

Vorlesung Semantic Web



TECHNISCHE
UNIVERSITÄT
DARMSTADT

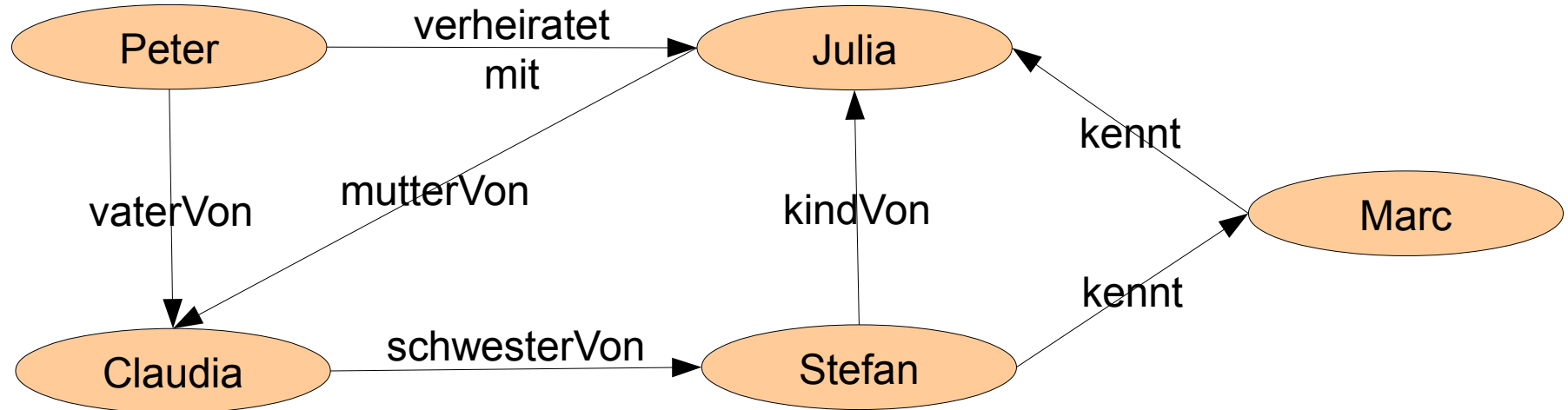
Vorlesung im Wintersemester 2011/2012

Dr. Heiko Paulheim

Fachgebiet Knowledge Engineering

Aufgabe 1

- Gegeben ist folgender RDF-Graph:



Aufgabe 1



- Wen kennt Stefan?

```
SELECT ?p WHERE { :Stefan :kennt ?p }
```

- Ergebnis: Marc

- Wer ist Kind von Julia?

```
SELECT ?k WHERE { ?k :kindVon :Julia }
```

- Ergebnis: Stefan

Aufgabe 1

- Hat Stefan Geschwister?
 - Dafür gibt es kein eigenes Prädikat, wir brauchen Hintergrundwissen
- Ist Julia verheiratet?

```
ASK ?p WHERE { :Julia :verheiratetMit ?p }
```

 - Antwort: false

Aufgabe 1

- Gegeben sind jetzt folgende zusätzliche Axiome in der T-Box:

```
:kennt a owl:SymmetricProperty .
:verheiratetMit rdfs:subPropertyOf :kennt .
:verheiratetMit a owl:SymmetricProperty .
:vaterVon rdfs:subPropertyOf :elternteilVon .
:mutterVon rdfs:subPropertyOf :elternteilVon .
:kindVon owl:inversePropertyOf :elternteilVon .
:kindVon rdfs:subPropertyOf :kennt .
:schwesterVon rdfs:subPropertyOf :geschwisterVon .
:geschwisterVon a owl:symmetricProperty .
:geschwisterVon rdfs:subPropertyOf :kennt .
```

