

Vorlesung Semantic Web



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Vorlesung im Wintersemester 2012/2013

Dr. Heiko Paulheim

Fachgebiet Knowledge Engineering

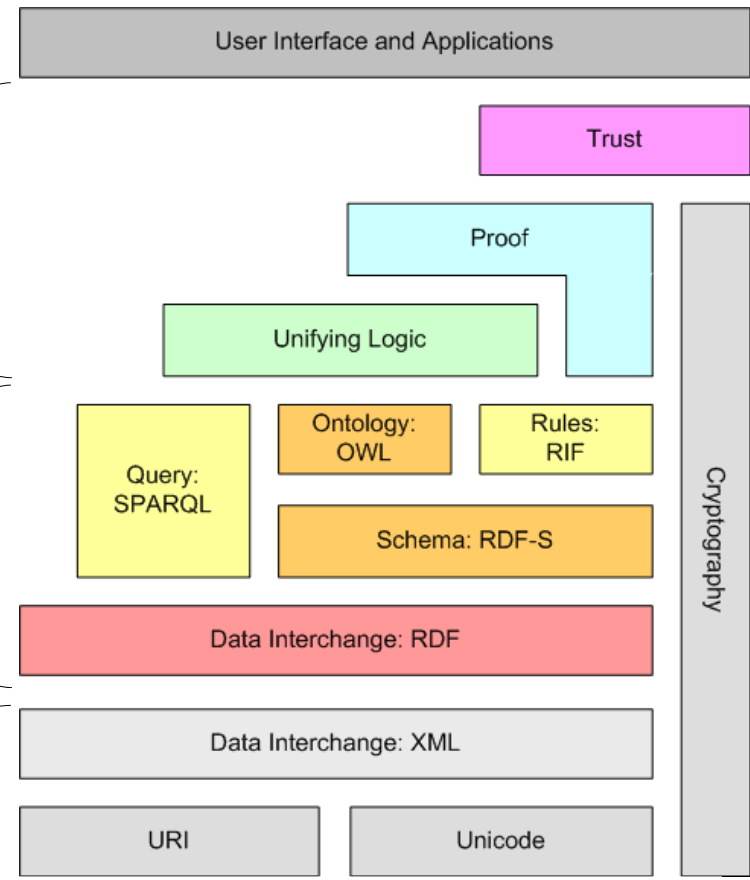
Semantic Web – Aufbau



here be dragons...

Semantic-Web-
Technologie
(Fokus der Vorlesung)

Technische
Grundlagen



Berners-Lee (2009): *Semantic Web and Linked Data*
<http://www.w3.org/2009/Talks/0120-campus-party-tbl/>

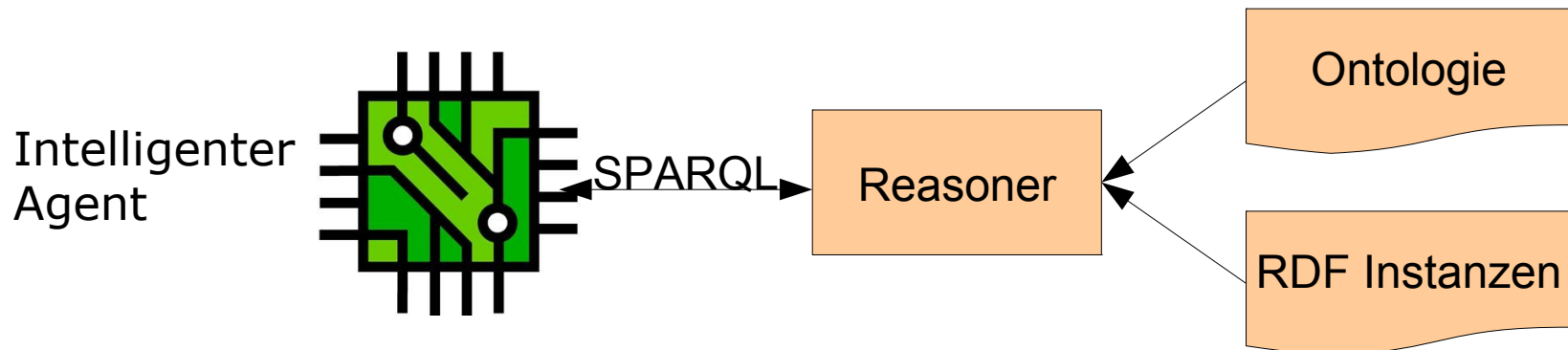
Was bisher geschah

- Komplexe Ontologien
 - RDF Schema ist leichtgewichtig
 - OWL kann mehr

- Reasoning
 - für RDF Schema können wir das schon
 - wie funktioniert Reasoning mit OWL?

Recap: Kombination von SPARQL & Reasoning

- Reasoning mit RDF Schema haben wir schon kennen gelernt
- Viele Reasoner haben auch eine SPARQL-Schnittstelle



Aufgaben für einen Reasoner

- Was würden wir gern von einem Reasoner wissen?
 - Subklassenbeziehung
 - z.B.: Sind alle Säugetiere Landbewohner?
 - Klassenäquivalenz
 - z.B.: Sind alle Wasserbewohner mit Flossen Fische und umgekehrt?
 - Klassendisjunktheit
 - z.B.: Gibt es Tiere, die zu den Säugetieren und zu den Vögeln gehören?
 - Klassenkonsistenz
 - z.B.: Kann es eierlegende Säugetiere geben?
 - Instanzbeziehung
 - z.B.: Ist Flipper ein Delfin?
 - Klassenaufzählung
 - z.B.: Liste alle Elefanten auf

Aufgaben für einen Reasoner

- Das sind ziemlich viele mögliche Aufgaben
- Macht einen Reasoner ziemlich kompliziert

