

# GRUNDLAGEN SEMANTIC WEB

Lehrveranstaltung im WS10/11  
Seminar für Computerlinguistik  
Universität Heidelberg

Dr. Sebastian Rudolph  
Institut AIFB  
Universität Karlsruhe

# RDF SCHEMA

Dr. Sebastian Rudolph

Einleitung und Ausblick

XML und URIs

Einführung in RDF

RDF Schema

Logik - Grundlagen

Semantik von RDF(S)

OWL - Syntax und Intuition

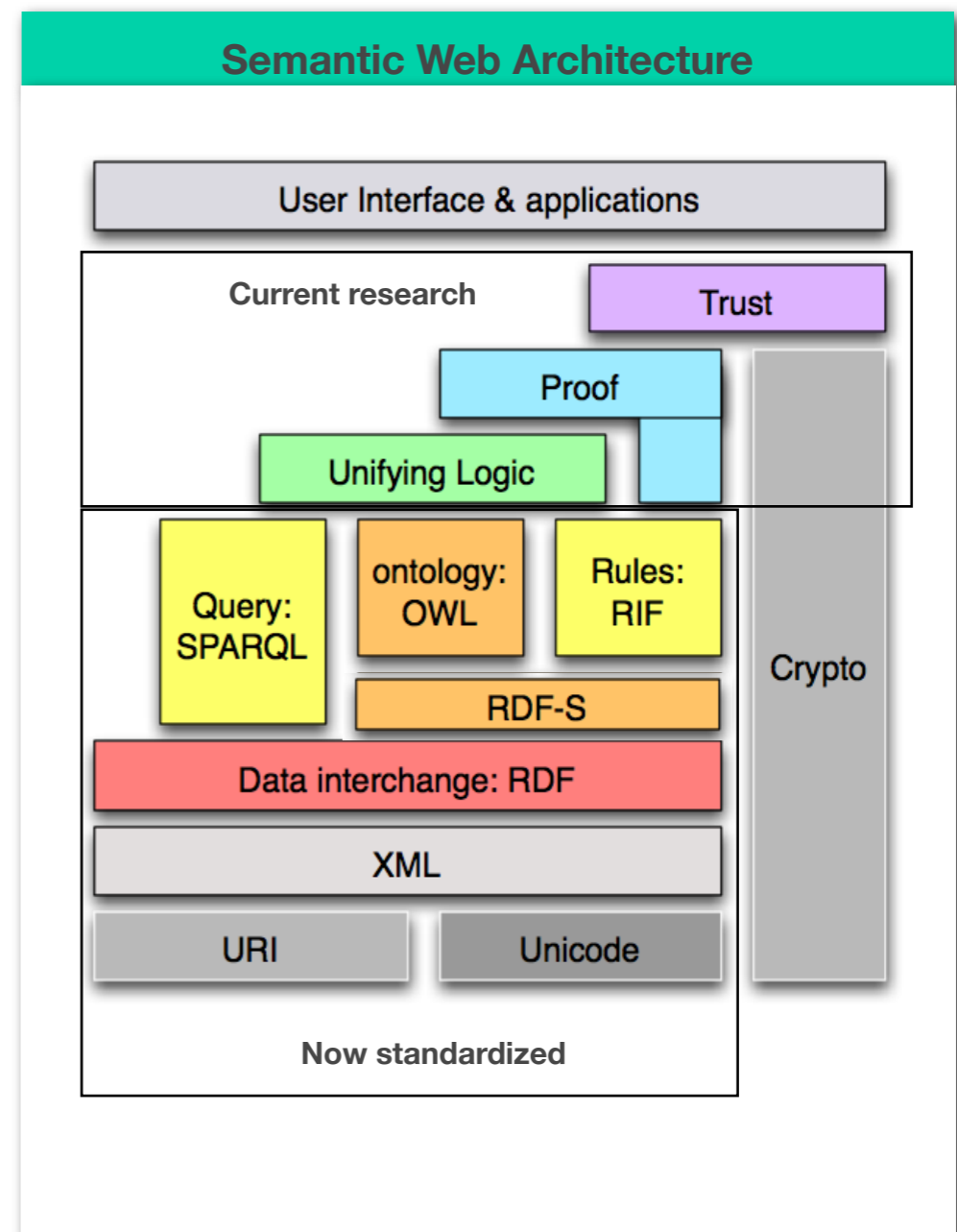
OWL - Semantik und Reasoning

SPARQL - Syntax und Intuition

Semantik von SPARQL und konjunktive Anfragen

OWL 1.1 - Syntax und Semantik

Semantic Web und Regeln



# RDF SCHEMA

Dr. Sebastian Rudolph

Einleitung und Ausblick

XML und URIs

Einführung in RDF

**RDF Schema**

Logik - Grundlagen

Semantik von RDF(S)