Aufgabe 1



▪ Es ergeben sich (u.a.) diese Folgerungen:

```
:Stefan :kindVon :Julia .  
:kindVon rdfs:subPropertyOf :kennt .  
→ :Stefan :kennt :Julia .
```

```
:Claudia :schwesterVon :Stefan .  
:schwesterVon rdfs:subPropertyOf :geschwisterVon .  
→ :Claudia :geschwisterVon :Stefan .
```

```
:Claudia :geschwisterVon :Stefan .  
:geschwisterVon rdfs:subPropertyOf :kennt .  
→ :Claudia :kennt :Stefan .
```

```
:Claudia :kennt :Stefan .  
:kennt a owl:SymmetricProperty .  
→ :Stefan :kennt :Claudia .
```

Aufgabe 1



- Wen kennt Stefan?

```
SELECT ?p WHERE { :Stefan :kennt ?p }
```

- Ergebnis: Marc, **Julia**, **Claudia**

- Analog: Wer ist Kind von Julia?

```
SELECT ?k WHERE { ?k :kindVon :Julia }
```

- Ergebnis: Stefan, **Claudia**

Aufgabe 1



- Hat Stefan Geschwister?

- geht jetzt so:

- ```
SELECT ?p WHERE { :Stefan :geschwisterVon ?p }
```

- Ergebnis: Claudia

- Ist Julia verheiratet?

- ```
ASK ?p WHERE { :Julia :verheiratetMit ?p }
```

- Antwort: true

- Folgt mit :veheiratetMit a owl:SymmetricProperty

Aufgabe 1



- Wäre es nützlich, das folgende Axiom noch hinzuzufügen?

`:elternteilVon rdfs:subPropertyOf :kennt .`

- Mit Hilfe dieses Axioms ist folgende Ableitung möglich

`?x :elternteilVon ?y . → ?x :kennt ?y .`

Aufgabe 1

- Das bekommt man aber auch so:

```
?x :elternTeilVon ?y .  
:kindVon owl:inversePropertyOf :elternTeilVon .  
→ ?y :kindVon ?x .
```

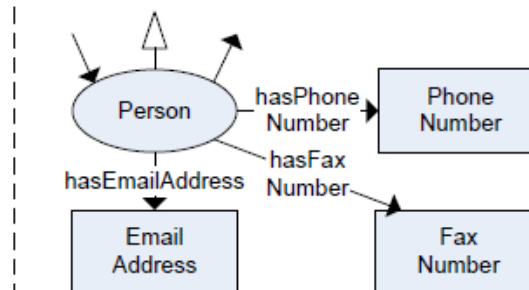
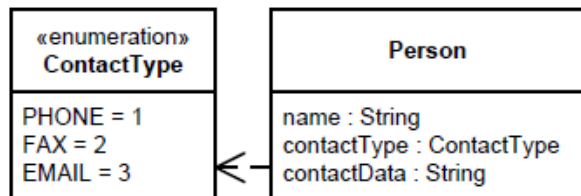
```
?y :kindVon ?x .  
:kindVon owl:subPropertyOf :kennt .  
→ ?y :kennt ?x .
```

```
?y :kennt ?x .  
:kennt a owl:symmetricProperty .  
→ ?x :kennt ?y .
```

- D.h.: das neue Axiom liefert uns keine neuen Folgerungsmöglichkeiten!

Aufgabe 2

- Gegeben sind ein Klassenmodell (links) und eine Ontologie (Mitte). Eine Beispiel-Serialisierung ist rechts angegeben.



```
:p0 rdf:type :Person.  
:p0 :hasPhoneNumber  
"+117432".
```

Aufgabe 2



▪ Beispiel: otm-j

```
@RDF(base="http://foo.bar/",  
      uri="Person")  
interface Person extends RDFResource {  
    @RDF("name")  
    String getName();  
  
    @RDF(???)  
    String getContactData();  
  
    @RDF(???)  
    ContactType getContactType();  
}
```

Aufgabe 2

- Was wir bräuchten
 - ein dynamischer Mechanismus, etwa so

```
@RDF(getContactType() == EMAIL ? hasEmailAddress)  
String getContactData();
```

```
ContactType getContactType();
```

- Das geben aber die derzeitigen direkten Programmier-Modelle nicht her
- Merke: für die Nutzung eines direkten Programmier-Modells ist es nötig, dass Klassenmodell und Ontologie sehr ähnlich sind

