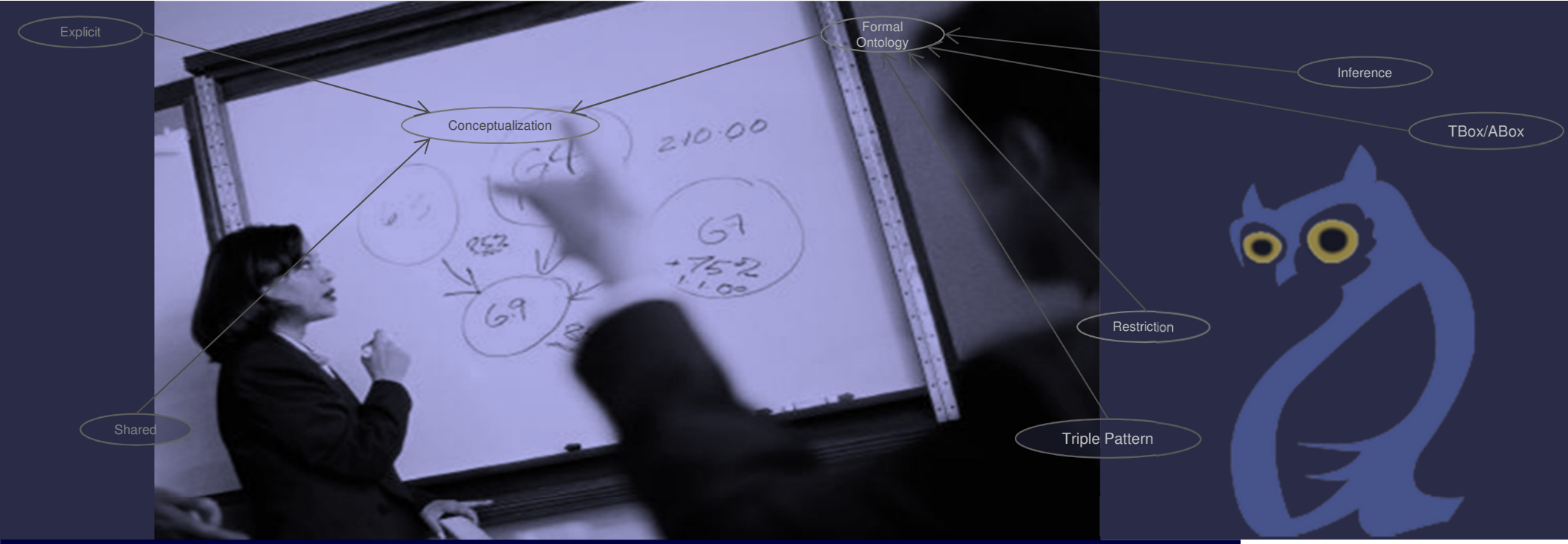


Semantisches Prozessmanagement und E-Business

Michael Fellmann

Lehrveranstaltung im SS 2013

UNIVERSITÄT OSNABRÜCK

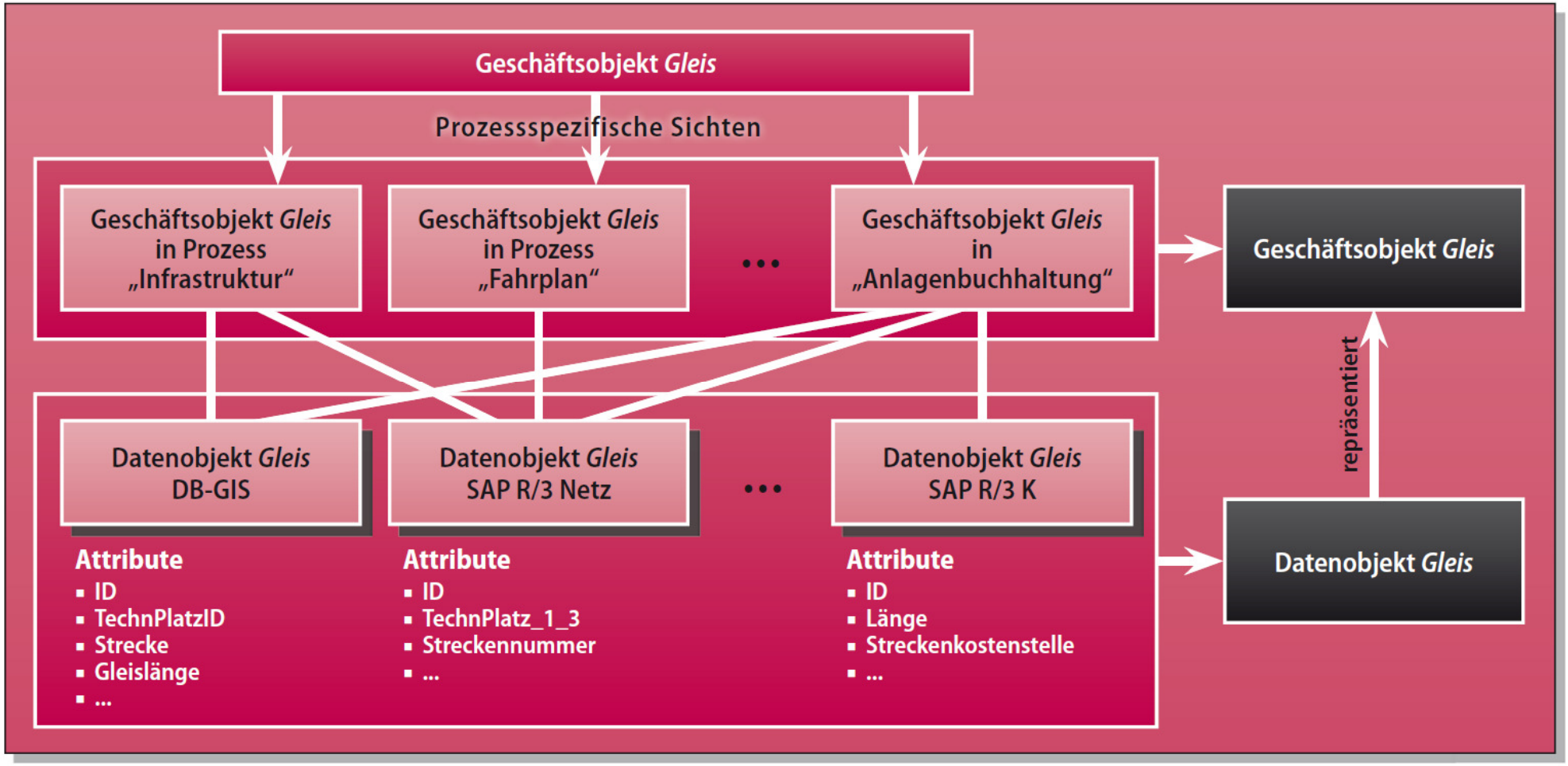
Institut für Informationsmanagement
und Unternehmensführung
michael.fellmann@uos.de

Agenda

- **OWL-Einführung**
  - Grundlagen und Terminologie
  - Klassen
  - Properties
  - Restriktionen auf Klassen
  - Primitive und definierte Klassen
  - Instanzen
- SPARQL zur Anfrage an RDF- und OWL-Ontologien

Motivation zur semantischen Beschreibung

Beispiel DB Netz AG – was ist ein „Gleis“?

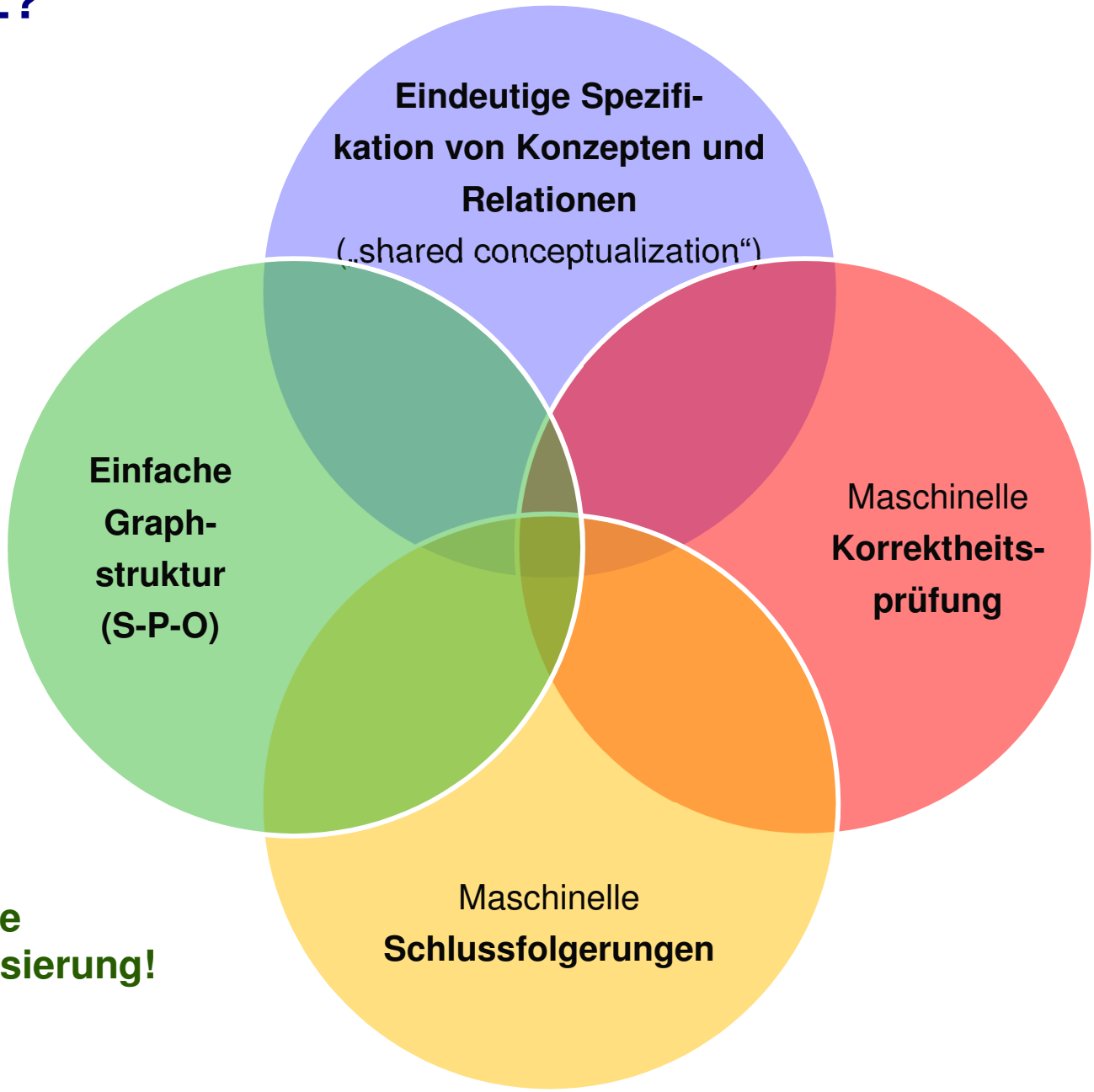


→ **Eine Ontologie erlaubt die Erfassung der verschiedenen Bedeutungsinhalte sowie die Auflösung von Synonymen und Homonymen (und weiteren „Defekten“)**

Quelle: Schmidt, A.; Otto, B.; Österle, H. (2010): Unternehmensweite Stammdatenintegration. In: WuM 5/2010, S. 47

Warum OWL?