OWL - Syntax und Intuition

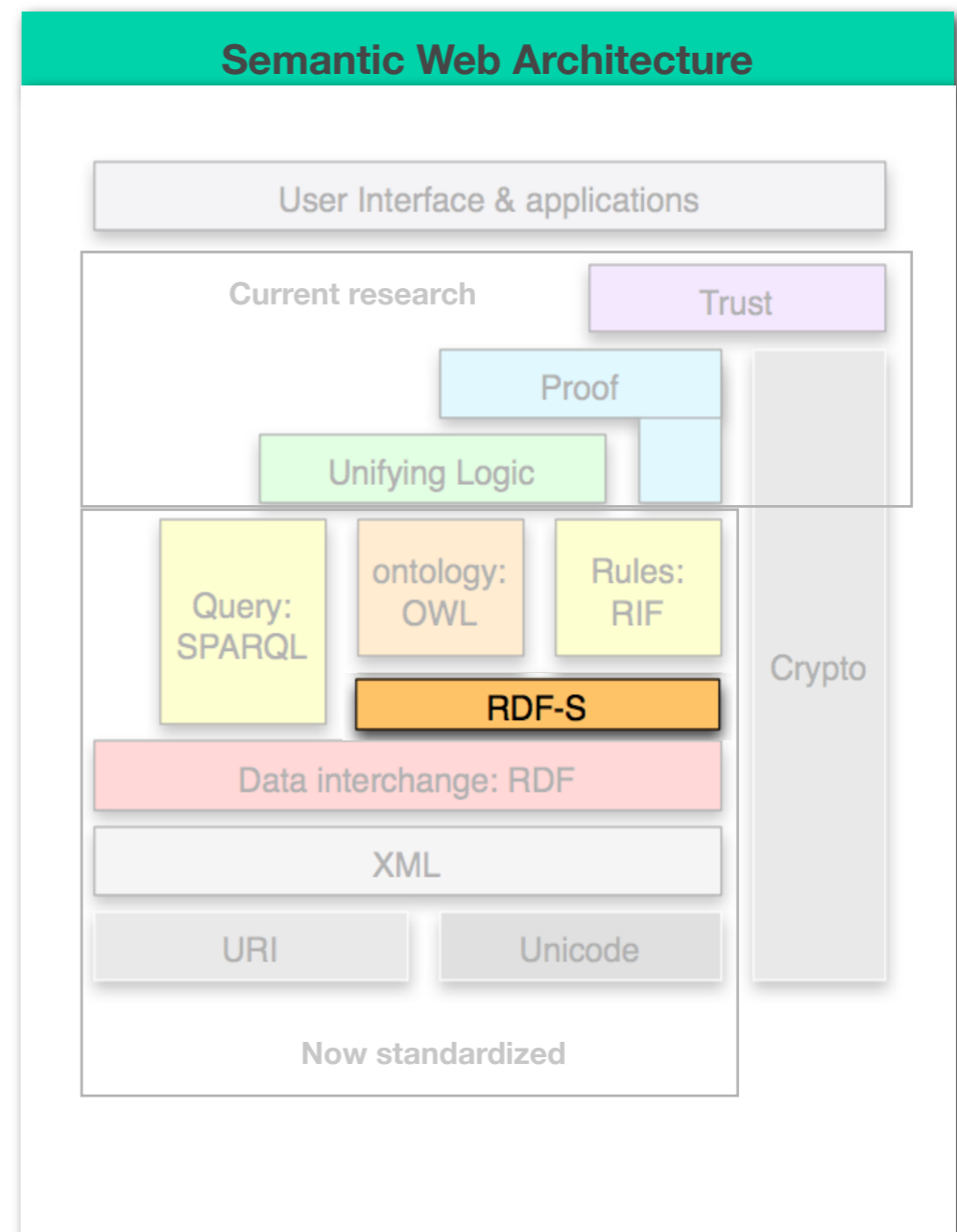
OWL - Semantik und Reasoning

SPARQL - Syntax und Intuition

Semantik von SPARQL und konjunktive Anfragen

OWL 1.1 - Syntax und Semantik

Semantic Web und Regeln



# AGENDA



- Motivation
- Klassen und Klassenhierarchien
- Propertys und Propertyhierarchien
- Einschränkungen auf Propertys
- offene Listen
- Reifikation
- zusätzliche Informationen in RDFS
- einfache Ontologien

