kadaster

RDFS OWL SHACL training

22-05-2019

Pano Maria

Ontologie:

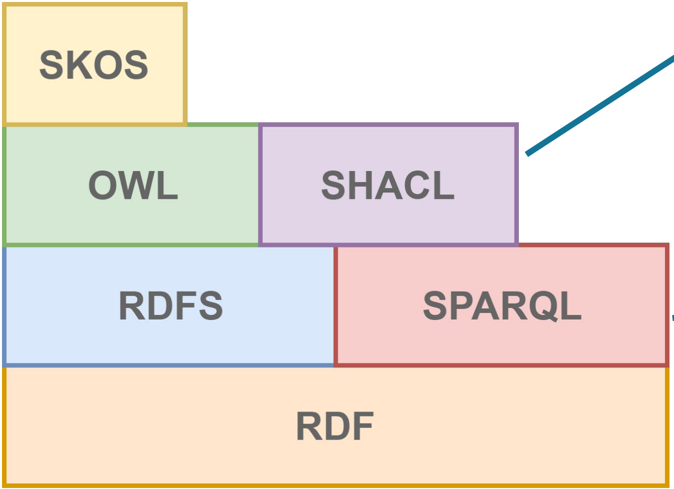
Kennisregels en -afleiding op basis van uitdrukkingen: Wat voor ding wordt hier beschreven?

Begrippen:

Wat bedoelen we als we deze term gebruiken?

Structuur:

Gebruik je die term wel correct? Dat mag je hier niet zeggen!



Vocabulaire:

Definiëren van gedeelde termen om dingen uit te drukken

Bevraging:

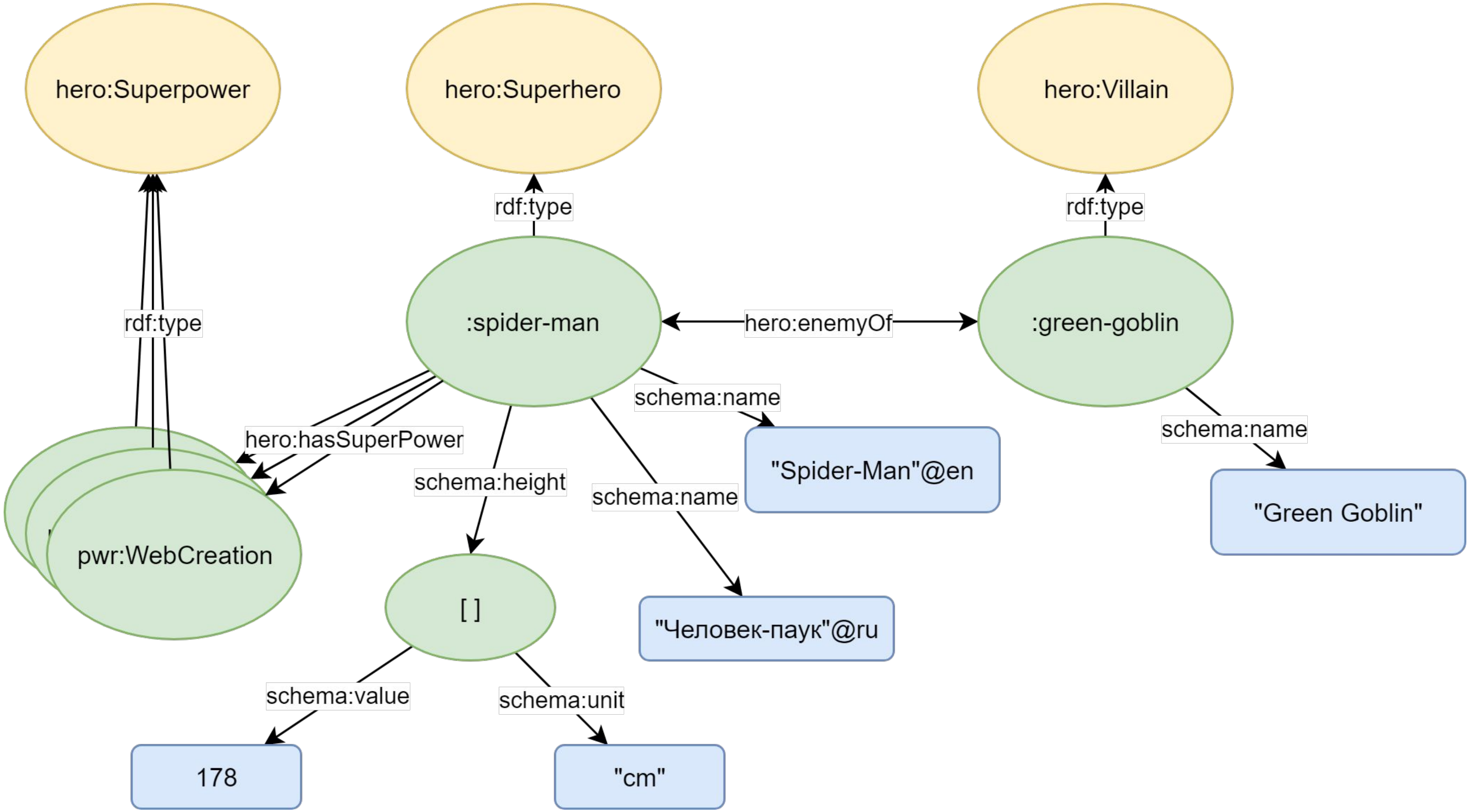
Wat is er allemaal gezegd?

Uitdrukkingen:

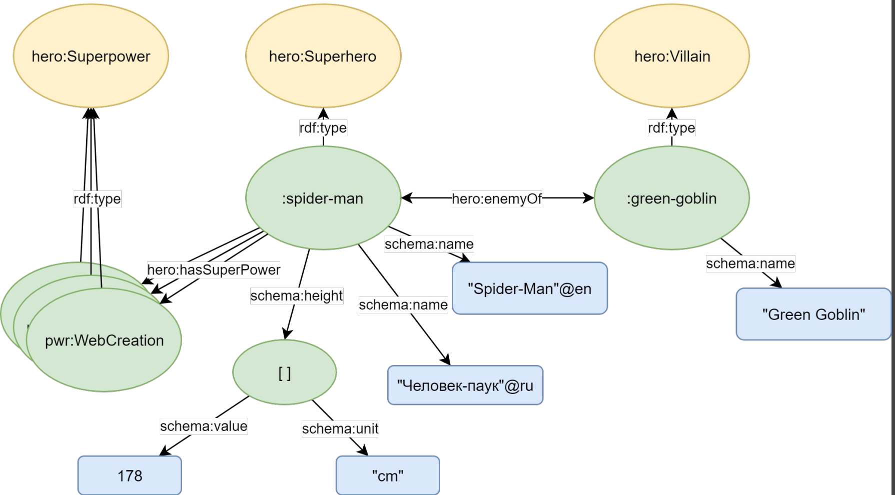
Simpele zinnen maken

Namen:

Dingen benoemen


hero:Superpower
hero:Superhero
hero:Villain
rdf:type
rdf:type
rdf:type
:spider-man
hero:enemyOf
:green-goblin
hero:hasSuperPower
schema:name
schema:name
schema:name
"Spider-Man"@en
"Green Goblin"
schema:height
pwr:WebCreation
[ ]
"Человек-паук"@ru
schema:value
schema:unit
178
"cm"

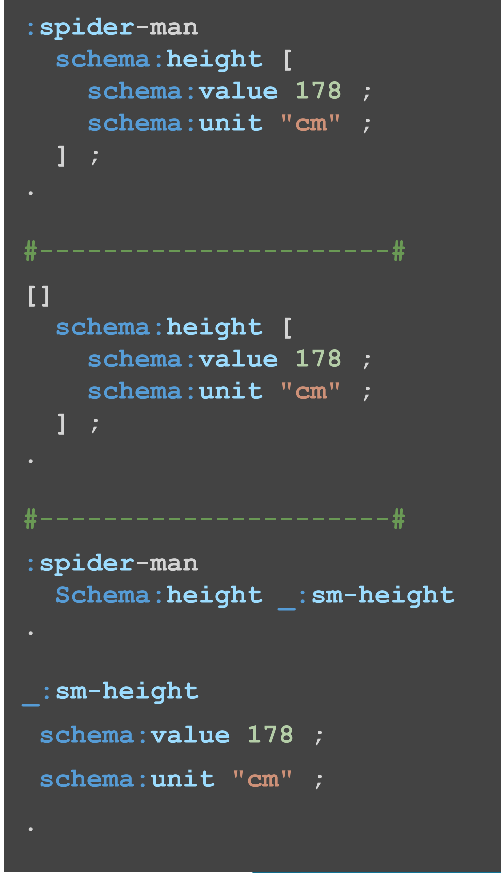
Turtle



```
@prefix : <http://example.org/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix schema: <http://schema.org/> .
@prefix hero: <http://data.superheros.com/def/superhero#> .
@prefix pwr: <http://data.superheros.com/id/superpower/> .

:green-goblin
  hero:enemyOf :spider-man ;
  rdf:type hero:Villain ;
  schema:name "Green Goblin" ;
.

:spider-man
  hero:enemyOf :green-goblin ;  # ; for statement continuation
  a hero:Superhero ;            # a is short-hand for rdf:type
  schema:name
    "Spider-Man"@en ,            # @ language tags
    "Человек-паук"@ru ;
  hero:hasSuperpower
    pwr:SpiderSense ,           # , multiple objects for same s,p
    pwr:SuperStrength ,
    pwr:WebCreation ;
  schema:height [               # a blank node (anonymous objects)
    schema:value 178 ;          # short-hand for "178"^^xsd:integer
    schema:unit "cm" ;
  ] ;
.
```

Blank Nodes

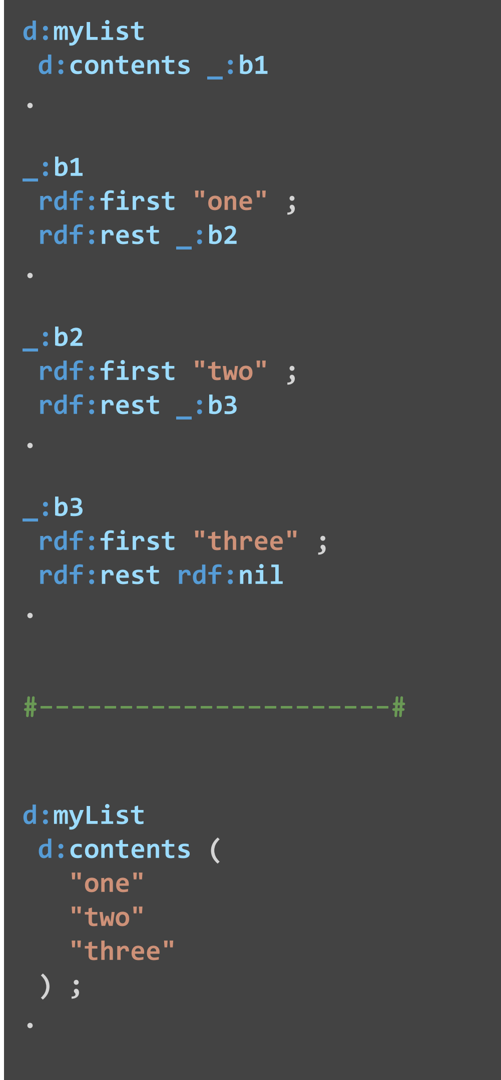
- Blank nodes
  - zijn anonieme nodes (zonder naam)
  - hebben lokale scope (uniek binnen document)
- Notatie:
  - _:x a hero:Superhero .
  - [ ] a hero:Superhero .
  - [a hero:Superhero] .
- Wordt gebruikt:
  - Wanneer je de URI van een ding niet weet
  - Wanneer je niet wilt dat iemand naar een ding verwijst
  - Voor complexe datatypes (bijv. waarde en meeteenheid)
  - Als compact syntax in specificaties (OWL, SHACL, RML, etc.)



```
:spider-man
  schema:height [
    schema:value 178 ;
    schema:unit "cm" ;
  ] ;
.

#---------------------#

[]
  schema:height [
    schema:value 178 ;
    schema:unit "cm" ;
  ] ;
.

#---------------------#

:spider-man
  Schema:height _:sm-height
.

_:sm-height
 schema:value 178 ;
 schema:unit "cm" ;
.
```

RDF Lists

- LISP style linked list
- Notatie
  - Turtle: Syntactic sugar
- Wordt gebruikt:
  - Als compact syntax in specificaties (OWL, SHACL..)
- Meestal niet nodig in instance data.
  - Geen natuurlijke uitdrukking



```
d:myList
 d:contents _:b1
.

_:b1
 rdf:first "one" ;
 rdf:rest _:b2
.

_:b2
 rdf:first "two" ;
 rdf:rest _:b3
.

_:b3
 rdf:first "three" ;
 rdf:rest rdf:nil
.


#-----------------------#


d:myList
 d:contents (
   "one"
   "two"
   "three"
 ) ;
.
```

Logic & Knowledge

Vastleggen, uitdrukken, afleiden van kennis.