- Naiver Forward-Chaining-Algorithmus
 - ist nicht besonders gut skalierbar
 - kann nicht mit Konjunktion umgehen
 - die gibt es aber in OWL, z.B. unionOf

Aufgaben für einen Reasoner

- Ansatz: alle Aufgaben auf ein gemeinsames Grundproblem zurückführen
- Zum Beispiel: Widerspruchsfreiheit
 - d.h.: es lässt sich nicht gleichzeitig eine Aussage und ihr Gegenteil aus T-Box und A-Box ableiten

Beispiel: Widerspruchsfreiheit

▪ Beispiel:

```
:Man a owl:Class .  
:Woman a owl:Class .  
:Man owl:disjointWith :Woman .  
  
:Alex a :Man .  
:Alex a :Woman .
```


Beispiel: Widerspruchsfreiheit

- Daraus folgt:

- $:Man \cap :Woman = \emptyset$

- ```
owl:Nothing owl:intersectionOf (:Man :Woman) .
```

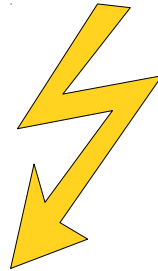
- $:Alex \in (:Man \cap :Woman)$

- ```
:Alex a [ a owl:Class; owl:intersectionOf (:Man :Woman) ] .
```

- also:

- $:Alex \in \emptyset$

- ```
:Alex a owl:Nothing .
```



# Neudefinition der Reasoner-Tasks



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

- Subklassenbeziehung  
Student  $\subseteq$  Person  $\Leftrightarrow$  „Jeder Student ist eine Person“
- Prüfungsmethode: indirekter Beweis
  - "Erfinde" eine Instanz  $i$
  - Setze Student( $i$ ) und  $\neg$ Person( $i$ )
  - Prüfe auf Widerspruchsfreiheit
    - bei Widerspruch: Student  $\subseteq$  Person muss gelten
    - kein Widerspruch: Student  $\subseteq$  Person kann nicht abgeleitet werden



# Beispiel: Subklassenbeziehung

- Ontologie:

```
:Student owl:subClassOf :UniversityMember .
:UniversityMember owl:subClassOf :Person .
```

- Neu hinzugefügte Instanz:

```
:i a :Student .
:i a [owl:complementOf :Person] .
```

# Beispiel: Subklassenbeziehung

- Aus

```
:i a :Student .
```

und

```
:Student owl:subClassOf :UniversityMember .
```

folgt

```
:i a :UniversityMember .
```

- analog folgt mit

```
:UniversityMember owl:subClassOf :Person .
```

- auch

```
:i a Person .
```

# Beispiel: Subklassenbeziehung

- Jetzt haben wir

```
:i a :Person .
```

```
:i a [owl:complementOf :Person] .
```

damit gilt

```
:i a [owl:intersectionOf (:Person
 [owl:complementOf :Person]))] .
```

- und folglich

```
:i a owl:Nothing .
```



# Neudefinition der Reasoner-Tasks



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

- Klassenäquivalenz
  - $\text{Person} \equiv \text{Mensch}$
- Das kann man auflösen in
  - $\text{Person} \subseteq \text{Mensch}$  und
  - $\text{Mensch} \subseteq \text{Person}$
- Also: zweimal Subklassenbeziehung zeigen
  - können wir schon
  
- Klassendisjunktheit
  - Sind C und D disjunkt?
  - "Erfinde" eine Instanz i
  - setze C(i) und D(i)
  - das haben wir schon gesehen (Beispiel: Alex)



- Kann eine Klasse Instanzen haben?
  - zum Beispiel: verheiratete Junggesellen

```
:Junggeselle owl:subClassOf :Mann .
:Junggeselle owl:subClassOf
 [a owl:Restriction;
 owl:onProperty :verheiratetMit;
 owl:cardinality 0] .
:VerheiratetePerson owl:subClassOf [
 a owl:Restriction;
 owl:onProperty :verheiratetMit;
 owl:cardinality 1] .

:VerheirateterJunggeselle owl:intersectionOf
 (:Junggeselle :VerheiratetePerson) .
```

- Kann eine Klasse Instanzen haben?
  - zum Beispiel: verheiratete Junggesellen

```
:Junggeselle owl:subClassOf :Mann .
:Junggeselle owl:subClassOf
 [a owl:Restriction;
 owl:onProperty :verheiratetMit;
 owl:cardinality 0] .
:VerheiratetePerson owl:subClassOf [
 a owl:Restriction;
 owl:onProperty :verheiratetMit;
 owl:cardinality 1] .

:VerheirateterJunggeselle owl:intersectionOf
 (:Junggeselle :VerheiratetePerson) .
```



# Klassenkonsistenz

- Prüfen auf Klassenkonsistenz:
  - "Erfinde" eine Instanz  $i$
  - Füge sie als Instanz der Klasse hinzu
  - Prüfe, ob ein Widerspruch entsteht

# Beispiel: Klassenkonsistenz



- In unserem Beispiel entsteht:

```
:i a [
 a owl:Restriction;
 owl:onProperty :verheiratetMit;
 owl:cardinality 1] .
```

- und

```
:i a [
 a owl:Restriction;
 owl:onProperty :verheiratetMit;
 owl:cardinality 0] .
```



# Aufgaben für einen Reasoner



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

- Instanzbeziehung
  - Ist Flipper ein Delfin?
- Prüfung:
  - setze  $\neg$ Delfin(Flipper)
  - prüfe auf Widerspruch
- Instanzen einer Klasse aufzählen
  - Prüfe die Instanzbeziehung für alle bekannten Individuen durch