Charakteristika



+ Flexible Visualisierung!

Agenda

- OWL-Einführung
  - **Grundlagen und Terminologie**
  - Klassen
  - Properties
  - Restriktionen auf Klassen
  - Primitive und definierte Klassen
  - Instanzen
- SPARQL zur Anfrage an RDF- und OWL-Ontologien

OWL 1 und Beschreibungslogiken

Zusammenhang

- **Beschreibungslogik $\mathcal{ALC}$ als Basis**
  - $\mathcal{ALC}$ (Attributive Language with Complements) ist die Grundlage vieler Beschreibungslogiken und unterstützt:
    - Klassen, Rollen, Individuen und deren Beschreibung mit den Konstrukten:
    - Konjunktion, Disjunktion, Existenz- und Allquantor sowie Negation.
  - Erweiterungen dieser Basissprache werden durch Hinzufügungen gekennzeichnet wie:

    $\mathcal{S}$ transitive Rollen

    $\mathcal{H}$ Rollen-Hierarchien

    $\mathcal{O}$ Klassen basierend auf der Aufzählung ihrer Mitglieder

    $\mathcal{I}$ inverse Rollen

    $\mathcal{N}$ Kardinalität des Auftretens von Rollen

    (D) für Datentypen

→ **OWL-DL entspricht der Beschreibungslogik** $\mathcal{SHOIN}$(D)

OWL 1 und Beschreibungslogiken

Zusammenhang

- **Eigenschaften von** $\mathcal{SHOIN}$(D)
  - Trotz eines relativ hohen Grades an Ausdrucksmächtigkeit vollständig berechenbar und entscheidbar.
    - Die **vollständige Berechenbarkeit** (engl. *computational completeness*) gibt an, dass jede theoretisch herleitbare Schlussfolgerung auch tatsächlich von einer Inferenzmaschine berechenbar ist.
    - Die **Entscheidbarkeit** gibt an, dass alle zu Schlussfolgerungen benötigten Berechnungen in endlicher Zeit möglich sind.

- **Eigenschaften von OWL-DL**
  - im Gegensatz zu OWL Full ist OWL-DL berechen- und entscheidbar.
  - „*OWL-DL supports those users who want the maximum expressiveness while retaining computational completeness (all conclusions are guaranteed to be computable) and decidability (all computations will finish in finite time)*“ (McGuiness, van Harmelen 2004).

OWL 2 im Überblick

Begriff, Sprachprofile

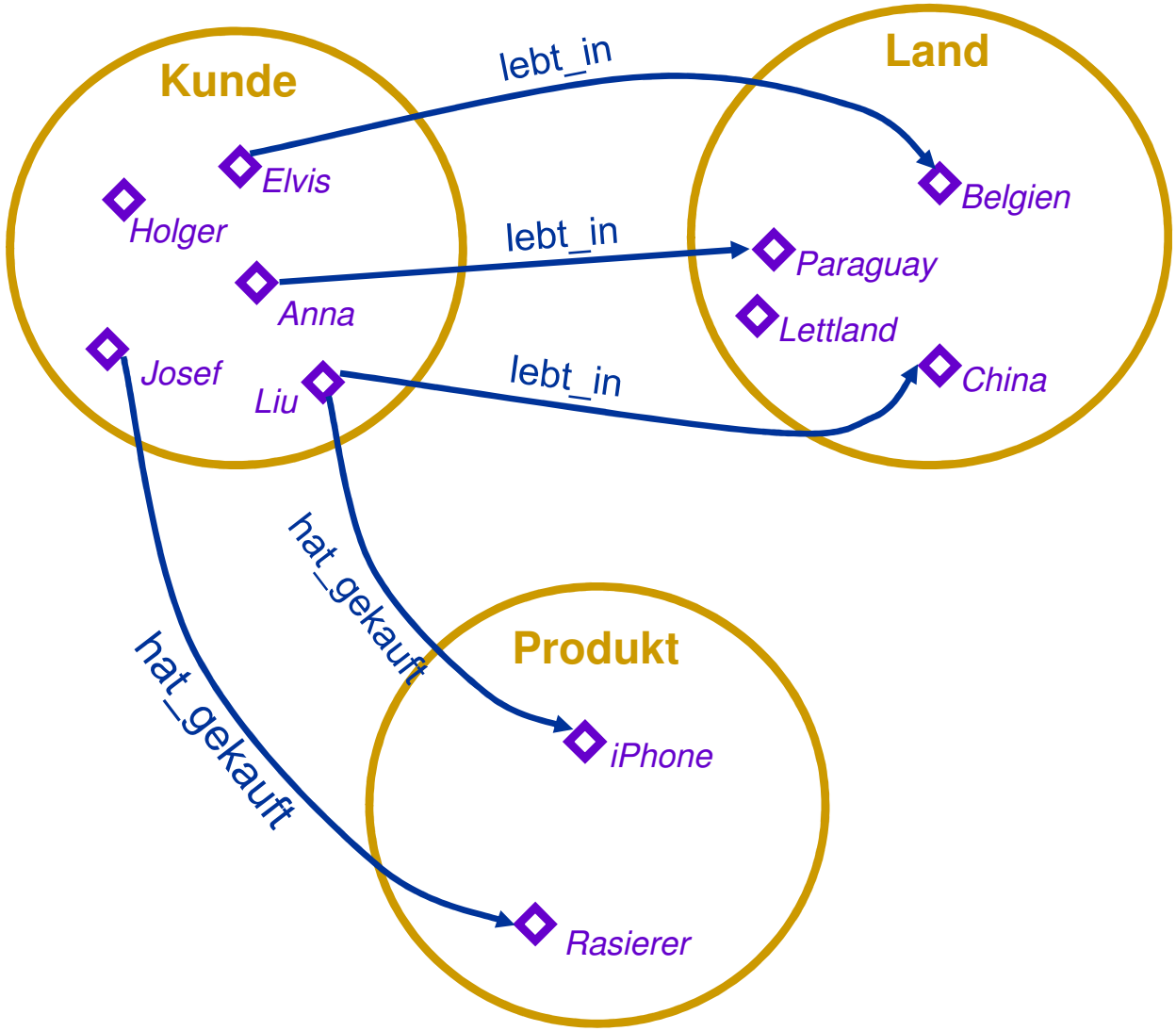
- **Was ist OWL (2.0)?**
  - OWL 2 ist eine Weiterentwicklung von OWL 1 und besitzt eine höhere Ausdrucksstärke.
  - In OWL 2 wurde der DL-Teil von OWL weiter aufgefächert in die Profile:

www.w3.org/2007/OWL/

    - **OWL 2 EL** („Existential Quantification“) – für große Ontologien mit großem „Schema“-Anteil (d.h. einer großen TBox), Schlussfolgerungen sind in polynomialer Zeit in Abhängigkeit der TBox möglich.
    - **OWL 2 QL** („Query Language“) – für Ontologien, die aus großen Mengen von Instanzdaten bestehen (d.h. eine große ABox aufweisen), über die im Rahmen von Anfragen geschlussfolgert werden muss. Dies ist mit einem garantierten Speicherverbrauch in Bezug auf die ABox-Größe möglich (LOGSPACE).
    - **OWL 2 RL** („Rule Language“) – kann mit einer Regelsprache implementiert werden und ist für Anwendungen, in denen Abstriche bei der Performanz zugunsten einer erhöhten Ausdrucksstärke hinnehmbar sind. Wesentliche Schlussfolgerungen sind in polynomialer Zeit in Abhängigkeit von der *gesamten* Ontologie möglich.

Bestandteile einer OWL-Ontologie

Grafische Übersicht über wesentliche Konstrukte



Bezeichnungen

	Klasse	(Konzept)
	Instanz	(Individuum)
	Property	(Rolle, Relation)

Bestandteile einer OWL-Ontologie

ABox und TBox

- **Eine Unterteilung** der in einer Beschreibungslogik ausgedrückten Fakten wird oft in eine „Terminological box“ (TBox) und eine „Assertional Box“ (Abox) vorgenommen.
- **TBox**
  - Auch „Schema“ genannt, enthält Wissen über Konzepte (Klassen) einer Domäne.
  - Beispiele:
    - Alle Professoren sind Personen.
    - Doktoranden haben mindestens einen Professor, der sie betreut.
- **ABox**
  - Auch „Daten“ genannt, enthält Wissen über Instanzen dieser Konzepte und deren Beziehungen untereinander.
  - Beispiele:
    - „Michael“ ist eine Instanz von „Doktorand“.
    - „Walter“ ist eine Instanz von „Person“.

→ TBox und ABox zusammen können als „Wissensbasis“ bezeichnet werden.

Agenda

- OWL-Einführung
  - Grundlagen und Terminologie
  - **Klassen**
  - Properties
  - Restriktionen auf Klassen
  - Primitive und definierte Klassen
  - Instanzen
- SPARQL zur Anfrage an RDF- und OWL-Ontologien

Klassen

Deklaration und Verwendung

- **Bedeutung von OWL-Klassen**
  - `owl:Class` ist eine Unterklasse von `rdfs:Class`.
  - Klassen fassen Individuen zusammen, die gleichartig beschrieben werden können.

- **Deklaration von OWL-Klassen**
  - Langform: Es wird eine RDF-Ressource deklariert, die als OWL-Klasse typisiert wird. Diese Form ist semantisch äquivalent zur Kurzform, wird jedoch im Folgenden aus Gründen der besseren Lesbarkeit nicht mehr weiter verwendet.
    - Beispiel:

```
<rdf:Description rdf:ID="ABC">
   <rdf:type rdf:resource="&owl;class"/>
</rdf:Description>
```

  - Kurzform: Es wird die von RDF bekannte Kurzschreibweise verwendet.
    - Beispiel:

```
<owl:Class rdf:ID="ABC"/>
```

Klassen

Vordefinierte Klassen, Klassenhierarchie

- **Vordefinierte Klassen in OWL**
  - `owl:Thing` ist die Klasse, die alles enthält.
  - `owl:Nothing` ist leer und enthält nichts.

DL-Syntax

$\top$ (Thing) $\bot$ (Nothing)

- **Beziehungen jeder Klasse zu den vordefinierten Klassen**
  - Jede Klasse ist Unterklasse von `owl:Thing`.
  - `owl:Nothing` ist Unterklasse jeder Klasse.