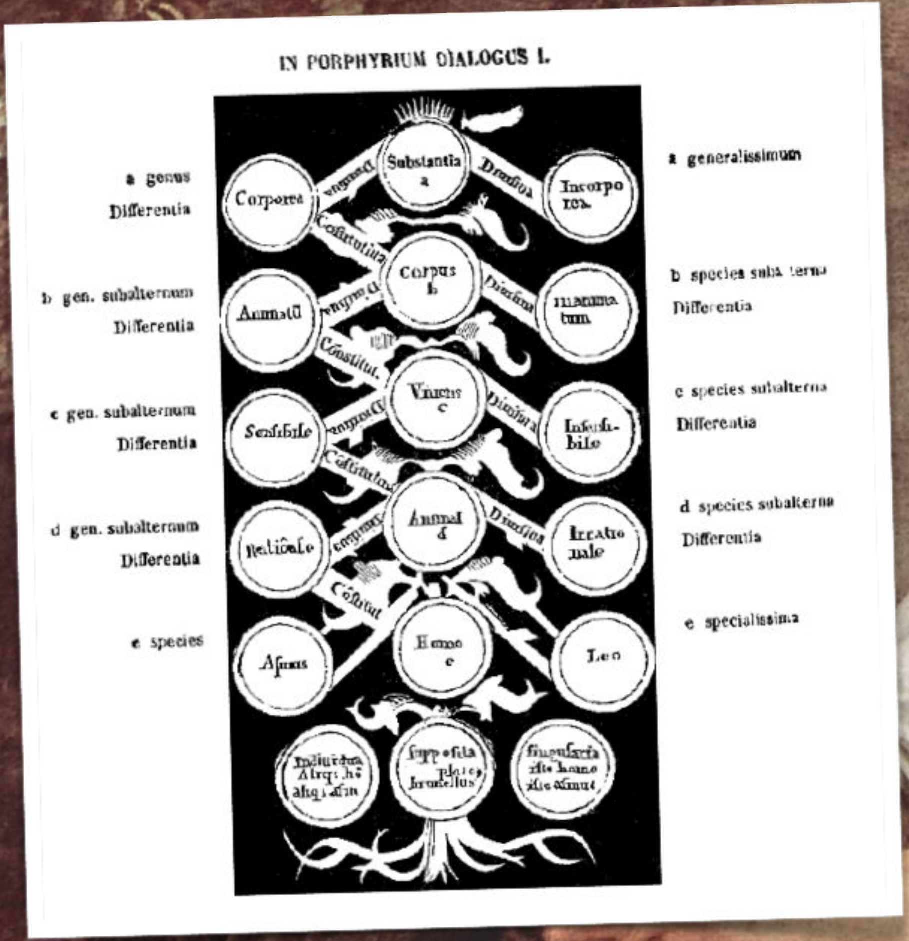
Logica is de wetenschap van het afleiden van kennis uit gegevens

All men are mortal;

Socrates is a man;

Therefore, Socrates is mortal.

The Universal Categories - Aristotle (384–322 BCE)



IN PORPHYRIUM DIALOGUS I.

Logica

- **Propositielogica (PL)**
  - Praten over waarheid van elementaire feiten
  - **Ondoenlijk verbose** voor representatie van kennis over **complexe domeinen**
  - **Efficient** voor reasoning

- **1e-orde logica (FOL)**
  - Praten over **objecten en** hun **relaties**
  - **Geschikt** voor vastleggen van kennis over de meeste **complexe domeinen**
  - Reasoning vaak **langzaam** en soms **onbeslisbaar**.
  - Gebaseerd op **set-theorie**

PL vs FOL

PL	FOL
○	○
Spiderman-is-a-Superhero	Superhero(Spiderman)
GreenGoblin-is-a-Superhero	Superhero(GreenGoblin)
Spiderman-and-GreenGoblin-are-enemies	enemyOf(Spiderman, GreenGoblin)

PL vs FOL

PL

○

Spiderman-is-a-Superhero
GreenGoblin-is-a-Superhero
Spiderman-and-GreenGoblin-are-enemies

FOL

○

Superhero(Spiderman)
Superhero(GreenGoblin)
enemyOf(Spiderman, GreenGoblin)

All superheroes are strong
∀s: Superhero(s) => Strong(s)

All superheroes can fly
∃s: Superhero(s) => CanFly(s)

Description Logics (DL)

- DLs zijn beperkte fragmenten van FOL
  - Gebaseerd op model-theorie
    - Geworteld in set-theorie
- Een DL modelleert **concepten**, **rollen** en **individuen** en hun **relaties**

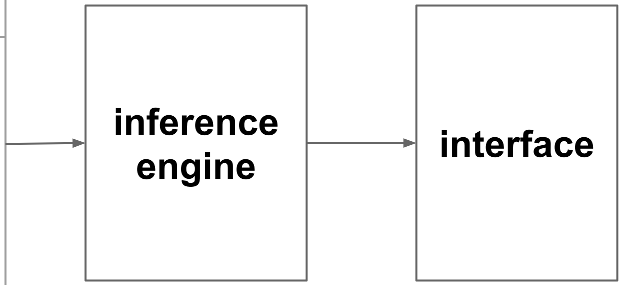
- DLs zijn beslisbaar (meestal)
- DLs zijn voldoende expressief (meestal)

Generieke DL architectuur

TBox	**Terminologische kennis** Kennis over concepten en hun rollen in een domein Schrijver ≡ Persoon ⊓ ∃ autheur.Boek

ABox	**Assertionele kennis**
	Kennis over individuen / entiteiten
	Schrijver(GeorgeOrwell)
	autheur(AnimalFarm, GeorgeOrwell)

RBox	**Rol-centrische kennis** Kennis over relaties tussen rollen coAutheur ⊑ autheur



inference engine

interface

RDFS + OWL

- Gebaseerd op DL

- RDFS voor simpele definitie van terminologie
  - Simpele kennis en logica

- OWL (Web Ontology Language)
  - complexere kennis en logica

DL hanteert geen Unique Name Assumption (UNA)

- In **databases** heeft elk ding een unieke naam
- In **DLs** kunnen dingen meer dan 1 naam hebben.
  - Dus, als twee dingen verschillende namen hebben zijn ze niet per se verschillend

Voorbeeld:

```
Superhero(Spiderman)
Superhero(IronMan)   hasFriend(Spiderman, IronMan)
Superhero(Hulk)      hasFriend(Spiderman, Hulk)
```

Hoeveel vrienden heeft Spiderman?

- ▸ DBs, met UNA:
- ▸ DLs, geen UNA:

DL hanteert geen Unique Name Assumption (UNA)

- In **databases** heeft elk ding een unieke naam
- In **DLs** kunnen dingen meer dan 1 naam hebben.
  - Dus, als twee dingen verschillende namen hebben zijn ze niet per se verschillend

Voorbeeld:

```
Superhero(Spiderman)
Superhero(IronMan)   hasFriend(Spiderman, IronMan)
Superhero(Hulk)      hasFriend(Spiderman, Hulk)
```

Hoeveel vrienden heeft Spiderman?

- DBs, met UNA: **2**
- DLs, geen UNA: **tenminste 1**

DL hanteert de Open World Assumption (OWA)

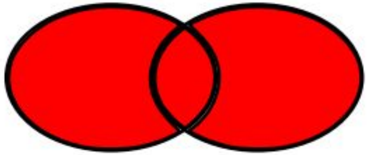
- **ER**, **UML**, **OO** hanteren **closed world** assumption (**CWA**)
- **CWA**:
  - Als een **feit niet bekend** is => **false**
- **OWA**:
  - Als een **feit niet bekend** is => **unknown**
  - Je kunt er nooit vanuit gaan dat je alle informatie hebt

- **CWA**:
  - Logische toepassing als informatie **compleet** beschikbaar is.
- **OWA**:
  - Logische toepassing voor **incomplete/open** informatiesystemen, zoals het Web.

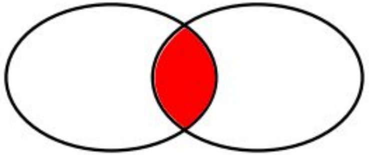
Vocabulaires

RDFS:

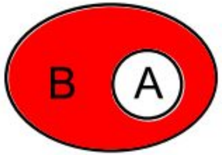
- Definitie van termen
- Minimale afleidingsregels (inference)
  - rdfs:subClassOf $\mathbf{Human \sqsubseteq Animal}$
  - rdfs:subPropertyOf $\mathbf{hasSon \sqsubseteq hasChild}$
  - rdfs:domain $\mathbf{T \sqsubseteq \forall hasParent^{-}.Human}$
  - rdfs:range $\mathbf{T \sqsubseteq \forall hasParent.Human}$
- Geen notie van incorrecte of inconsistente inferences.



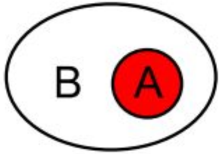
Union (A or B)



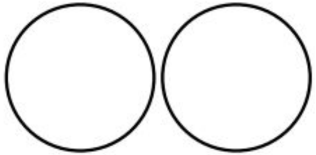
Intersection (A and B)



Complement
(complement of A inside B)



Set-subset
(A is subset of B)



Disjoint sets

RDFS - subClassOf

If S contains:	then S RDFS entails
xxx rdf:type rdfs:Class .	xxx rdfs:subClassOf rdfs:Resource .
xxx rdfs:subClassOf yyy . zzz rdf:type xxx .	zzz rdf:type yyy .
xxx rdf:type rdfs:Class .	xxx rdfs:subClassOf xxx .
xxx rdfs:subClassOf yyy . yyy rdfs:subClassOf zzz .	xxx rdfs:subClassOf zzz .

```
@prefix : <http://example.org/> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

:Ability a rdfs:Class .

:SuperhumanAbility a rdfs:Class ;
  rdfs:subClassOf :Ability .
.
```

Properties are first class citizens!