# Aufgaben für einen Reasoner



- Was würden wir gern von einem Reasoner wissen?
  - Subklassenbeziehung
    - z.B.: Sind alle Säugetiere Landbewohner?
  - Klassenäquivalenz
    - z.B.: Sind alle Wasserbewohner mit Flossen Fische und umgekehrt?
  - Klassendisjunktheit
    - z.B.: Gibt es Tiere, die zu den Säugetieren und zu den Vögeln gehören?
  - Klassenkonsistenz
    - z.B.: Kann es eierlegende Säugetiere geben?
  - Instanzbeziehung
    - z.B.: Ist Flipper ein Delfin?
  - Klassenaufzählung
    - z.B.: Liste alle Elefanten auf

# Aufgaben für einen Reasoner

- Wir haben jetzt gesehen:
  - alle möglichen Aufgaben lassen sich auf dieselbe Aufgabe herunterbrechen
  - nämlich Prüfung auf Konsistenz
- Jetzt muss ein Reasoner also nur noch einen Task implementieren

# Tableau-Verfahren

- Das heute am häufigsten eingesetzte Reasoning-Verfahren
- kommt aus dem Bereich der Beschreibungslogiken
- Ziel:
  - Eine logische Faktenbasis auf Widersprüche prüfen
- Verfahren:
  - Aussagen ableiten, bis ein Widerspruch entsteht

## ▪ Klassen und Instanzen

- $C(x)$   $\leftrightarrow x \text{ a } C$  .
- $R(x,y)$   $\leftrightarrow x R y$  .
- $C \sqsubseteq D$   $\leftrightarrow C \text{ rdfs:subClassOf } D$
- $C \equiv D$   $\leftrightarrow C \text{ owl:equivalentClass } D$
- $C \sqsubseteq \neg D$   $\leftrightarrow C \text{ owl:disjointWith } D$
- $C \equiv \neg D$   $\leftrightarrow C \text{ owl:complementOf } D$
- $C \equiv D \sqcap E$   $\leftrightarrow C \text{ owl:intersectionOf } (D E)$  .
- $C \equiv D \sqcup E$   $\leftrightarrow C \text{ owl:unionOf } (D E)$  .
- $T$   $\leftrightarrow \text{owl:Thing}$
- $\perp$   $\leftrightarrow \text{owl:Nothing}$

## ▪ Restriktionen

- $C \sqsubseteq \forall R.D$       $\leftrightarrow C \text{ owl:subClassOf } [ a \text{ owl:Restriction; owl:onProperty } R; \text{ owl:allValuesFrom } D ] .$
- $C \sqsubseteq \exists R.D$       $\leftrightarrow C \text{ owl:subClassOf } [ a \text{ owl:Restriction; owl:onProperty } R; \text{ owl:someValuesFrom } D ] .$
- $C \sqsubseteq \geq nR$       $\leftrightarrow C \text{ owl:subClassOf } [ a \text{ owl:Restriction; owl:onProperty } R; \text{ owl:minCardinality } n ] .$
- $\exists R.T \sqsubseteq C$       $\leftrightarrow R \text{ rdfs:domain } C .$
- $T \sqsubseteq \forall R.C$       $\leftrightarrow R \text{ rdfs:range } C .$



- Abkürzung für universell gültige Aussagen

- $D \sqcap E \quad \leftrightarrow \forall x: x \in D \sqcap E$
- $D \sqcup E \quad \leftrightarrow \forall x: x \in D \sqcup E$
- $\exists R.C \quad \leftrightarrow \forall x: x \in \exists R.C$   
 $\quad \quad \quad \leftrightarrow \forall x: \exists y: y \in C \wedge R(x,y)$
- $\forall R.C \quad \leftrightarrow \forall x: x \in \forall R.C$   
 $\quad \quad \quad \leftrightarrow \forall x: R(x,y) \rightarrow y \in C$

- Ziel bei Description Logics:

- Logikdefinition möglichst in dieser Form

# Vorbereitung: Ontologien in Negationsnormalform (NNF)



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

- Negationsnormalform:
  - $\exists$  und  $\equiv$  kommen nicht vor
  - Negation nur vor atomaren Klassen oder Aussagen
- Eine vereinfachte Schreibweise für Ontologien
- Von Tableau-Reasonern als Eingabe verlangt
- Wie kommt man da hin?

# Ontologien in Negationsnormalform (NNF)

- Elimination von  $\exists$ :
  - Ersetze  $C \exists D$  durch  $\neg C \sqcup D$
  - Kurzschreibweise für:  $\forall x: \neg C(x) \vee D(x)$
- Warum gilt diese Äquivalenz?
  - $C \exists D$  ist äquivalent zu  $C(x) \rightarrow D(x)$

| C(x)  | D(x)  | $C(x) \rightarrow D(x)$ | $\neg C(x) \vee D(x)$ |
|-------|-------|-------------------------|-----------------------|
| true  | true  | true                    | true                  |
| true  | false | false                   | false                 |
| false | true  | true                    | true                  |
| false | false | true                    | true                  |

# Ontologien in Negationsnormalform

- Elimination von  $\equiv$ :
  - Ersetze  $C \equiv D$  durch  $C \sqsubseteq D$  und  $D \sqsubseteq C$
  - Verfahren wie gehabt

# Ontologien in Negationsnormalform (NNF)

## ▪ Transformationsregeln

- $\text{NNF}(C) = C$  (für atomare  $C$ )
- $\text{NNF}(\neg C) = \neg C$  (für atomare  $C$ )
- $\text{NNF}(\neg \neg C) = C$
- $\text{NNF}(C \sqcup D) = \text{NNF}(C) \sqcup \text{NNF}(D)$
- $\text{NNF}(C \sqcap D) = \text{NNF}(C) \sqcap \text{NNF}(D)$
- $\text{NNF}(\neg(C \sqcap D)) = \text{NNF}(\neg C) \sqcup \text{NNF}(\neg D)$
- $\text{NNF}(\neg(C \sqcup D)) = \text{NNF}(\neg C) \sqcap \text{NNF}(\neg D)$
- $\text{NNF}(\forall R.C) = \forall R.\text{NNF}(C)$
- $\text{NNF}(\exists R.C) = \exists R.\text{NNF}(C)$
- $\text{NNF}(\neg \forall R.C) = \exists R.\text{NNF}(\neg C)$
- $\text{NNF}(\neg \exists R.C) = \forall R.\text{NNF}(\neg C)$

