

CC7220-1

LA WEB DE DATOS

PRIMAVERA 2022

LECTURE 10: SHAPES

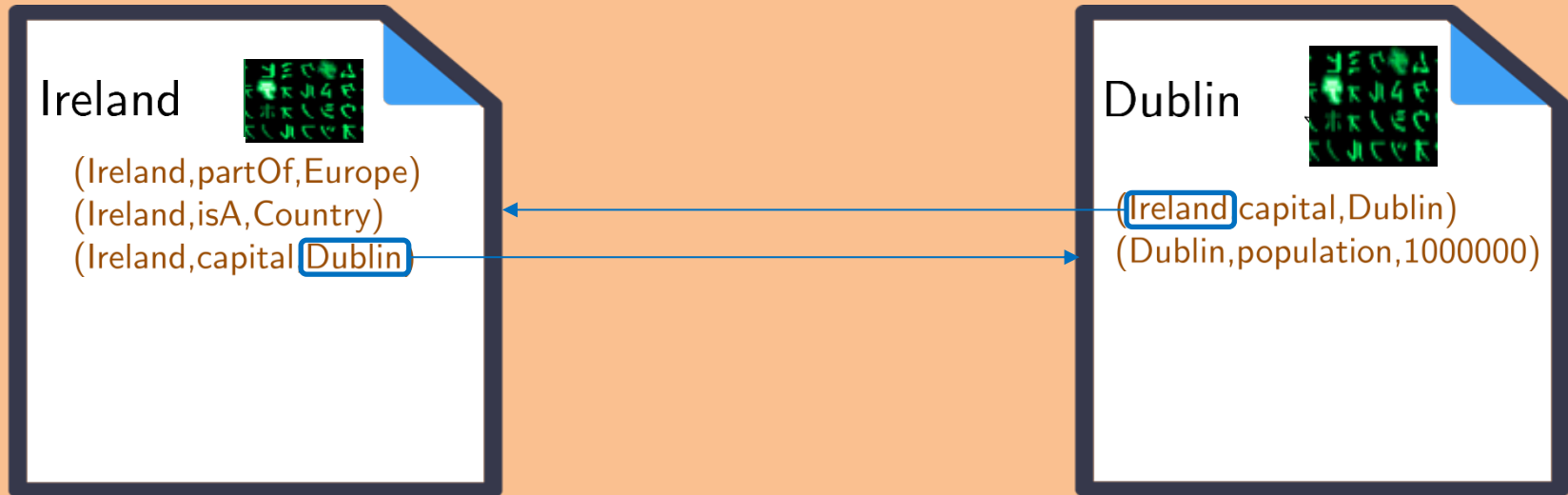
Aidan Hogan

aidhog@gmail.com

PREVIOUSLY ...

SEMANTIC WEB: DATA, LOGIC, QUERY, LINKS

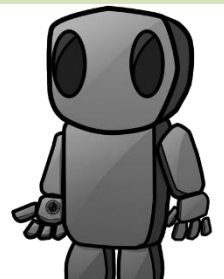
DATA:



LOGIC: $“(b, \text{capital}, a) \rightarrow (a, \text{partOf}, b)”$
 $“(a, \text{partOf}, b), (b, \text{partOf}, c) \rightarrow (a, \text{partOf}, c)”$

QUERY: $“(x, \text{partOf}, y)?”$

OUTPUT: $\{(x \mapsto \text{Ireland}, y \mapsto \text{Europe}),$
 $(x \mapsto \text{Dublin}, y \mapsto \text{Ireland}),$
 $(x \mapsto \text{Dublin}, y \mapsto \text{Europe})\}$



SEMANTIC WEB: DATA, LOGIC, QUERY, LINKS

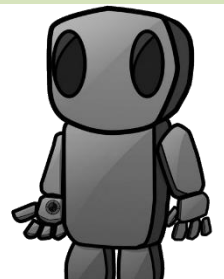
DATA:



LOGIC: $(b, \text{capital}, a) \rightarrow (a, \text{partOf}, b)$
 $(a, \text{partOf}, b), (b, \text{partOf}, c) \rightarrow (a, \text{partOf}, c)$

QUERY: $(x, \text{partOf}, y)?$

OUTPUT: $\{(x \mapsto \text{Ireland}, y \mapsto \text{Europe}),$
 $(x \mapsto \text{Dublin}, y \mapsto \text{Ireland}),$
 $(x \mapsto \text{Dublin}, y \mapsto \text{Europe})\}$



GRAPHS ...

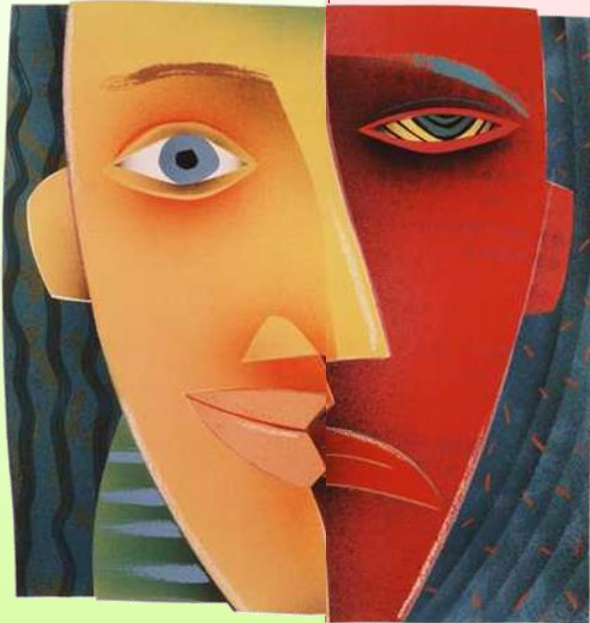
GRAPH DATA: PROS AND CONS

0.0000079 M_J

Planet

0.00314 M_J

We don't have to impose a structure (schema) from the start



We don't have a structure (schema) imposed from the start

G-type Star

...

1 M_{\odot}



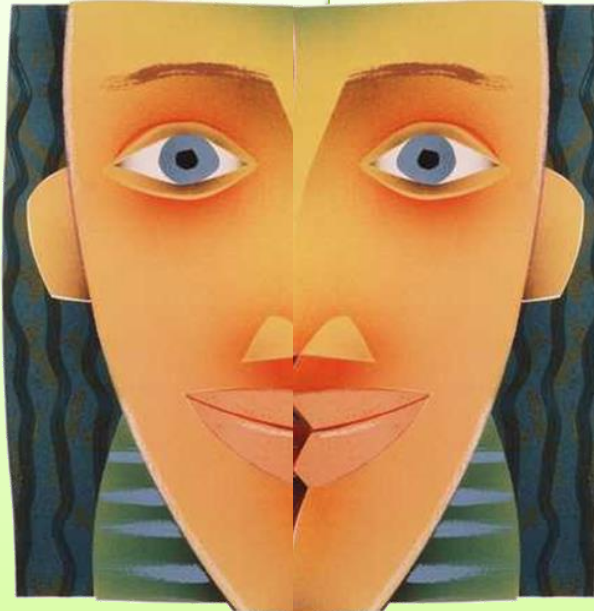
GRAPH DATA: PROS AND PROS

0.0000079 M_J

Planet

0.00314 M_J

We don't have to impose a structure (schema) from the start



... and we can impose a schema any time we like

G-type Star

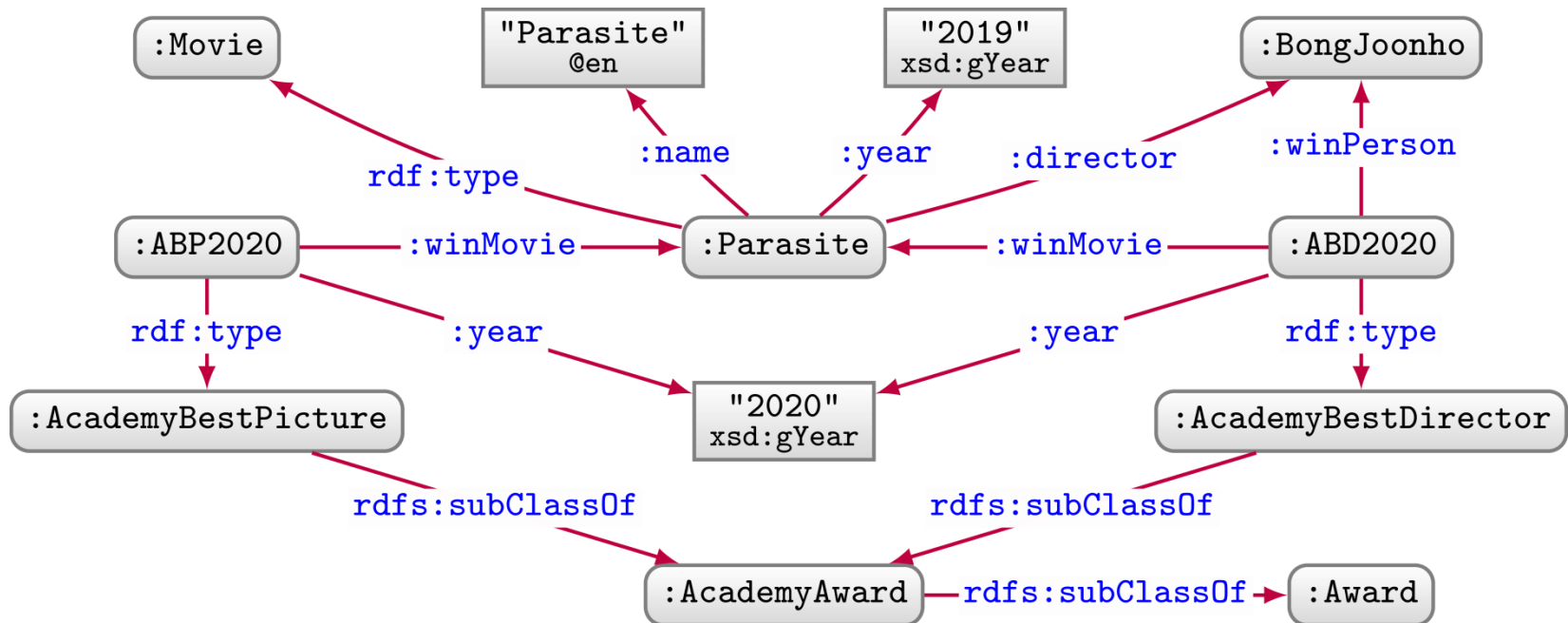
...

1 M_\odot

So how can we define and impose a schema for graphs?

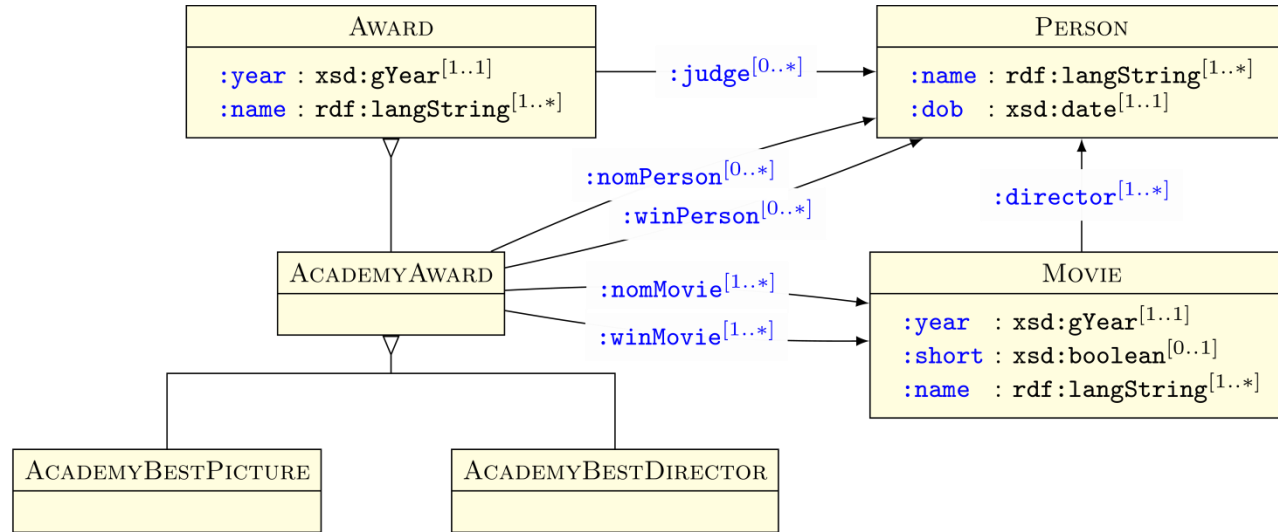
SHAPES

GRAPH DATA: VALIDATION

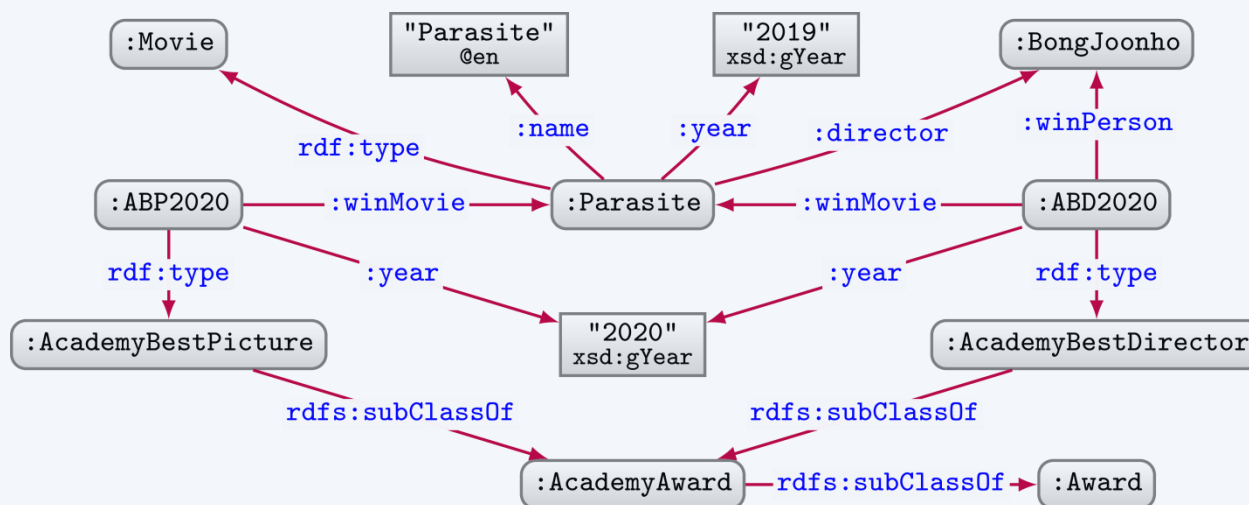
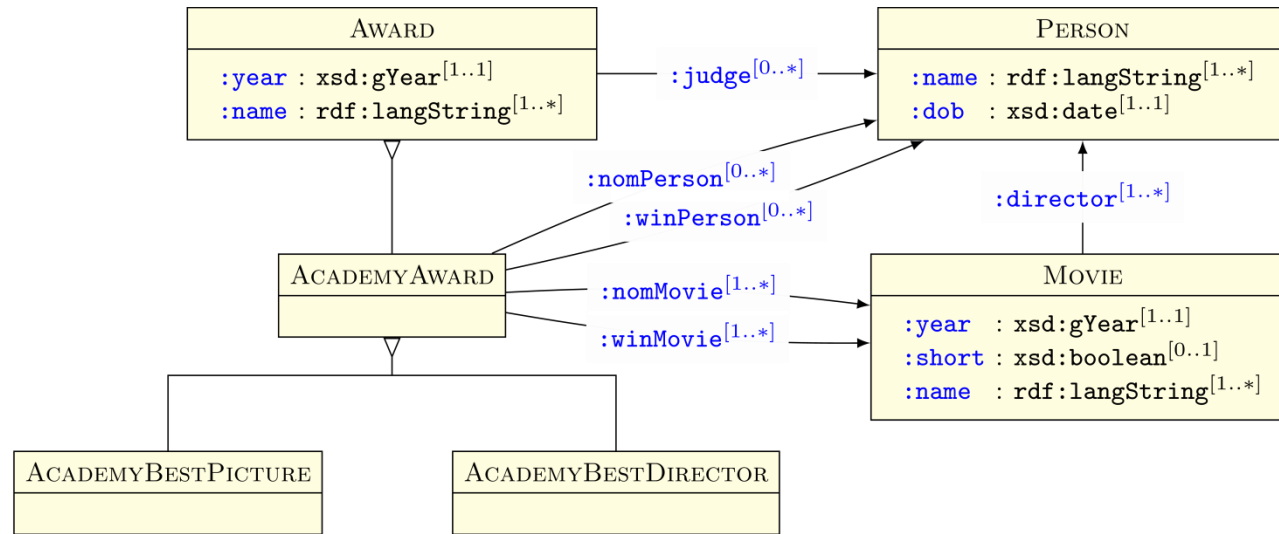


Is this graph "complete"?
Does it have "errors"?
How do we define "completeness" and "errors"?