Aufgabe 3

- Sie sollen eine Ontologie für ein Autohaus modellieren. Folgende Informationen sind gegeben:
 - Autos werden von verschiedenen Firmen hergestellt.
 - Jedes Auto hat bestimmte Kennzahlen (Beschleunigung, Geschwindigkeit, Verbrauch).
 - Autos können darüber hinaus Zubehör haben, z.B. Anhängerkupplung, Kindersitze oder Dachgepäckträger.
 - Familienautos haben immer einen Kindersitz als Zubehör inklusive.
 - Alle Kombis verfügen mindestens über eine Anhängerkupplung oder einen Dachgepäckträger.

Aufgabe 3



- Autos werden von verschiedenen Firmen hergestellt.

```
:Auto a owl:Class .  
:Firma a owl:Class .  
:hergestelltVon a owl:ObjectProperty .  
:hergestelltVon rdfs:domain :Auto .  
:hergestelltVon rdfs:range :Firma .
```

- Eventuell wollen wir auch noch das:

```
:Auto rdfs:subClassOf [  
  a owl:Restriction ;  
  owl:onProperty :hergestelltVon ;  
  owl:cardinality 1^^xsd:integer ] .
```

- auch das wäre noch möglich:

```
:hergestelltVon a owl:FunctionalProperty .
```

Aufgabe 3



- Jedes Auto hat bestimmte Kennzahlen (Beschleunigung, Geschwindigkeit, Verbrauch).

```
:hatBeschleunigung a owl:DatatypeProperty .  
:hatBeschleunigung rdfs:domain :Auto .  
:hatBeschleunigung rdfs:range xsd:integer .
```

...

- Eventuell wollen wir auch noch das:

```
:Auto rdfs:subClassOf [  
  a owl:Restriction ;  
  owl:onProperty :hatBeschleunigung ;  
  owl:cardinality 1^^xsd:integer ] .
```


Aufgabe 3



- Autos können darüber hinaus Zubehör haben, z.B. Anhängerkupplung, Kindersitze oder Dachgepäckträger.

```
:Zubehör a owl:Class .  
:Anhängerkupplung rdfs:subClassOf :Zubehör .  
:Kindersitz rdfs:subClassOf :Zubehör .  
:Dachgepäckträger rdfs:subClassOf :Zubehör .  
  
:hatZubehör a owl:ObjectProperty .  
:hatZubehör rdfs:domain :Auto .  
:hatZubehör rdfs:range :Zubehör .
```

Aufgabe 3



- Familienautos haben immer einen Kindersitz als Zubehör inklusive.

```
:FamilienAuto rdfs:subClassOf
  :Auto,
  [ a owl:Restriction ;
    owl:onProperty :hatZubehör ;
    owl:someValuesFrom :Kindersitz ] .
```

Aufgabe 3

- Alle Kombis verfügen mindestens über eine Anhängerkupplung oder einen Dachgepäckträger.

```
:Kombi rdfs:subClassOf
  :Auto,
  [ a owl:Restriction ;
    owl:onProperty :hatZubehör ;
    owl:someValuesFrom [
      owl:unionOf (:Anhängerkupplung, :Dachgepäckträger )
    ]
  ] .
```

Aufgabe 3



- Welche Komplexität hat die Ontologie?
- Zunächst: wir haben owl:unionOf verwendet
 - damit sind wir schon mal nicht mehr in OWL Lite!
- Wir sind also in OWL DL
 - wir haben verwendet (alle Optionen ausschöpfend):
 - Kardinalitätsrestriktionen, funktionale Properties, Datentypen
 - also: SNF(D)

Aufgabe 4

- Gegeben drei Häuser, ein braunes, ein gelbes, und ein blaues, die nebeneinander stehen und von 1-3 nummeriert sind. In jedem der Häuser wohnt eine Person. Jede der Personen hat ein anderes Hobby (Fußball, Rugby oder Eishockey) und ein anderes Lieblingsessen (Pfannkuchen, Waffeln oder Eis). Gegeben sind darüber hinaus noch folgende Fakten:
 - Die Person in Haus 2 isst am liebsten Pfannkuchen.
 - Haus 3 ist nicht braun.
 - Die Person in Haus 3 spielt nicht Rugby.
 - Die Person, die am liebsten Waffeln isst, wohnt im gelben Haus.
 - Haus 3 ist nicht gelb.
 - Die Person in Haus 2 spielt Fußball.