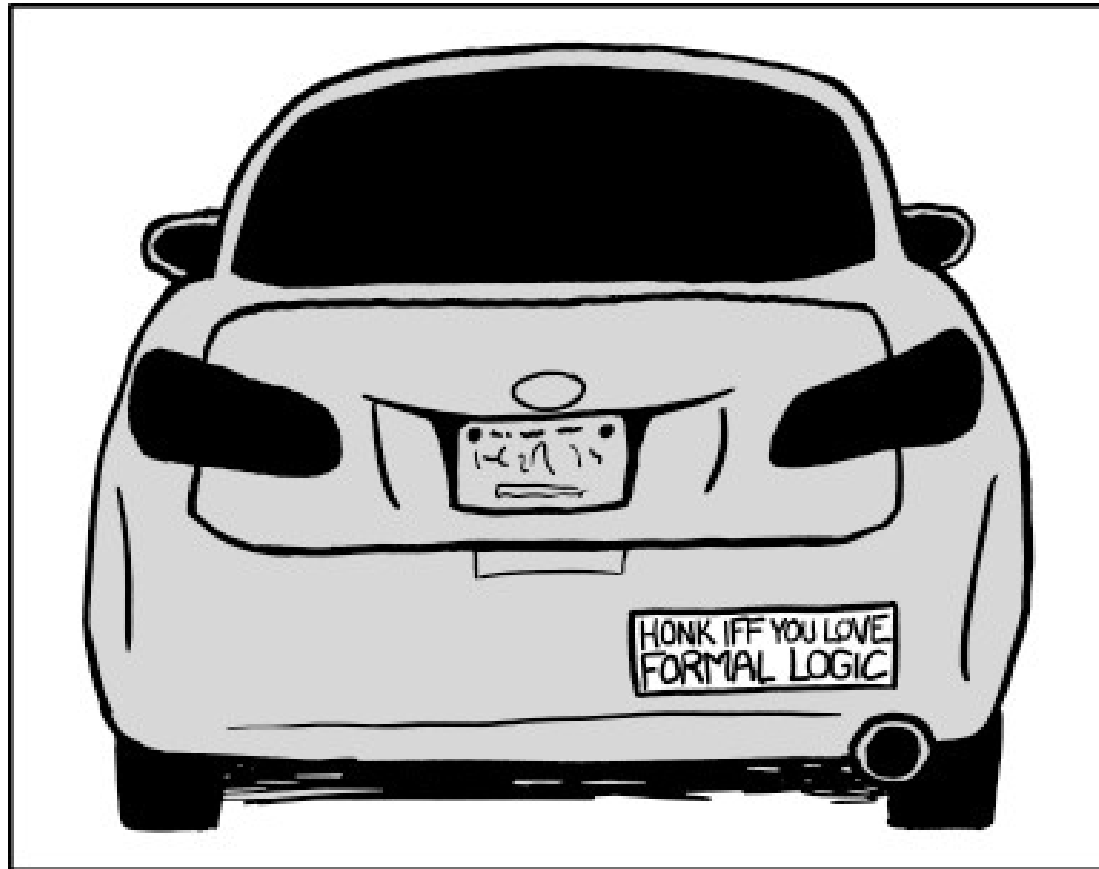
# Ontologien in Negationsnormalform (NNF)

- Beispiel: ein Verein kann natürliche und juristische Personen als Mitglieder haben
  - `:hasMember rdfs:range [ owl:unionOf (:NaturalPerson :LegalPerson) ]`
  - $T \sqsubseteq \forall \text{hasMember} . (\text{NaturalPerson} \sqcup \text{LegalPerson})$
  - $\neg T \sqcup \forall \text{hasMember} . (\text{NaturalPerson} \sqcup \text{LegalPerson})$
  - Kürzer:  
 $\forall \text{hasMember} . (\text{NaturalPerson} \sqcup \text{LegalPerson})$
- Damit sind alle Aussagen in NNF  
(keine Negation vor nicht-Elementaren Ausdrücken)

# Ontologien in Negationsnormalform (NNF)



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT



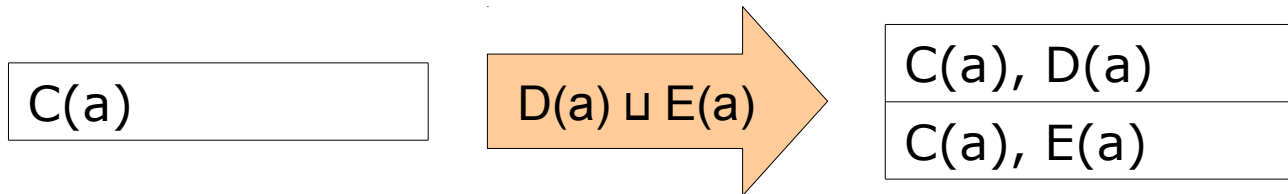
<http://xkcd.com/1033/>

# Der einfache Tableau-Algorithmus



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

- Tableau: Sammlung von abgeleiteten Aussagen
  - wird nach und nach erweitert
  - wie beim Forward Chaining
- Bei Konjunktion
  - Teilen in zwei Tableaus