SHAPES GRAPH: VALIDATING SCHEMA



SHAPES GRAPH: VALIDATE RDF GRAPHS

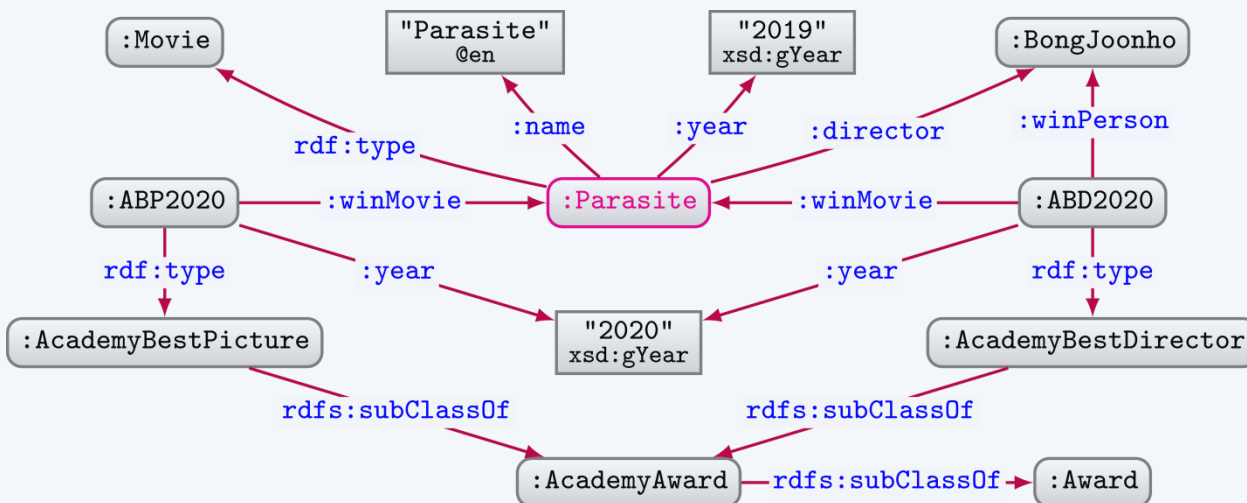
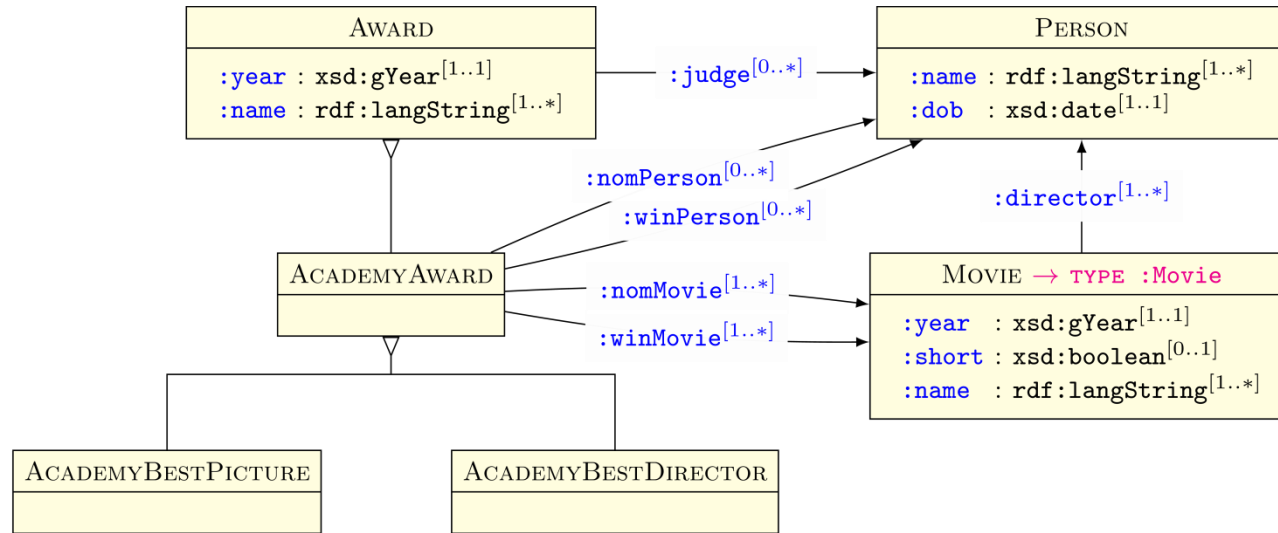


Does it pass?

👉

We have not yet defined a **target** for a shape, so we don't know which shape applies to which node in the data

SHAPES GRAPH: DEFINE A TARGET

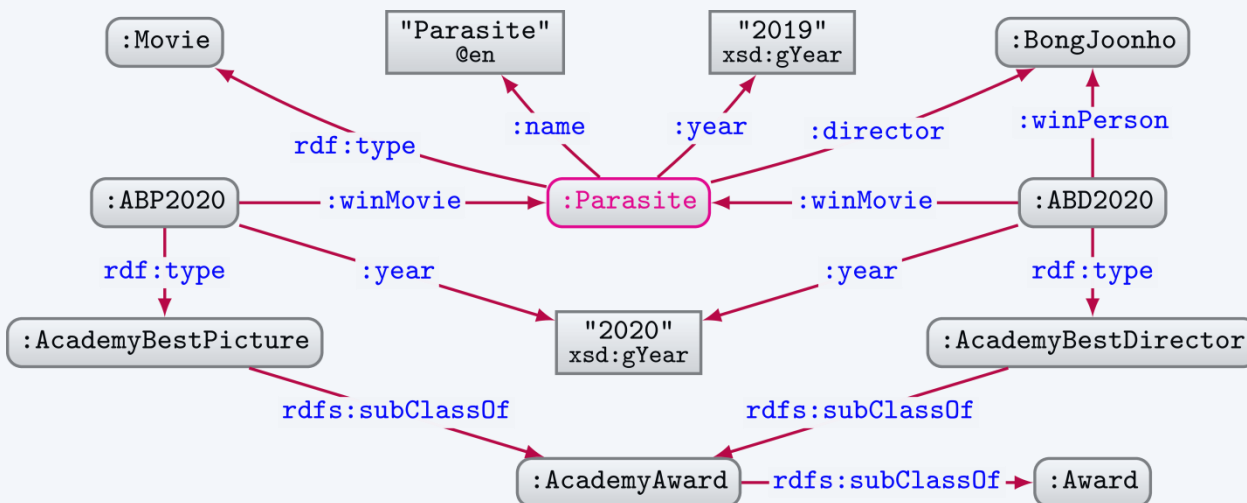
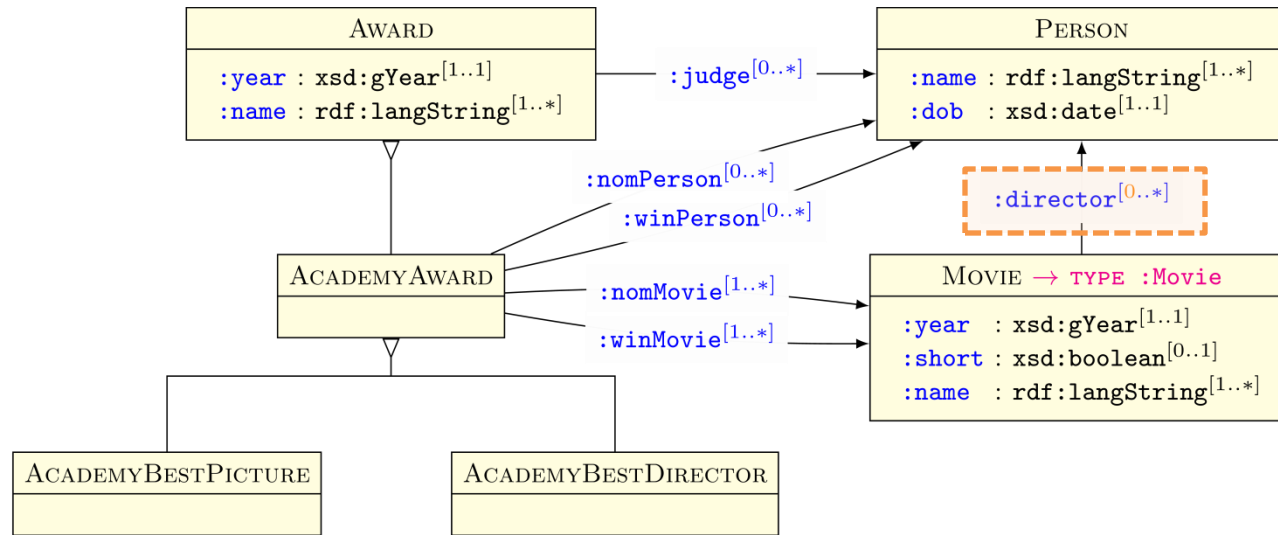


Does it pass?

No. :Parasite does not have a director satisfying PERSON.

We are missing a name and a date of birth for Bong Joon-ho!

SHAPES GRAPH: MULTIPLICITY

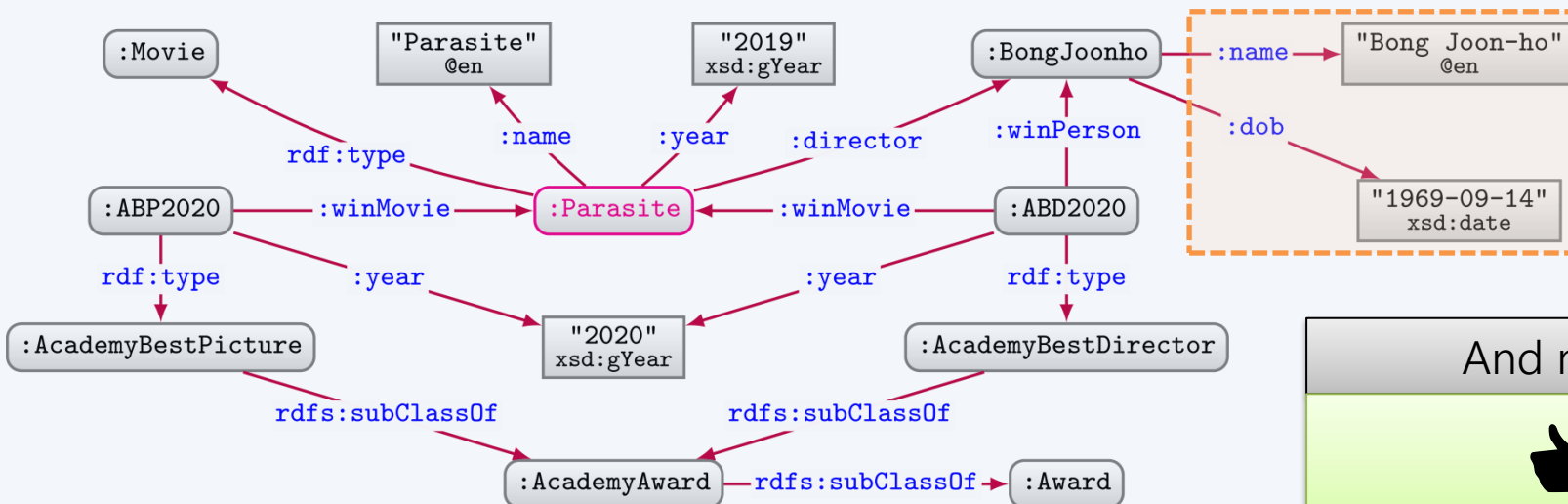
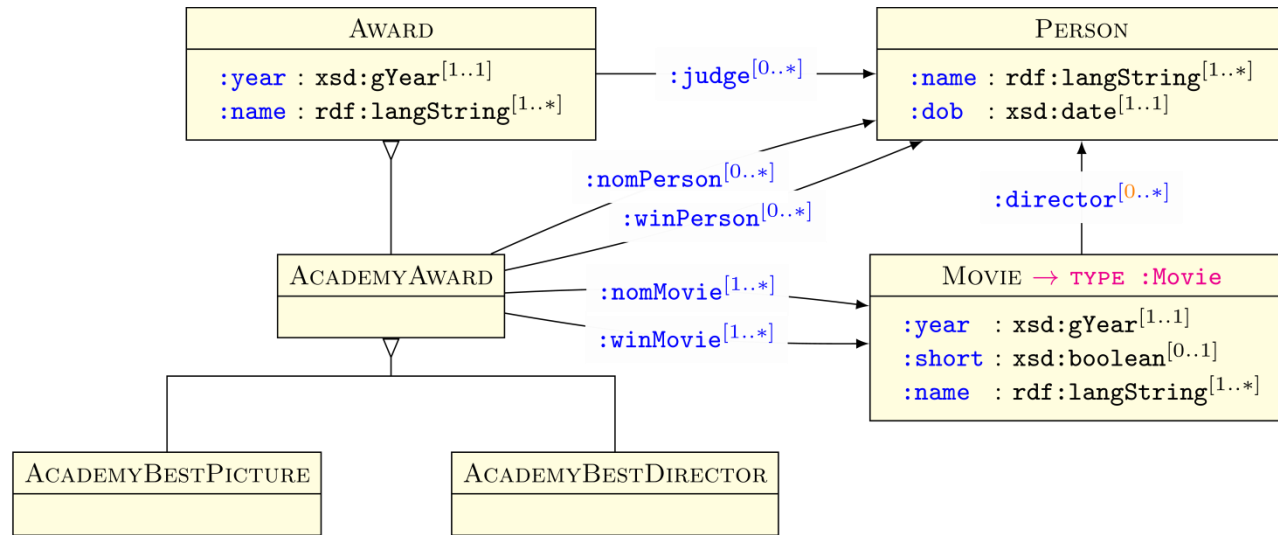


How about now?


No. Any director of :Parasite must still satisfy PERSON.

We are missing a name and a date of birth for Bong Joon-ho!