- **Klassenhierarchien**
  - Zur Bildung einer Klassenhierarchie wird `rdfs:subClassOf` verwendet.
    - Beispiel: Klasse „LKW“ ist eine Unterklasse von „Automobil“.

```
<owl:Class rdf:ID="LKW">
  <rdfs:subClassOf rdf:resource="#Automobil"/>
</owl:Class>
```

DL-Syntax

$\text{LKW} \sqsubseteq \text{Automobil}$

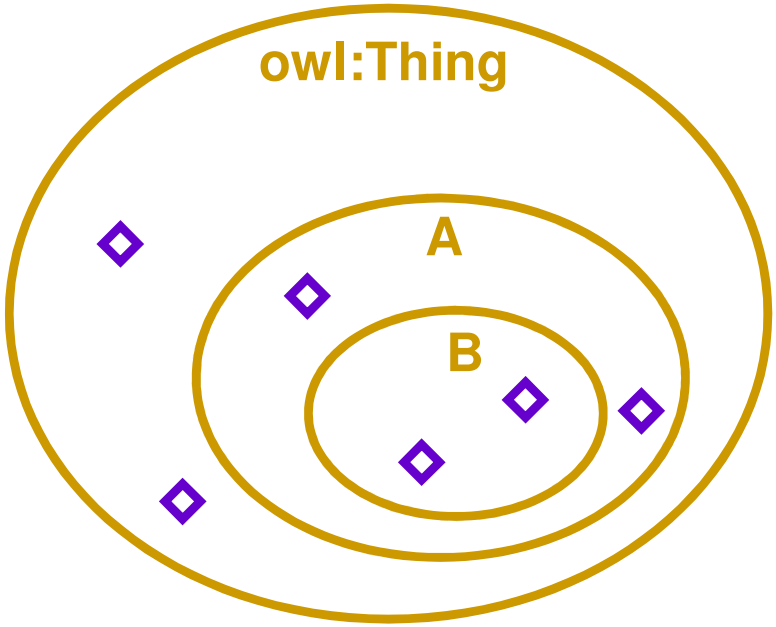
Klassen

Klassenhierarchie

DL-Syntax

$B \sqsubseteq A \sqsubseteq \top$

- **Grafische Veranschaulichung**
  - owl:Thing ist Oberklasse aller OWL-Klassen.
  - Die Subklassenbeziehung kann auch als „is-a“ gelesen werden: Alle Instanzen von B sind auch Instanzen von A sowie Instanzen von „Thing“.



Klassen

Ansätze zur Klassenbildung

- **Top-down-Ansatz**
  - (1) Allgemeinste Klasse einer Domäne formulieren (z.B. „Thing“)
  - (2) Speziellere Unterklassen entwickeln (z.B. „Objekt“, „Prozess“ etc.)
- **Bottom-up-Ansatz**
  - (1) Speziellste Klasse wird definiert (z.B. „Lieferantenadresse“)
  - (2) Definition von allgemeineren Oberklassen (z.B. „Adresse“)
- **Kombinierter Ansatz (Middle-out)**
  - (1) In beide Richtungen
  - (2) Man beginnt mit den hervorstechendsten/ intuitiv wichtigsten Konzepten (vgl. Basisklassen im Kontext der Prototypentheorie)

Klassen

Einfache Klassenbeziehungen

- **Disjunkte Klassen mit `owl:disjointWith`**
  - Zwei Klassen können keine gemeinsamen Instanzen besitzen.
  - Beispiel:

```
<owl:Class rdf:ID="Männlich">
  <owl:disjointWith rdf:resource="#Weiblich"/>
</owl:Class>
```

DL-Syntax

$$\text{Männlich} \sqsubseteq \neg\,\text{Weiblich}$$

- **Äquivalente Klassen mit `owl:equivalentClass`**
  - Zwei Klassen gleichen sich.
  - Beispiel:

```
<owl:Class rdf:ID="Automobil">
  <owl:equivalentClass rdf:resource="#Kraftfahrzeug"/>
</owl:Class>
```

DL-Syntax

$$\text{Automobil} \equiv \text{Kraftfahrzeug}$$

Komplexe Klassen

Konstruktoren

- **Schnittmenge mit `owl:intersectionOf`**
  - Interpretierbar als UND-Operator auf Klassen. Individuen müssen beiden Klassen zugleich angehören.
  - Beispiel:

```
<owl:Class rdf:ID="Amphibienfahrzeug">
  <rdfs:subClassOf>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Landfahrzeug"/>
        <owl:Class rdf:about="#Wasserfahrzeug"/>
      </owl:intersectionOf>
    </owl:Class>
  </rdfs:subClassOf>
</owl:Class>
```

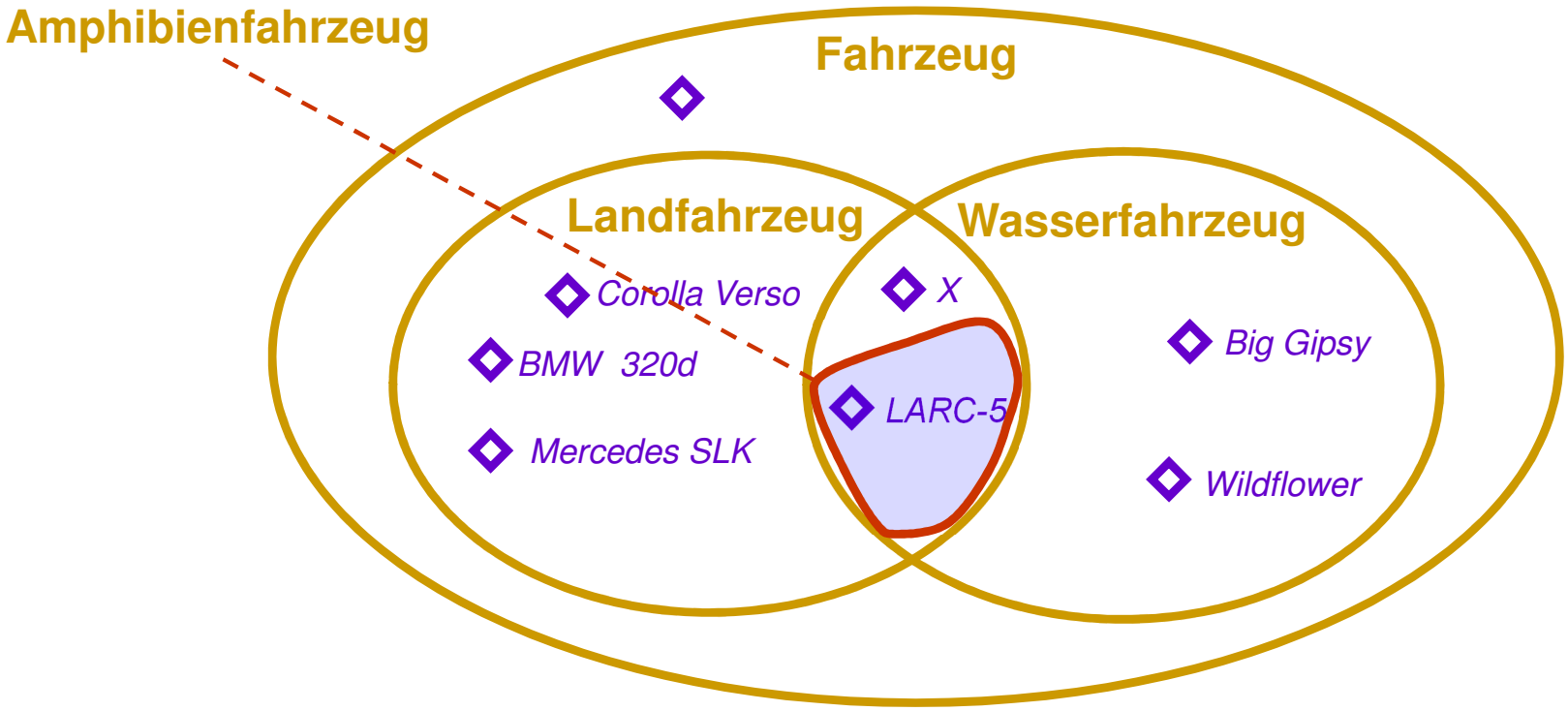
DL-Syntax

$$\text{Amphibienfahrzeug} \sqsubseteq \text{Landfahrzeug} \sqcap \text{Wasserfahrzeug}$$

Komplexe Klassen

Konstruktoren

- **Schnittmenge mit `owl:intersectionOf`**
  - Grafische Veranschaulichung des Beispiels:



DL-Syntax

$$\text{Amphibienfahrzeug} \sqsubseteq \text{Landfahrzeug} \sqcap \text{Wasserfahrzeug}$$

OWL-Übung

- **Aufgabe: Überlegen Sie…**
  - Gegeben sei das folgende Objekt:

Einflügeliges Bodeneffektfahrzeug von Alexander Lippisch (Erstflug 1970)

  - Begründen Sie unter Bezugnahme auf das Objekt, warum es sinnvoll ist, die Klasse „Amphibienfahrzeug“ als *Teilmenge* der Schnittmenge aus Landfahrzeug und Wasserfahrzeug zu definieren (und nicht als deren Äquivalent)!