# Wann ist eine Wissensbasis widerspruchsfrei?



- Tableau wird kontinuierlich erweitert und geteilt
- Widerspruchsfreiheit, wenn...
  - keine weiteren Axiome erzeugt werden können
  - mindestens ein Teiltabelleau ohne Widerspruch bleibt
  - Ein Teiltabelleau enthält einen Widerspruch, wenn es ein Axiom und sein direktes Gegenteil enthält
    - z.B. Mensch(Hans) und  $\neg$ Mensch(Hans)
    - man sagt dann, dass das Teiltabelleau *abgeschlossen* ist

# Der einfache Tableau-Algorithmus



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

- Gegeben: eine Wissensbasis  $W$  in NNF
- Solange nicht alle Teiltableaus abgeschlossen sind
  - \* Wähle ein nicht abgeschlossenes Teiltableau  $T$  und ein  $A \in W \cup T$
  - Wenn  $A$  noch nicht in  $T$  enthalten
    - Wenn  $A$  eine A-Box-Aussage ist
      - füge  $A$  zu  $T$  hinzu
      - zurück zu \*
    - Wenn  $A$  eine T-Box-Aussage ist
      - Wähle ein Individuum  $a \in W \cup T$
      - Füge  $A(a)$  zu  $T$  hinzu
      - zurück zu \*
  - sonst
    - Erweitere das Tableau mit Konsequenzen aus  $A$
    - zurück zu \*



# Der einfache Tableau-Algorithmus



- Erweiterungsregeln für eine Aussage A

| Nr | Aussage            | Aktion                                                                             |
|----|--------------------|------------------------------------------------------------------------------------|
| 1  | $C(a)$             | Füge $C(a)$ hinzu                                                                  |
| 2  | $R(a,b)$           | Füge $R(a,b)$ hinzu                                                                |
| 3  | $C$                | Wähle ein Individuum $a$ , füge $C(a)$ hinzu                                       |
| 4  | $(C \cap D)(a)$    | Füge $C(a)$ und $D(a)$ hinzu                                                       |
| 5  | $(C \sqcup D)(a)$  | Teile das Tableau in $T1$ und $T2$ .<br>Füge $C(a)$ zu $T1$ , $D(a)$ zu $T2$ hinzu |
| 6  | $(\exists R.C)(a)$ | Füge $R(a,b)$ und $C(b)$ für ein <i>neues</i> Individuum $b$ hinzu                 |
| 7  | $(\forall R.C)(a)$ | Für alle $b$ mit $R(a,b) \in T$ : füge $C(b)$ hinzu                                |

# Ein einfaches Beispiel



- Gegeben folgende Ontologie:
  - :Animal owl:unionOf (:Mammal :Bird :Fish :Insect ) .
  - :Animal owl:disjointWith :Human .
  - :Seth a :Human .
  - :Seth a :Insect .
  
- Ist diese Wissensbasis konsistent?

# Ein einfaches Beispiel

- Gegeben folgende Ontologie:
  - :Animal owl:disjointWith :Human .
  - :Animal owl:unionOf (:Mammal :Bird :Fish :Insect) .
  - :Seth a :Human .
  - :Seth a :Insect .
  
- Dieselbe Ontologie in DL-NNF:
  - $\neg \text{Animal} \sqcup \neg \text{Human}$
  - $\text{Animal} \sqcup (\neg \text{Mammal} \sqcap \neg \text{Bird} \sqcap \neg \text{Fish} \sqcap \neg \text{Insect})$
  - $\neg \text{Animal} \sqcup (\text{Mammal} \sqcup \text{Bird} \sqcup \text{Fish} \sqcup \text{Insect})$
  - Human(Seth)
  - Insect(Seth)
  
- Wie funktioniert jetzt das Reasoning? Ausprobieren!

# Der einfache Tableau-Algorithmus



- Erweiterungsregeln für eine Aussage A

| Nr | Aussage            | Aktion                                                                             |
|----|--------------------|------------------------------------------------------------------------------------|
| 1  | $C(a)$             | Füge $C(a)$ hinzu                                                                  |
| 2  | $R(a,b)$           | Füge $R(a,b)$ hinzu                                                                |
| 3  | $C$                | Wähle ein Individuum $a$ , füge $C(a)$ hinzu                                       |
| 4  | $(C \cap D)(a)$    | Füge $C(a)$ und $D(a)$ hinzu                                                       |
| 5  | $(C \sqcup D)(a)$  | Teile das Tableau in $T1$ und $T2$ .<br>Füge $C(a)$ zu $T1$ , $D(a)$ zu $T2$ hinzu |
| 6  | $(\exists R.C)(a)$ | Füge $R(a,b)$ und $C(b)$ für ein <i>neues</i> Individuum $b$ hinzu                 |
| 7  | $(\forall R.C)(a)$ | Für alle $b$ mit $R(a,b) \in T$ : füge $C(b)$ hinzu                                |

# Ein einfaches Beispiel



Human(Seth), Insect(Seth)

| Nr | Aussage | Aktion          |
|----|---------|-----------------|
| 1  | C(a)    | Füge C(a) hinzu |

# Ein einfaches Beispiel



Human(Seth), Insect(Seth),  
 $(\neg\text{Animal} \sqcup \neg\text{Human})(\text{Seth})$

| Nr | Aussage | Aktion                                  |
|----|---------|-----------------------------------------|
| 3  | C       | Wähle ein Individuum a, füge C(a) hinzu |



# Ein einfaches Beispiel



Human(Seth), Insect(Seth),  
 $\neg$ Animal(Seth)

Human(Seth), Insect(Seth),  
 $\neg$ Human(Seth)

| Nr | Aussage           | Aktion                                                               |
|----|-------------------|----------------------------------------------------------------------|
| 5  | $(C \sqcup D)(a)$ | Teile das Tableau in T1 und T2.<br>Füge C(a) zu T1, D(a) zu T2 hinzu |