SHAPES GRAPH: VALIDATION

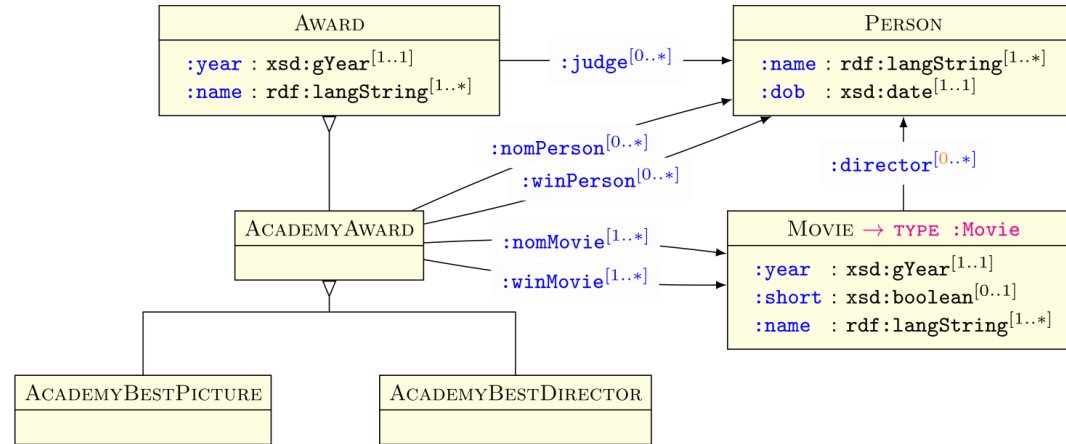


And now?



SHAPES vs. RDFS/OWL AND SPARQL

SHAPES VS. RDFS/OWL



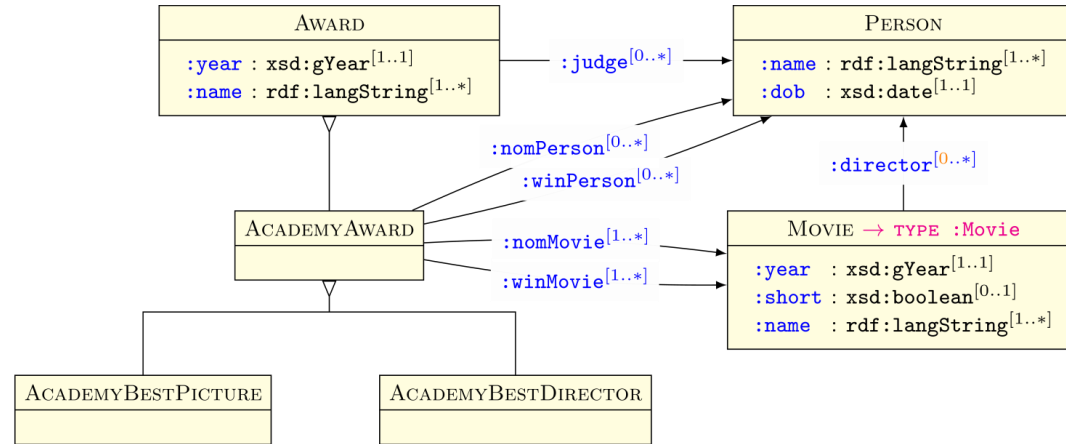
```
#[...]
:Award rdfs:subClassOf
  [ owl:allValuesFrom xsd:gYear ; owl:onProperty :year ] ,
  [ owl:cardinality 1 ; owl:onProperty :year ] ,
  [ owl:allValuesFrom rdf:langString ; owl:onProperty :name ] ,
  [ owl:minCardinality 1 ; owl:onProperty :name ] ,
  [ owl:allValuesFrom :Person ; owl:onProperty :judge ] .

AcademyAward rdfs:subClassOf :Award ,
  [ owl:allValuesFrom :Person ; owl:onProperty :nomPerson ] ,
  [ owl:allValuesFrom :Person ; owl:onProperty :winPerson ] ,
  [ owl:allValuesFrom :Movie ; owl:onProperty :nomMovie ] ,
  [ owl:minCardinality 1 ; owl:onProperty :nomMovie ] ,
  [ owl:minCardinality 1 ; owl:onProperty :winMovie ] .
#[...]
```



OWL assumes OWA and no UNA. Cannot easily detect missing data or duplicated values!

SHAPES VS. SPARQL



```
#[...]
```

```
# finds constraint violations
```

```
SELECT DISTINCT ?movie WHERE {
```

```
  ?movie a :Movie .
```

```
  OPTIONAL { ?movie :year ?year1 . }
```

```
  OPTIONAL { ?movie :year ?year2 . }
```

```
  FILTER(!bound(?year1) || datatype(?year1)!=xsd:gYear || ?year1 != ?year2)
```

```
  OPTIONAL { ?movie :short ?short1 . }
```

```
  OPTIONAL { ?movie :short ?short2 . }
```

```
  FILTER(bound(?short1) && (datatype(?short1)!=xsd:boolean || ?short1 != ?short2))
```

```
  OPTIONAL { ?movie :name ?name . }
```

```
  FILTER(!bound(?name) || lang(?name)!="")
```

```
  OPTIONAL { ?movie :director ?director . }
```

```
  # check that ?director satisfies Person shape
```

```
}
```

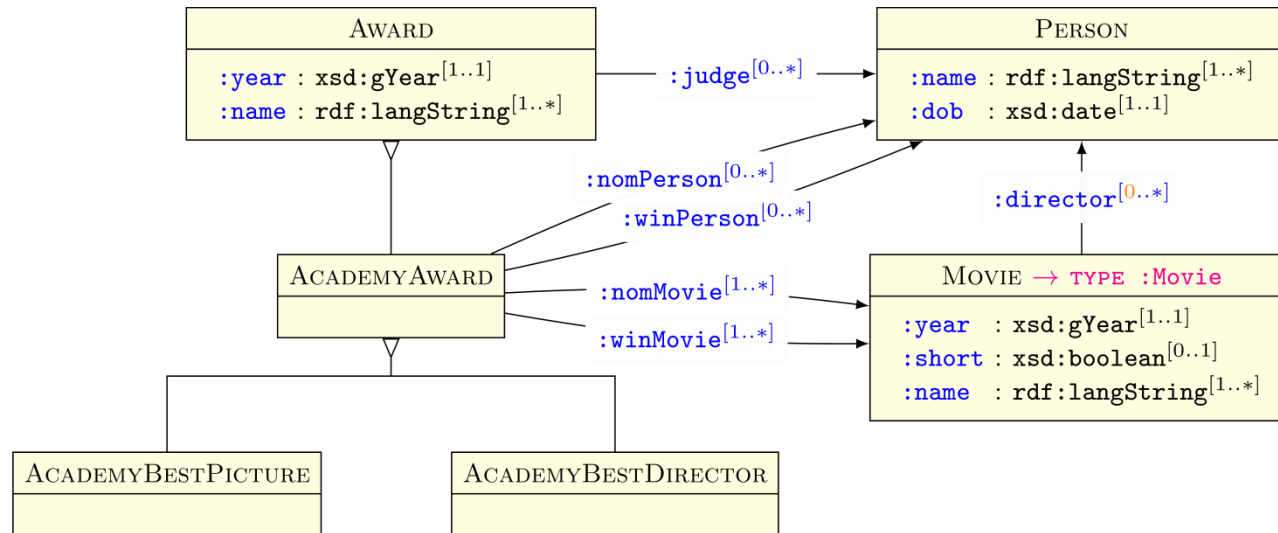


Correct semantics, but difficult to express these types of constraints.

SHACL:

SHAPES CONSTRAINT LANGUAGE

SHAPES GRAPH: HOW DO WE DEFINE THEM?



So how do we define shapes graphs?

Shapes Constraint Language (SHACL)

W3C Recommendation 20 July 2017



This version:

<https://www.w3.org/TR/2017/REC-shacl-20170720/>

Latest published version:

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

Latest editor's draft:

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

Implementation report:

<https://w3c.github.io/data-shapes/data-shapes-test-suite/>

Previous version:

<https://www.w3.org/TR/2017/PR-shacl-20170608/>

Editors:

[Holger Knublauch](#), [TopQuadrant, Inc.](#)

[Dimitris Kontokostas](#), [University of Leipzig](#)

Repository:

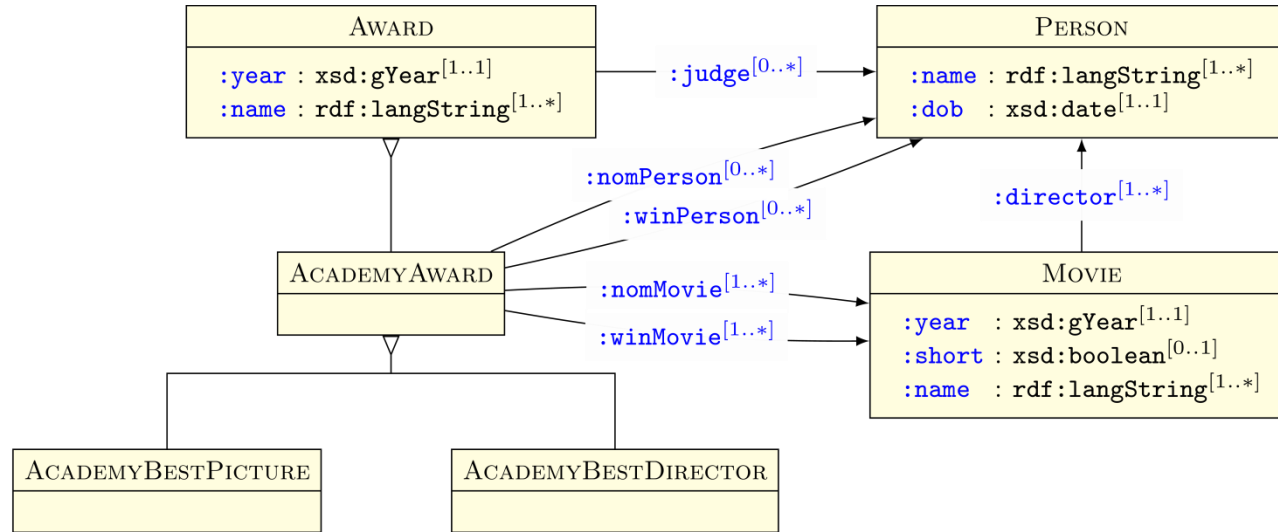
[GitHub](#)

[Issues](#)

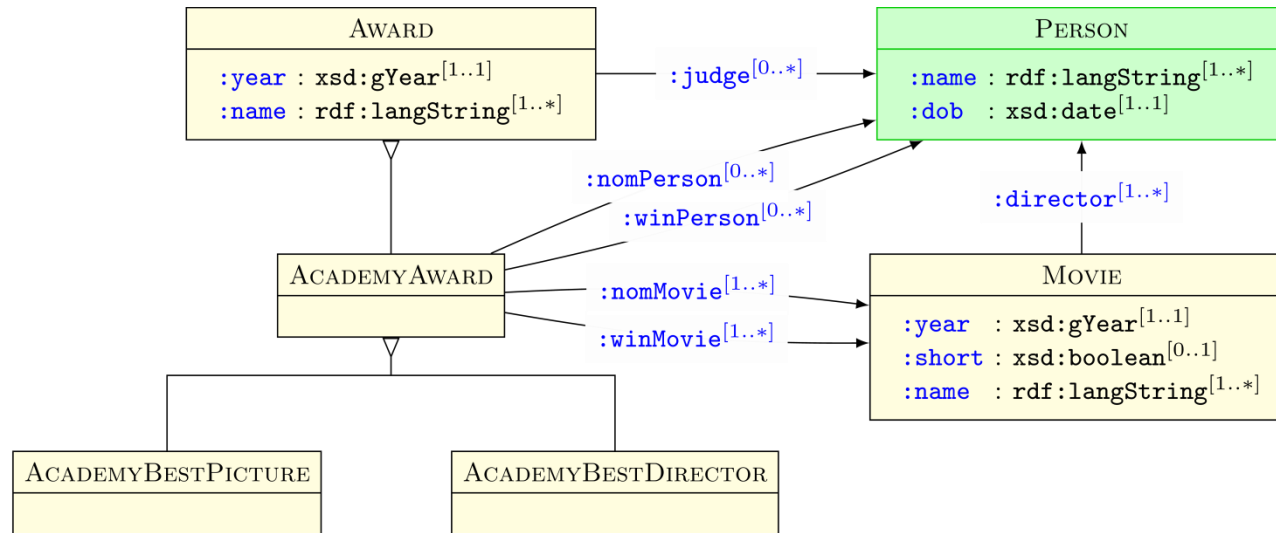
Test Suite:

[SHACL Test Suite](#)

SHACL: SHAPES GRAPH



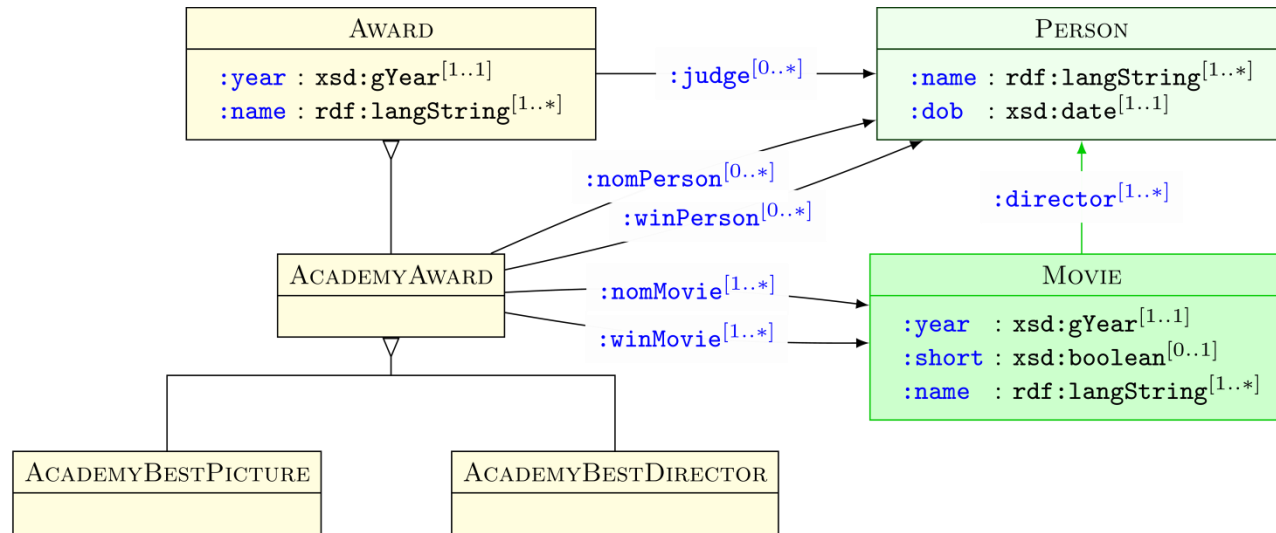
SHACL: NODE AND PROPERTY SHAPES



```
@prefix : <http://ex.org/data/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix s: <http://ex.org/shapes/> .
@prefix sh: <http://www.w3.org/ns/shacl#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.

s:Person a sh:NodeShape ;
  sh:property [ sh:path :name ; sh:datatype rdf:langString ; sh:minCount 1 ] ;
  sh:property [ sh:path :dob ; sh:datatype xsd:date ; sh:minCount 1 ; sh:maxCount 1 ] .
```