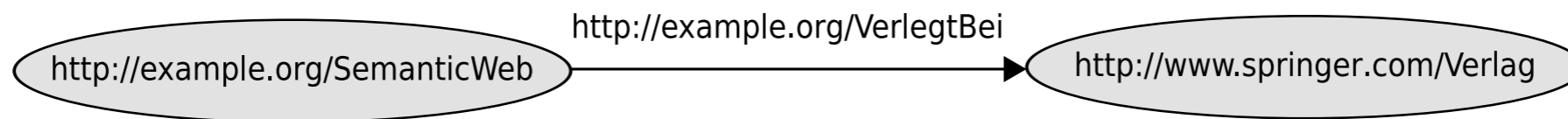
# AGENDA



- **Motivation**
- Klassen und Klassenhierarchien
- Propertys und Propertyhierarchien
- Einschränkungen auf Propertys
- offene Listen
- Reifikation
- zusätzliche Informationen in RDFS
- einfache Ontologien

# SCHEMAWISSEN MIT RDFS

- RDF bietet universelle Möglichkeit zur Kodierung von faktischen Daten im Web:



- = Aussagen über einzelne Ressourcen (Individuen) und deren Beziehungen
- wünschenswert: Aussagen über generische Mengen von Individuen (Klassen), z.B. Verlage, Organisationen, Personen etc.

# SCHEMAWISSEN MIT RDFS



- weiterhin wünschenswert: Spezifikation der logischen Zusammenhänge zwischen Individuen, Klassen und Beziehungen, um möglichst viel Semantik des Gegenstandsbereiches einzufangen, z.B.:  
*"Verlage sind Organisationen."*  
*"Nur Personen schreiben Bücher."*
- in Datenbanksprache: Schemawissen

# SCHEMAWISSEN MIT RDFS

- RDF Schema (RDFS):
  - Teil der W3C Recommendation zu RDF
  - ermöglicht Spezifikation von *schematischem* (auch: *terminologischem*) Wissen
  - spezielles RDF-Vokabular (also: jedes RDFS-Dokument ist ein RDF-Dokument)
  - Namensraum (i.d.R. abgekürzt mit rdfs:) :  
<http://www.w3.org/2000/01/rdf-schema#>



# SCHEMAWISSEN MIT RDFS



- RDF Schema (RDFS):
  - jedoch: Vokabular nicht themengebunden (wie z.B. bei FOAF), sondern generisch
  - erlaubt die Spezifikation (von Teilen) der Semantik beliebiger RDF-Vokabulare (ist also eine Art „Metavokabular“)
  - Vorteil: jede Software mit RDFS-Unterstützung interpretiert jedes vermittels RDFS definierte Vokabular korrekt
  - Funktionalität macht RDFS zu einer Ontologiesprache (für leichtgewichtige - engl.: lightweight - Ontologien)
  - „A little semantics goes a long way.“