# Ein einfaches Beispiel



Human(Seth), Insect(Seth),  
 $\neg$ Animal(Seth)

Animal  $\sqcup$  ( $\neg$ Mammal  $\sqcap$   $\neg$ Bird  $\sqcap$   $\neg$ Fish  $\sqcap$   $\neg$ Insect)(Seth)

Human(Seth), Insect(Seth),  
 $\neg$ Human(Seth)

| Nr | Aussage | Aktion                                  |
|----|---------|-----------------------------------------|
| 3  | C       | Wähle ein Individuum a, füge C(a) hinzu |

# Ein einfaches Beispiel



Human(Seth), Insect(Seth),

$\neg$ Animal(Seth)

Animal(Seth)

Human(Seth), Insect(Seth),

$\neg$ Animal(Seth)

$(\neg$ Mammal  $\wedge$   $\neg$ Bird  $\wedge$   $\neg$ Fish  $\wedge$   $\neg$ Insect)(Seth)

Human(Seth), Insect(Seth),

$\neg$ Human(Seth)

| Nr | Aussage           | Aktion                                                               |
|----|-------------------|----------------------------------------------------------------------|
| 5  | $(C \sqcup D)(a)$ | Teile das Tableau in T1 und T2.<br>Füge C(a) zu T1, D(a) zu T2 hinzu |

# Ein einfaches Beispiel



Human(Seth), Insect(Seth),  
 $\neg$ Animal(Seth)  
Animal(Seth)

Human(Seth), Insect(Seth),  
 $\neg$ Animal(Seth)  
 $(\neg$ Mammal  $\wedge$   $\neg$ Bird  $\wedge$   $\neg$ Fish  $\wedge$   $\neg$ Insect)(Seth)  
 $\neg$ Mammal(Seth)  $\wedge$   $\neg$ Bird(Seth)  $\wedge$   $\neg$ Fish(Seth)  $\wedge$   $\neg$ Insect(Seth)

Human(Seth), Insect(Seth),  
 $\neg$ Human(Seth)

| Nr | Aussage           | Aktion                       |
|----|-------------------|------------------------------|
| 4  | $(C \wedge D)(a)$ | Füge $C(a)$ und $D(a)$ hinzu |

# Noch ein Beispiel



## ▪ Gegeben folgende Ontologie:

```
:Woman rdfs:subClassOf :Person .
:Man rdfs:subClassOf :Person .
:hasChild rdfs:domain :Person .
:hasChild rdfs:range :Person .
:Peter :hasChild :Julia .
:Julia a :Woman .
:Peter a :Man .
```

# Noch ein Beispiel

- in DL-NNF:
  - ¬Man  $\sqcup$  Person
  - ¬Woman  $\sqcup$  Person
  - ¬ $\exists$ hasChild.T  $\sqcup$  Person
  - $\forall$ hasChild.Person
  - hasChild(Peter,Julia)
  - Woman(Julia)
  - Man(Peter)

# Noch ein Beispiel

hasChild(Peter,Julia)

| Nr | Aussage  | Aktion              |
|----|----------|---------------------|
| 2  | $R(a,b)$ | Füge $R(a,b)$ hinzu |

# Noch ein Beispiel

hasChild(Peter,Julia), Woman(Julia)

| Nr | Aussage | Aktion          |
|----|---------|-----------------|
| 1  | C(a)    | Füge C(a) hinzu |



# Noch ein Beispiel



hasChild(Peter,Julia), Woman(Julia),  
 $(\neg \exists \text{hasChild.T} \sqcup \text{Person})(\text{Peter})$

| Nr | Aussage | Aktion                                  |
|----|---------|-----------------------------------------|
| 3  | C       | Wähle ein Individuum a, füge C(a) hinzu |

# Noch ein Beispiel



hasChild(Peter,Julia), Woman(Julia),  
 $(\neg\exists\text{hasChild.T} \sqcup \text{Person})(\text{Peter})$ ,  
 $\neg\exists\text{hasChild.T}(\text{Peter})$

hasChild(Peter,Julia), Woman(Julia),  
 $(\neg\exists\text{hasChild.T})(\text{Peter})$ ,  
Person(Peter)

| Nr | Aussage           | Aktion                                                               |
|----|-------------------|----------------------------------------------------------------------|
| 5  | $(C \sqcup D)(a)$ | Teile das Tableau in T1 und T2.<br>Füge C(a) zu T1, D(a) zu T2 hinzu |

# Noch ein Beispiel

hasChild(Peter,Julia), Woman(Julia),  
( $\neg\exists$ hasChild.T)(Peter),  
 $\neg$ ParentsOfSons(Peter)

hasChild(Peter,Julia), Woman(Julia),  
( $\neg\exists$ hasChild.T)(Peter),  
Person(Peter),  
 $\neg$ hasChild(Peter,b0),T(b0)

| Nr | Aussage            | Aktion                                                             |
|----|--------------------|--------------------------------------------------------------------|
| 6  | $(\exists R.C)(a)$ | Füge $R(a,b)$ und $C(b)$ für ein <i>neues</i> Individuum $b$ hinzu |

# Noch ein Beispiel

hasChild(Peter,Julia), Woman(Julia),  
( $\neg$ ParentsOfSons  $\sqcup$   $\exists$ hasChild.Man)(Peter),  
 $\neg$ ParentsOfSons(Peter)

hasChild(Peter,Julia), Woman(Julia),  
( $\neg$  $\exists$ hasChild.T)(Peter),  
Person(Peter),  
 $\neg$ hasChild(Peter,b0),T(b0),  
 $\neg$ hasChild(Peter,b1),T(b1),  
...