RDFS - subPropertyOf

If S contains:	then S RDFS entails recognizing D:
xxx rdfs:subPropertyOf yyy . yyy rdfs:subPropertyOf zzz .	xxx rdfs:subPropertyOf zzz .
xxx rdf:type rdf:Property .	xxx rdfs:subPropertyOf xxx .
aaa rdfs:subPropertyOf bbb . xxx aaa yyy .	xxx bbb yyy .

```
@prefix : <http://example.org/> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

:hasAbility a rdf:Property .

:hasSuperhumanAbility a rdf:Property ;
 rdfs:subPropertyOf :hasAbility .
.
```

RDFS - domain & range

If S contains:	then S RDFS entails
aaa rdfs:domain xxx . yyy aaa zzz .	yyy rdf:type xxx .
aaa rdfs:range xxx . yyy aaa zzz .	zzz rdf:type xxx .

```
@prefix : <http://example.org/> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .


:hasAbility a rdf:Property ;
  rdfs:domain :Being ;
  rdfs:range :Ability ;
.
```

Let op! Global scope!

If S contains:	then S RDFS entails
aaa rdfs:domain xxx . yyy aaa zzz .	yyy rdf:type xxx .

```turtle
@prefix : <http://example.org/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

:Superhero a rdfs:Class .

:hasName a rdf:Property ;
  rdfs:domain :Superhero ;
  rdfs:range xsd:string ;
.
```

Let op! Global scope!

If S contains:	then S RDFS entails
aaa rdfs:domain xxx . yyy aaa zzz .	yyy rdf:type xxx .

```
@prefix : <http://example.org/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

:Superhero a rdfs:Class .

:hasName a rdf:Property ;
  rdfs:domain :Superhero ;
  rdfs:range xsd:string ;
.

# instance data
:LoisLane a :Human ;
  :hasName "Lois Lane" ;
.
```

Let op! Global scope!

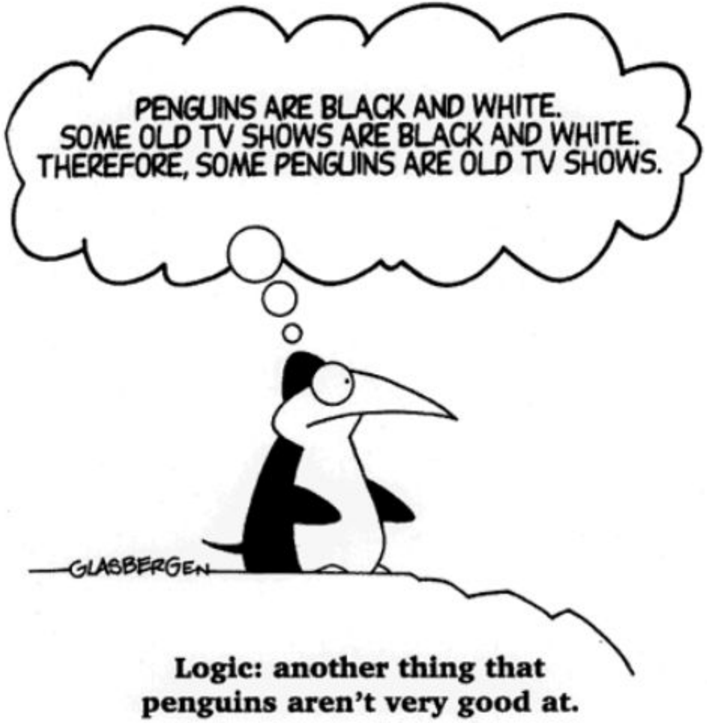
If S contains:	then S RDFS entails
aaa rdfs:domain xxx . yyy aaa zzz .	yyy rdf:type xxx .

```
@prefix : <http://example.org/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

:Superhero a rdfs:Class .

:hasName a rdf:Property ;
  rdfs:domain :Superhero ;
  rdfs:range xsd:string ;
.

# instance data
:LoisLane a :Human ;
  :hasName "Lois Lane" ;
.

# inferred
:LoisLane a :Superhero .
```

RDFS inferencing

RDFS Inferencing (kennisafleiding)

- Geen notie van incorrecte of inconsistente inferences. Volgt gewoon de regels.


PENGUINS ARE BLACK AND WHITE.
SOME OLD TV SHOWS ARE BLACK AND WHITE.
THEREFORE, SOME PENGUINS ARE OLD TV SHOWS.
GLASBERGEN
Logic: another thing that penguins aren't very good at.

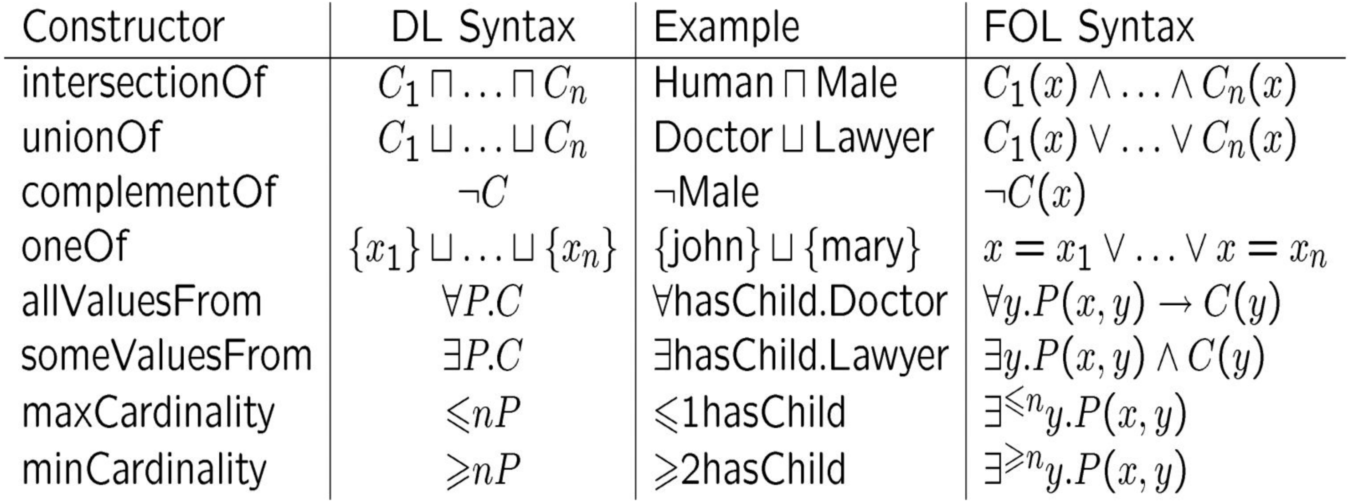
Ontologieën

Een Ontologie is een beschrijving van een domein in termen van categorieën van concepten (Class), instanties van concepten en hun relaties (Property) tot elkaar.

OWL - Web **Ontology** Language

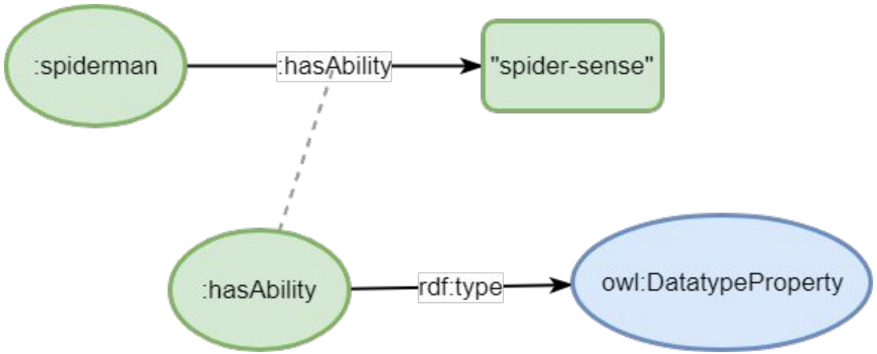
OWL constructs

- OWL gebouwd op 20+ jaar DL onderzoek
  - Goed gedefinieerde (modeltheorie-)semantiek



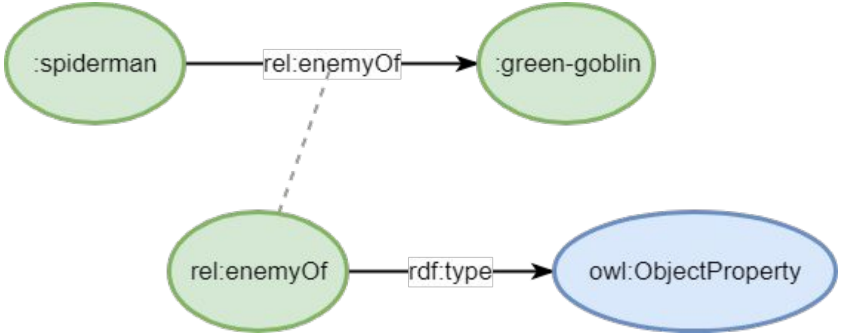
Constructor	DL Syntax	Example	FOL Syntax
intersectionOf	$C_1 \sqcap \ldots \sqcap C_n$	Human $\sqcap$ Male	$C_1(x) \wedge \ldots \wedge C_n(x)$
unionOf	$C_1 \sqcup \ldots \sqcup C_n$	Doctor $\sqcup$ Lawyer	$C_1(x) \vee \ldots \vee C_n(x)$
complementOf	$\neg C$	$\neg$Male	$\neg C(x)$
oneOf	$\{x_1\} \sqcup \ldots \sqcup \{x_n\}$	{john} $\sqcup$ {mary}	$x = x_1 \vee \ldots \vee x = x_n$
allValuesFrom	$\forall P.C$	$\forall$hasChild.Doctor	$\forall y.P(x,y) \rightarrow C(y)$
someValuesFrom	$\exists P.C$	$\exists$hasChild.Lawyer	$\exists y.P(x,y) \wedge C(y)$
maxCardinality	$\leqslant nP$	$\leqslant$1hasChild	$\exists^{\leqslant n} y.P(x,y)$
minCardinality	$\geqslant nP$	$\geqslant$2hasChild	$\exists^{\geqslant n} y.P(x,y)$

OWL - DatatypeProperty



```
@prefix : <http://example.org/> .
@prefix owl: <http://www.w3.org/2002/07/owl#>.

:hasAbility a owl:DatatypeProperty .

:spiderman :hasAbility "spider-sense" .
```

OWL - ObjectProperty



```
@prefix : <http://example.org/> .
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix rel: <http://www.example.org/rel#> .

rel:enemyOf a owl:ObjectProperty .

:spiderman rel:enemyOf :green-gobin .
```

OWL - inverseOf

```
@prefix : <http://example.org/> .
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix rel: <http://www.perceive.net/schemas/relationship/> .

rel:memberOf a owl:ObjectProperty .

rel:hasMember a owl:ObjectProperty .

rel:memberOf owl:inverseOf rel:hasMember .

:spiderman rel:memberOf :Avengers .
```

OWL - inverseOf

```
@prefix : <http://example.org/> .
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix rel: <http://www.perceive.net/schemas/relationship/> .

rel:memberOf a owl:ObjectProperty .

rel:hasMember a owl:ObjectProperty .

rel:memberOf owl:inverseOf rel:hasMember .

:spiderman rel:memberOf :Avengers .

# inferred:

:Avengers rel:hasMember :spiderman .
```

OWL - sameAs & differentFrom

- Remember: Geen UNA!

- **sameAs** vaak als mapping-relatie.
- **differentFrom** vaak resultaat van inference.

```
@prefix : <http://example.org/> .
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix super: <http://www.superheroes.net/hero/> .

:Spiderman a :Superhero .

super:spmn a super:Hero .

super:grngbln a super:Hero .


:spiderman owl:sameAs super:spmn .

:spiderman owl:differentFrom super:grngbln .
```

OWL - sameAs & differentFrom

- Remember: Geen UNA!