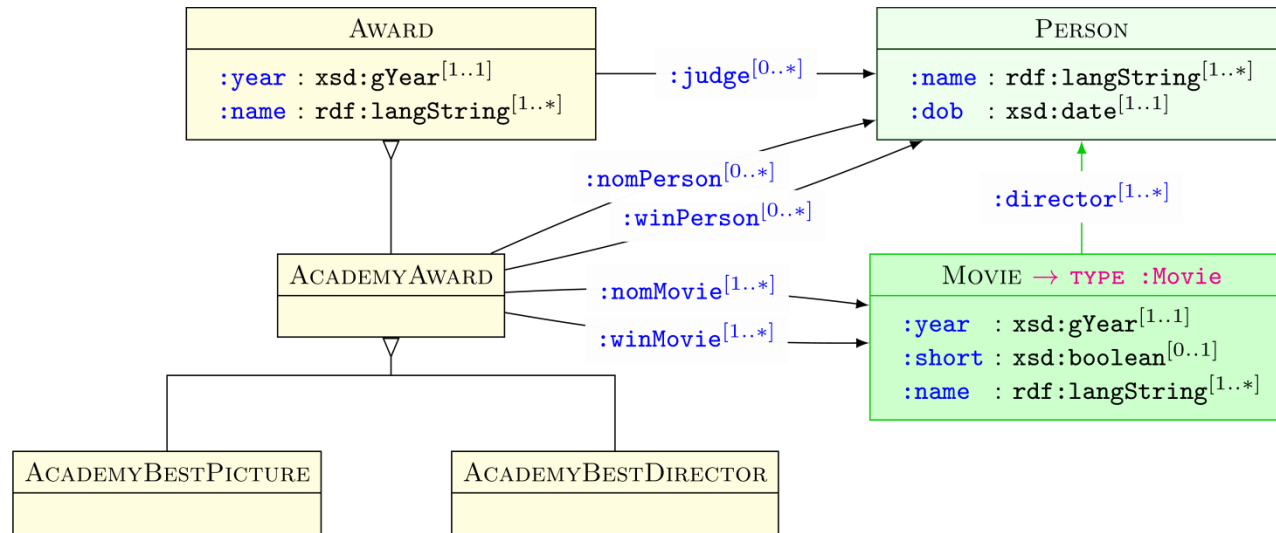
SHACL: REFERENCING NODE SHAPES



```
#[...]
s:Person a sh:NodeShape ; #[...]

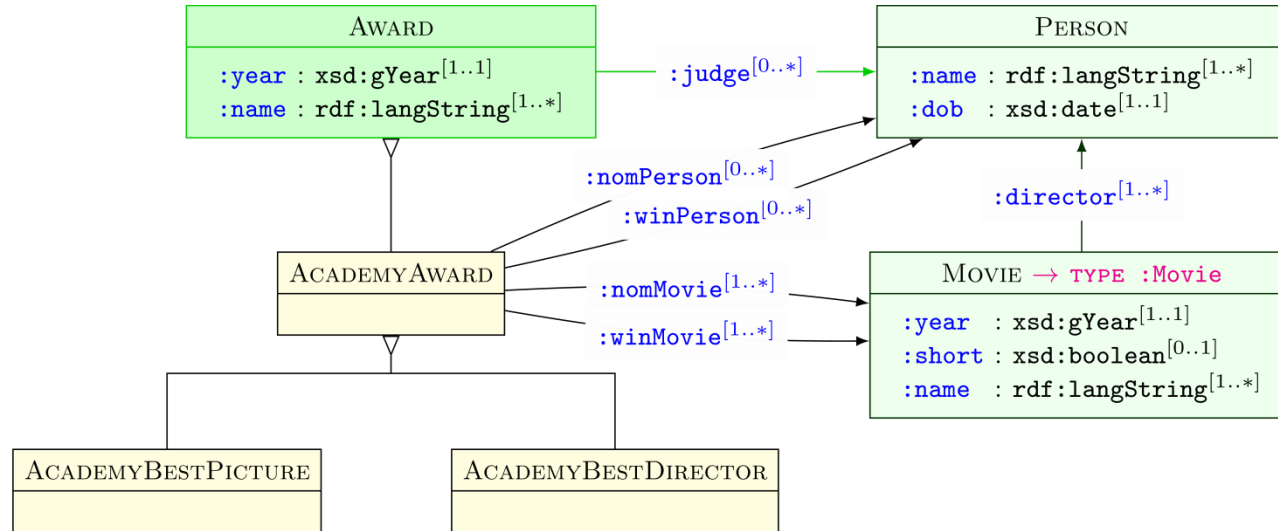
s:Movie a sh:NodeShape ;
  sh:property [ sh:path :year ; sh:datatype xsd:gYear ; sh:maxCount 1 ; sh:minCount 1 ] ;
  sh:property [ sh:path :short ; sh:datatype xsd:boolean ; sh:maxCount 1 ] ;
  sh:property [ sh:path :name ; sh:datatype rdf:langString ; sh:minCount 1 ] ;
  sh:property [ sh:path :director ; sh:node s:Person ; sh:minCount 1 ] .
```


SHACL: TARGETS



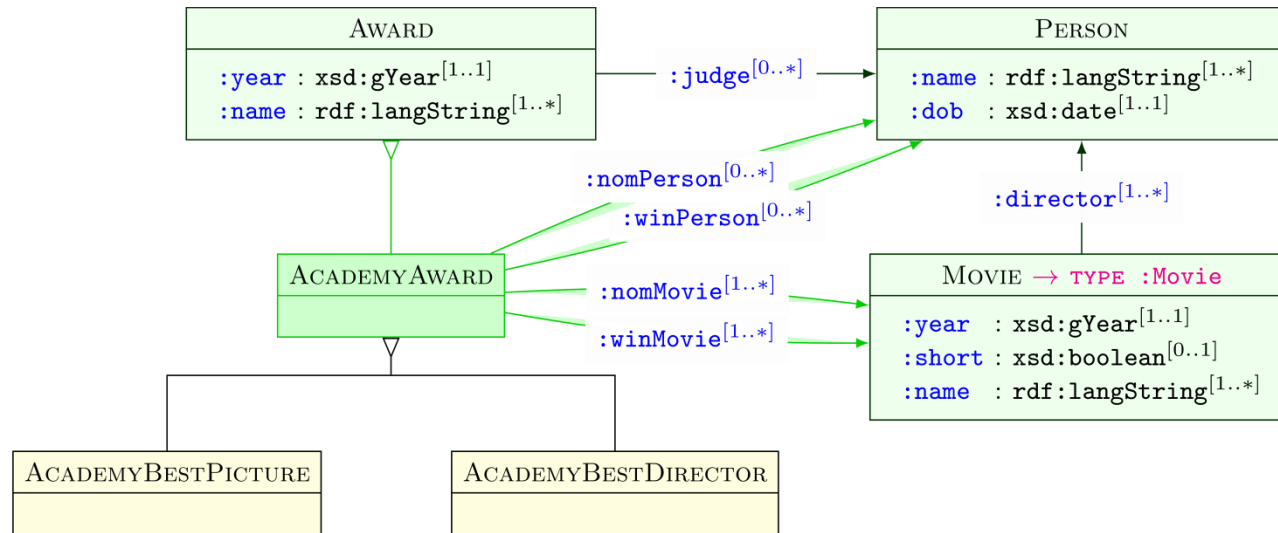
```
#[...]  
s:Person a sh:NodeShape ; #[...]  
  
s:Movie a sh:NodeShape ;  
  sh:targetClass :Movie ;  
  sh:property [ sh:path :year ; sh:datatype xsd:gYear ; sh:maxCount 1 ; sh:minCount 1 ] ;  
  sh:property [ sh:path :short ; sh:datatype xsd:boolean ; sh:maxCount 1 ] ;  
  sh:property [ sh:path :name ; sh:datatype rdf:langString ; sh:minCount 1 ] ;  
  sh:property [ sh:path :director ; sh:node s:Person ; sh:minCount 1 ] .
```

SHACL: INHERITANCE



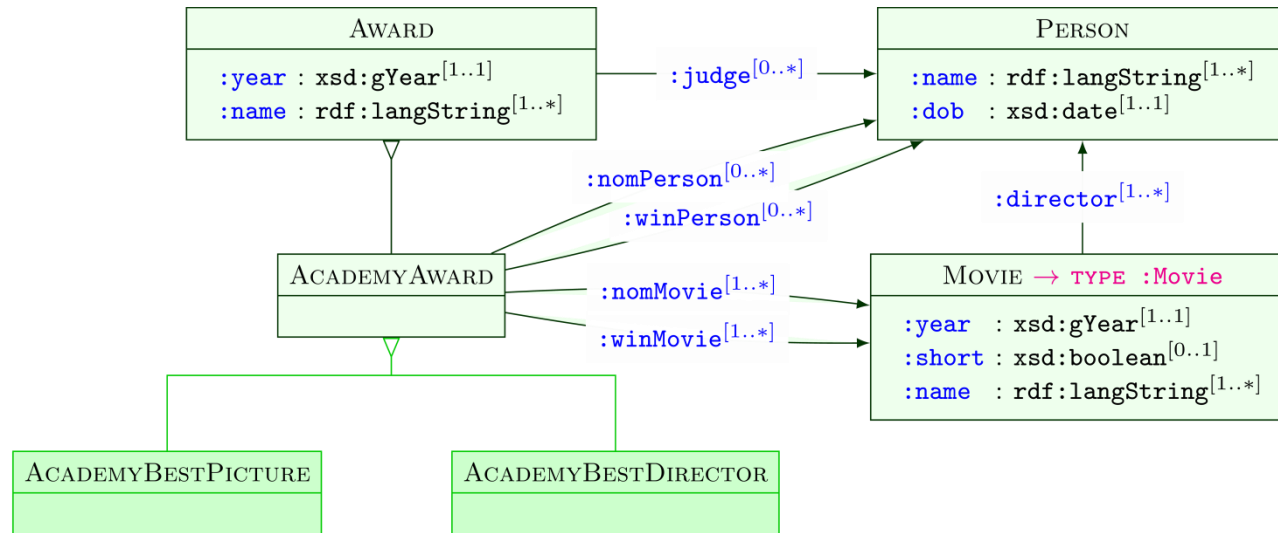
```
#[...]  
s:Person a sh:NodeShape ; #[...]  
s:Movie a sh:NodeShape ; #[...]  
  
s:Award a sh:NodeShape ;  
  sh:property [ sh:path :year ; sh:datatype xsd:gYear ; sh:maxCount 1 ; sh:minCount 1 ] ;  
  sh:property [ sh:path :name ; sh:datatype rdf:langString ; sh:minCount 1 ] ;  
  sh:property [ sh:path :judge ; sh:node s:Person ] .
```

SHACL: INHERITANCE



```
#[...]  
s:Person a sh:NodeShape ; #[...]  
s:Movie a sh:NodeShape ; #[...]  
s:Award a sh:NodeShape ; #[...]  
  
s:AcademyAward a sh:NodeShape ;  
  sh:node s:Award ;  
  sh:property [ sh:path :nomPerson ; sh:node s:Person ] ;  
  sh:property [ sh:path :winPerson ; sh:node s:Person ] ;  
  sh:property [ sh:path :nomMovie ; sh:node s:Movie ; sh:minCount 1 ] ;  
  sh:property [ sh:path :winMovie ; sh:node s:Movie ; sh:minCount 1 ] .
```

SHACL: INHERITANCE



```
#[...]
s:Person a sh:NodeShape ; #[...]
s:Movie a sh:NodeShape ; #[...]
s:Award a sh:NodeShape ; #[...]
s:AcademyAward a sh:NodeShape ; #[...]

s:AcademyBestPicture a sh:NodeShape ;
  sh:node s:AcademyAward .

s:AcademyBestDirector a sh:NodeShape ;
  sh:node s:AcademyAward .
```

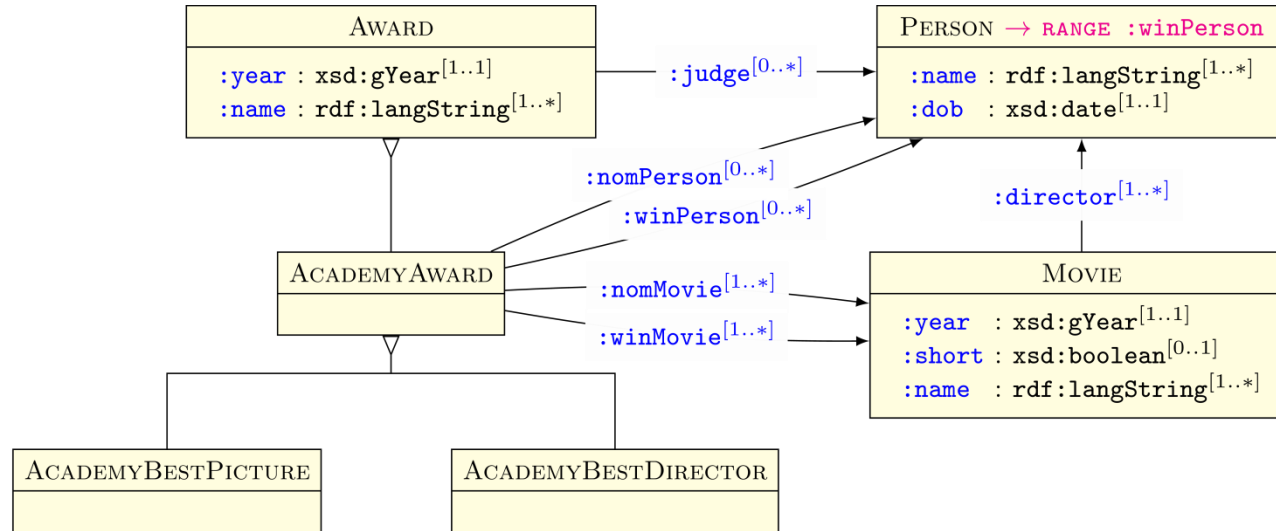
SHACL:

TARGETS

SHACL: TARGETS

- `sh:targetClass` instances of a class
- `sh:targetSubjectsOf` domain of a property
- `sh:targetObjectsOf` range of a property
- `sh:targetNode` a specific node