| Nr | Aussage            | Aktion                                                             |
|----|--------------------|--------------------------------------------------------------------|
| 6  | $(\exists R.C)(a)$ | Füge $R(a,b)$ und $C(b)$ für ein <i>neues</i> Individuum $b$ hinzu |

# Tableau-Algorithmus mit Blocking



- Terminiert nicht unbedingt
- Man kann beliebig viele neue Axiome erzeugen

| Nr | Aussage            | Aktion                                                             |
|----|--------------------|--------------------------------------------------------------------|
| 6  | $(\exists R.C)(a)$ | Füge $R(a,b)$ und $C(b)$ für ein <i>neues</i> Individuum $b$ hinzu |

- Idee: wenn keine neue Information erzeugt wird, dann verhindere die Anwendung von Regel 6
  - d.h., wenn schon eine gleichartige Instanz  $b_0$  für  $a$  erzeugt wurde, dann blockiere die Auswahl von Regel 6 auf  $a$ .

# Tableau-Algorithmus mit Blocking



- Gegeben: eine Wissensbasis  $W$  in NNF  
Solange nicht alle Teiltableaus abgeschlossen sind  
**und weitere Aussagen erzeugt werden können**
  - \* Wähle ein nicht abgeschlossenes Teiltableau  $T$   
und ein  $A \in W \cup T$ , **das nicht blockiert ist**  
Wenn  $A$  noch nicht in  $T$  enthalten
    - Wenn  $A$  eine A-Box-Aussage ist
      - füge  $A$  zu  $T$  hinzu
      - zurück zu \*
    - Wenn  $A$  eine T-Box-Aussage ist
      - Wähle ein Individuum  $a \in W \cup T$
      - Füge  $A(a)$  zu  $T$  hinzu
      - zurück zu \*
  - sonst
    - Erweitere das Tableau mit Konsequenzen aus  $A$
    - zurück zu \*

# Tableau-Algorithmus: Zusammenfassung

- Ein Algorithmus für Beschreibungslogiken
  - passt auf OWL Lite und DL
- Wir haben Beispiele für einige OWL-Konstrukte gesehen
  - Auch andere OWL-DL-Ausdrücke können in DL "übersetzt" werden
  - und auch hier gibt es Expansionsregeln
  - allerdings machen die das Reasoning nicht immer leichter
    - benötigen teilweise komplizierte Strategien
    - dynamisches Blockieren und Deblockieren

# Tableau-Algorithmus: Optimierungsansätze



- Gegeben: eine Wissensbasis  $W$  in NNF  
Solange nicht alle Teiltableaus abgeschlossen sind  
und weitere Aussagen erzeugt werden können
  - \* Wähle ein nicht abgeschlossenes Teiltableau  $T$   
und ein  $A \in W \cup T$ , das nicht blockiert ist  
Wenn  $A$  noch nicht in  $T$  enthalten
    - Wenn  $A$  eine A-Box-Aussage ist
      - füge  $A$  zu  $T$  hinzu
      - zurück zu \*
    - Wenn  $A$  eine T-Box-Aussage ist
      - Wähle ein Individuum  $a \in W \cup T$
      - Füge  $A(a)$  zu  $T$  hinzu
      - zurück zu \*
  - sonst
    - Erweitere das Tableau mit Konsequenzen aus  $A$
    - zurück zu \*



# Reasoning: aktuelle Forschung



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

- Gute Heuristiken für Auswahl von Axiomen
- Unterstützte Expressivität erhöhen
  - z.B. auch Regeln ermöglichen
  - dabei Skalierbarkeit erlauben
- Paralleles Reasoning

# Tableau-Algorithmus: Implementierungen



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

- Fact
  - University of Manchester, kostenlos
  - SHIQ
- Fact++/JFact
  - Weiterentwicklung von Fact, kostenlos
  - SHOIQ(mit etwas D), OWL-DL + OWL2
- Pellet
  - Clark & Parsia, kostenlos für akademische Nutzung
  - SHOIN(D), OWL-DL + OWL2
- RacerPro
  - Racer Systems, kommerzielles Produkt
  - SHIQ(D)

# Reasoning mit OWL: Alternative Ansätze

- Wiederverwendung von Reasonern aus anderen Bereichen
  - z.B. Datalog-Reasoner (KAON)
  - z.B. First Order Logic Theorem Prover (Hoolet)

# Zur Erholung

- Gönnen wir unserem Reasoner doch mal etwas Zerstreuung...

|   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|
| 5 | 3 |   |   | 7 |   |   |   |   |
| 6 |   |   | 1 | 9 | 5 |   |   |   |
|   | 9 | 8 |   |   |   |   | 6 |   |
| 8 |   |   |   | 6 |   |   |   | 3 |
| 4 |   |   | 8 |   | 3 |   |   | 1 |
| 7 |   |   |   | 2 |   |   |   | 6 |
|   | 6 |   |   |   |   | 2 | 8 |   |
|   |   |   | 4 | 1 | 9 |   |   | 5 |
|   |   |   |   | 8 |   |   | 7 | 9 |

# Sudoku in OWL

- Was haben wir?
- Zunächst mal eine abgeschlossene Klasse von Zahlen:

```
:Zahl a owl:Class ;
```

```
owl:oneOf (:1 :2 :3 :4 :5 :6 :7 :8 :9) .
```

- Und eine ganze Menge von Feldern
  - die wir mit diesen Zahlen füllen wollen
  - machen wir's uns einfach: die Felder sind auch Zahlen
  - wir wollen wissen, welches Feld gleich welcher Zahl ist

# Sudoku als OWL-Problem

## ▪ 81 Felder:

c1\_11 a :Zahl .  
c1\_21 a :Zahl .  
...  
c1\_33 a :Zahl .  
c2\_11 a :Zahl .  
...  
c9\_33 a :Zahl .