- **sameAs** vaak als mapping-relatie.
- **differentFrom** vaak resultaat
  van inference.

```
@prefix : <http://example.org/> .
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix super: <http://www.superheroes.net/hero/> .

:Spiderman a :Superhero .

super:spmn a super:Hero .

super:grngbln a super:Hero .


:spiderman owl:sameAs super:spmn .

:spiderman owl:differentFrom super:grngbln .

# inference rules:
{?X owl:sameAs ?Y} => {?Y owl:sameAs ?X}.
{?X owl:sameAs ?Y. ?Y owl:sameAs ?Z} => {?X owl:sameAs ?Z}.
{?X owl:sameAs ?Y. ?X owl:differentFrom ?Y} => false.

{?A owl:differentFrom ?B} => {?B owl:differentFrom ?A}.
```

OWL - equivalentClass

```
@prefix : <http://example.org/> .
@prefix owl: <http://www.w3.org/2002/07/owl#>.

:SuperhumanAbility a owl:Class .

:Superpower a owl:Class .

:SuperhumanAbility owl:equivalentClass :Superpower .

:SuperStrength a :Superpower .
```

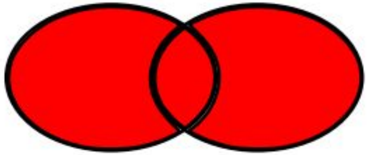
OWL - equivalentClass

```
@prefix : <http://example.org/> .
@prefix owl: <http://www.w3.org/2002/07/owl#>.

:SuperhumanAbility a owl:Class .

:Superpower a owl:Class .

:SuperhumanAbility owl:equivalentClass :Superpower .

:SuperStrength a :Superpower .

# inferred
:Superpower rdfs:subClassOf :SuperhumanAbility .
:SuperhumanAbility rdfs:subClassOf :Superpower .

:SuperStrength a :SuperPower .
:SuperStrength a :SuperhumanAbility .
```

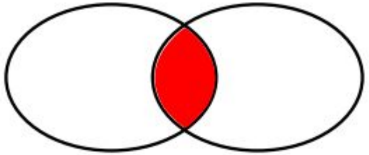
OWL - equivalentClass

```
@prefix : <http://example.org/> .
@prefix owl: <http://www.w3.org/2002/07/owl#>.

:SuperhumanAbility a owl:Class .

:Superpower a owl:Class .

:SuperhumanAbility owl:equivalentClass :Superpower .

:SuperStrength a :Superpower .

# inferred
:Superpower rdfs:subClassOf :SuperhumanAbility .
:SuperhumanAbility rdfs:subClassOf :Superpower .

:SuperStrength a :SuperPower .
:SuperStrength a :SuperhumanAbility .
```

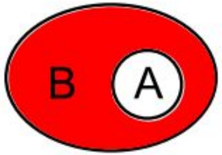
OWL set expressions and class constructors



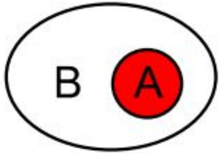
Union (A or B)



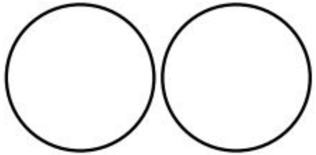
Intersection (A and B)



Complement
(complement of A inside B)

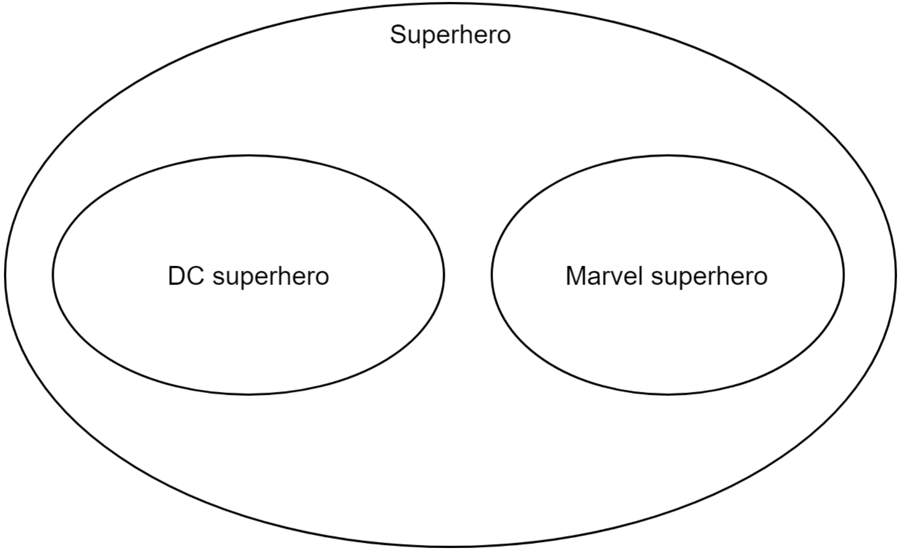


Set-subset
(A is subset of B)



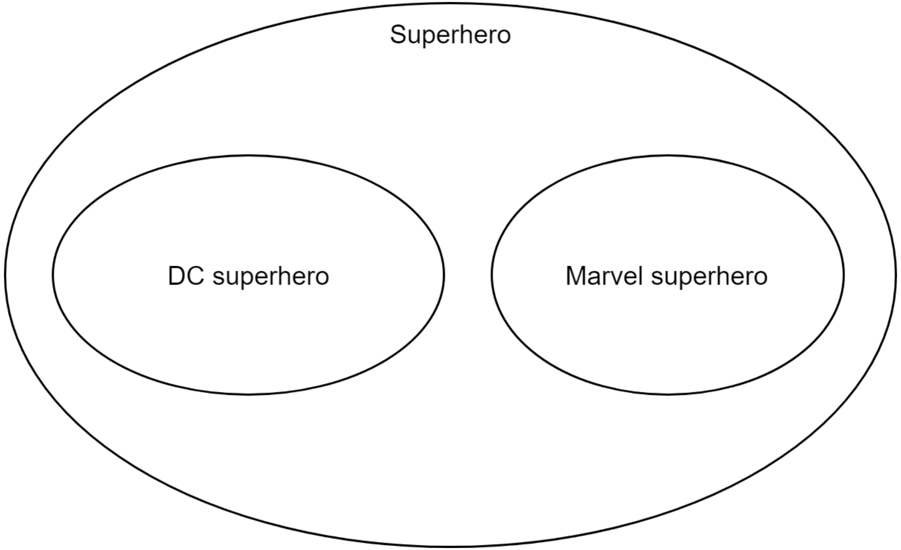
Disjoint sets

OWL - disjointWith



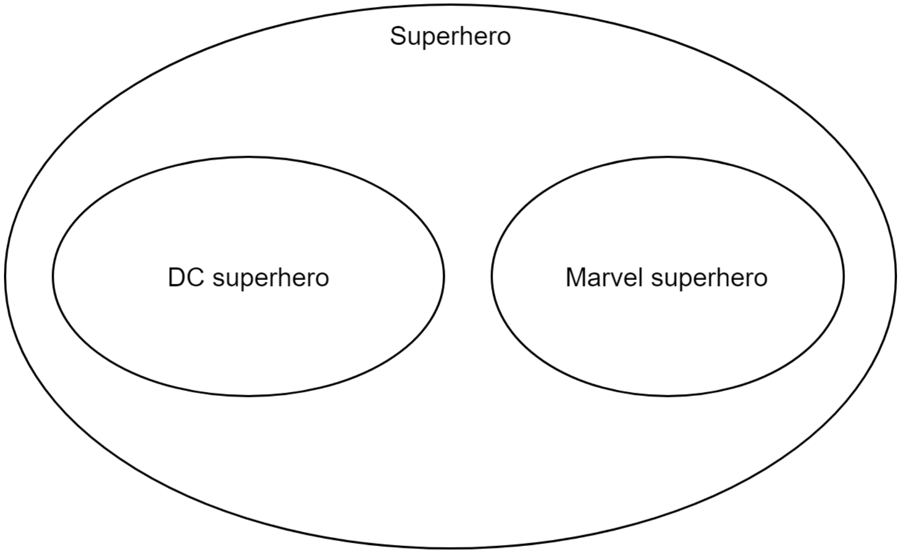
```
@prefix : <http://example.org/> .
@prefix owl: <http://www.w3.org/2002/07/owl#>.

:Superhero a owl:Class .

:DCSuperhero a owl:Class ;
  rdfs:subClassOf :Superhero ;
.

:MarvelSuperhero a owl:Class ;
  rdfs:subClassOf :Superhero ;
.

:DCSuperhero owl:disjointWith :MarvelSuperhero .

:Superman a :DCSuperhero .
:Spiderman a :MarvelSuperhero .
```

OWL - disjointWith



```
@prefix : <http://example.org/> .
@prefix owl: <http://www.w3.org/2002/07/owl#>.

:Superhero a owl:Class .

:DCSuperhero a owl:Class ;
  rdfs:subClassOf :Superhero ;
.

:MarvelSuperhero a owl:Class ;
  rdfs:subClassOf :Superhero ;
.

:DCSuperhero owl:disjointWith :MarvelSuperhero .

:Superman a :DCSuperhero .
:Spiderman a :MarvelSuperhero .


# inferred:
:Superman owl:differentFrom :Spiderman .
:Spiderman owl:differentFrom :Superman .
```

OWL - disjointWith

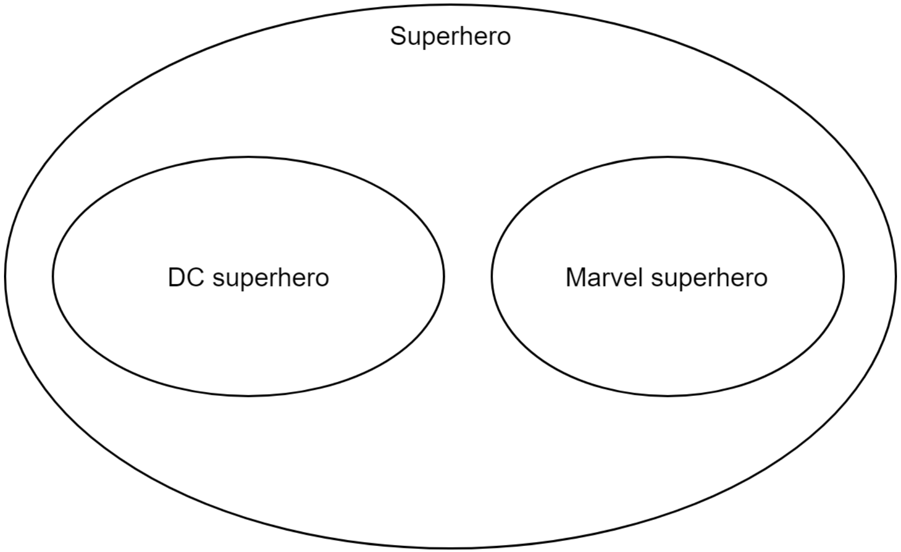

Superhero

DC superhero

Marvel superhero

```
@prefix : <http://example.org/> .
@prefix owl: <http://www.w3.org/2002/07/owl#>.

:Superhero a owl:Class .

:DCSuperhero a owl:Class ;
 rdfs:subClassOf :Superhero ;
.

:MarvelSuperhero a owl:Class ;
 rdfs:subClassOf :Superhero ;
.

:DCSuperhero owl:disjointWith :MarvelSuperhero .

:Batman a :DCSuperhero .
:Batman a :MarvelSuperhero .
```