Komplexe Klassen

Konstruktoren

- **Vereinigungsmenge mit `owl:unionOf`**
  - Interpretierbar als ODER-Operator auf Klassen. Individuen müssen einer oder beiden Klassen angehören.
  - Beispiel:

```
<owl:Class rdf:ID="WasserOderLandfahrzeug">
  <rdfs:subClassOf>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Landfahrzeug"/>
        <owl:Class rdf:about="#Wasserfahrzeug"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:subClassOf>
</owl:Class>
```

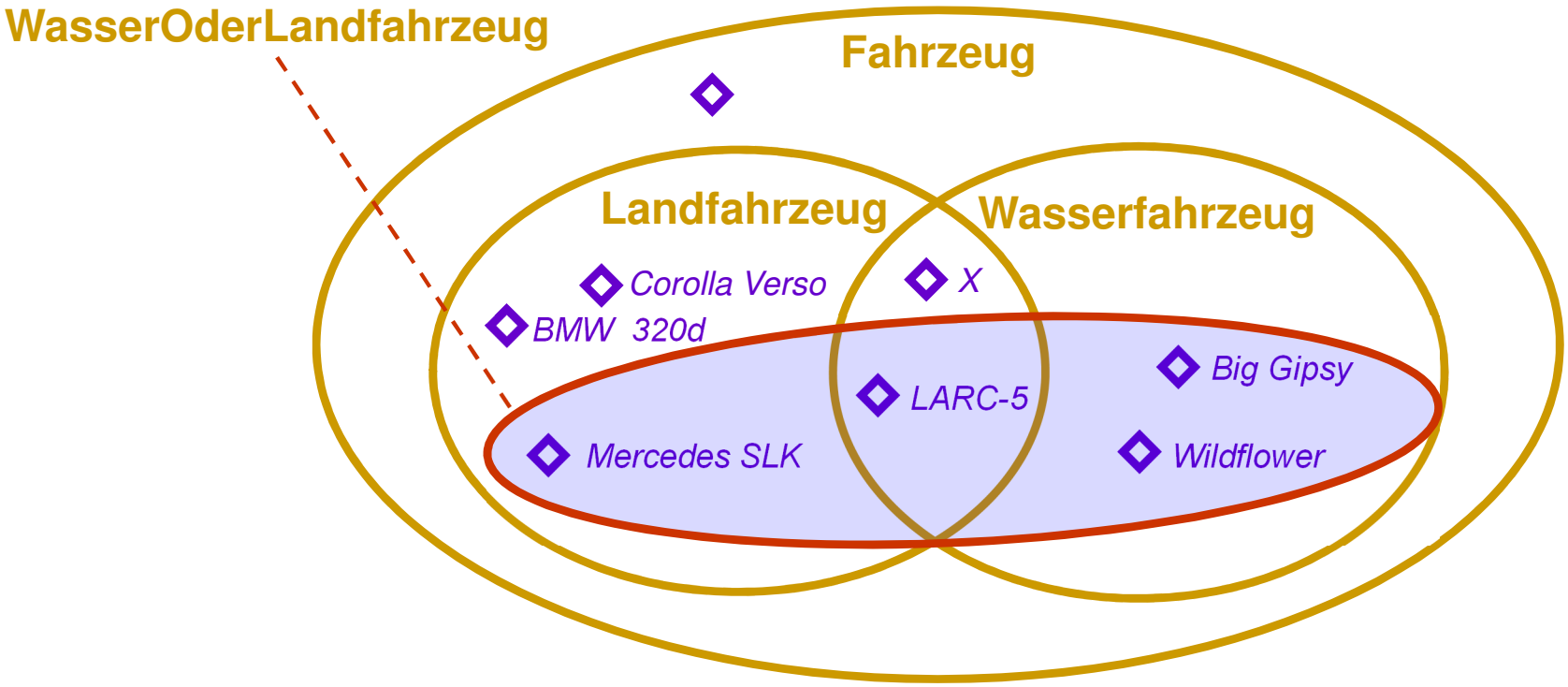
DL-Syntax

$$\text{WasserOderLandfahrzeug} \sqsubseteq \text{Landfahrzeug} \sqcup \text{Wasserfahrzeug}$$

Komplexe Klassen

Konstruktoren

- **Vereinigungsmenge mit `owl:unionOf`**
  - Grafische Veranschaulichung des Beispiels:



DL-Syntax

$$\text{WasserOderLandfahrzeug} \sqsubseteq \text{Landfahrzeug} \sqcup \text{Wasserfahrzeug}$$

Komplexe Klassen

Konstruktoren

- **Komplementmenge mit `owl:complementOf`**
  - Interpretierbar als NICHT-Operator (d.h. Negation). Individuen dürfen nicht einer angegebenen Klasse angehören.
  - Beispiel:

```
<owl:Class rdf:ID="BodenOderLuftOderRaumfahrzeug">
  <rdfs:subClassOf>
    <owl:Class>
      <owl:complementOf rdf:resource="#Wasserfahrzeug" />
    </owl:Class>
  </rdfs:subClassOf>
</owl:Class>
```

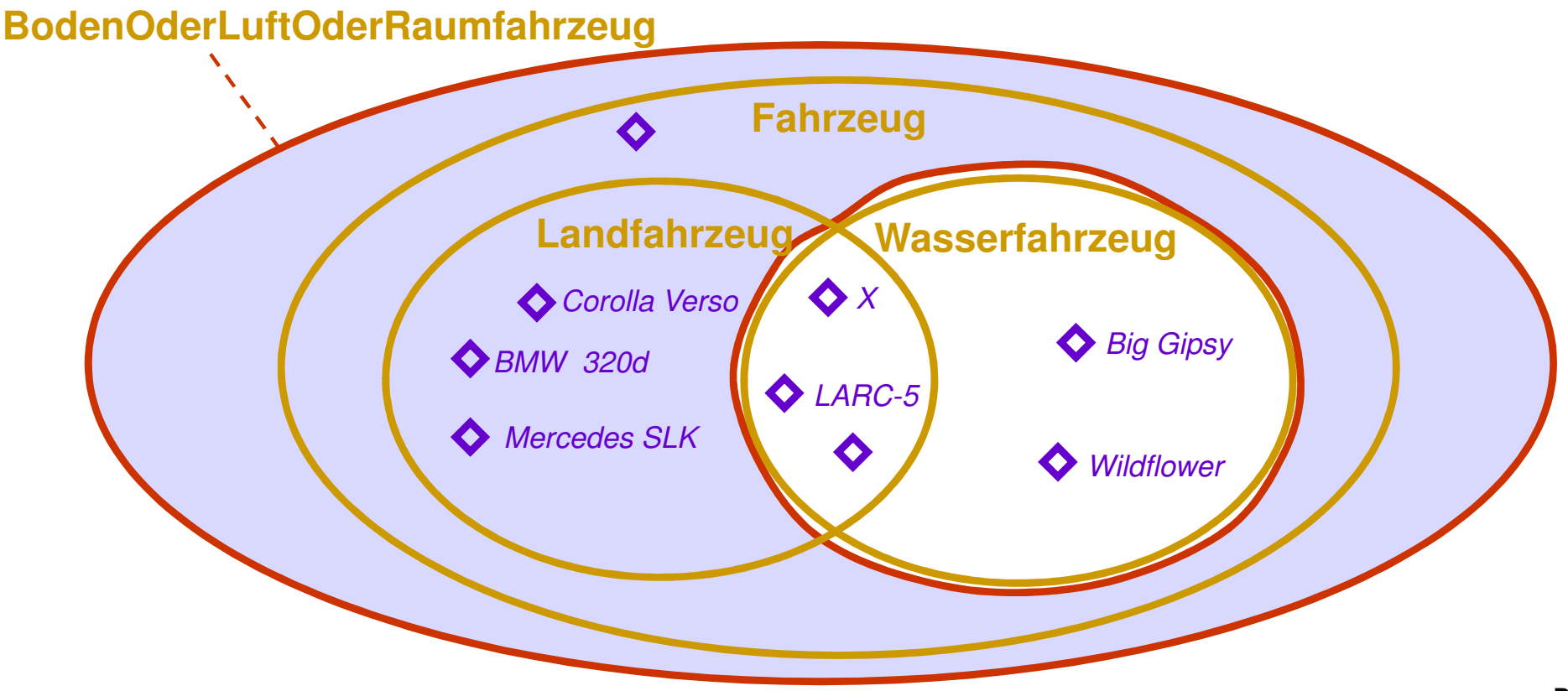
DL-Syntax

$$\text{BodenOderLuftOderRaumfahrzeug} \sqsubseteq \neg\text{Wasserfahrzeug}$$

Komplexe Klassen

Konstruktoren

- **Komplementmenge mit `owl:complementOf`**
  - Grafische Veranschaulichung des Beispiels:



Problem: Klasse ist zu groß,
besser wäre:
BLR $\sqsubseteq$ Fahrzeug $\sqcap \neg$Wasserfahrzeug

DL-Syntax

BodenOderLuftOderRaumfahrzeug $\sqsubseteq \neg$Wasserfahrzeug

Klassen und Inferenz – Vorbemerkungen

Inferenz und Inferenzmaschinen

- **Der Begriff „Inferenz“**
  - Auch als „Folgerung“, „Schlussfolgerung“ oder „Schließen“ bezeichnet.
  - Es geht grundlegend um die logisch korrekte Ableitung von neuen Aussagen (Fakten) aus bestehenden Aussagen.
- **Inferenzmaschine**
  - Eine Inferenzmaschine (engl. auch „Inference engine“, „Reasoner“ oder „Classifier“ genannt) leitet durch Schlussfolgerung neue Aussagen aus einer bestehenden Wissensbasis ab.
  - Teilweise wird auch davon gesprochen, dass durch maschinelles Schlussfolgern implizit (in der Wissensbasis) vorhandenes Wissen expliziert wird.
  - Aufgaben von Inferenzmaschinen sind u.a.:
    - Konsistenzprüfung (Consistency Checking)
    - Explizite Ober-/Unterbeziehungen prüfen oder entdecken (Subsumption Checking)
    - Gleichheitsprüfung (Equivalence Checking)
    - Instanziierungsprüfung (Instantiation Checking)

Klassen und Inferenz – Vorbemerkungen

OWA und GWA, Implementierungen von Inferenzmaschinen

- **Annahme einer offenen oder geschlossenen Welt**
  - **Die Geschlossene-Welt-Annahme** beim Schlussfolgern (Closed World Reasoning)
    - Annahme, dass alle relevanten Fakten in der Wissensbasis vorhanden sind.
    - Fehlende bzw. unauffindbare Informationen können als Fehler interpretiert werden.
  - **Die Offene-Welt-Annahme** beim Schlussfolgern (Open World Reasoning)
    - Annahme, dass die Wissensbasis unvollständig ist.
    - Fehlende Informationen werden als unbekannt (nicht: falsch) eingestuft.

- **Inferenzmaschinen für OWL-DL**
  - Es existieren leistungsfähige Inferenzmaschinen wie
    - Pellet
    - HermiT
    - FACT++
    - Racer

Klassen und Inferenz

Beispiele für Schlussfolgerungen

- **Durch logisches Schließen (Inferenz)** können einer Ontologie auf der Basis der verwendeten Sprachkonstrukte automatisch (mittels einer Inferenzmaschine) neue Fakten hinzugefügt werden. Im Folgenden werden hierfür Beispiele angegeben.

- **Nutzung der transitiven Subklassenbeziehung**
  - Gegeben: TBox: Fakultätsmitglied $\sqsubseteq$ Person
    Professor $\sqsubseteq$ Fakultätsmitglied

Schlussfolgerung: Professor $\sqsubseteq$ Person

- **Nutzung der Subklassen- und Äquivalenzbeziehung**
  - Gegeben: TBox: Professor $\sqsubseteq$ Fakultätsmitglied
    Professor $\equiv$ Hochschullehrer

Schlussfolgerung: Hochschullehrer $\sqsubseteq$ Fakultätsmitglied

Agenda

- OWL-Einführung
  - Grundlagen und Terminologie
  - Klassen
  - **Properties**
  - Restriktionen auf Klassen
  - Primitive und definierte Klassen
  - Instanzen
- SPARQL zur Anfrage an RDF- und OWL-Ontologien

Abgrenzung: Properties und Eigenschaften

Grundlegende Bemerkungen

- **Properties** sind Beziehungen zwischen Instanzen von Klassen, diese werden in der Literatur auch als *Rollen* oder (gelegentlich) *Relationen* bezeichnet.
  - In RDF wird auch davon gesprochen, dass Ressourcen (Subjekte) Properties (Prädikate) besitzen, deren Wert (Objekt) andere Ressourcen sind.
  - In OWL werden Properties, die Instanzen verbinden, als **Objekt-Properties** bezeichnet.
- **Eigenschaften** sind Beziehungen zwischen Instanzen von Klassen und Datenwerten.
  - In RDF wird auch davon gesprochen, dass Ressourcen (Subjekte) Properties (Prädikate) besitzen, deren Werte (Objekte) Literale sind.
  - In OWL werden Eigenschaften als spezielle Properties aufgefasst: Sie werden als **Daten-Properties** bezeichnet.