|       |       |  |       |       |  |  |  |  |
|-------|-------|--|-------|-------|--|--|--|--|
| c1_11 | c1_12 |  | c2_11 | c2_12 |  |  |  |  |
| c1_21 |       |  |       |       |  |  |  |  |
|       |       |  |       |       |  |  |  |  |
| c4_11 |       |  |       |       |  |  |  |  |
|       |       |  |       |       |  |  |  |  |
|       |       |  |       |       |  |  |  |  |
|       |       |  |       |       |  |  |  |  |
|       |       |  |       |       |  |  |  |  |
|       |       |  |       |       |  |  |  |  |

# Sudoku als OWL-Problem

- Felder in einem Quadranten sind verschieden:

```
c1_11 owl:differentFrom
 c1_12, c1_13, ..., c1_33 .
c1_12 owl:differentFrom
 c1_13, c1_21, ..., c1_33 .
...
c1_32 owl:differentFrom
 c1_33 .
c2_11 owl:differentFrom
 c2_12, c2_13, ..., c1_33 .
...
c9_32 owl:differentFrom
 c9_33 .
```

|       |       |  |       |       |  |  |  |  |
|-------|-------|--|-------|-------|--|--|--|--|
| c1_11 | c1_12 |  | c2_11 | c2_12 |  |  |  |  |
| c1_21 |       |  |       |       |  |  |  |  |
|       |       |  |       |       |  |  |  |  |
| c4_11 |       |  |       |       |  |  |  |  |
|       |       |  |       |       |  |  |  |  |
|       |       |  |       |       |  |  |  |  |
|       |       |  |       |       |  |  |  |  |
|       |       |  |       |       |  |  |  |  |
|       |       |  |       |       |  |  |  |  |

# Sudoku als OWL-Problem

- Felder in einer Zeile sind verschieden:

```
c1_11 owl:differentFrom
 c1_12, c1_13, ..., c3_13 .
...
```

|       |       |  |       |       |  |  |  |  |
|-------|-------|--|-------|-------|--|--|--|--|
| c1_11 | c1_12 |  | c2_11 | c2_12 |  |  |  |  |
| c1_21 |       |  |       |       |  |  |  |  |
|       |       |  |       |       |  |  |  |  |
| c4_11 |       |  |       |       |  |  |  |  |
|       |       |  |       |       |  |  |  |  |
|       |       |  |       |       |  |  |  |  |
|       |       |  |       |       |  |  |  |  |
|       |       |  |       |       |  |  |  |  |
|       |       |  |       |       |  |  |  |  |



# Sudoku als OWL-Problem

- Felder in einer Spalte sind verschieden:

```
c1_11 owl:differentFrom
 c1_21, c1_31, ..., c3_31 .
...
```

|       |       |  |       |       |  |  |  |  |
|-------|-------|--|-------|-------|--|--|--|--|
| c1_11 | c1_12 |  | c2_11 | c2_12 |  |  |  |  |
| c1_21 |       |  |       |       |  |  |  |  |
|       |       |  |       |       |  |  |  |  |
| c4_11 |       |  |       |       |  |  |  |  |
|       |       |  |       |       |  |  |  |  |
|       |       |  |       |       |  |  |  |  |
|       |       |  |       |       |  |  |  |  |
|       |       |  |       |       |  |  |  |  |
|       |       |  |       |       |  |  |  |  |

# Sudoku in OWL

- Letzter Schritt: bekannte Zahlen einsetzen

```
c1_11 owl:sameAs :5 .
c1_12 owl:sameAs :3 .
c1_21 owl:sameAs :6 .
...
```

|   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|
| 5 | 3 |   |   | 7 |   |   |   |   |
| 6 |   |   | 1 | 9 | 5 |   |   |   |
|   | 9 | 8 |   |   |   |   | 6 |   |
| 8 |   |   |   | 6 |   |   |   | 3 |
| 4 |   |   | 8 |   | 3 |   |   | 1 |
| 7 |   |   |   | 2 |   |   |   | 6 |
|   | 6 |   |   |   |   | 2 | 8 |   |
|   |   |   | 4 | 1 | 9 |   |   | 5 |
|   |   |   |   | 8 |   |   | 7 | 9 |

# Komplexität unseres OWL-Problems

- Wir haben
  - Klassen, und zwar eine abgeschlossene
- d.h., eine Ontologie der Komplexität SO.
- Wer kann das?
  - Fact: SHIQ :-)
  - RacerPro: SHIQ(D) :-)
  - Pellet: SHOIN(D) :-)
  - Hermit: SHOIQ :-)

# Komplexität unseres OWL-Problems

- Wir haben
  - Klassen, und zwar eine abgeschlossene
- d.h., eine Ontologie der Komplexität SO.
- Wer kann das?
  - Fact: SHIQ :-)
  - RacerPro: SHIQ(D) :-)
  - Pellet: SHOIN(D) :-)
  - Hermit: SHOIQ :-)

# Sudoku in OWL

- Probe aufs Exempel in Protégé
- Vereinfacht, damit wir nicht so viel tippen müssen

|   |   |   |  |
|---|---|---|--|
|   | 4 |   |  |
|   |   | 3 |  |
|   |   | 2 |  |
| 1 |   |   |  |

# Zusammenfassung

- Reasoning auf Ontologien
  - lässt sich auf Konsistenzprüfung zurückführen
  - in OWL DL: Description Logic Reasoning
- Tableau-Verfahren
  - erzeuge alle möglichen Aussagen
  - verzweige bei "oder"
  - blockiere unproduktive Axiome

# Vorlesung Semantic Web



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

Vorlesung im Wintersemester 2012/2013

Dr. Heiko Paulheim

Fachgebiet Knowledge Engineering