OWL - disjointWith



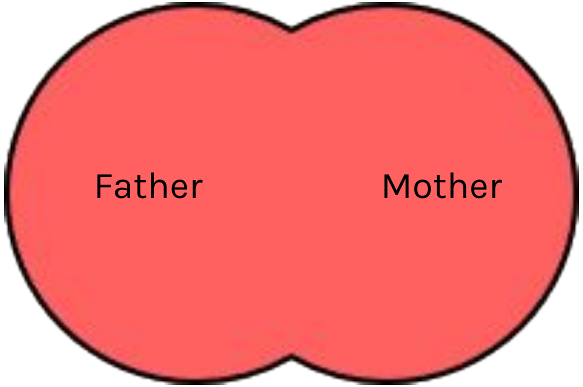
```
@prefix : <http://example.org/> .
@prefix owl: <http://www.w3.org/2002/07/owl#>.

:Superhero a owl:Class .

:DCSuperhero a owl:Class ;
 rdfs:subClassOf :Superhero ;
.

:MarvelSuperhero a owl:Class ;
 rdfs:subClassOf :Superhero ;
.

:DCSuperhero owl:disjointWith :MarvelSuperhero .

:Superman a :DCSuperhero .
:Spiderman a :MarvelSuperhero .

:Batman a :DCSuperhero .
:Batman a :MarvelSuperhero .

# inferred:

FALSE (Logically inconsistent)
```

OWL - Closed classes

- Met **owl:oneOf** specificeer je een **gesloten** enumeratie.

```
:Ability a owl:Class .

:Sight a :Ability .
:Hearing a :Ability .
:Touch a :Ability .
:Taste a :Ability .
:Smell a :Ability .

:BasicSenses a owl:Class ;
  owl:oneOf (
    :Sight
    :Hearing
    :Touch
    :Taste
    :Smell
  )
.
```

OWL - Union of



```
@prefix : <http://example.org/> .
@prefix owl: <http://www.w3.org/2002/07/owl#>.

:Parent a owl:Class ;
 owl:equivalentClass [
   owl:unionOf (
     :Father
     :Mother
   )
 ] ;
.
```

OWL Property Restrictions

Restricties op waardes van properties:

- owl:hasValue
- owl:allValuesFrom
- owl:someValuesFrom

Restricties op kardinaliteit van properties:

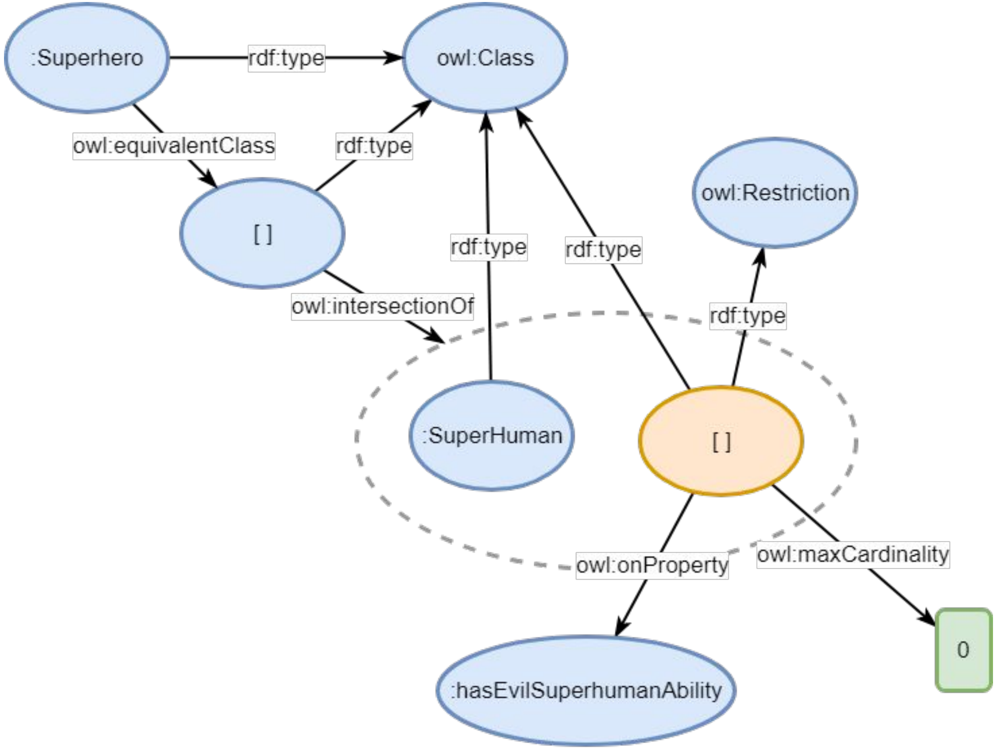
- owl:cardinality
- owl:minCardinality
- owl:maxCardinality

```
:SupermansEnemy a owl:Class ;
 owl:subClassOf [
   a owl:Restriction ;
   owl:onProperty :hasEnemy ;
   owl:hasValue :Superman ;
 ] ;
.
```

```
:Being a owl:Class ;
 owl:equivalentClass [
   a owl:Restriction ;
   owl:onProperty :hasAbility ;
   owl:allValuesFrom :Ability ;
 ] ;
.
```

```
:Tetralogy a owl:Class ;
 rdfs:subClassOf [
   a owl:Restriction ;
   owl:onProperty :hasVolumes ;
   owl:cardinality 4 ;
 ] ;
.
```

OWL - Complex Classes



```
:Superhero a owl:Class ;
 owl:equivalentClass [
   a owl:Class ;
   owl:intersectionOf (
     :SuperHuman
     [
       a owl:class, owl:Restriction ;
       owl:onProperty :hasEvilSuperhumanAbility ;
       owl:maxCardinality 0 ;
     ]
   )
 ] ;
.
```

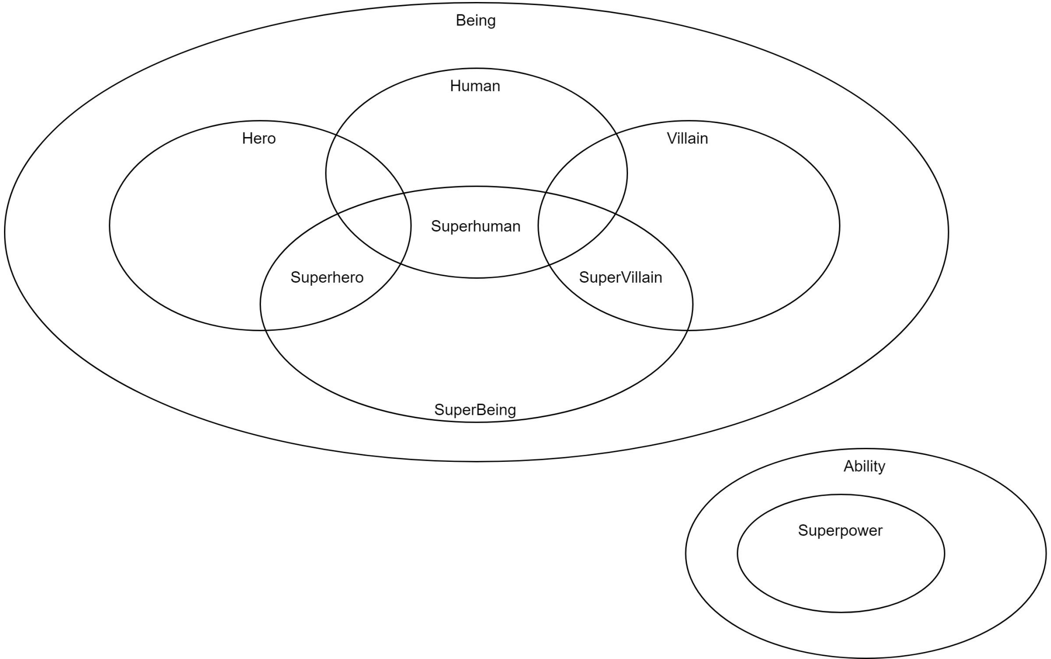
Pauze

LINKED OPEN DATA
OPEN LICENS
★ On the web
★★ Machine-readable data
★★★ Non-proprietary format
★★★★ RDF standards
★★★★★ Linked RDF
IS YOUR DATA 5★?

Exercise 001

Exercise - Superhero ontology

- Exercise 001

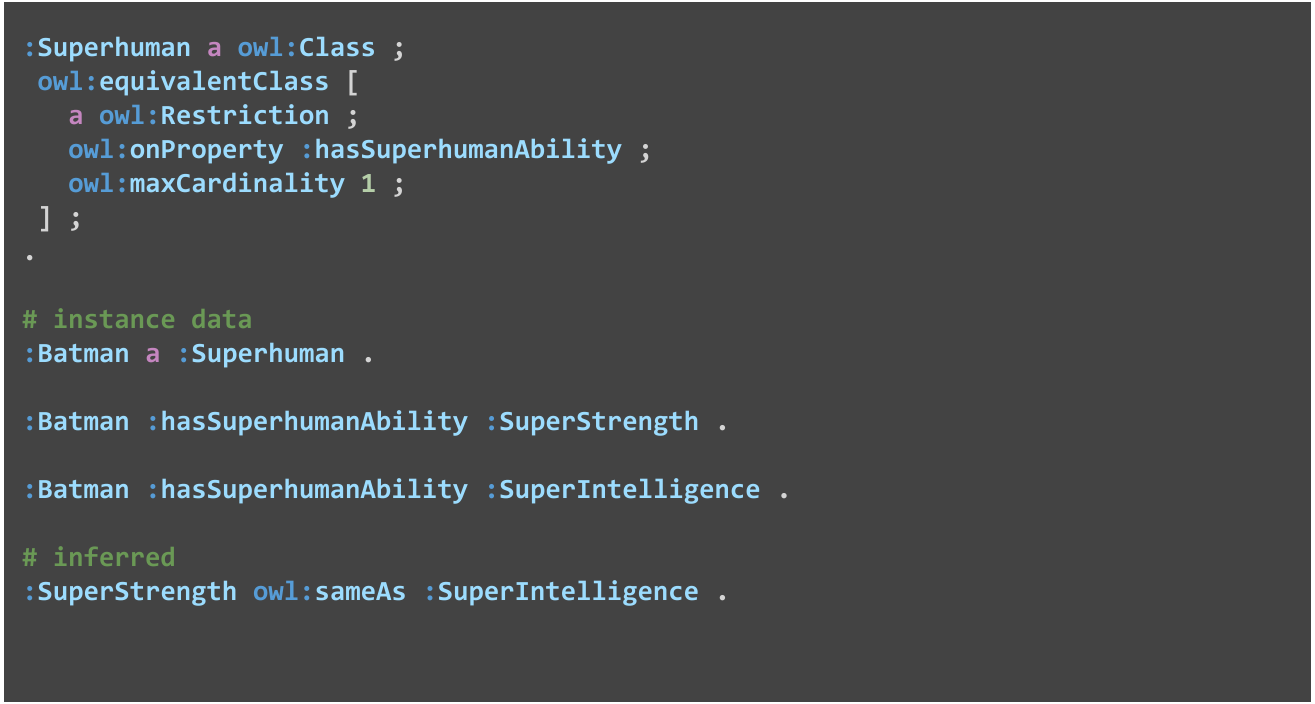


Exercise solutions 001

OWL data validation?

```
:Superhuman a owl:Class ;
 owl:equivalentClass [
   a owl:Restriction ;
   owl:onProperty :hasSuperhumanAbility ;
   owl:maxCardinality 1 ;
 ] ;
.

# instance data
:Batman a :Superhuman .

:Batman :hasSuperhumanAbility :SuperStrength .

:Batman :hasSuperhumanAbility :SuperIntelligence .
```