SHACL: TARGETING RANGE

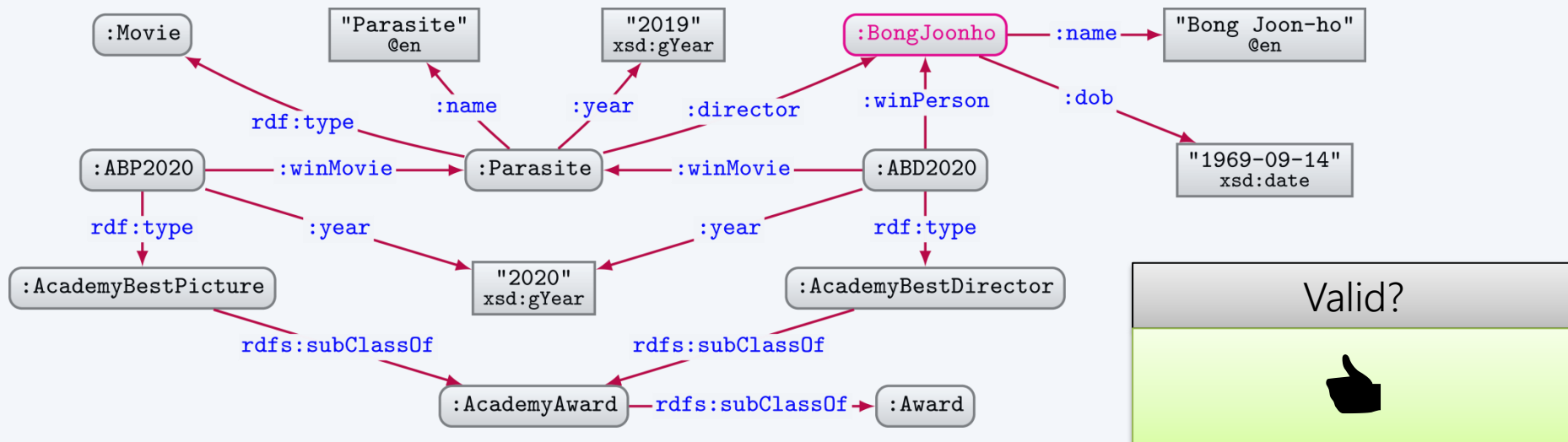
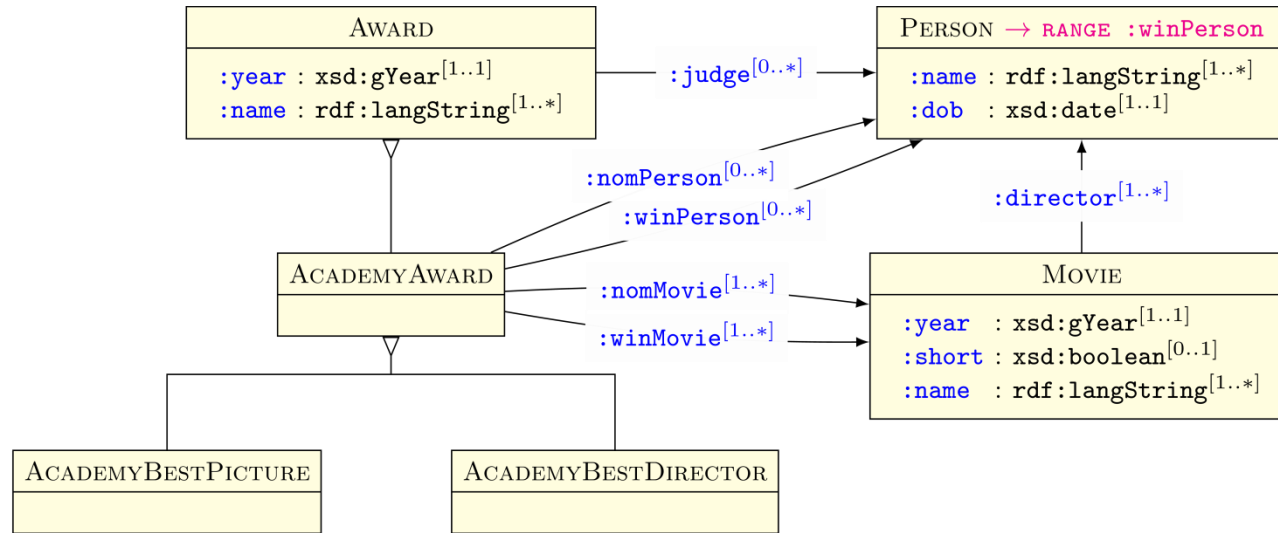


```
@prefix : <http://ex.org/data/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix s: <http://ex.org/shapes/> .
@prefix sh: <http://www.w3.org/ns/shacl#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.

s:Person a sh:NodeShape ;
  sh:targetObjectsOf :winPerson ;
  sh:property [ sh:path :name ; sh:datatype rdf:langString ; sh:minCount 1 ] ;
  sh:property [ sh:path :dob ; sh:datatype xsd:date ; sh:minCount 1 ; sh:maxCount 1 ] .

#[...]
```

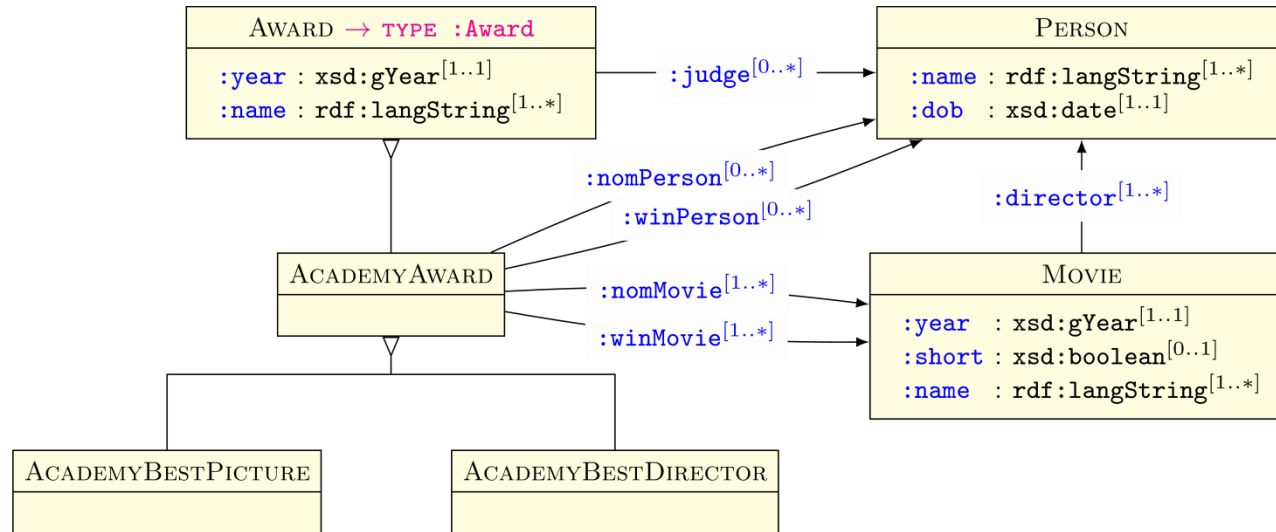
SHACL: TARGETING RANGE



Valid?



SHACL: TARGETING SUBCLASSES

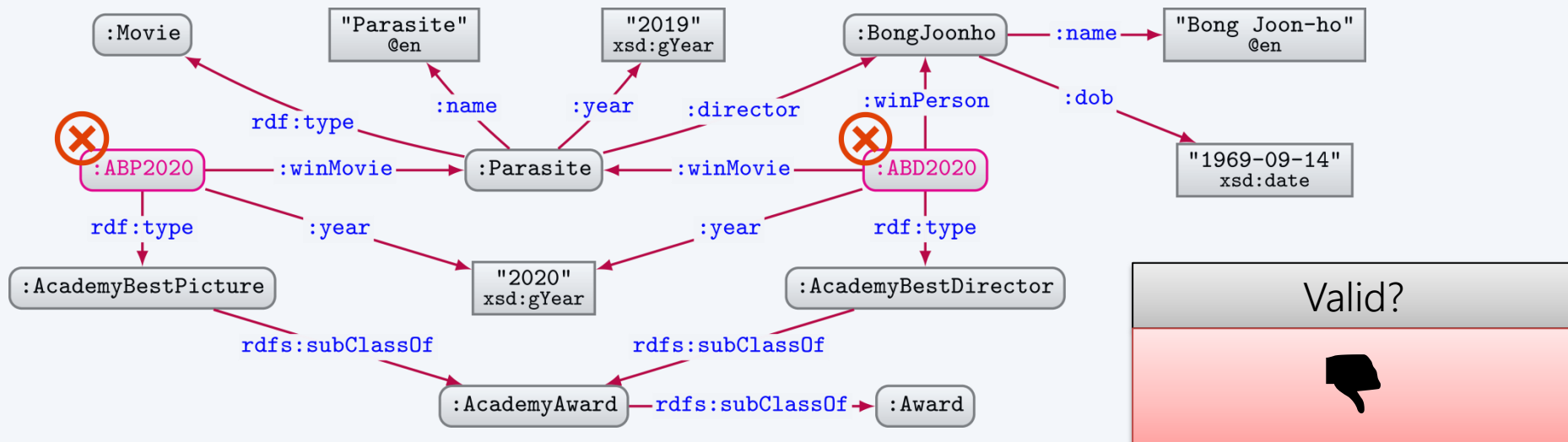
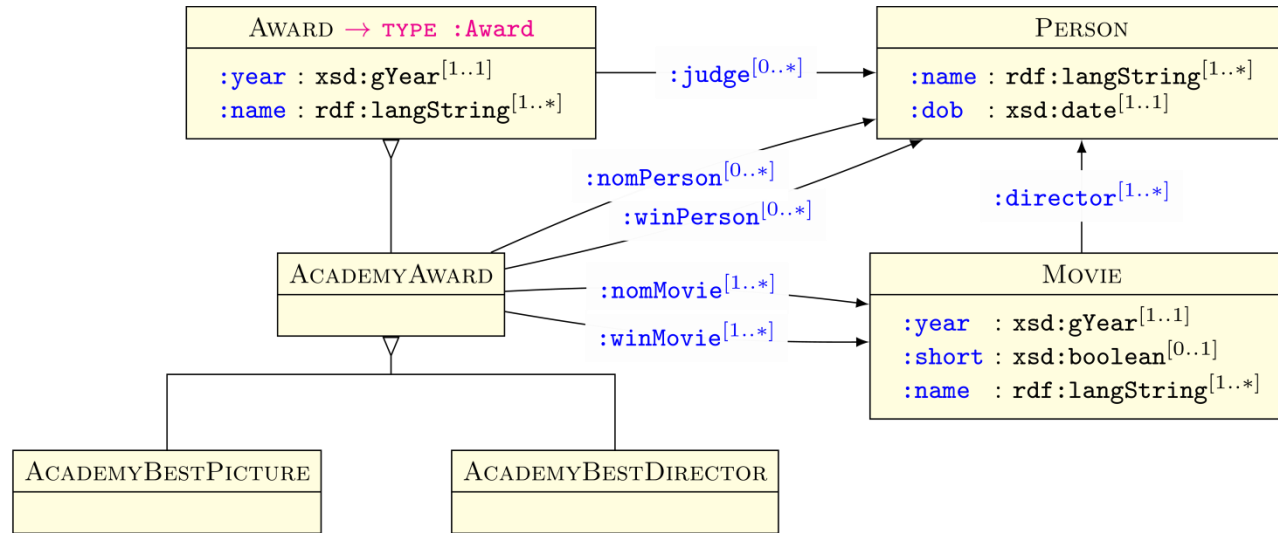


```
#[...]
s:Person a sh:NodeShape ; #[...]
s:Movie a sh:NodeShape ; #[...]

s:Award a sh:NodeShape ;
  sh:targetClass :Award ;
  sh:property [ sh:path :year ; sh:datatype xsd:gYear ; sh:maxCount 1 ; sh:minCount 1 ] ;
  sh:property [ sh:path :name ; sh:datatype rdf:langString ; sh:minCount 1 ] ;
  sh:property [ sh:path :judge ; sh:node s:Person ] .

#[...]
```

SHACL: TARGETS SUBCLASSES



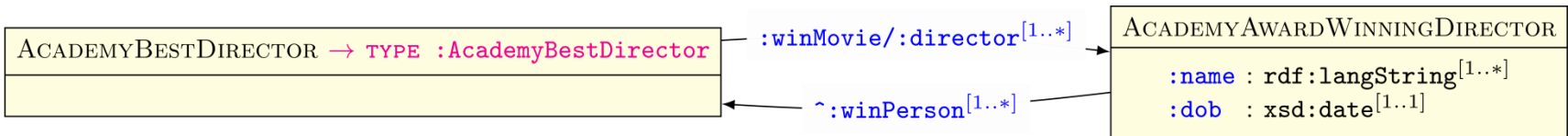
SHACL:

PATHS

SHACL: PATHS

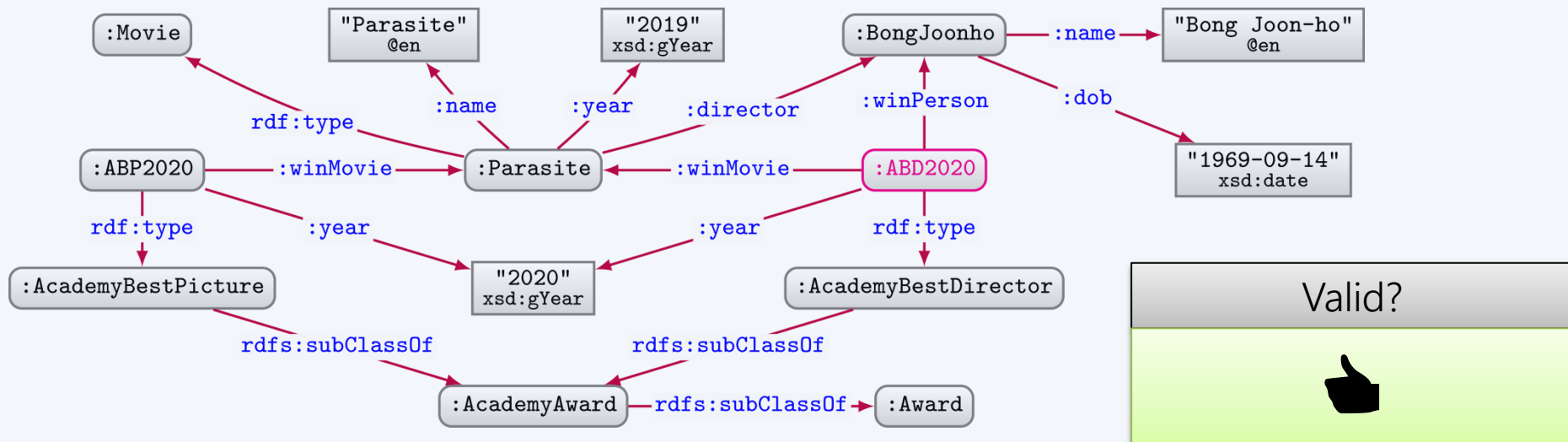
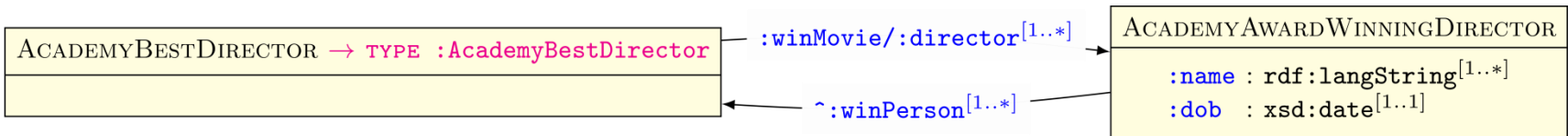
- `:p` property `(:p)`
- `[sh:inversePath e]` inverse of *e* `(^e)`
- `(e f)` *e* then *f* `(e/f)`
- `[sh:alternativePath (e f)]` *e* or *f* `(e|f)`
- `[sh:zeroOrMorePath e]` recursive *e* `(e*`)
- `[sh:oneOrMorePath e]` recursive+ *e* `(e+`)
- `[sh:zeroOrOnePath e]` optional *e* `(e?)`

SHACL: PATHS

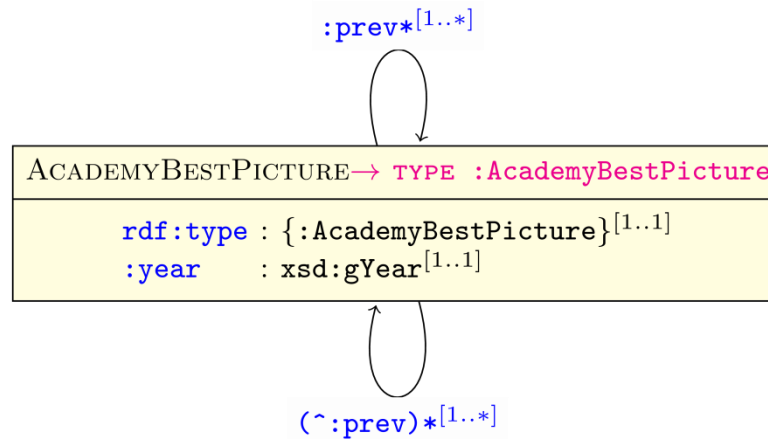


```
#[...]  
  
s:AcademyBestDirector a sh:NodeShape ;  
  sh:targetClass :AcademyBestDirector .  
  sh:property [  
    sh:path ( :winMovie :director ) ;  
    sh:node s:AcademyAwardWinningDirector ; sh:minCount 1  
  ] .  
  
s:AcademyAwardWinningDirector a sh:NodeShape ;  
  sh:property [ sh:path :name ; sh:datatype rdf:langString ; sh:minCount 1 ] ;  
  sh:property [ sh:path :dob ; sh:datatype xsd:date ; sh:minCount 1 ; sh:maxCount 1 ] ;  
  sh:property [  
    sh:path [ sh:inversePath :winPerson ] ;  
    sh:node s:AcademyBestDirector ; sh:minCount 1  
  ] .  
  
#[...]
```