→ **Im Rahmen der Vorlesung werden zur Reduzierung der Bezeichnungsvielfalt die OWL-nahen Bezeichnungen „Objekt-Property“ und „Daten-Property“ gewählt.**

Properties

Charakterisierung, Deklaration

- **Charakterisierung von Properties**
  - Properties in OWL sind Spezialisierungen von `rdf:Property` und besitzen daher
    - eine **Domäne**, die über `rdfs:domain` angegeben wird
    - und einen **Wertebereich**, der über `rdfs:range` angegeben wird.
- **Beispiele zur Deklaration**
  - Objekt-Property:

```
<owl:ObjectProperty rdf:ID="besitzt">
  <rdfs:domain rdf:resource="#Person"/>
  <rdfs:range rdf:resource="#Objekt"/>
</owl:ObjectProperty>
```

DL-Syntax

$\exists$besitzt. $\top \sqsubseteq$ Person

$\top \sqsubseteq \forall$besitzt.Objekt

  - Daten-Property:

```
<owl:DatatypeProperty rdf:ID="hatAlter">
  <rdfs:domain rdf:resource="#Person"/>
  <rdfs:range rdf:resource="&xsd;int"/>
</owl:DatatypeProperty>
```

DL-Syntax

$\exists$hatAlter. $\top \sqsubseteq$ Person

$\top \sqsubseteq \forall$hatAlter.(xs:int)

Daten-Properties

Verwendbare Datentypen

- **Grundsätzlich** können die von der XML-Schema-Spezifikation bekannten Datentypen verwendet werden.

- **Ausgewählte Datentypen**

Name	Bedeutung	Beispiel
`xsd:float`	Gleitkommazahlen	2.25
`xsd:int`	Ganzzahlen	4711
`xsd:boolean`	Wahrheitswerte	true
`xsd:string`	Zeichenkette	abcdef…
`xsd:double`	Gleitkommazahlen mit hoher Präzision	3.14159…
`xsd:date`	Datum	2002-09-24
`xsd:dateTime`	Datum mit Uhrzeit	2002-05-30T09:00:00

→ **In OWL 2 können darüber hinaus durch Mechanismen von XML-Schema eigene Typen definiert werden.**

Beziehungen zwischen Properties

Gleichartigkeit und Umgekehrtheit

- **Äquivalente Properties**
  - `owl:equivalentProperty` zur Deklaration von Properties, deren Extension gleich ist, die sich aber hinsichtlich ihrer Intension unterscheiden können
  - `owl:sameAs` zur Definition synonymer Properties.
  - Beispiel:

```
<owl:ObjectProperty rdf:ID="hatEhefrau">
  <owl:equivalentProperty rdf:resource="#hatGattin"/>
</owl:ObjectProperty>
```

DL-Syntax

$$\text{hatEhefrau} \equiv \text{hatGattin}$$

- **Inverse Properties** (nur bei Objektproperties möglich)
  - `owl:inverseProperty` zur Deklaration inverser („umgekehrter“) Properties
  - Beispiel:

```
<owl:ObjectProperty rdf:ID="hatEhefrau">
  <owl:inverseOf rdf:resource="#hatEhemann"/>
</owl:ObjectProperty>
```

DL-Syntax

$$\text{hatEhefrau} \equiv \text{hatEhemann}^{-}$$

Zusätzliche Beziehungen zwischen Properties in OWL 2

Disjunktivität und Propertyketten

- **Disjunkte Properties**
  - Erlauben eine zusätzliche Einschränkung gültiger Ontologien.
    Es gilt stets: $R_1(x, y) \rightarrow \neg R_2(y, x)$.
  - Somit können einfache Wenn-Dann-Beziehungen spezifiziert werden, die als Restriktion genutzt werden können.
    - Beispiel: Wenn ein Kunde das Property „hatNeukundenrabatt“ besitzt, dann darf er nicht das Property „hatStammkundenRabattStufe“ besitzen.

- **Property-Ketten** (nur bei Objektproperties möglich)
  - Das Auftreten eines Properties zwischen zwei Instanzen in der Wissensbasis wird an das Auftreten einer Kette von Properties zwischen den beteiligten Instanzen gebunden.
    - Beispiel: Der Bruder des Vaters ist der Onkel.
      Schematisch: $R_{Vater}(x, y), R_{Bruder}(y, z) \rightarrow R_{Onkel}(x, z)$.

Properties mit besonderen Eigenschaften

Funktionalität

- **Funktionale Properties**
  - **`owl:FunctionalProperty`** zur Deklaration von Properties, die maximal einen Wert für jede Instanz besitzen dürfen (äquivalent zur Kardinalitätsrestriktion von 1).
  - Lässt Schlüsse der Form zu: R(x, y) und R(x, z) → y = z.
    - Beispiele: hatVorgesetzen, hatMutter.

DL-Syntax

$\top \sqsubseteq \leq 1\ \text{hatVorgesetzten}$

- **Inversfunktionale Properties** (nur bei Objekt-Properties möglich)
  - **`owl:InverseFunctionalProperty`** zur Deklaration von Properties, für die verschiedene Instanzen nicht den gleichen Wert besitzen können.
  - Lässt Schlüsse der Form zu: R(x, y) und R(z, y) → x = z.
    - Beispiele: hatSSN, hatPersonalausweisnummer.

DL-Syntax

$\top \sqsubseteq \leq 1\ \text{hatSSN}^{-}$

Properties mit besonderen Eigenschaften

Transitivität, Symmetrie

- **Transitive Properties** (nur bei Objekt-Properties möglich)
  - `owl:TransitiveProperty` zur Deklaration
  - Lässt Schlüsse der Form zu: R(x, y) und R(y, z) → R(x, z).
    - Beispiele: subRegionVon, hatNachfahre.

DL-Syntax

$subRegion^{+} \equiv subRegion$

- **Symmetrische Properties** (nur bei Objekt-Properties möglich)
  - `owl:SymmetricProperty` zur Deklaration
  - Lässt Schlüsse der Form zu: R(x, y) → R(y, x).
    - Beispiele: hatPartner, istGeschwisterVon.

DL-Syntax

$hatPartner \equiv hatPartner^{-}$

- **In OWL 2 sind weitere Eigenschaften möglich**
  - Reflexive und irreflexive Properties. Irreflexivität erlaubt den Schluss: R(x, y) → y ≠ x.
  - Asymmetrische Properties. Es gilt stets: R(x, y) → ¬R(y, x).

Properties und Inferenz

Beispiele für Schlussfolgerungen

- **Nutzung der Property-Hierarchie und -Transitivität**
  - Gegeben:

    TBox: Fluss

    $\text{mündetIn} \sqsubseteq \text{fließtIn}$

    $\text{fließtIn}^{+} \equiv \text{fließtIn}$

    ABox: Fluss(pegnitz), Fluss(main), Fluss(rhein)

    mündetIn(pegnitz, main), mündetIn(main, rhein)

  Schlussfolgerung (u. a.): fließtIn(pegnitz, rhein)

- **Nutzung der Property-Hierarchie und inverser Properties**
  - Gegeben:

    TBox: Firma

    $\text{istGeschäftspartner} \equiv \text{istGeschäftspartner}^{-}$

    $\text{istGeschäftspartner} \sqsubseteq \text{kennt}$

    ABox: Firma(a), Firma(b), istGeschäftspartner(a, b)

  Schlussfolgerung (u. a.): kennt(b, a)

Agenda

- OWL-Einführung
  - Grundlagen und Terminologie
  - Klassen
  - Properties
  - **Restriktionen auf Klassen**
  - Primitive und definierte Klassen
  - Instanzen
- SPARQL zur Anfrage an RDF- und OWL-Ontologien

Restriktionen auf Klassen

Grundlegendes

- **Property-Restriktionen sind Bedingungen**, welche die zulässigen Property-Werte einschränken, die Instanzen einer Klasse aufweisen dürfen.
  - Festlegung durch `<owl:Restriction>...</owl:Restriction>`
  - Die Restriktion muss ein `owl:onProperty`-Element enthalten und ein oder mehrere der folgenden Deklarationen:
    - `owl:cardinality, owl:minCardinality, owl:maxCardinality`
    - `owl:allValuesFrom`
    - `owl:someValuesFrom`
    - `owl:hasValue`
- **Weitere Restriktionen in OWL 2**
  - Qualifizierte Kardinalitätsrestriktionen
  - Selbstbezüge (owl:hasSelf)
  - Restriktionen über Datentypdefinitionen (vgl. Abschn. „Benutzerdefinierte Datentypen“)

Restriktionen auf Klassen

Kardinalitäts-Restriktion

- **Unter- und Obergrenze der Kardinalität** (Auftretenshäufigkeit)
  - Für das Auftreten eines Properties bei einer Instanz kann
    - eine genaue Anzahl mit **`owl:cardinality`**, oder
    - eine Untergrenze mit **`owl:minCardinality`** und/oder eine Obergrenze mit **`owl:maxCardinality`** festgelegt werden.
  - Beispiel:

```
<owl:Class rdf:ID="Vorlesung">
   <rdfs:subClassOf>
     <owl:Restriction>
       <owl:onProperty rdf:resource="#hatDozent"/>
       <owl:minCardinality
          rdf:datatype="&xsd;int">1</owl:minCardinality>
     </owl:Restriction>
   </rdfs:subClassOf>
 </owl:Class>
```