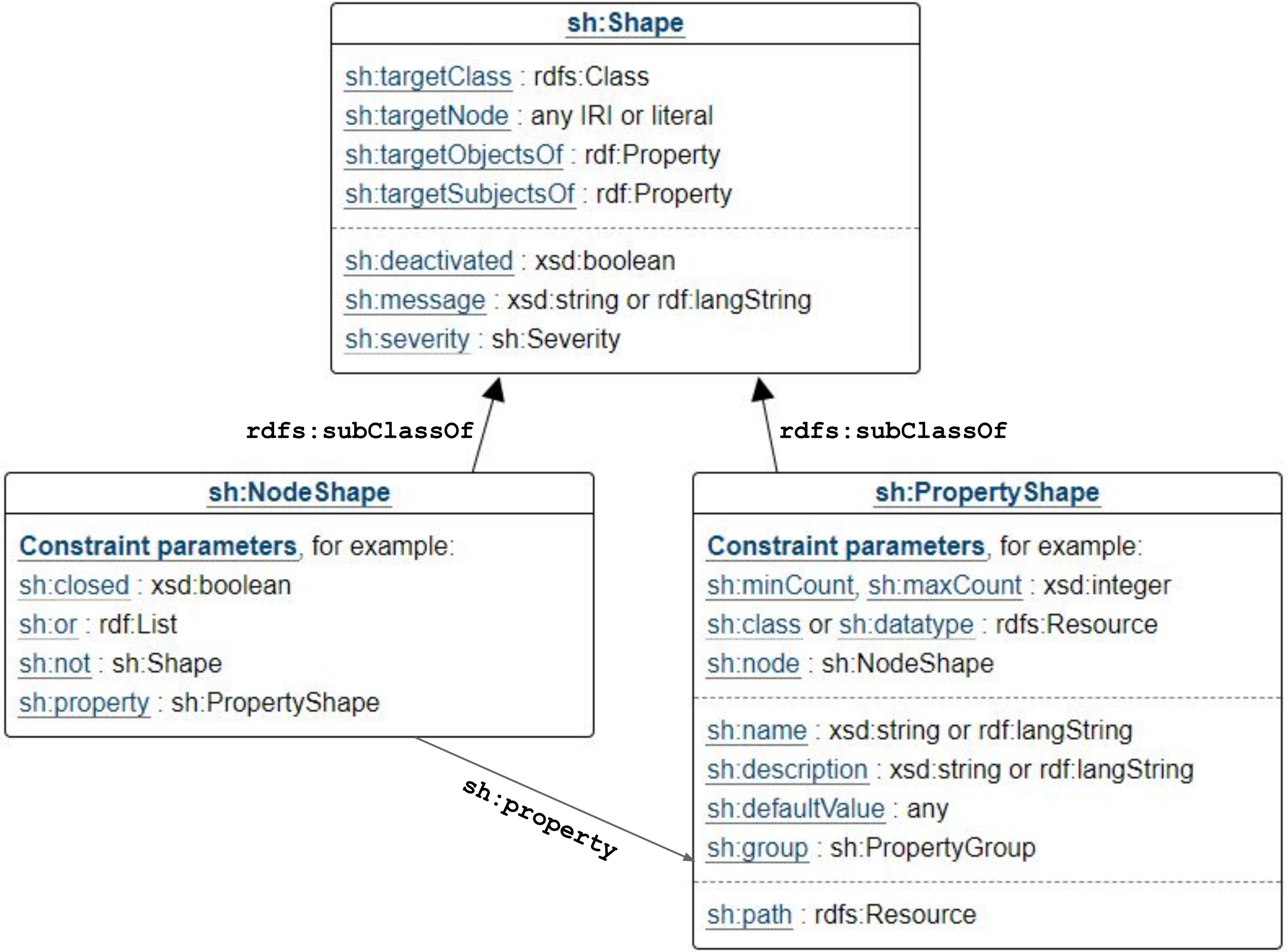
OWL data validation?



```
:Superhuman a owl:Class ;
 owl:equivalentClass [
   a owl:Restriction ;
   owl:onProperty :hasSuperhumanAbility ;
   owl:maxCardinality 1 ;
 ] ;
.

# instance data
:Batman a :Superhuman .

:Batman :hasSuperhumanAbility :SuperStrength .

:Batman :hasSuperhumanAbility :SuperIntelligence .

# inferred
:SuperStrength owl:sameAs :SuperIntelligence .
```

SHACL - Shapes Constraint Language

- W3C Recommendation sinds 2017
- Primair voor **validatie** van RDF data.
- Maakt “closed world view” op RDF data mogelijk.
- SHACL is uitgedrukt in RDF en dus descriptief en machineleesbaar
- Voornamelijk geïmplementeerd in SPARQL, maar ook een implementatie in JavaScript



sh:Shape

sh:targetClass : rdfs:Class
sh:targetNode : any IRI or literal
sh:targetObjectsOf : rdf:Property
sh:targetSubjectsOf : rdf:Property

sh:deactivated : xsd:boolean
sh:message : xsd:string or rdf:langString
sh:severity : sh:Severity

rdfs:subClassOf

rdfs:subClassOf

sh:NodeShape

Constraint parameters, for example:
sh:closed : xsd:boolean
sh:or : rdf:List
sh:not : sh:Shape
sh:property : sh:PropertyShape

sh:property

sh:PropertyShape

Constraint parameters, for example:
sh:minCount, sh:maxCount : xsd:integer
sh:class or sh:datatype : rdfs:Resource
sh:node : sh:NodeShape

sh:name : xsd:string or rdf:langString
sh:description : xsd:string or rdf:langString
sh:defaultValue : any
sh:group : sh:PropertyGroup

sh:path : rdfs:Resource

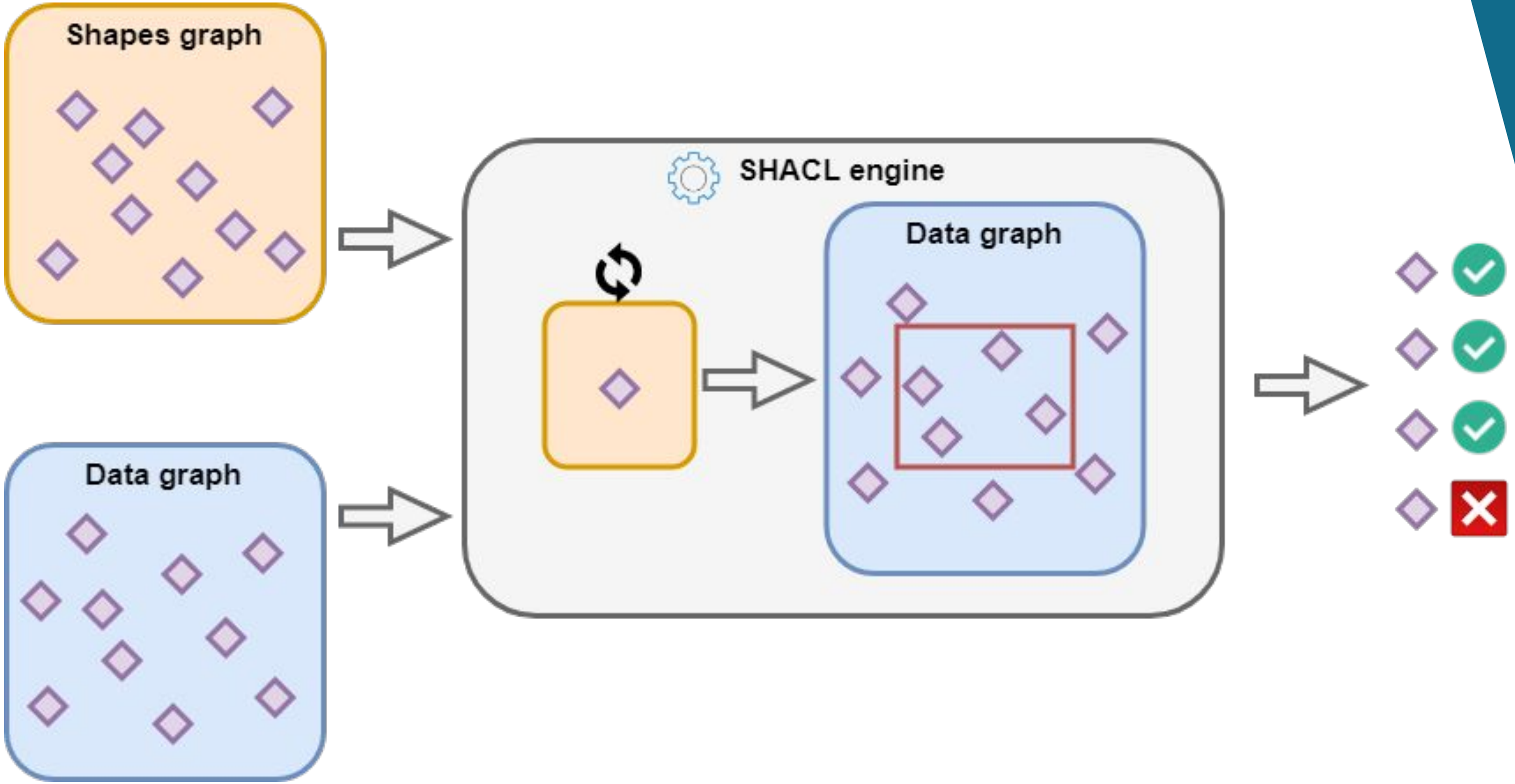
NodeShapes en PropertyShapes

Een **NodeShape** beschrijft de vorm van entiteiten.

Een **PropertyShape** beschrijft de vorm van een eigenschap van een entiteit.