SHACL: PATHS

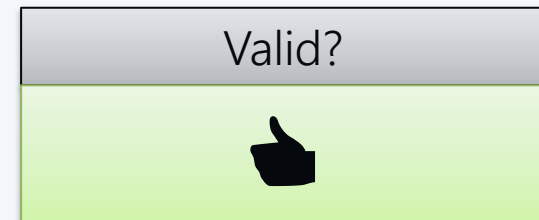
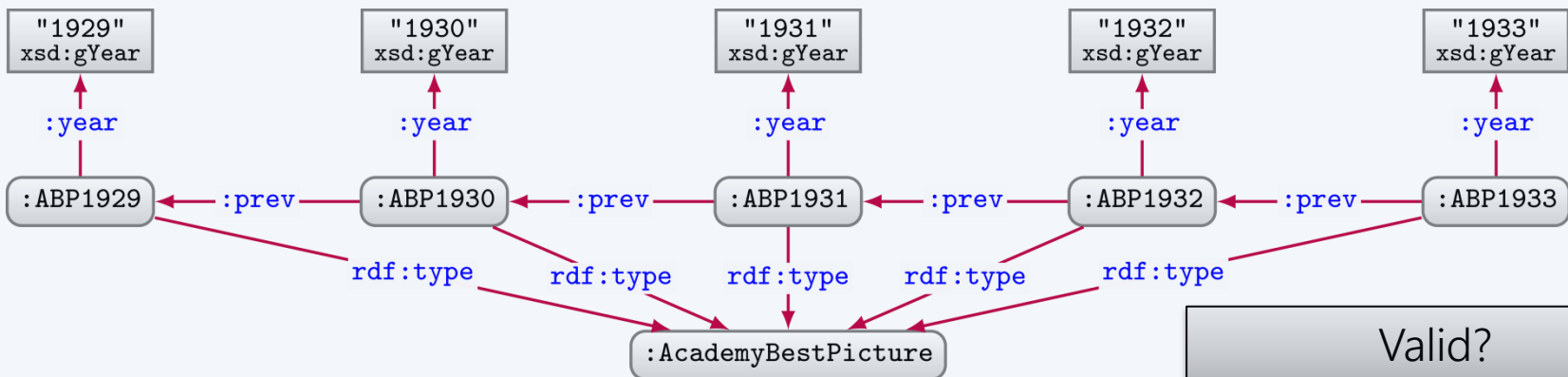
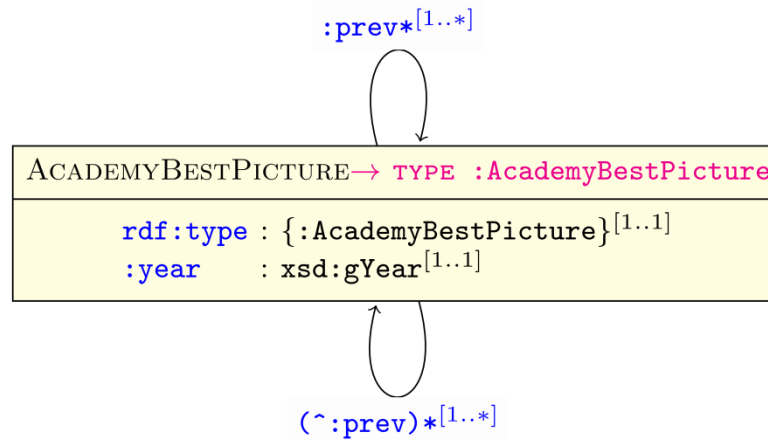


SHACL: PATHS



```
#[...]  
  
s:AcademyBestPicture a sh:NodeShape ;  
  sh:targetClass :AcademyBestPicture ;  
  sh:class :AcademyBestPicture ;  
  sh:property [ sh:path :year ; sh:datatype xsd:gYear ] ;  
  sh:property [  
    sh:path [ sh:zeroOrMorePath :prev ] ;  
    sh:node s:AcademyBestPicture ; sh:minCount 1  
  ] ;  
  sh:property [  
    sh:path [ sh:zeroOrMorePath [ sh:inversePath :prev ] ] ;  
    sh:node s:AcademyBestPicture ; sh:minCount 1  
  ] .  
  
#[...]
```

SHACL: PATHS



SHACL:

CORE CONSTRAINTS

SHACL: SHAPE CONSTRAINTS

Given ϕ a shape and e a path ...

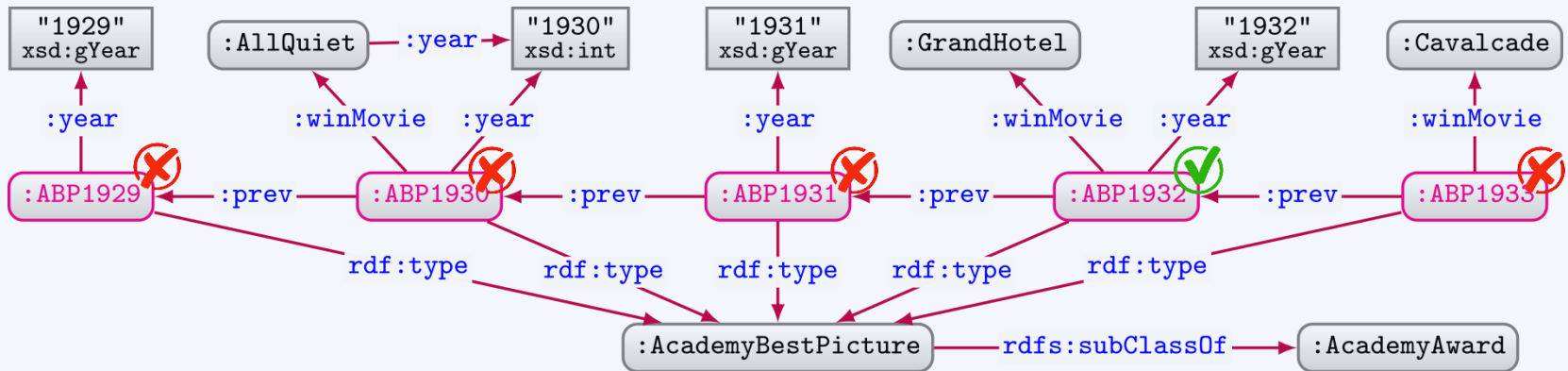
- `sh:node ϕ` all target nodes in N satisfy shape ϕ
- `sh:property ϕ, e` all value nodes V_n^e connected by path e from each target node $n \in N$ satisfy shape ϕ

```
#[...]
s:Movie a sh:NodeShape ;
  sh:property [ sh:path :year ; sh:datatype xsd:gYear ] .

s:Award a sh:NodeShape ;
  sh:property [ sh:path :year ; sh:datatype xsd:gYear ; sh:minCount 1 ] .

s:AcademyAward a sh:NodeShape ;
  sh:targetClass :AcademyAward ;
  sh:node s:Award ;
  sh:property [ sh:path :winMovie ; sh:node s:Movie ; sh:minCount 1 ] .
```

SHACL: SHAPE CONSTRAINTS



```
#[...]  
s:Movie a sh:NodeShape ;  
  sh:property [ sh:path :year ; sh:datatype xsd:gYear ] .  
  
s:Award a sh:NodeShape ;  
  sh:property [ sh:path :year ; sh:datatype xsd:gYear ; sh:minCount 1 ] .  
  
s:AcademyAward a sh:NodeShape ;  
  sh:targetClass :AcademyAward ;  
  sh:node s:Award ;  
  sh:property [ sh:path :winMovie ; sh:node s:Movie ; sh:minCount 1 ] .
```

SHACL: BOOLEAN CONSTRAINTS

Assuming that $\phi, \phi_1, \dots, \phi_n$ are shapes then ...

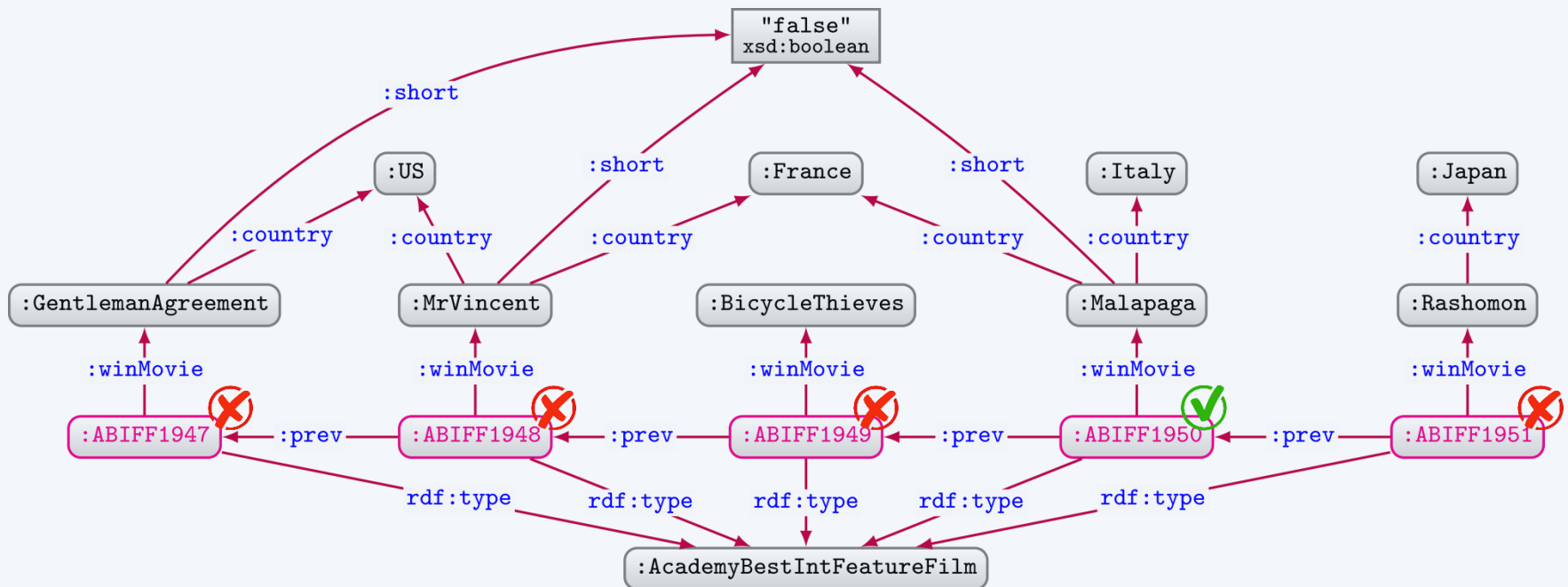
- `sh:not ϕ` negation ($\neg\phi$)
- `sh:or ($\phi_1 \dots \phi_n$)` disjunction ($\phi_1 \vee \dots \vee \phi_n$)
- `sh:and ($\phi_1 \dots \phi_n$)` conjunction ($\phi_1 \wedge \dots \wedge \phi_n$)
- `sh:xone ($\phi_1 \dots \phi_n$)` excl. disjunction ($\phi_1 \oplus \dots \oplus \phi_n$)

... are also shapes.

```
#[...]
s:FromUS a sh:NodeShape ; sh:property [ sh:path :country ; sh:hasValue :US ] .
s:FeatureMovie a sh:NodeShape ; sh:property [ sh:path :short ; sh:hasValue false ] .

s:AcademyBestIntFeatureFilm a sh:NodeShape ;
  sh:targetClass :AcademyBestIntFeatureFilm ;
  sh:property [
    sh:path :winMovie ;
    sh:and ( [ sh:not s:FromUS ] s:FeatureMovie ) ;
    sh:minCount 1 ; sh:maxCount 1
  ] .
```

SHACL: BOOLEAN CONSTRAINTS



```
#[...]
s:FromUS a sh:NodeShape ; sh:property [ sh:path :country ; sh:hasValue :US ] .
s:FeatureMovie a sh:NodeShape ; sh:property [ sh:path :short ; sh:hasValue false ] .

s:AcademyBestIntFeatureFilm a sh:NodeShape ;
  sh:targetClass :AcademyBestIntFeatureFilm ;
  sh:property [
    sh:path :winMovie ;
    sh:and ( [ sh:not s:FromUS ] s:FeatureMovie ) ;
    sh:minCount 1 ; sh:maxCount 1
  ] .
```

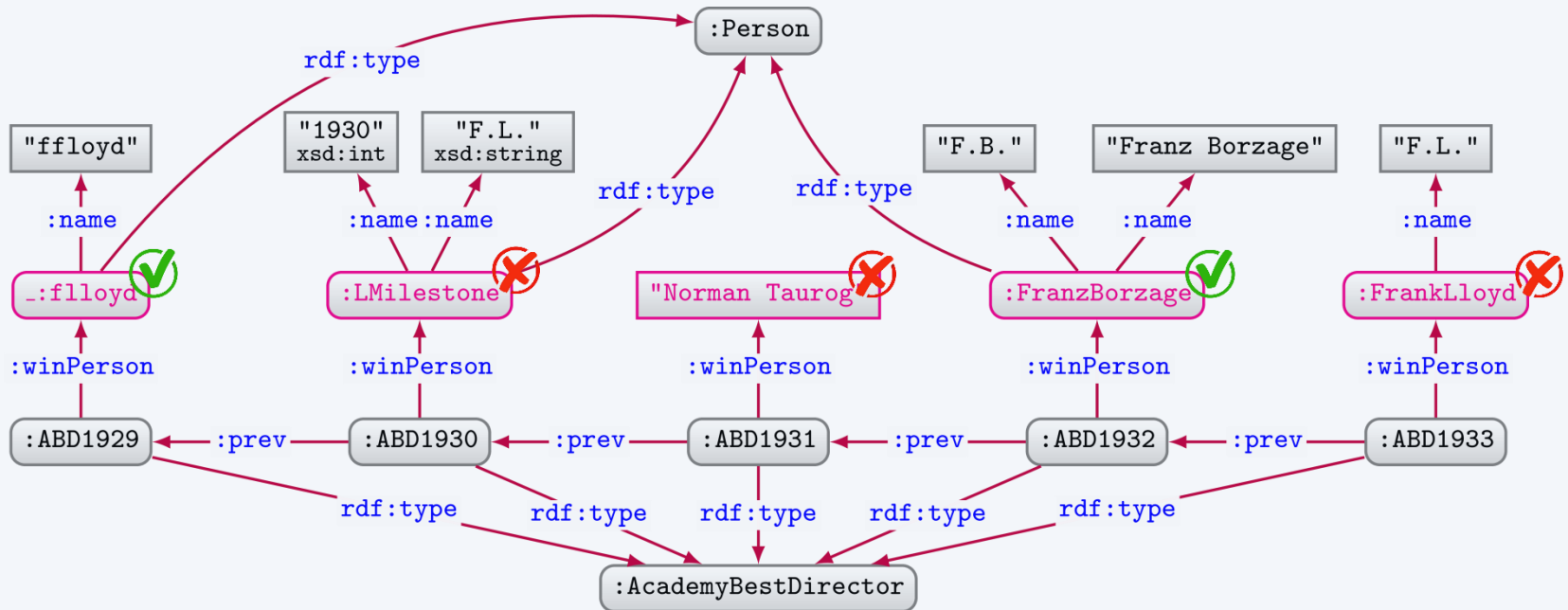
SHACL: VALUE TYPE CONSTRAINTS

For each node $n \in N$ it holds that:

- `sh:class` C n is an instance of class C
- `sh:datatype` D n has the datatype D
- `sh:nodeKind` K n is of kind K
 - K can be: `sh:BlankNode`, `sh:IRI`, `sh:Literal`, `sh:BlankNodeOrIRI`, `sh:BlankNodeOrLiteral`, `sh:IRIOrLiteral`

```
#[...]  
  
s:Person a sh:NodeShape ;  
  sh:targetObjectsOf :winPerson ;  
  sh:class :Person ;  
  sh:nodeKind sh:BlankNodeOrIRI ;  
  sh:property [ sh:path :name ; sh:datatype xsd:string ; sh:minCount 1 ] .
```

SHACL: VALUE TYPE CONSTRAINTS



```
#[...]
```

```
s:Person a sh:NodeShape ;  
  sh:targetObjectsOf :winPerson ;  
  sh:class :Person ;  
  sh:nodeKind sh:BlankNodeOrIRI ;  
  sh:property [ sh:path :name ; sh:datatype xsd:string ; sh:minCount 1 ] .
```

SHACL: VALUE RANGE CONSTRAINTS

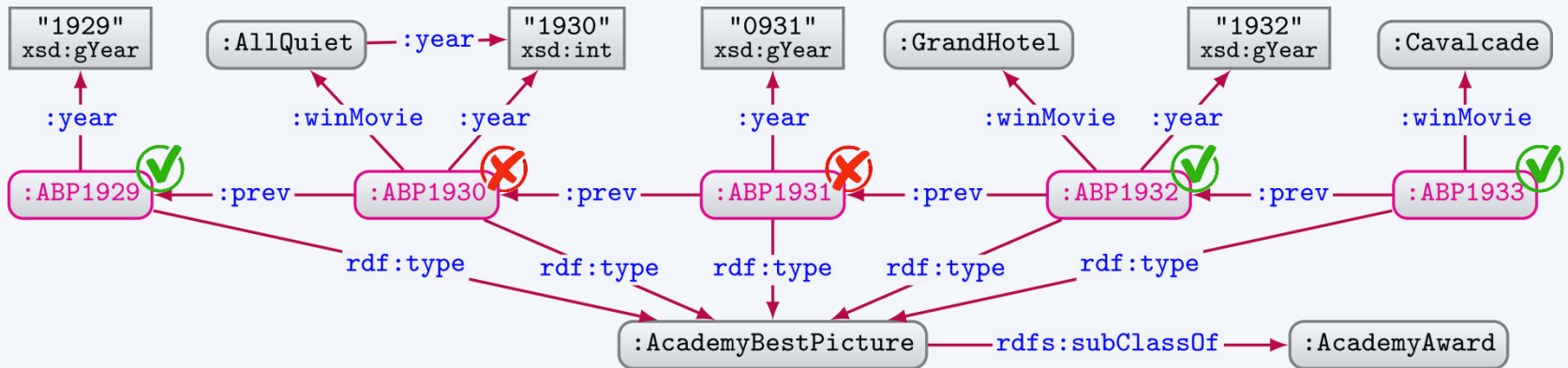
For each node $n \in N$ it holds that:

- `sh:minInclusive m` $n \geq m$
- `sh:maxInclusive m` $n \leq m$
- `sh:minExclusive m` $n > m$
- `sh:maxExclusive m` $n \leq m$

```
#[...]
```

```
s:AcademyBestPicture a sh:NodeShape ;  
  sh:targetClass :AcademyBestPicture ;  
  sh:property [ sh:path :year ; sh:datatype xsd:gYear ; sh:minInclusive "1929"^^xsd:gYear ] .
```


SHACL: VALUE RANGE CONSTRAINTS



```
#[...]
```

```
s:AcademyBestPicture a sh:NodeShape ;  
  sh:targetClass :AcademyBestPicture ;  
  sh:property [ sh:path :year ; sh:datatype xsd:gYear ; sh:minInclusive "1929"^^xsd:gYear ] .
```

SHACL: VALUE RANGE CONSTRAINTS

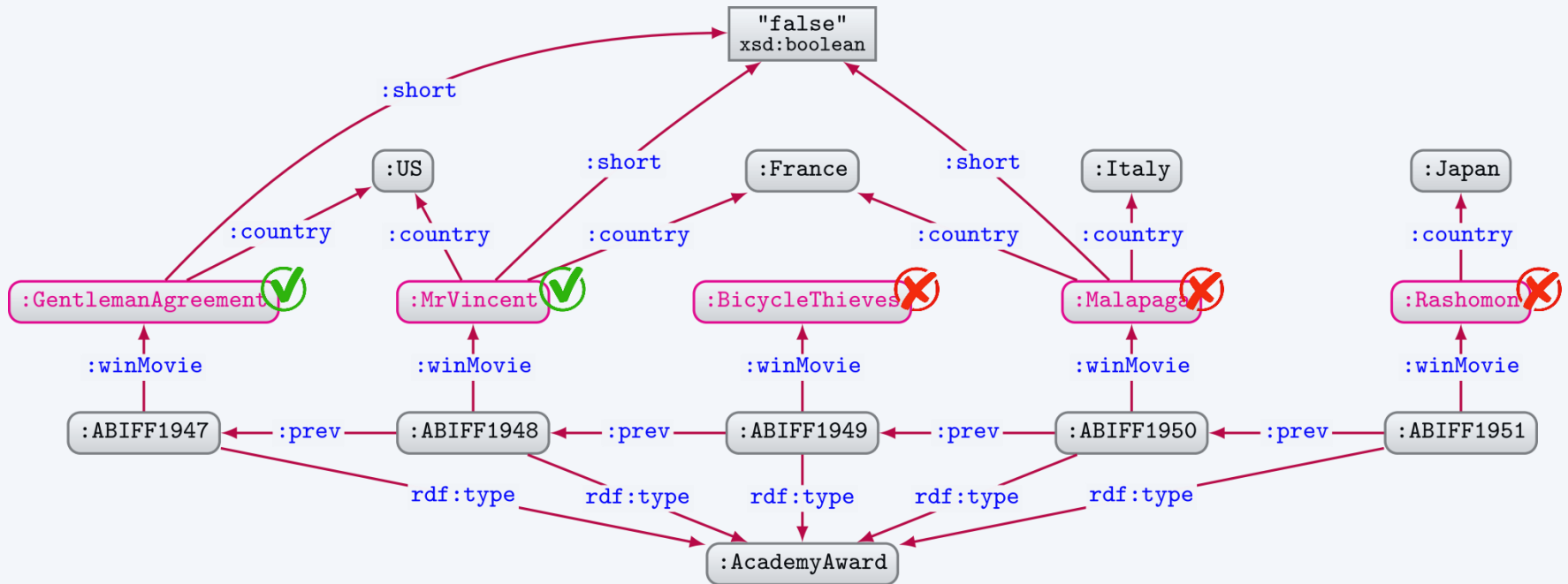
For the set of nodes N it holds that:

- `sh:hasValue v` $v \in N$
- `sh:in ($v_1 \dots v_n$)` $N \subseteq \{ v_1, \dots, v_n \}$

```
#[...]
```

```
s:USMovie a sh:NodeShape ;  
  sh:targetObjectsOf :winMovie ;  
  sh:property [ sh:path :country ; sh:hasValue :US ] .
```

SHACL: VALUE RANGE CONSTRAINTS



```
# [...]
```

```
s:USMovie a sh:NodeShape ;  
  sh:targetObjectsOf :winMovie ;  
  sh:property [ sh:path :country ; sh:hasValue :US ] .
```

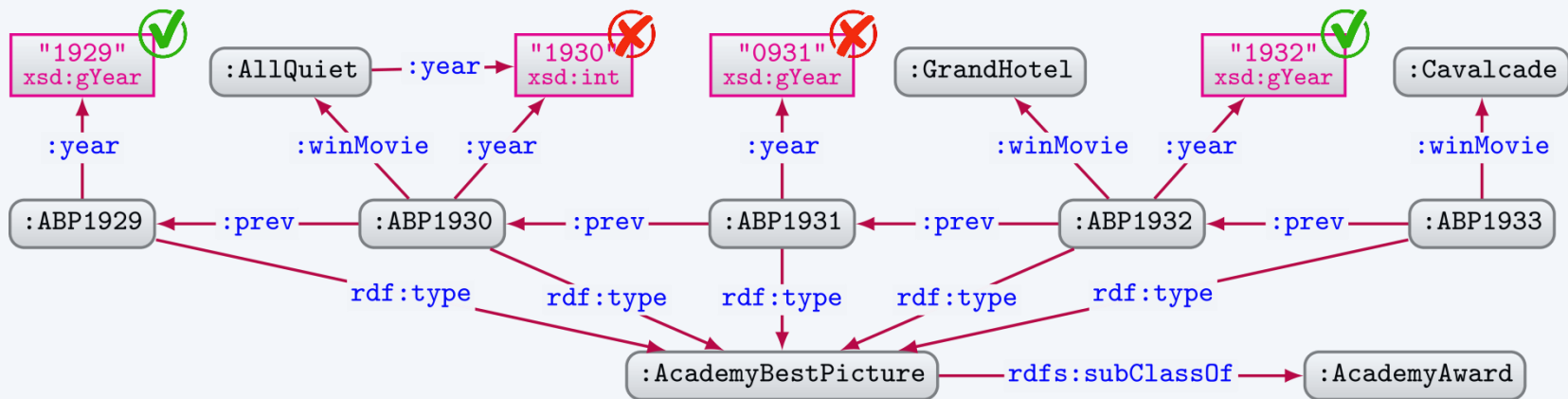
SHACL: STRING-BASED CONSTRAINTS

Each node $n \in N$ is not a blank node and:

- `sh:minLength m` has string length $\geq m$
- `sh:maxLength m` has string length $\leq m$
- `sh:pattern p, f` matches regex p with flags f

```
#[...]  
  
s:ModernYear a sh:NodeShape ;  
  sh:targetObjectsOf :year ;  
  sh:datatype xsd:gYear ;  
  sh:minLength 4 ;  
  sh:maxLength 4 ;  
  sh:pattern "(19|20)[0-9][0-9]" ;  
  sh:flags "i" . # case insensitive pattern (just for the example)
```

SHACL: STRING-BASED CONSTRAINTS



```
#[...]  
  
s:ModernYear a sh:NodeShape ;  
  sh:targetObjectsOf :year ;  
  sh:datatype xsd:gYear ;  
  sh:minLength 4 ;  
  sh:maxLength 4 ;  
  sh:pattern "(19|20)[0-9][0-9]" ;  
  sh:flags "i" . # case insensitive pattern (just for the example)
```

SHACL: STRING-BASED CONSTRAINTS

Each node $n \in N$ is a literal and:

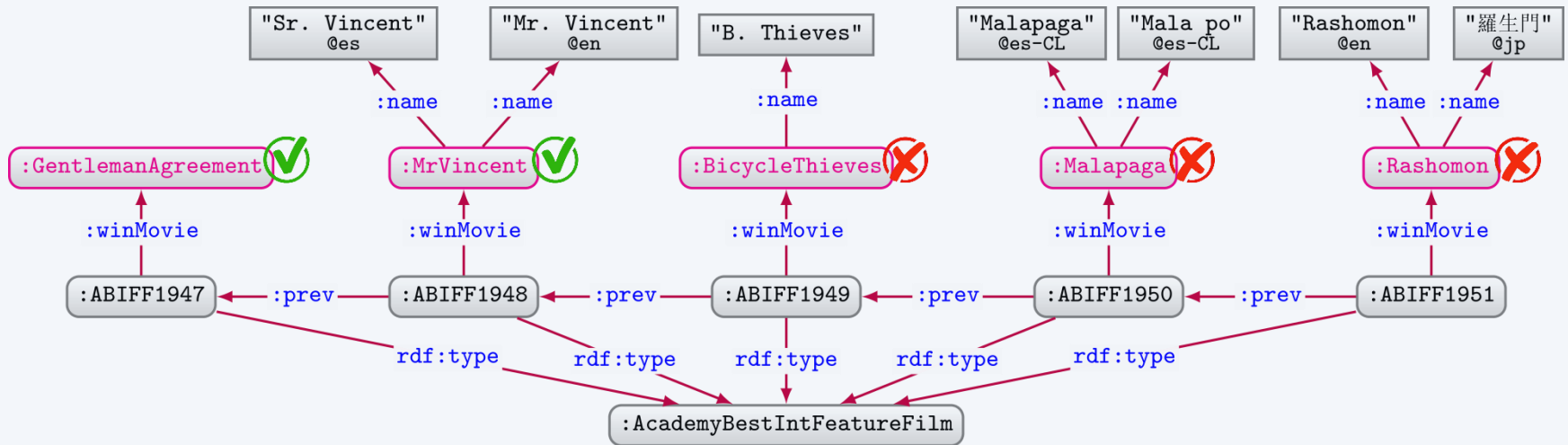
- `sh:languageIn ($l_1 \dots l_n$)` has a lang. tag matching $l \in \{l_1, \dots, l_n\}$

Each node $n \in N$:

- `sh:uniqueLang true` has no lang. tag, an empty lang. tag or or a lang. tag unique in N

```
#[...]  
  
s:MovieHasUniqueEnglishSpanishName a sh:NodeShape ;  
  sh:targetObjectsOf :winMovie ;  
  sh:property [  
    sh:path :name ;  
    sh:languageIn ( "es" "en" ) ; # will also match "es-CL", etc.  
    sh:uniqueLang true  
  ] .
```

SHACL: STRING-BASED CONSTRAINTS



```
#[...]
```

```
s:MovieHasUniqueEnglishSpanishName a sh:NodeShape ;  
  sh:targetObjectsOf :winMovie ;  
  sh:property [  
    sh:path :name ;  
    sh:languageIn ( "es" "en" ) ; # will also match "es-CL", etc.  
    sh:uniqueLang true  
  ] .
```

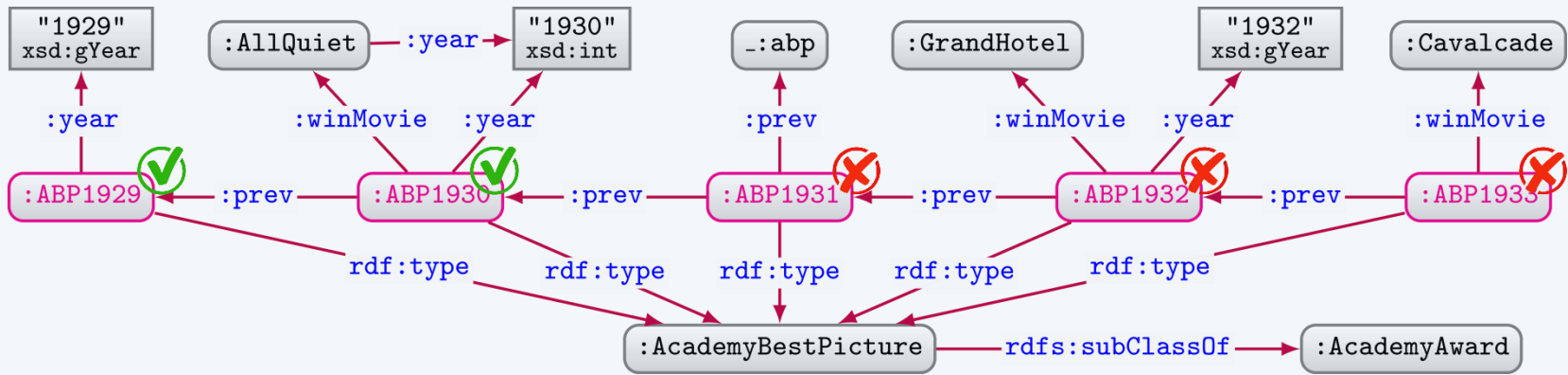
SHACL: CARDINALITY CONSTRAINTS

Given a path e , for each node $n \in N$, there are ...

