Vorlesung Semantic Web

TECHNISCHE UNIVERSITÄT DARMSTADT

Vorlesung im Wintersemester 2012/2013

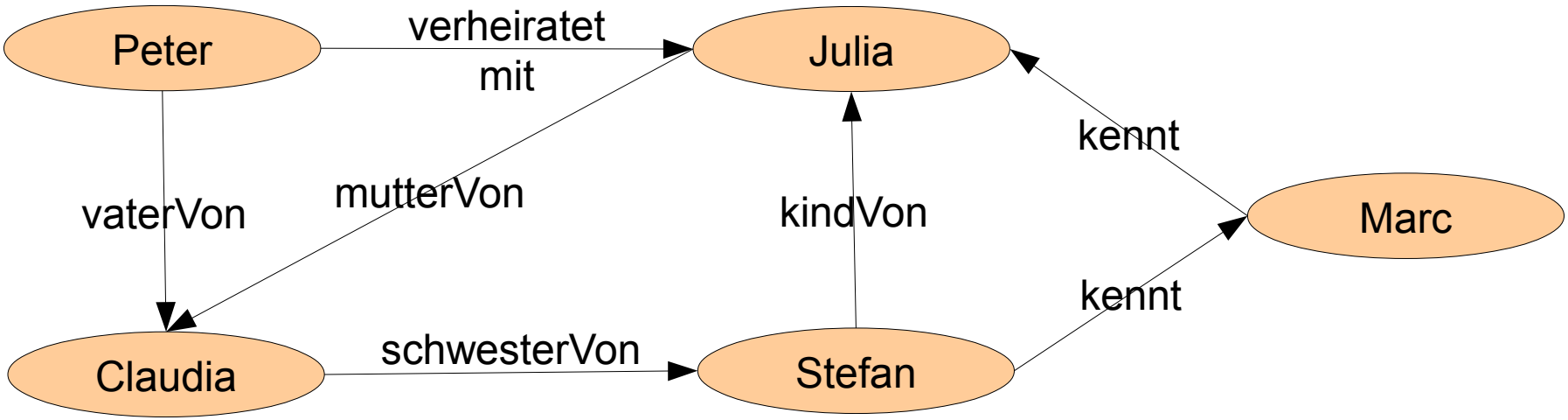
Dr. Heiko Paulheim

Fachgebiet Knowledge Engineering

Aufgabe 1

TECHNISCHE UNIVERSITÄT DARMSTADT

- Gegeben ist folgender RDF-Graph:



Aufgabe 1

TECHNISCHE UNIVERSITÄT DARMSTADT

- 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

TECHNISCHE UNIVERSITÄT DARMSTADT

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

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

  - Antwort: `false`

Aufgabe 1

TECHNISCHE UNIVERSITÄT DARMSTADT

- 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

TECHNISCHE UNIVERSITÄT DARMSTADT

- 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

TECHNISCHE UNIVERSITÄT DARMSTADT

- 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

TECHNISCHE UNIVERSITÄT DARMSTADT

- 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

TECHNISCHE UNIVERSITÄT DARMSTADT

- 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

TECHNISCHE UNIVERSITÄT DARMSTADT

- 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 3

TECHNISCHE UNIVERSITÄT DARMSTADT

- Zwischen Objekten der Klasse *Number* sind unter anderem die Relationen *previous, next, lessThan, greaterThan* und *primefactor* definiert.

- Formulieren Sie unter Verwendung (nur) dieser Relationen folgende Abfragen in SPARQL:
  - Finde alle geraden Zahlen.
  - Finde alle Zahlen, die direkter Nachfolger eines ihrer Primfaktoren sind.
  - Finde alle ungeraden Zahlen.
  - Finde alle Primzahlen.
  - Finde alle Nicht-Primzahlen.
  - Finde alle Primzahl-Zwillinge (d.h. zwei Primzahlen, bei denen die eine um zwei größer ist als die andere).