Een **NodeShape** kan een **PropertyShape** declareren met `sh:property`

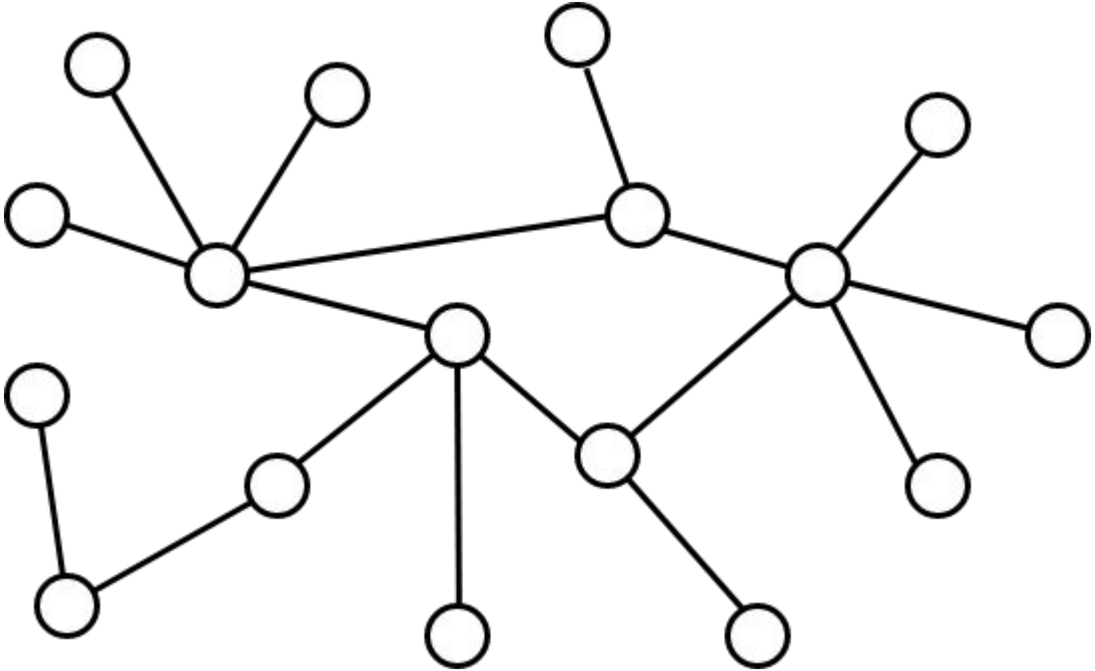
SHACL validation engine



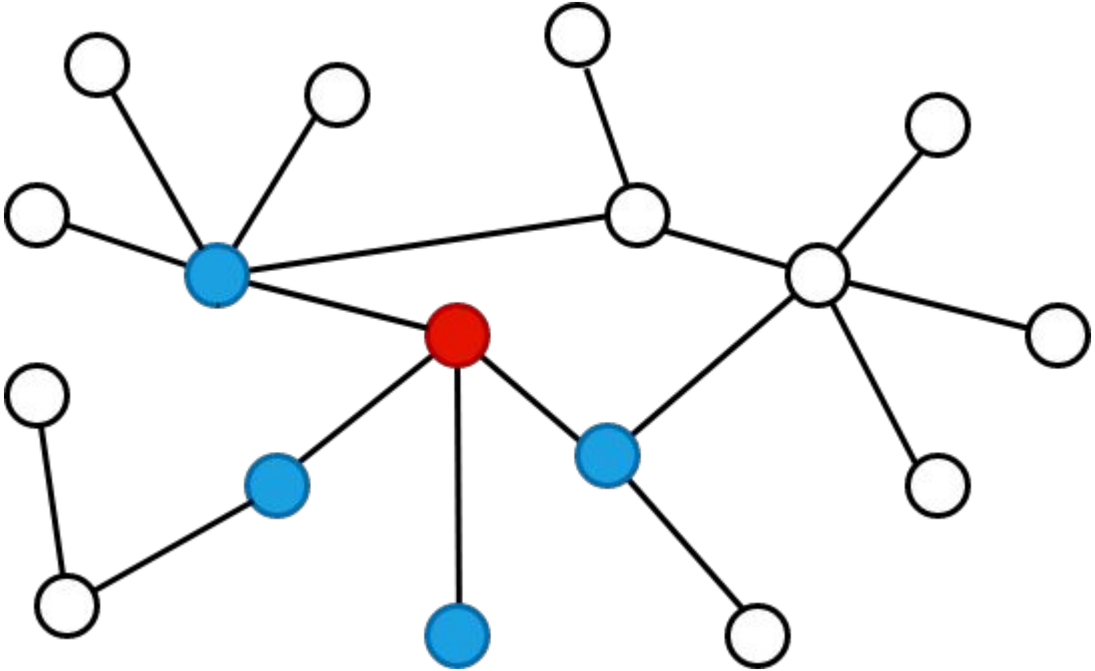
Shape voorbeeld

```
ex:PersonShape
  a sh:NodeShape ;
  sh:targetClass ex:Person ;        # Applies to all persons
  sh:property [                     # _:b1
    sh:path ex:ssn ;                # constrains the values of ex:ssn
    sh:maxCount 1 ;
    sh:datatype xsd:string ;
    sh:pattern "^\\d{3}-\\d{2}-\\d{4}$" ;
    sh:message "wrong ex:ssn" ;     # Message upon shape violation
  ] ;
  sh:property [                     # _:b2
    sh:path ex:worksFor ;
    sh:class ex:Company ;
    sh:nodeKind sh:IRI ;
  ] ;
.
```

Targets



Targets



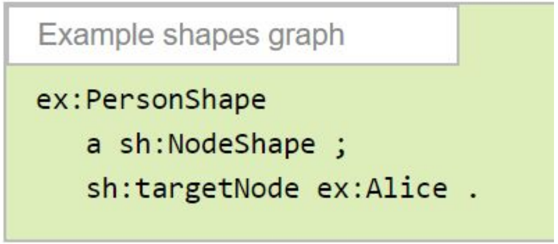
Targets

Target is een eigenschap van een shape die aangeeft welke nodes in de data graph gevalideerd worden door de shape.

SHACL-core definieert verschillende soorten targets:

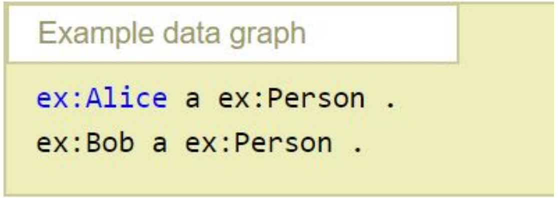
- Node targets (sh:targetNode)
- Class-based Targets (sh:targetClass)
- Implicit Class Targets
- Subjects-of targets (sh:targetSubjectsOf)
- Objects-of targets (sh:targetObjectsOf)

Node targets



Example shapes graph

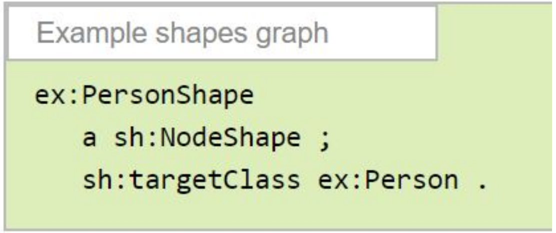
```
ex:PersonShape
   a sh:NodeShape ;
   sh:targetNode ex:Alice .
```



Example data graph

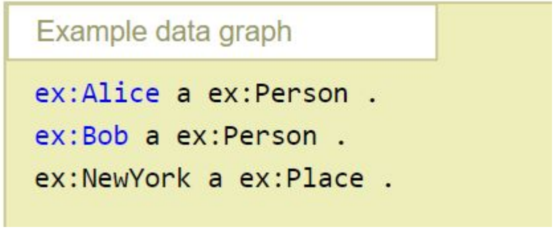
```
ex:Alice a ex:Person .
ex:Bob a ex:Person .
```

Class-based targets



Example shapes graph

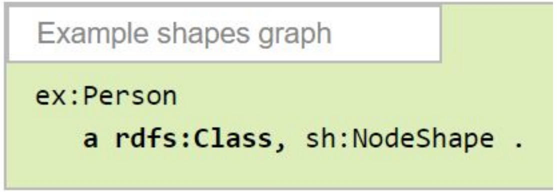
```
ex:PersonShape
   a sh:NodeShape ;
   sh:targetClass ex:Person .
```



Example data graph

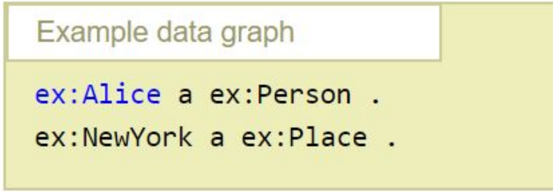
```
ex:Alice a ex:Person .
ex:Bob a ex:Person .
ex:NewYork a ex:Place .
```

Implicit Class targets



Example shapes graph

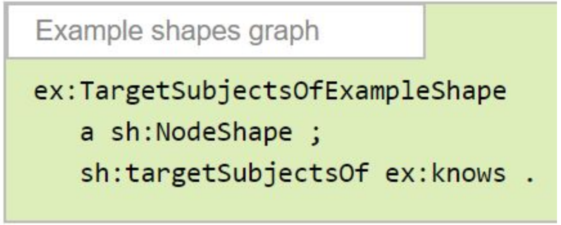
```
ex:Person
   a rdfs:Class, sh:NodeShape .
```



Example data graph

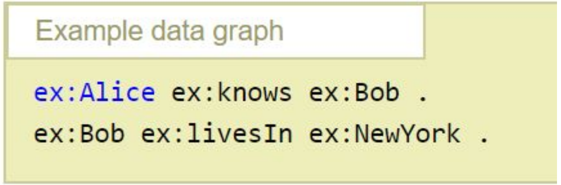
```
ex:Alice a ex:Person .
ex:NewYork a ex:Place .
```

Subjects-of targets



Example shapes graph

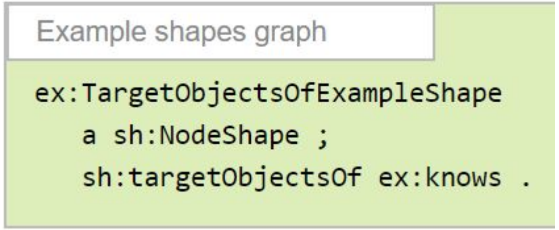
```
ex:TargetSubjectsOfExampleShape
   a sh:NodeShape ;
   sh:targetSubjectsOf ex:knows .
```



Example data graph

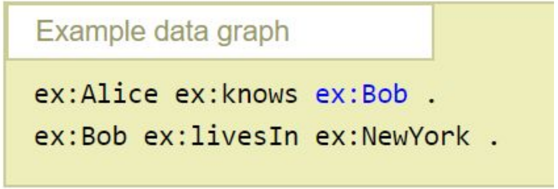
```
ex:Alice ex:knows ex:Bob .
ex:Bob ex:livesIn ex:NewYork .
```

Objects-of targets



Example shapes graph

```
ex:TargetObjectsOfExampleShape
    a sh:NodeShape ;
    sh:targetObjectsOf ex:knows .
```

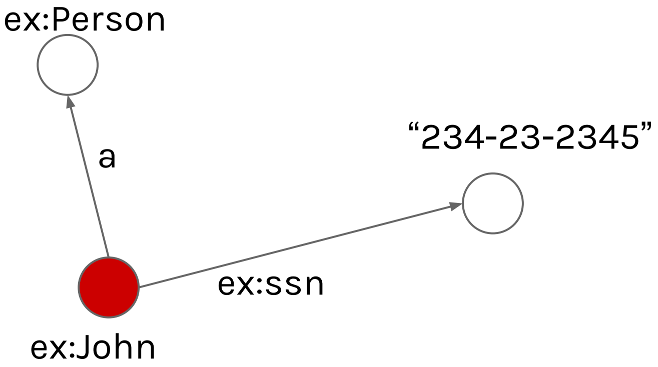


Example data graph

```
ex:Alice ex:knows ex:Bob .
ex:Bob ex:livesIn ex:NewYork .
```

Beschrijven van properties

- sh:property
    - Heeft altijd een sh:path!