DL-Syntax

$$\text{Vorlesung} \sqsubseteq \geq 1\ \text{hatDozent}$$

Restriktionen auf Klassen

Qualifizierte Kardinalitäts-Restriktion in OWL 2

- **Qualifizierte Kardinalität** (Auftretenshäufigkeit mit Typangabe)
  - Für das Auftreten eines Properties bei einer Instanz kann die Anzahl (min, max, exakt) in Abhängigkeit des Wertebereichs festgelegt werden.
  - Beispiel:

```
<owl:Class rdf:ID="Pruefung">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hatPruefer"/>
      <owl:onClass rdf:resource="#Professor"/>
      <owl:minQualifiedCardinality
         rdf:datatype="&xsd;nonNegativeInteger"
         >1</owl:minQualifiedCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

DL-Syntax (Erweiterung)

$$\text{Pruefung} \sqsubseteq \geq 1\ \text{hatPruefer.Professor}$$

→ **Kombinierbar mit unqualifizierter Kardinalitätsangabe (z.B. max. 3 Prüfer)**

Restriktionen auf Klassen

Wertebereichs-Restriktion

- **Einschränkungen des Wertebereiches**
  - Für das Auftreten eines Properties bei einer Instanz kann vorgeschrieben werden, dass dieses Property **nur bestimmte Werte** annehmen darf.
  - Beispiel:

```
<owl:Class rdf:ID="Vorlesung">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hatDozent"/>
      <owl:allValuesFrom rdf:resource="#Lehrpersonal"/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

DL-Syntax

$$\text{Vorlesung} \sqsubseteq \forall \text{hatDozent.Lehrpersonal}$$

Restriktionen auf Klassen

Existenz-Restriktion

- **Forderung nach mindestens einem Wert**
  - Für das Auftreten eines Properties bei einer Instanz kann vorgeschrieben werden, dass dieses **mindestens einen Wert** eines bestimmten Objekttyps aufweist.
  - Beispiel:

```
<owl:Class rdf:ID="Pruefung">
   <rdfs:subClassOf>
     <owl:Restriction>
       <owl:onProperty rdf:resource="#hatPruefer"/>
       <owl:someValuesFrom rdf:resource="#Professor"/>
     </owl:Restriction>
   </rdfs:subClassOf>
 </owl:Class>
```

DL-Syntax

$$\text{Prüfung} \sqsubseteq \exists \text{hatPruefer.Professor}$$

Restriktionen auf Klassen

Vorgabewert-Restriktion

- **Festlegung eines Vorgabewertes**
  - Für das Auftreten eines Properties bei einer Instanz kann **ein fester Wert vorgegeben** werden.
  - Beispiel:

```
<owl:Class rdf:ID="NiederlassungUSA">
   <rdfs:subClassOf>
     <owl:Restriction>
       <owl:onProperty rdf:resource="#hatLand"/>
       <owl:hasValue rdf:resource="#USA"/>
     </owl:Restriction>
   </rdfs:subClassOf>
 </owl:Class>
```

DL-Syntax

$$\text{NiederlassungUSA} \sqsubseteq \exists \text{hatLand}.\{\text{USA}\}$$

→ **`owl:hasValue` kann zur Spezifikation von Relationen oder Eigenschaften auf Klassenebene verwendet werden.**

Restriktionen auf Klassen und Inferenz

Beispiele für Schlussfolgerungen

- **Nutzung der Kardinalitäts-Restriktion**
  - Gegeben:

    TBox: Fahrzeug $\sqsubseteq$ =1 hatHalter

    $\exists$hatHalter.$\top$ $\sqsubseteq$ Fahrzeug

    ABox: hatHalter(a4, sabine), hatHalter(a4, bine)

  Schlussfolgerung: sabine $\equiv$ bine

- **Nutzung eines Vorgabewertes**
  - Gegeben:

    TBox: NiederlassungUSA $\sqsubseteq$ $\exists$hatLand.{USA}

    ABox: NiederlassungUSA(san_francisco)

  Schlussfolgerung: hatLand(san_francisco, USA)

- Viele Inferenz-Beispiele, die auf Restriktionen basieren, benötigen definierte Klassen.

  → siehe daher auch die Inferenzbeispiele im Bereich „definierte Klassen“.

Agenda

- OWL-Einführung
  - Grundlagen und Terminologie
  - Klassen
  - Properties
  - Restriktionen auf Klassen
  - **Primitive und definierte Klassen**
  - Instanzen
- SPARQL zur Anfrage an RDF- und OWL-Ontologien

Primitive und definierte Klassen

Vorbemerkungen

- **Primitive Klassen**
  - Auch „partielle“ Klassen genannt, da sie nur **notwendige Bedingungen**, jedoch keine hinreichenden angeben.
  - Die Klassenmitgliedschaft von Individuen wird i.d.R. manuell bspw. durch einen Ontologiekonstrukteur oder Ontologienutzer deklariert.
- **Definierte Klassen**
  - Auch „vollständige“ Klassen genannt, da sie **notwendige und hinreichende Bedingungen** angeben, um die Mitgliedschaft einer Instanz zu entscheiden.
  - Die Klassenmitgliedschaft kann automatisch durch eine Inferenzmaschine geschlossen werden.

→ **Definierte Klassen nutzen das Potenzial maschineller „Intelligenz“.**

Primitive Klassen

Beispiel für eine Klasse mit ausschließlich notwendigen Bedingungen

- **Klasse „Unternehmensprozess“**
  - Notwendige Bedingung: Unternehmensprozesse werden von mindestens einer Person oder einer Maschine ausgeführt.
  - RDF/XML:

```
<owl:Class rdf:ID="Unternehmensprozess">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hatAkteur"/>
      <owl:someValuesFrom>
        <owl:Class>
          <owl:unionOf rdf:parseType="Collection">
            <owl:Class rdf:about="#Mitarbeiter"/>
            <owl:Class rdf:about="#Maschine"/>
          </owl:unionOf>
        </owl:Class>
      </owl:someValuesFrom>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

DL-Syntax

$$\text{Unternehmensprozess} \sqsubseteq \exists \text{hatAkteur}.(\text{Mitarbeiter} \sqcup \text{Maschine})$$

Definierte Klassen

Beispiel für eine Klasse mit notwendigen und hinreichenden Bedingungen

- **Klasse „KritischerProzess“**
  - Notwendige Bedingung: Der Prozess wird von einem Mitarbeiter ausgeführt.
  - Hinreichende Bedingung: Der Prozess unterstützt ein strategisches Unternehmensziel.
  - RDF/XML:

```
<owl:Class rdf:ID="KritischerProzess">
  <owl:equivalentClass>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hatZiel"/>
      <owl:someValuesFrom rdf:resource="#StrategischesZiel"/>
    </owl:Restriction>
  </owl:equivalentClass>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hatAkteur"/>
      <owl:someValuesFrom rdf:resource="#Mitarbeiter"/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

DL-Syntax

$$\text{KritischerProzess} \equiv \exists \text{hatZiel.StrategischesZiel}$$

$$\text{KritischerProzess} \sqsubseteq \exists \text{hatAkteur.Mitarbeiter}$$

Definierte Klassen und Inferenz

Beispiele für Schlussfolgerungen

- **Nutzung definierter Klassen und der Vereinigungsmenge**
  - Gegeben: TBox: $\text{Kunde} \equiv \text{Käufer} \sqcup \text{Interessent}$

Schlussfolgerungen:

$\text{Käufer} \sqsubseteq \text{Kunde}$

$\text{Interessent} \sqsubseteq \text{Kunde}$

- **Nutzung definierter Klassen, der Subklassenbeziehung und Existenz-Restriktion**
  - Gegeben:

TBox: $\text{Prozess} \equiv \exists \text{hatErgebnis}.\text{Produkt}$

$\text{Kreditvergabe} \equiv \exists \text{hatErgebnis}.\text{Kredit}$

$\text{Kredit} \sqsubseteq \text{Produkt}$

Schlussfolgerung:

$\text{Kreditvergabe} \sqsubseteq \text{Prozess}$

Agenda

- OWL-Einführung
  - Grundlagen und Terminologie
  - Klassen
  - Properties
  - Restriktionen auf Klassen
  - Primitive und definierte Klassen
  - **Instanzen**
- SPARQL zur Anfrage an RDF- und OWL-Ontologien

Instanzen

Grundlegendes

- **Es existieren zwei Arten von Instanzen**
  - **Klassen-Instanzen** (Individuen)
    - Beispiel:

```
<Prozess rdf:ID="Auftragsbearbeitung"/>
```

→ **Entspricht dem Erzeugen von Instanzen in RDF.**

  - **Property-Instanzen**
    - Beispiel:

```
<Prozess rdf:ID="Auftragsbearbeitung">
  <hatMitarbeiter>5</hatMitarbeiter>
</Prozess>
```

Property

→ **Properties in OWL entsprechen Prädikaten in Subjekt-Prädikat-Objekt-Sätzen.**

Instanzen

Beziehungen zwischen Instanzen

- **Gleichheit mit `owl:sameAs`**
  - Deklariert, dass verschiedene Bezeichner dasselbe Individuum bezeichnen.
  - Wichtig für Inferenz, da in OWL die *Non Unique Name Assumption* gilt!
    - Beispiel:

```
<rdf:Description rdf:ID="rechnung">
  <owl:sameAS rdf:resource="#faktura"/>
</rdf:Description>
```

- **Unterschiedenheit mit `owl:differentFrom`**
  - Gegenteil von sameAs.
  - Deklariert, dass zwei Individuen verschieden voneinander sind.
    - Beispiel: Verwendung ähnlich wie **`owl:sameAs`**.

Instanzen und Inferenz

Beispiele für Schlussfolgerungen

- **Nutzung definierter Klassen, Gleichheit und der Vereinigungsmenge**
  - Gegeben:

    TBox: FlussUndStrassenfahrzeug $\equiv$ Landfahrzeug $\sqcap$ Wasserfahrzeug

    ABox: Landfahrzeug(hans_trippels_car),
    Wasserfahrzeug(amphicar)
    sameAs(amphicar, hans_trippels_car)

  Schlussfolgerungen:

  FlussUndStrassenfahrzeug(amphicar),
  FlussUndStrassenfahrzeug(hans_trippels_car)

- **Nutzung def. Klassen, Unterschiedenheit u. der qualifizierten Kardinalitäts-Restriktion**
  - Gegeben:

    TBox: AntragMitDoppelterPrüfung $\equiv$ $=2$hatPrüfer.Person

    ABox: hatPrüfer(urlaubsantrag, chef),
    hatPrüfer(urlaubsantrag, sachbearbeiter),
    differentFrom(chef, sachbearbeiter)

  Schlussfolgerung:

  AntragMitDoppelterPrüfung(urlaubsantrag)

Namenskonventionen

Empfehlungen (nicht nur für Instanzen)

- **Namenskonventionen für Ontologie definieren und konsequent anwenden**
  - Ontologie leichter verständlich
  - Beugt einigen Modellierungsfehlern vor
- **Verschiedene zu betrachtende Konventionen**
  - Großschreibung und Trennzeichen
    - Klassen meist groß, Properties meist klein.
    - Bei Bezeichnern aus zwei Wörtern Trennzeichen verwenden wie „_“, „-“ oder zusammenschreiben (sog. „CamelCase“-Schreibweise).
  - Bei Klassen sollte möglichst konsequent der Singular verwendet werden.
  - Präfix- und Suffix-Konventionen:
    - Properties von Klassen abgrenzen (z.B. „hatGeburtstag“ vs. „Geburtstag“).
    - Keine unnötigen Suffixe (z.B. bei direkten Unterklassen den Namen der Oberklasse nicht wiederholen (Ausnahmen sind zur Verbesserung der Verständlichkeit teilweise aber möglich).