Aufgabe 3

TECHNISCHE UNIVERSITÄT DARMSTADT

- Finde alle geraden Zahlen.
- Idee: gerade Zahlen haben 2 als Primfaktor.

```
SELECT ?n WHERE {?n :primefactor :2}
```

Aufgabe 3

TECHNISCHE UNIVERSITÄT DARMSTADT

- Finde alle Zahlen, die Nachfolger eines ihrer Primfaktoren sind
- Das geht relativ straight forward:

```
SELECT ?n WHERE   { ?n :primefactor ?x .
                    ?x :next ?n }
```

Aufgabe 3

TECHNISCHE UNIVERSITÄT DARMSTADT

- Finde alle ungeraden Zahlen
- *Idee:* bei ungeraden Zahlen ist die nächste Zahl gerade

```
SELECT ?n WHERE {?n :next ?next .
                 ?next :primefactor :2}
```

Aufgabe 3

TECHNISCHE UNIVERSITÄT DARMSTADT

- Finde alle Nicht-Primzahlen
- *Idee:* Alle Zahlen, die einen Primfaktor größer als 1 und kleiner sich selbst haben

```
SELECT ?n WHERE { ?n :primefactor ?x .
                  ?x :greaterThan :1 .
                  ?x :lessThan ?n }
```

Aufgabe 3

TECHNISCHE UNIVERSITÄT DARMSTADT

- Finde alle Primzahlen
- *Idee:* es darf keinen Primfaktor zwischen 1 und n geben
  - Verneinungstrick

```
SELECT ?n WHERE     { ?n a :Number }
          OPTIONAL  { ?n :primefactor ?x .
                      ?x :greaterThan :1 .
                      ?x :lessThan ?n }
          FILTER    { !BOUND(?x) }
```

Aufgabe 3

TECHNISCHE UNIVERSITÄT DARMSTADT

- Einfacherer Lösungsweg:
  - Primzahlen sind alle Zahlen, die Primfaktoren von größeren Zahlen sind:

```
SELECT ?n WHERE { ?x :primefactor ?n }
```

Aufgabe 3

TECHNISCHE UNIVERSITÄT DARMSTADT

- Finde alle Primzahlzwillinge
- Kombination aus doppelter Lösung der vorherigen

```
SELECT ?n1 ?n2 WHERE    { ?n1 :next ?n . ?n :next ?n2 }
               OPTIONAL { ?n1 :primefactor ?x1 .
                          ?x1 :greaterThan :1 .
                          ?x1 :lessThan ?n1 }
               OPTIONAL { ?n2 :primefactor ?x2 .
                          ?x2 :greaterThan :1 .
                          ?x2 :lessThan ?n2 }
               FILTER   { !BOUND(?x1) && !BOUND(?x2) }
```

Aufgabe 3

TECHNISCHE UNIVERSITÄT DARMSTADT

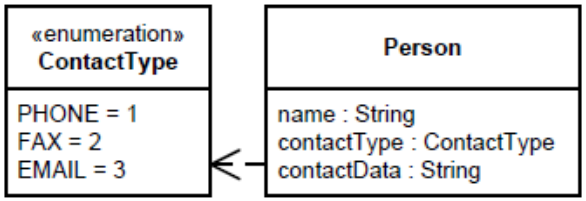
- Alternative mit dem einfacheren Lösungsweg:

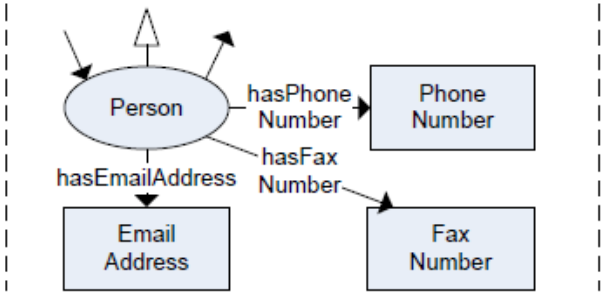
```
SELECT ?n1 ?n2 WHERE    { ?n1 :next ?n . ?n :next ?n2 .
                          ?x1 :primefactor ?n1 .
                          ?x2 :primefactor ?n2 }
```