Aufgabe 4

- Wir haben es zu tun mit vier Klassen:
 - Personen, Hobbys, Essen, und Häusern.
 - alle vier sind abgeschlossen und bestehen aus verschiedenen Individuen

- Zum Beispiel Hobby:

```
Hobby a owl:Class .  
Hobby owl:oneOf (:fußball :eishockey :rugby) .  
:fußball owl:differentFrom :eishockey, :rugby .
```

- Bei Häusern haben wir zwei Aufzählungen:

```
Haus a owl:Class .  
Haus owl:oneOf (:linkesHaus :mittleresHaus :rechtesHaus) .  
Haus owl:oneOf (:braunesHaus, :gelbesHaus, :blauesHaus) .
```

Aufgabe 4

- Personen werden zu Häusern, Hobbys und Essen zugeordnet.
- Dabei hat jede Person genau ein Haus, Hobby, ...,
 - und jedes Haus wird nur von einer Person bewohnt
 - jedes Hobby nur von einer Person ausgeführt
 - jedes Essen nur von einer Person gegessen
- D.h.: alle drei Properties sind sowohl Funktional als auch invers Funktional

```
:wohntIn a owl:ObjectProperty, owl:FunctionalProperty,  
         owl:InverseFunctionalProperty .
```

```
:wohntIn rdfs:domain :Person .
```

```
:wohntIn rdfs:range :Haus .
```

```
:Person rdfs:subClassOf [  
  a owl:Restriction ;  
  owl:onProperty :wohntIn ;  
  owl:cardinality 1^^xsd:integer ] .
```

Aufgabe 4



- Häuser sind Nachbarhäuser von anderen (symmetrisch):

```
:nachbarhausVon a owl:ObjectProperty,  
                  owl:SymmetricProperty .  
:linkesHaus :nachbarhausVon :mittleresHaus .  
:mittleresHaus :nachbarhausVon :rechtesHaus .
```

- Das ist soweit das Hintergrundwissen

Aufgabe 4

- Die Person in Haus 2 isst am liebsten Pfannkuchen.
- Wir definieren uns hier eine Person, die in Haus 2 lebt und Pfannkuchen isst:

```
p0 a :Person .  
p0 :wohntIn :mittleresHaus .  
p0 :isst :Pfannkuchen .
```

- Haus 3 ist nicht braun.

```
:braunesHaus owl:differentFrom :rechtesHaus .
```

Aufgabe 4



- Die Person in Haus 3 spielt nicht Rugby.

```
p1 a :Person .
p1 :wohntIn :rechtesHaus .
_x a owl:NegativeObjectPropertyAssertion [
  owl:sourceIndividual :p1;
  owl:targetIndividual :rugby;
  owl:assertionProperty :hatHobby ] .
```

- Die Person, die am liebsten Waffeln isst, wohnt im gelben Haus.

```
p2 a :Person .
p2 :wohntIn :gelbesHaus .
p2 :isst :Waffeln .
```

Aufgabe 4

- Haus 3 ist nicht gelb.

```
:rechtesHaus owl:differentFrom :gelbesHaus .
```

- Die Person in Haus 2 spielt Fußball.

```
p3 a :Person .  
p3 :wohntIn :mittleresHaus .  
p3 :hatHobby :Fußball .
```

- Moment mal, jetzt haben wir ja vier Personen (p0-p3)?!
- Aber es gilt: Non Unique Name Assumption
- Hoffnung: der Reasoner wird's richten

Aufgabe 4

- Ob Du wirklich richtig stehst,
siehst Du, wenn der Reasoner angeht...

Vorlesung Semantic Web



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Vorlesung im Wintersemester 2011/2012

Dr. Heiko Paulheim

Fachgebiet Knowledge Engineering