Schlussbetrachtung

zur OWL-Einführung

- **Was ist OWL?**
  - OWL ist eine Sprache zur Formulierung von Ontologien.
  - Mit OWL kann Wissen formal repräsentiert werden und es existieren effiziente Algorithmen für Schlussfolgerungen.

- **…und was nicht?**
  - OWL ist keine Programmiersprache (es ist eine deklarative Sprache, um den „Zustand der Welt“ oder „wahres Wissen“ zu beschreiben).
  - OWL ist keine Schemasprache wie XML-Schema (syntaktisch korrekte Dokumente zu beschreiben, ist nicht das Ziel von OWL).
  - Ontologien sind keine Datenbanken (können aber von DB-Technologien profitieren).

→ **OWL ist eine Wissensrepräsentationssprache, die maschinelle Schlussfolgerungen erlaubt!**

Agenda

- OWL-Einführung
  - Grundlagen und Terminologie
  - Klassen
  - Properties
  - Restriktionen auf Klassen
  - Primitive und definierte Klassen
  - Benutzerdefinierte Datentypen
  - Instanzen
- **SPARQL zur Anfrage an RDF- und OWL-Ontologien**

SPARQL

Grundlegendes

- **Begriff, Standardisierung**
  - SPARQL steht für „SPARQL Query Language for RDF“
  - Es handelt sich um einen W3C-Standard (Prud'hommeaux, Seaborne 2006), der gegenwärtig (Sommer 2010) überarbeitet wird zur Version 1.1.

w3.org/TR/rdf-sparql-query

- **Merkmale von SPARQL**
  - SPARQL ermöglicht Anfragen auf einer strukturellen Ebene basierend auf der Graph-Struktur der RDF-Tripel.
  - Im RDF-Graphen können Muster gefunden werden, wozu u.a. Variablen und Konstrukte für optionale Bereiche oder Vereinigungen zur Verfügung stehen.
  - SPARQL besitzt eine (entfernt) an SQL erinnernde Syntax mit Schlüsselwörtern wie SELECT, FROM, WHERE, ORDER BY etc.

SPARQL

Struktur von Anfragen

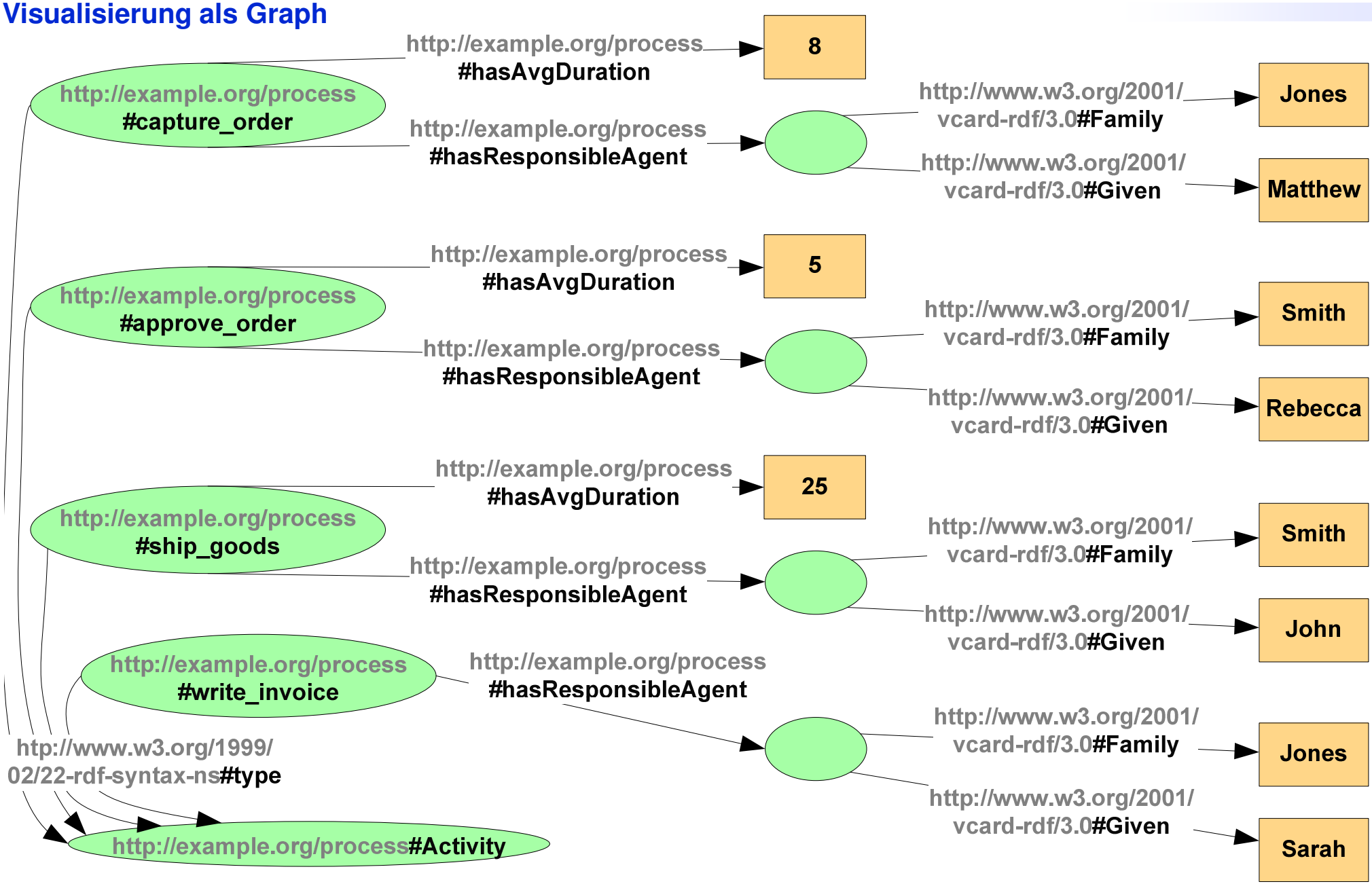
- **Schematischer Aufbau einer einfachen SPARQL-Anfrage**

```
SELECT ?var1 ... ?varN
WHERE { pattern1 . pattern2 .   ...   . patternN . }
```

  - Zunächst werden im Anschluss an SELECT diejenigen Variablen ausgewählt, die als Rückgabewerte der Anfrage dienen.
  - Anschließend wird mit Hilfe von WHERE ein Muster bestehend aus Tripeln formuliert, das zur Suche im RDF-Graphen verwendet wird.
    - Einzelne Tripel werden dabei mit einem Punkt abgeschlossen.
    - Alternativen, Filter und die Reihenfolge der Rückgabe sind dabei ebenfalls spezifizierbar.
  - SPARQL unterstützt ebenfalls Namensräume, diese werden zu Beginn (vor SELECT) im Format **`PREFIX prefixname: <URI>`** notiert.

SPARQL – Daten für Beispielanfragen

Visualisierung als Graph



SPARQL – Daten für Beispielanfragen

Darstellung in Turtle-Notation

```
@prefix vCard  : <http://www.w3.org/2001/vcard-rdf/3.0#> .
@prefix        : <http://example.org/process#> .

:capture_order :hasAvgDuration "8" ;
               :hasResponsibleAgent [
                    vCard:Family "Jones" ; vCard:Given "Matthew" ] .

:approve_order :hasAvgDuration "5" ;
               :hasResponsibleAgent [
                    vCard:Family "Smith" ; vCard:Given "Rebecca" ] .

:ship_goods    :hasAvgDuration "25" ;
               :hasResponsibleAgent [
                    vCard:Family "Smith" ; vCard:Given "John" ] .

:write_invoice :hasResponsibleAgent [
                    vCard:Family "Jones" ; vCard:Given "Sarah" ] .
```

Graphmuster

Beispiele für einfache Graphmuster

- **Gebe eine Liste mit allen Aktivitäten aus**
  - Anfrage:

```
PREFIX : <http://example.org/process#">
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#">

SELECT ?activity
WHERE { ?activity rdf:type :Activity }
```

  - Ergebnis:

activity
:capture_order
:approve_order
:ship_goods
:write_invoice

Abkürzung von „**rdf:type**“ durch „**a**“ möglich!

Hinweis: Präfixe werden im Folgenden nur bei ihrer ersten Verwendung deklariert.

Graphmuster

Beispiele für einfache Graphmuster

- **Gebe alle Aktivitäten mit ihren Bearbeitungsdauern aus**
  - Anfrage:

```
SELECT ?activity ?duration
WHERE {?activity :hasAvgDuration ?duration}
```

  - Ergebnis:

activity	duration
:capture_order	8
:approve_order	5
:ship_goods	25

Filter

Beispiel: Numerischer Vergleich

- **Suche alle Aktivitäten, die mehr als 10 Zeiteinheiten benötigen**
  - Anfrage:

```
SELECT ?activity
WHERE {
    ?activity :hasAvgDuration ?duration .
     FILTER (?duration >= 10)
}
```

  - Ergebnis:

activity
ship_goods

Weitere Möglichkeiten von SPARQL

Lösungsmodifikatoren

- **Lösungsmodifikationen**
  - Optionale Graphmuster mit dem Schlüsselwort OPTIONAL
    - Beispiel: `{?x :hat_name ?y} OPTIONAL {?x :hat_gehalt ?z}`
  - Alternative Graphmuster mit dem Schlüsselwort UNION
    - Beispiel: `{?x :hat_mobil_nr ?y} UNION {?x :hat_handy_nr ?y }`
  - Sortierung mit ORDER BY
    - Beispiele: `ORDER BY ?x ?y, ORDER BY DESC(?x)`
  - Ausschnitt aus der Ergenismenge mit OFFSET und LIMIT
    - Beispiele: `OFFSET 10, LIMIT 5`
  - Eindeutige Variablen-Werte-Kombinationen mit DISTINCT
    - Beispiel: `SELECT DISTINCT ?student`

Weitere Anfrageformen

ASK, DESCRIBE und Weitere

- **ASK**
  - Verwendung, wenn nur eine Ja/Nein-Antwort erforderlich ist.
  - Beispiel: `ASK { :erstelle_rechnung  rdf:type :Activity }`

- **DESCRIBE**
  - Verwendung, wenn alle Fakten über ein Objekt zurückgegeben werden sollen.
  - Beispiel: `DESCRIBE :erstelle_rechnung`

- **Weitere Anfrageformen und deren Verwendung**
  - CONSTRUCT: Konstruktion von Graphen im Kontext von Anfragen.
  - UPDATE: Änderung von Fakten.
  - INSERT DATA: Eingabe von Fakten.
  - DELETE: Löschung von Fakten.