- `sh:maxCount m` at most m nodes in V_n^e
- `sh:minCount m` at least m nodes in V_n^e
- `sh:qualifiedMinCount m, ϕ` at least m nodes in V_n^e that satisfy ϕ
- `sh:qualifiedMaxCount m, ϕ` at most m nodes in V_n^e that satisfy ϕ

```
#[...]  
  
s:FirstInSeries a sh:NodeShape ;  
  sh:property [ sh:path :prev ; sh:maxCount 0 ] .  
  
s:AcademyBestPicture a sh:NodeShape ;  
  sh:targetClass :AcademyBestPicture ;  
  sh:property [  
    sh:path [ sh:zeroOrMorePath :prev ] ;  
    sh:qualifiedMaxCount 1 ;  
    sh:qualifiedMinCount 1 ;  
    sh:qualifiedValueShape s:FirstInSeries  
  ] .
```


SHACL: CARDINALITY CONSTRAINTS



```
#[...]  
  
s:FirstInSeries a sh:NodeShape ;  
  sh:property [ sh:path :prev ; sh:maxCount 0 ] .  
  
s:AcademyBestPicture a sh:NodeShape ;  
  sh:targetClass :AcademyBestPicture ;  
  sh:property [  
    sh:path [ sh:zeroOrMorePath :prev ] ;  
    sh:qualifiedMaxCount 1 ;  
    sh:qualifiedMinCount 1 ;  
    sh:qualifiedValueShape s:FirstInSeries  
  ] .
```

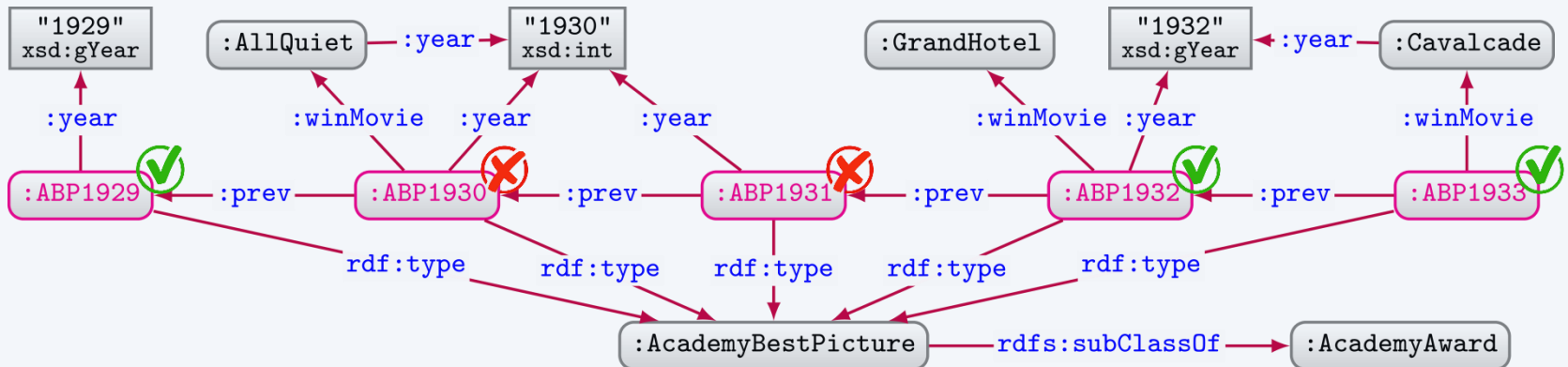
SHACL: PROPERTY-PAIR CONSTRAINTS

Given a path e , for each target node $n \in N$, it holds that ...

- `sh:equals p` $V_n^e = V_n^f$
- `sh:disjoint p` $V_n^e \cap V_n^f = \emptyset$
- `sh:lessThan p` $\max(V_n^e) < \min(V_n^f)$
- `sh:lessThanOrEquals p` $\max(V_n^e) \leq \min(V_n^f)$

```
#[...]  
  
s:AcademyBestPicture a sh:NodeShape ;  
  sh:targetClass :AcademyBestPicture ;  
  sh:property [  
    sh:path ( :winMovie :year ) ;  
    sh:lessThan :year  
  ] ;  
  sh:property [  
    sh:path [  
      [ sh:alternativePath (  
        [ sh:oneOrMorePath [ sh:inversePath :prev ] ] [ sh:oneOrMorePath :prev ]  
      ) ]  
      :year  
    ] ;  
    sh:disjoint :year  
  ] .
```

SHACL: PROPERTY-PAIR CONSTRAINTS



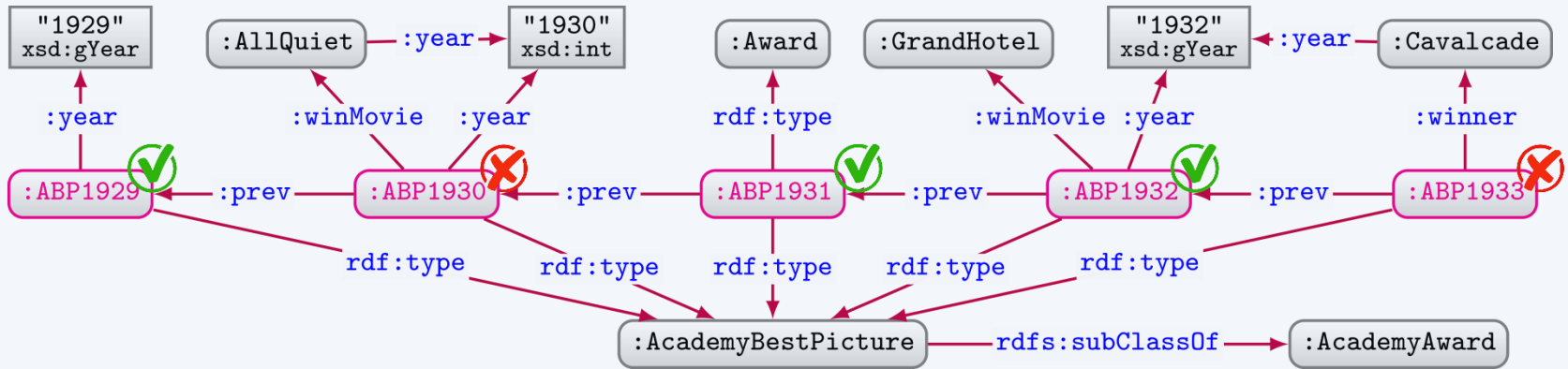
```
#[...]  
  
s:AcademyBestPicture a sh:NodeShape ;  
  sh:targetClass :AcademyBestPicture ;  
  sh:property [  
    sh:path ( :winMovie :year ) ;  
    sh:lessThan :year  
  ] ;  
  sh:property [  
    sh:path [  
      [ sh:alternativePath (  
        [ sh:oneOrMorePath [ sh:inversePath :prev ] ] [ sh:oneOrMorePath :prev ]  
      ) ]  
    :year  
  ] ;  
  sh:disjoint :year  
] .
```

SHACL: CLOSED CONSTRAINTS

- `sh:closed true` only properties in shapes graph allowed
 - `sh:ignoredProperties` optional list of exceptions

```
#[...]
s:AcademyAward a sh:NodeShape ;
  sh:targetClass :AcademyAward ;
  sh:property [ sh:path :winMovie ; sh:nodeKind sh:IRI ] ;
  sh:property [ sh:path :year ; sh:datatype xsd:gYear ] ;
  sh:closed true ; sh:ignoredProperties ( :prev rdf:type ) .
```

SHACL: CLOSED CONSTRAINTS

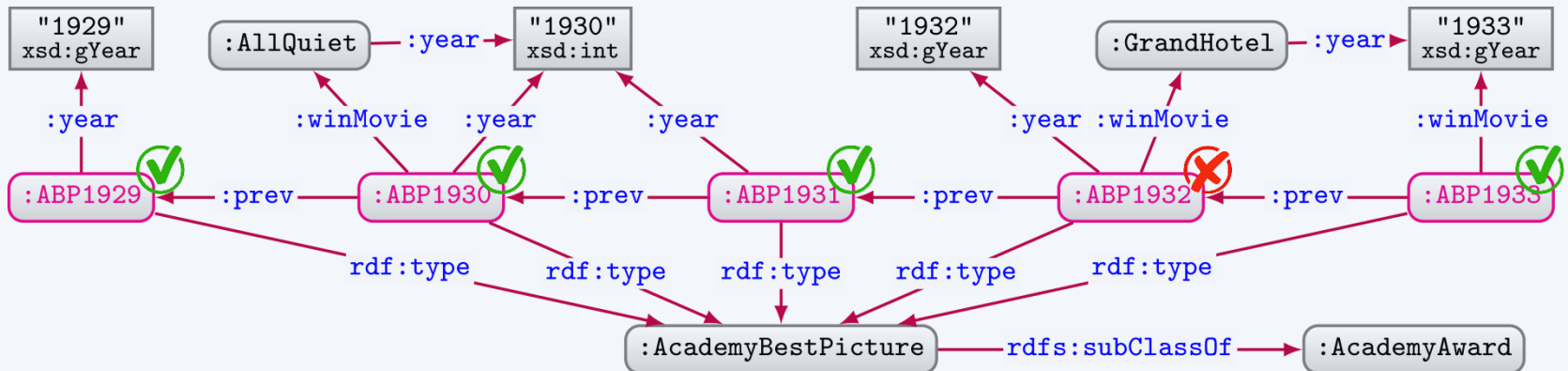


```
#[...]
s:AcademyAward a sh:NodeShape ;
  sh:targetClass :AcademyAward ;
  sh:property [ sh:path :winMovie ; sh:nodeKind sh:IRI ] ;
  sh:property [ sh:path :year ; sh:datatype xsd:gYear ] ;
  sh:closed true ; sh:ignoredProperties ( :prev rdf:type ) .
```

SHACL:

SPARQL CONSTRAINTS

SHACL: SPARQL CONSTRAINTS



```
#[...]
s:AcademyBestPicture a sh:NodeShape ;
  sh:targetClass :AcademyBestPicture ;
  sh:sparql [
    a sh:SPARQLConstraint ;
    sh:select """
      PREFIX : <http://ex.org/data/>
      PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
      SELECT $this (:year AS ?path) ?value
      WHERE {
        $this :year ?value .
        $this :winMovie/:year ?yearM .
        FILTER(xsd:int(str(?yearM))+1 != xsd:int(str(?value))
          && xsd:int(str(?yearM)) != xsd:int(str(?value)))
      }
    """
  ] .
```

Results indicate
constraint violations.

\$this will be replaced
with target nodes.

SHEx:

SHAPE EXPRESSIONS LANGUAGE

Shape Expressions Language 2.1

Final Community Group Report 8 October 2019



This version:

<http://shex.io/shex-semantic-20191008/>

Latest published version:

<http://shex.io/shex-semantic/>

Editor's draft:

<https://shexspec.github.io/spec/>

Previous version:

<http://shex.io/shex-semantic-20181122/>

Test suite:

<https://github.com/shexSpec/shexTest>

Bug tracker:

[File a bug \(open bugs\)](#)

Editors:

[Eric Prud'hommeaux \(W3C/MIT\)](#)

[Ivka Boneva \(University of Lille\)](#)

[Jose Emilio Labra Gayo \(University of Oviedo\)](#)

[Gregg Kellogg \(Spec-Ops\)](#)

Participate:

[GitHub shexSpec/spec](#)

[File a bug](#)

[Commit history](#)

[Pull requests](#)

SHEx: SHAPE EXPRESSIONS LANGUAGE

```
prefix : <http://ex.org/data/>
prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
prefix xsd: <http://www.w3.org/2001/XMLSchema#>
```

```
:Award {
  :year    xsd:gYear ;
  :name    rdf:langString + ;
  :judge   @:Person *
}
```

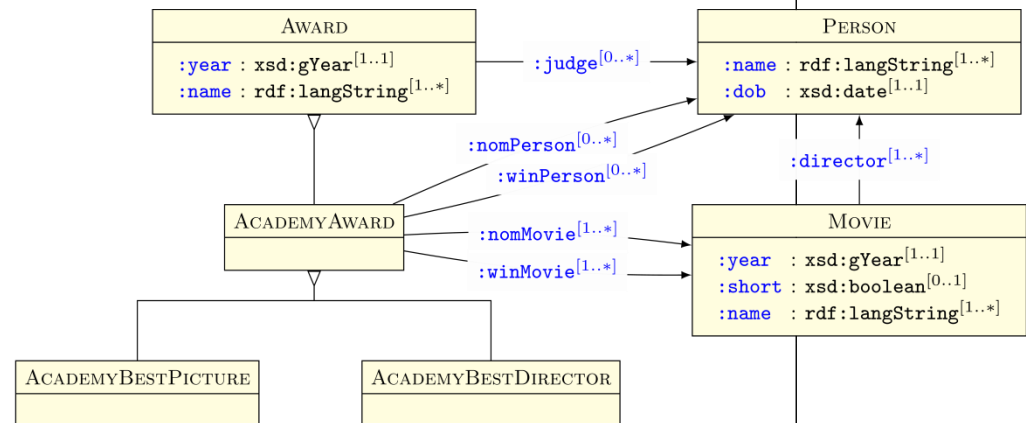
```
:AcademyAward @:Award AND {
  :winMovie @:Movie + ;
  :nomMovie @:Movie + ;
  :winPerson @:Person * ;
  :nomPerson @:Person *
}
```

```
:AcademyBestDirector @:AcademyAward
```

```
:AcademyBestPicture @:AcademyAward
```

```
:Person {
  :name    rdf:langString + ;
  :dob     xsd:date
}
```

```
:Movie {
  :year    xsd:gYear ;
  :short   xsd:boolean ? ;
  :name    rdf:langString + ;
  :director @:Person +
}
```



SHEx: USED BY WIKIDATA



EntitySchema

[Discussion](#)

[Read](#)

[View history](#)

actor (E25)

```
PREFIX p: <http://www.wikidata.org/prop/>
PREFIX wd: <http://www.wikidata.org/entity/>
PREFIX wdt: <http://www.wikidata.org/prop/direct/>

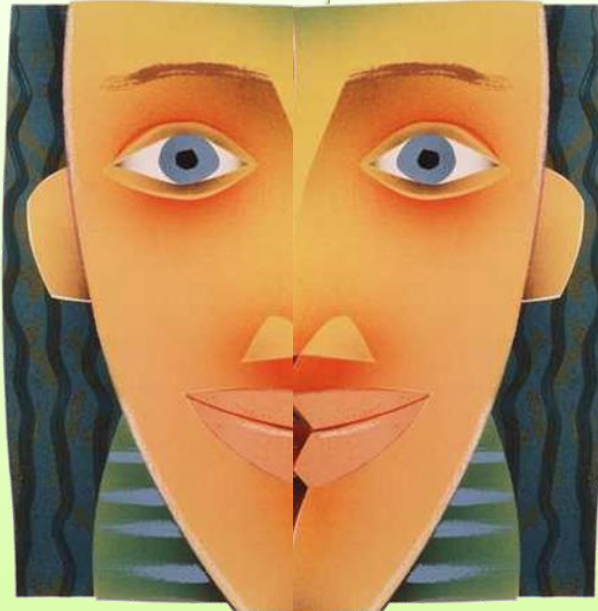
start = @<actor>

<actor> {
  wdt:P31 [wd:Q5] ;
  wdt:P21 . ;
  wdt:P106 . * ;
  wdt:P106 [ wd:Q33999 wd:Q21169216]
}
```

SHAPES:

(OPTIONAL) VALIDATING SCHEMA!

We don't have to impose a
structure (schema)
from the start



... and we can impose a
schema any time we like
using shapes!

QUESTIONS?

