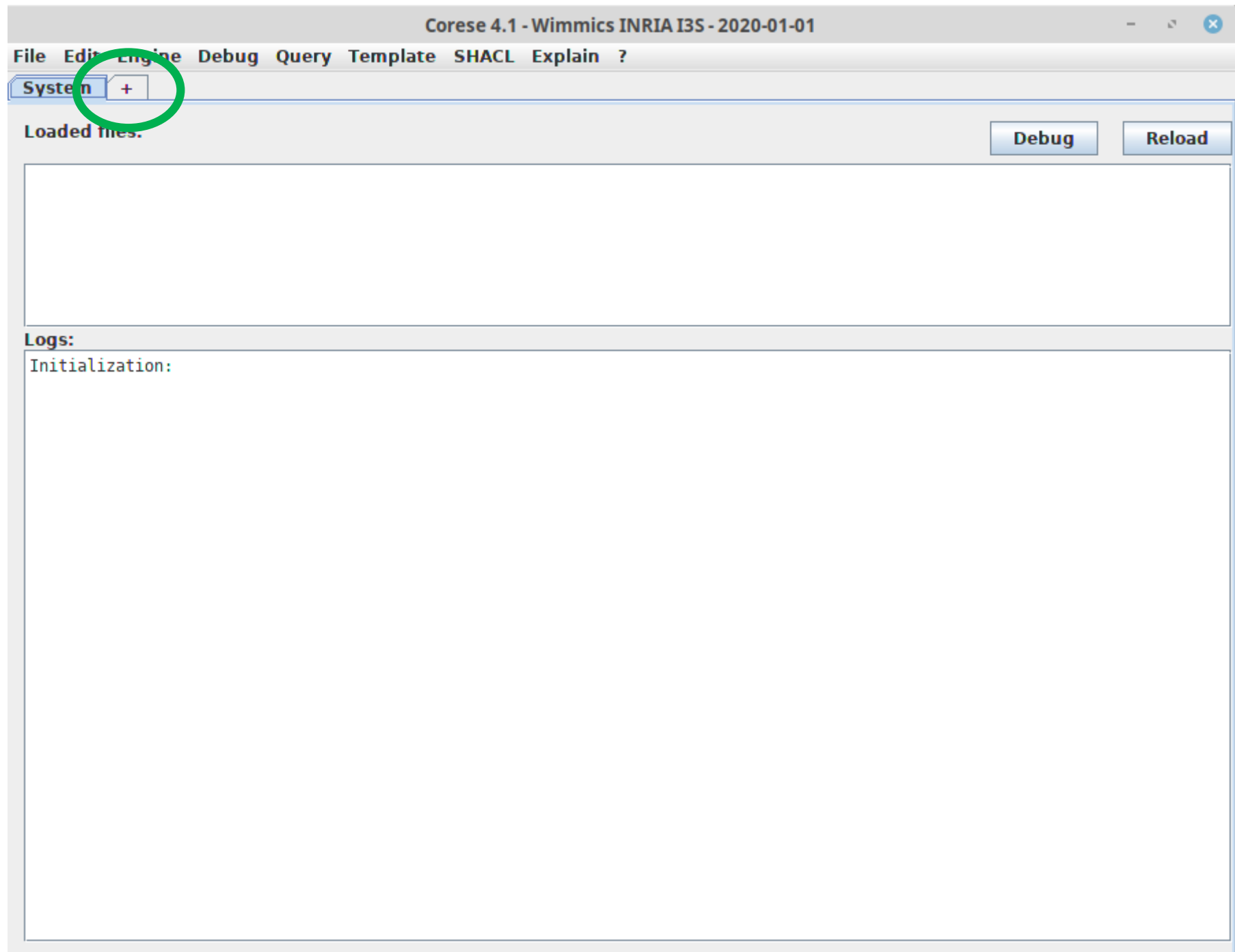
Load RDF graph

Load RDF schema

Load SHACL graph

Click + and click SHACL





Corese 4.1 - Wimmics INRIA I3S - 2020-01-01

File Edit Engine Debug Query Template SHACL Explain ?

System Query1 x +

Query SHACL Stop Kill Validate to SPIN to SPARQL Search Refresh stylesheet Defau

```
1 select * where {  
2   ?x ?p ?y  
3 }  
4
```

Graph XML/RDF Table Validate

# ShEx vs SHACL

- There is another shape language called ShEx !

<https://shex.io/shex-primer/>