  } **Jena/ARQ-spezifisch**

Abkürzungen in SPARQL-Anfragen

Abkürzungen der N3-Syntax sind anwendbar

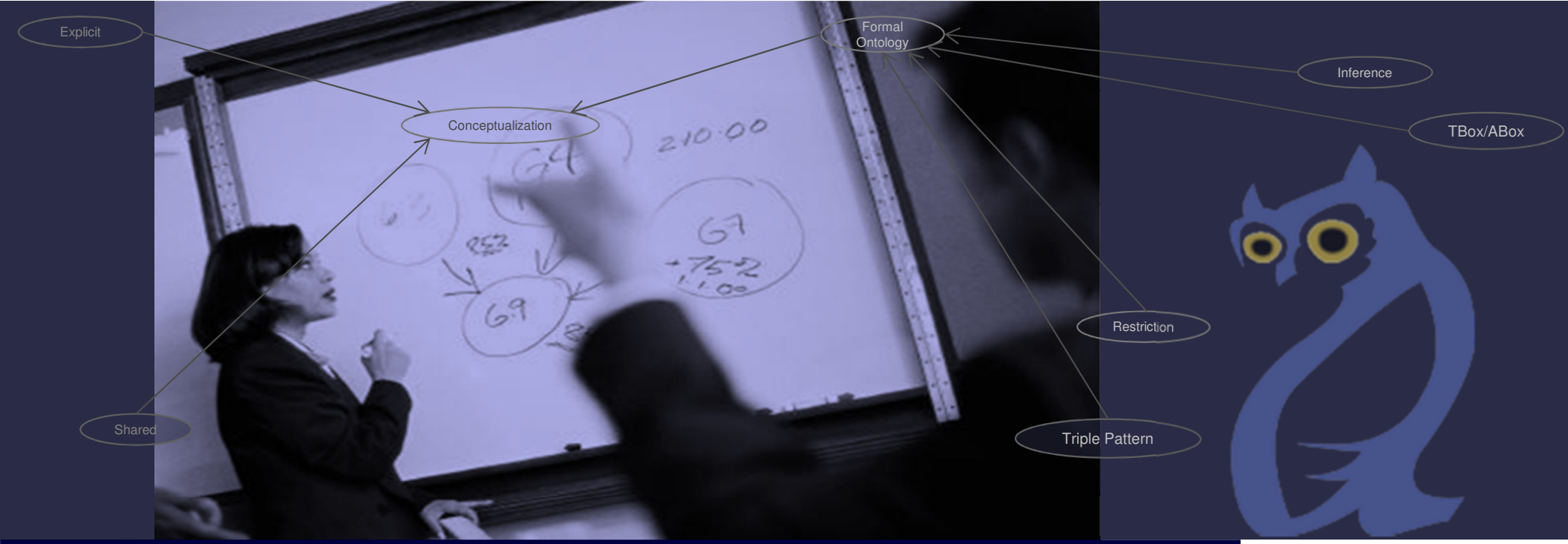
- **Mehrere Graphmuster mit gleichem Subjekt**
  - Beispiel Langform: `s1 p1 o1 . s1 p2 o2 .`
  - Beispiel Kurzform: `s1 p1 o1 ; p2 o2 .`
- **Mehrere Graphmuster mit gleichem Subjekt und Prädikat**
  - Beispiel Langform: `s1 p1 o1 . s1 p1 o2 .`
  - Beispiel Kurzform: `s1 p1 o1 , o2 .`
- **Graphmuster mit anonymem Knoten**
  - Anonyme Knoten können an der Subjekt- oder Objekt-Position eines Graphmusters auftreten. Im unteren Bsp. wird ein anonymer Knoten an der Objekt-Position gezeigt.
  - Beispiel Langform: `s1 p1 o1 . o1 p2 o2 .`
  - Beispiel Kurzform: `s1 p1 [ p2 o2 ] .`

→ **Die vorgestellten Abkürzungen können kombiniert werden!**

Schlussbemerkung

zu Anfragesprachen

- **Anfragesprachen**
  - Es existiert eine Vielfalt an Anfragesprechen.
  - Für OWL sind derzeit vor allem DL-Anfragen und SPARQL-Anfragen relevant
    - DL-Anfragen eignen sich vor allem, wenn komplexe Klassenbeschreibungen zur Anfrage eingesetzt werden sollen.
    - SPARQL-Anfragen eignen sich vor allem, wenn Instanzen und deren Rückgabe als Tabellen intendiert werden.
  - Zur Anfrage in semantischen Wikis werden teils eigene Sprachen wie SMW-QL vorgeschlagen. Diese sind allerdings weniger flexibel einsetzbar als SPARQL.



Semantisches Prozessmanagement und E-Business

Michael Fellmann

Lehrveranstaltung im SS 2013

UNIVERSITÄT OSNABRÜCK

Institut für Informationsmanagement
und Unternehmensführung
michael.fellmann@uos.de

ANHANG

ANHANG – DL-Syntax

Übersicht

	Concepts	
$\mathcal{ALC}$	Atomic	A, B
	Not	$\neg C$
	And	$C \sqcap D$
	Or	$C \sqcup D$
	Exists	$\exists R.C$
	For all	$\forall R.C$
$\mathcal{Q}$ $(\mathcal{N})$	At least	$\geq n\ R.C$ $(\geq n\ R)$
	At most	$\leq n\ R.C$ $(\leq n\ R)$
$\mathcal{O}$	Nominal	$\{i_1, \ldots, i_n\}$

	Roles	
	Atomic	R
$\mathcal{I}$	Inverse	R^-

Ontology (=Knowledge Base)

	Concept Axioms (TBox)	
	Subclass	$C \sqsubseteq D$
	Equivalent	$C \equiv D$

	Role Axioms (RBox)	
$\mathcal{H}$	Subrole	$R \sqsubseteq S$
$\mathcal{S}$	Transitivity	Trans(S)

	Assertional Axioms (ABox)	
	Instance	C(a)
	Role	R(a,b)
	Same	$a = b$
	Different	$a \neq b$

Quelle: Krötzsch, Hitzler, Rudolph (2008): Vorlesungsunterlagen zur Veranstaltung „Semantic Web Technologies I “ am AIFB, Universität Karlsruhe.
Siehe auch: www.semantic-web-grundlagen.de

ANHANG – DL-Syntax

Klassenkonstruktoren

Constructor	DL Syntax	Example
intersectionOf	$C_1 \sqcap \ldots \sqcap C_n$	Human $\sqcap$ Male
unionOf	$C_1 \sqcup \ldots \sqcup C_n$	Doctor $\sqcup$ Lawyer
complementOf	$\neg C$	$\neg$Male
oneOf	$\{x_1\} \sqcup \ldots \sqcup \{x_n\}$	{john} $\sqcup$ {mary}
allValuesFrom	$\forall P.C$	$\forall$hasChild.Doctor
someValuesFrom	$\exists P.C$	$\exists$hasChild.Lawyer
maxCardinality	$\leqslant nP$	$\leqslant$1hasChild
minCardinality	$\geqslant nP$	$\geqslant$2hasChild

Quelle: Krötzsch, Hitzler, Rudolph (2008): Vorlesungsunterlagen zur Veranstaltung „Semantic Web Technologies I “ am AIFB, Universität Karlsruhe.
Siehe auch: www.semantic-web-grundlagen.de

ANHANG – DL-Syntax

Axiome

Axiom	DL Syntax	Example
subClassOf	$C_1 \sqsubseteq C_2$	$\mathsf{Human} \sqsubseteq \mathsf{Animal} \sqcap \mathsf{Biped}$
equivalentClass	$C_1 \equiv C_2$	$\mathsf{Man} \equiv \mathsf{Human} \sqcap \mathsf{Male}$
disjointWith	$C_1 \sqsubseteq \neg C_2$	$\mathsf{Male} \sqsubseteq \neg\mathsf{Female}$
sameAs	$\{x_1\} \equiv \{x_2\}$	$\{\mathsf{President\_Bush}\} \equiv \{\mathsf{G\_W\_Bush}\}$
differentFrom	$\{x_1\} \sqsubseteq \neg\{x_2\}$	$\{\mathsf{john}\} \sqsubseteq \neg\{\mathsf{peter}\}$
subPropertyOf	$P_1 \sqsubseteq P_2$	$\mathsf{hasDaughter} \sqsubseteq \mathsf{hasChild}$
equivalentProperty	$P_1 \equiv P_2$	$\mathsf{cost} \equiv \mathsf{price}$
inverseOf	$P_1 \equiv P_2^-$	$\mathsf{hasChild} \equiv \mathsf{hasParent}^-$
transitiveProperty	$P^+ \sqsubseteq P$	$\mathsf{ancestor}^+ \sqsubseteq \mathsf{ancestor}$
functionalProperty	$\top \sqsubseteq \leqslant 1 P$	$\top \sqsubseteq \leqslant 1 \mathsf{hasMother}$
inverseFunctionalProperty	$\top \sqsubseteq \leqslant 1 P^-$	$\top \sqsubseteq \leqslant 1 \mathsf{hasSSN}^-$

Quelle: Krötzsch, Hitzler, Rudolph (2008): Vorlesungsunterlagen zur Veranstaltung „Semantic Web Technologies I “ am AIFB, Universität Karlsruhe.
Siehe auch: www.semantic-web-grundlagen.de

SPARQL mit SemQuu praktisch anwenden

Download im „Dateien“-Bereich der Vorlesung!

Optionen zur Validierung

Regeln erfassen u. speichern

Optionen für Anfragen

Ergebnis-Bereich

URI der Ontologie

SPARQL Anfragen mit Vorschlags-Funktion

SPARQL Anfragen

Firefox

SemQuu — Semantic Querying

localhost:8080/semquu/

SemQuu — Semantic Querying

– Doc

http://localhost:8080/semquu/ontologies/process/kb.owl

Load and validate

(show options)

+ Rules

– Query

Add Pattern: -- subject -- predicate -- object

Add Other: -- Construct

Return Vars: ?function ?event

```
SELECT ?function ?event
WHERE {
 ?function :flow ?event . ?
}
```

?event
?function
?x
?y
?z

Execute Reset Save SELECT ASK DESCRIBE UPDATE (insert queries) (show prefixes) (help)

(show options)

Result

function	event
Protokoll_in_festgelegten_Ordner_ablegen	Protokoll_ist_abgelegt
Grafik_erstellen	Grafik_ist_erstellt
Pruefmittel_checken	Pruefmittel_sind_in_Ordnung
Pruefwerte_in_Pruefmittelkarte_eintragen	Pruefwerte_sind_eingetragen

Auf Fragen, Bugs und Anregungen freut sich:
Michael.Fellmann@uos.de