Aufgabe 3

TECHNISCHE UNIVERSITÄT DARMSTADT

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






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

Aufgabe 3

TECHNISCHE UNIVERSITÄT DARMSTADT

- 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 3

TECHNISCHE UNIVERSITÄT DARMSTADT

- 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

TECHNISCHE UNIVERSITÄT DARMSTADT

- Weiteres Problem, das auftreten könnte:
  - Gegeben

```
:hans a :Person .
:hans :hasPhoneNumber +12345 .
:hans :hasFaxNumber +12346 .
```

  - Welche Kontaktinformation sollen wir verwenden, welche verwerfen?
  - Oder zwei Personenobjekte speichern?
    - Wie wissen wir dann, dass die zusammen gehören?

- Fragen über Fragen, die wir uns mit einem direkten Programmiermodell einhandeln!

Aufgabe 4

TECHNISCHE UNIVERSITÄT DARMSTADT

- 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. Anhängekupplung, Kindersitze oder Dachgepäckträger.
  - Familienautos haben immer einen Kindersitz als Zubehör inklusive.
  - Alle Kombis verfügen mindestens über eine Anhängekupplung oder einen Dachgepäckträger.

Aufgabe 4

TECHNISCHE UNIVERSITÄT DARMSTADT

- 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 4

TECHNISCHE UNIVERSITÄT DARMSTADT

- 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 4

TECHNISCHE UNIVERSITÄT DARMSTADT

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

```
:Zubehör a owl:Class .
:Anhängekupplung rdfs:subClassOf :Zubehör .
:Kindersitz rdfs:subClassOf :Zubehör .
:Dachgepäckträger rdfs:subClassOf :Zubehör .

:hatZubehör a owl:ObjectProperty .
:hatZubehör rdfs:domain :Auto .
:hatZubehör rdfs:range :Zubehör .
```

Aufgabe 4

TECHNISCHE UNIVERSITÄT DARMSTADT

- Familienautos haben immer einen Kindersitz als Zubehör inklusive.

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

Aufgabe 4

TECHNISCHE UNIVERSITÄT DARMSTADT

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

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

Aufgabe 4

TECHNISCHE UNIVERSITÄT DARMSTADT

- 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):
  - Kardinalitätsrestriktrionen, funktionale Properties, Datentypen
  - also: SNF(D)

Aufgabe 5

TECHNISCHE UNIVERSITÄT DARMSTADT

- 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 5

TECHNISCHE UNIVERSITÄT DARMSTADT

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

- Hobby:

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

- Essen:

```
Essen a owl:Class .
Essen owl:oneOf (:pfannkuchen :waffeln :eis) ....
```

Aufgabe 5

TECHNISCHE UNIVERSITÄT DARMSTADT

- Bei Häusern haben wir zwei Aufzählungen:

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

- Hier ist die Non-unique Name Assumption nützlich:
  - Ein Haus kann z.B. gleichzeitig linkes Haus und gelbes Haus sein

Aufgabe 5

TECHNISCHE UNIVERSITÄT DARMSTADT

- Personen werden zu Häusern, Hobbys und Essen zugeordnet.
- Dabei hat jede Person genau ein Haus, Hobby, …,
  - und jedes Haus wird nur von genau einer Person bewohnt
  - jedes Hobby nur von genau einer Person ausgeführt
  - jedes Essen nur von genau 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 5

TECHNISCHE UNIVERSITÄT DARMSTADT

- Häuser sind Nachbarhäuser von anderen (symmetrisch):

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

- Das ist soweit das Hintergrundwissen

Aufgabe 5

TECHNISCHE UNIVERSITÄT DARMSTADT

- 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 5

TECHNISCHE UNIVERSITÄT DARMSTADT

- 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 5

TECHNISCHE UNIVERSITÄT DARMSTADT

- 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 5

TECHNISCHE UNIVERSITÄT DARMSTADT

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

Vorlesung Semantic Web

TECHNISCHE UNIVERSITÄT DARMSTADT

Vorlesung im Wintersemester 2012/2013

Dr. Heiko Paulheim

Fachgebiet Knowledge Engineering