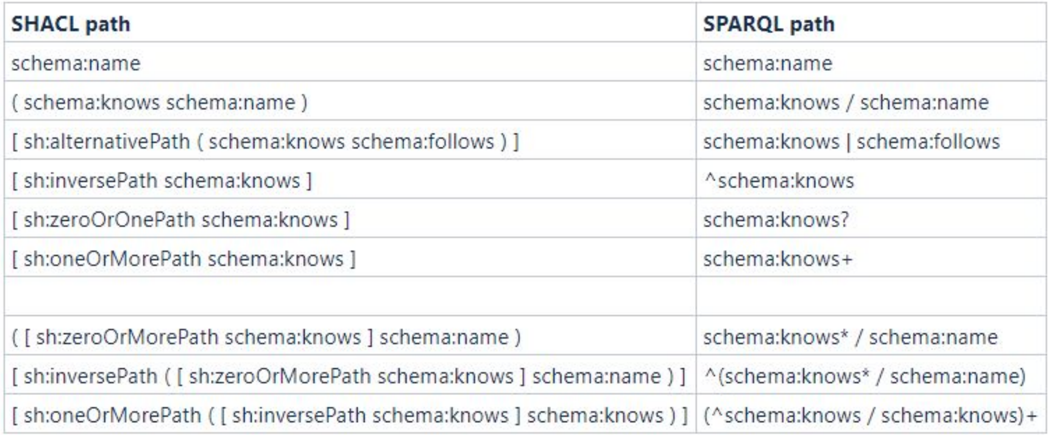


```
ex:PersonShape
  a sh:NodeShape ;
  sh:targetClass ex:Person ;
  sh:property [
    sh:path ex:ssn ;
    sh:maxCount 1 ;
    sh:datatype xsd:string ;
    sh:pattern "^\\d{3}-\\d{2}-\\d{4}$" ;
    sh:message "wrong ex:ssn" ;
  ]
```

Exercise 002

Exercise solutions 002

Complexere paden



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

```
ex:PersonShape
  a sh:NodeShape ;
  sh:targetClass ex:Person ;
  sh:property [
    sh:path [ schema:knows / schema:name ] ;
    sh:minCount 1 ;
    sh:datatype xsd:string ;
  ] ;
.
```

```
ex:ConsideredEnemyByOther a sh:NodeShape ;
 sh:targetClass ex:Person ;
 sh:property [
   sh:path [ sh:inversePath :hasEnemy ] ;
   sh:minCount 1 ;
 ] ;
.
```

Waardetype constraints

- sh:class
- sh:nodeKind
- sh:datatype

```
ex:ClassAndNodeKindExampleShape
  a sh:NodeShape ;
  sh:targetNode ex:Bob, ex:Alice, ex:Carol ;
  sh:property [
    sh:path ex:address ;
    sh:class ex:PostalAddress ;
    sh:nodeKind sh:IRI
    #ander opties: sh:BlankNode, sh:Literal, sh:BlankNodeOrIRI,
    #              sh:BlankNodeOrLiteral en sh:IRIOrLiteral
  ] ;
.



ex:DatatypeExampleShape
  a sh:NodeShape ;
  sh:targetNode ex:Alice, ex:Bob, ex:Carol ;
  sh:property [
    sh:path ex:age ;
    sh:datatype xsd:integer ;
  ]
.
```

Validation results

```
[
  a sh:ValidationReport ;
  sh:conforms false ;
  sh:result [
    a sh:ValidationResult ;
    sh:resultSeverity sh:Violation ;
    sh:focusNode ex:Bob ;
    sh:resultPath ex:age ;
    sh:value "twenty two" ;
    sh:resultMessage "ex:age expects a literal of datatype xsd:integer."
;
    sh:sourceConstraintComponent sh:DatatypeConstraintComponent ;
    sh:sourceShape ex:PersonShapeAge ;
  ]
] .
```

sh:severity, sh:message

```
ex:MyShape
  a sh:NodeShape ;
  sh:targetNode ex:MyInstance ;
  sh:property [
    # Violations of sh:minCount and
    # sh:datatype are produced as warnings
    sh:path ex:myProperty ;
    sh:minCount 1 ;
    sh:datatype xsd:string ;
    sh:severity sh:Warning ;
  ] ;
  sh:property [
    # The default severity here is
    # sh:Violation
    sh:path ex:myProperty ;
    sh:maxLength 10 ;
    sh:message "Too many characters"@en ;
    sh:message "Zu viele Zeichen"@de ;
  ]
.
```

```
# Gegeven:
ex:MyInstance
  ex:myProperty "http://toomanycharacters"^^xsd:anyURI .

# Resultaat:
[
  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 ;
  ] ,
  [ a sh:ValidationResult ;
    sh:resultSeverity sh:Violation ;
    sh:focusNode ex:MyInstance ;
    sh:resultPath ex:myProperty ;
    sh:value "http://toomanycharacters"^^xsd:anyURI ;
    sh:resultMessage "Too many characters"@en ;
    sh:resultMessage "Zu viele Zeichen"@de ;
    sh:sourceConstraintComponent sh:MaxLengthConstraintComponent ;
    sh:sourceShape _:b2 ;
  ]
] .
```

Kardinaliteit constraints

- sh:minCount
- sh:maxCount

```
ex:MinMaxCountExampleShape
  a sh:NodeShape ;
  sh:targetNode ex:Bob ;
  sh:property [
    sh:path ex:birthDate ;
    sh:maxCount 1 ;
    sh:minCount 0 ; # 0 is default en kan weggelaten worden
  ]
.
```

String constraints

- sh:minLength
- sh:maxLength
- sh:pattern

```
ex:PasswordExampleShape
  a sh:NodeShape ;
  sh:targetNode ex:Bob, ex:Alice ;
  sh:property [
    sh:path ex:password ;
    sh:minLength 8 ;
    sh:maxLength 10 ;
    sh:pattern "^(?=.*\\d)(?=.*[a-z])(?=.*[A-Z]).{8,10}$" ;
  ] ;
.
```

Logical constraints (1)

- sh:not
- sh:and

```
ex:NotExampleShape
  a sh:NodeShape ;
  sh:targetNode ex:InvalidInstance1 ;
  sh:not [
    a sh:PropertyShape ;
    sh:path ex:property ;
    sh:minCount 1 ;
  ]
.

ex:AndExampleShape
  a sh:NodeShape ;
  sh:targetNode ex:ValidInstance, ex:InvalidInstance ;
  sh:and (
    [
      sh:path ex:property ;
      sh:minCount 1 ;
    ]
    [
      sh:path ex:property ;
      sh:maxCount 1 ;
    ]
  )
.
```

Logical constraints (2)

- sh:or
- sh:xone

```
ex:OrConstraintExampleShape
  a sh:NodeShape ;
  sh:targetNode ex:Bob ;
  sh:or (
    [
      sh:path ex:firstName ;
      sh:minCount 1 ;
    ]
    [
      sh:path ex:givenName ;
      sh:minCount 1 ;
    ]
  )
.

ex:XoneConstraintExampleShape
  a sh:NodeShape ;
  sh:targetClass ex:Person ;
  sh:xone (
    [
      sh:property [
        sh:path ex:fullName ;
        sh:minCount 1 ;
      ]
    ]
    [
      sh:property [
        sh:path ex:firstName ;
        sh:minCount 1 ;
      ] ;
      sh:property [
        sh:path ex:lastName ;
        sh:minCount 1 ;
      ]
    ]
  )
.
```

Shape-gebaseerde constraints (1)

- sh:qualifiedValueShape
  - sh:qualifiedMinCount
  - sh:qualifiedMaxCount

```
ex:QualifiedValueShapeExampleShape
  a sh:NodeShape ;
  sh:targetNode
ex:QualifiedValueShapeExampleValidResource ;
  sh:property [
    sh:path ex:parent ;
    sh:minCount 2 ;
    sh:maxCount 2 ;
    sh:qualifiedValueShape [
      sh:path ex:gender ;
      sh:hasValue ex:female ;
    ] ;
    sh:qualifiedMinCount 1 ;
  ]
.
```

Shape-gebaseerde constraints (2)

- sh:node

```
ex:AddressShape
  a sh:NodeShape ;
  sh:property [
    sh:path ex:postalCode ;
    sh:datatype xsd:string ;
    sh:maxCount 1 ;
  ]
.

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

Waarde constraints

- sh:hasValue
- sh:in

```
ex:StanfordGraduate
  a sh:NodeShape ;
  sh:targetNode ex:Alice ;
  sh:property [
    sh:path ex:alumniOf ;
    sh:hasValue ex:Stanford ;
  ]
.
```

```
ex:InExampleShape
  a sh:NodeShape ;
  sh:targetNode ex:RainbowPony ;
  sh:property [
    sh:path ex:color ;
    sh:in ( ex:Pink ex:Purple ) ;
  ]
.
```

Constraint beïnvloedende componenten

- sh:closed
- sh:ignoredProperties

```
ex:ClosedShapeExampleShape
  a sh:NodeShape ;
  sh:targetNode ex:Alice, ex:Bob ;
  sh:closed true ;
  sh:ignoredProperties ( rdf:type ) ;
  sh:property [
    sh:path ex:firstName ;
  ] ;
  sh:property [
    sh:path ex:lastName ;
  ]
.
```

Exercise 003 & 004

Exercise solutions 003 & 004

SHACL - Advanced Features

- SPARQL based constraints
- SPARQL targets
- SHACL rules

SPARQL-based constraints

```
ex:LanguageExamplePropertyShape
  a sh:PropertyShape ;
  sh:targetClass ex:Country ;
  sh:path ex:germanLabel ;
  sh:sparql [
    a sh:SPARQLConstraint ;    # This triple is optional
    sh:message "Values are literals with German language tag.";
    sh:prefixes ex: ;
    sh:select """
      SELECT $this ?value
      WHERE {
        $this $PATH ?value .
        FILTER (
         !isLiteral(?value) || !langMatches(lang(?value), "de")
       )
      }
      """ ;
  ]
.
```

SHACL advanced features (1)

- Custom targets

```
ex:USCitizenShape
  a sh:NodeShape ;
  sh:target [
    a sh:SPARQLTarget ;
    sh:prefixes ex: ;
    sh:select """
      SELECT ?this
      WHERE {
        ?this a ex:Person .
        ?this ex:bornIn ex:USA .
      }
      """ ;
  ] ;
 ...
```

SHACL advanced features (2)

- SHACL rules

```
ex:Rectangle
  a rdfs:Class, sh:NodeShape ;
  rdfs:label "Rectangle" ;
  sh:property [
    sh:path ex:height ;
    sh:datatype xsd:integer ;
    sh:maxCount 1 ;
    sh:minCount 1 ;
    sh:name "height" ;
  ] ;
  sh:property [
    sh:path ex:width ;
    sh:datatype xsd:integer ;
    sh:maxCount 1 ;
    sh:minCount 1 ;
    sh:name "width" ;
  ] ;
  sh:rule [
    a sh:TripleRule ;
    sh:subject sh:this ;
    sh:predicate rdf:type ;
    sh:object ex:Square ;
    sh:condition ex:Rectangle ;
    sh:condition [
      sh:property [
        sh:path ex:width ;
        sh:equals ex:height ;
      ] ;
    ] ;
  ]
.
```

SHACL advanced features (3)

- SHACL **SPARQL** rules

```
cimow_str:Gebied a sh:NodeShape ;
 sh:target [
   sh:select """
   SELECT ?this
   WHERE {
     ?x ogc:hasGeometry ?this .
     ?this a ogc:Geometry
   } """
 ] ;
 sh:rule cimow_str:Gebied_Attributes
.

cimow_str:Gebied_Attributes a sh:SPARQLRule ;
   sh:construct """
     CONSTRUCT {
       ?gebied a cimow:Gebied .
       ?gebied a cimow:Locatie .
       ?gebied cimow:identificatie ?identificatie .
       ?gebied ogc:hasGeometry $this .
       $this rdf:type ?type .
       $this cimow:idealisatie ?idealisatie .
       $this ogc:asWKT ?wkt .
       $this pdok_pdok:asWKT-RD ?wktRD .
     }
     WHERE {
       $this rdf:type ?type .
       FILTER (?type not in (ro:Geometrie))
       $this ogc:asWKT ?wkt .
       $this pdok_pdok:asWKT-RD ?wktRD .
       OPTIONAL {
         $this ro:idealisatie ?idealisatie .
       }
       BIND(IRI(CONCAT(STR($this), '_Gebied')) as ?gebied)
       BIND(CONCAT(STRAFTER(STR($this),
'http://data.informatiehuisruimte.nl/ro/id/geometry/'), '_Gebied') as ?identificatie)
     }
     """ ;
   sh:condition cimow_str:hasTekstobjects
.
```

Andere SHACL use cases

- https://www.w3.org/TR/shacl-ucr/
- Kan gebruikt worden om closed world perspectieven op data te beschrijven.
    - Daarmee ook om object-georiënteerde perspectieven te beschrijven.
        - UI
        - APIs
        - Autorisatie scopes

Extra: Final exercise

Take the ontology from exercise 001.

Create a SHACL shapes graph that covers the ontology!