# AGENDA

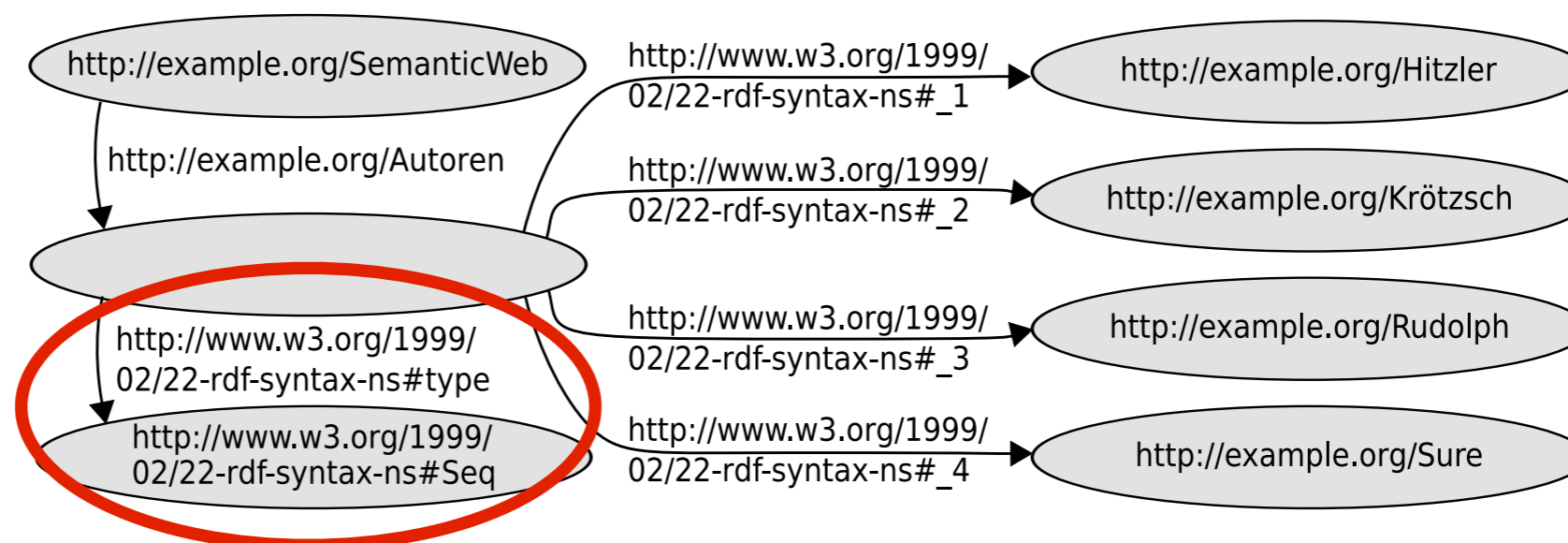


- Motivation
- **Klassen und Klassenhierarchien**
- Propertys und Propertyhierarchien
- Einschränkungen auf Propertys
- offene Listen
- Reifikation
- zusätzliche Informationen in RDFS
- einfache Ontologien

# KLASSEN UND INSTANZEN



- Typisierung von Ressourcen bereits in RDF zur Kennzeichnung von Listen:



- Prädikat `rdf:type` weist dem Subjekt das Objekt als Typ zu
- Objekt aufgefasst als Bezeichner für *Klasse*, der die durch das Subjekt bezeichnete Ressource (als sog. *Instanz*) angehört

# KLASSEN UND INSTANZEN



`ex:SemanticWeb rdf:type ex:Lehrbuch .`

- charakterisiert „Semantic Web - Grundlagen“ als Instanz der (neu definierten) Klasse „Lehrbuch“
- Klassenzugehörigkeit ist nicht exklusiv, z.B. mit o.g. Tripel gleichzeitig möglich:  
`ex:SemanticWeb rdf:type ex:Unterhaltsam .`
- allgemein: a priori syntaktisch keine eindeutige Unterscheidung zwischen Individuen- und Klassenbezeichnern möglich
- auch in der Realität Charakterisierung manchmal schwierig, beispielsweise für <http://www.un.org/#URI>

# DIE KLASSE ALLER KLASSEN



- jedoch manchmal eindeutige Kennzeichnung einer URI als Klassenbezeichner wünschenswert
- möglich durch Typung der betreffenden URI als `rdfs:Class`

```
es:Lehrbuch rdf:type rdfs:Class .
```

- `rdfs:Class` ist also die „Klasse aller Klassen“ und enthält sich damit auch selbst, d.h. das folgende Tripel ist immer wahr:

```
rdfs:Class rdf:type rdfs:Class .
```

# UNTERKLASSEN - MOTIVATION



- gegeben Tripel  
`ex:SemanticWeb rdf:type ex:Lehrbuch .`
- Problem: Suche nach Instanzen der Klasse  
`ex:Buch` liefert kein Resultat
- Möglichkeit: Hinzufügen von Tripel  
`ex:SemanticWeb rdf:type ex:Buch .`
- löst das Problem aber nur für die eine Ressource  
`ex:SemanticWeb`
- automatisches Hinzufügen für alle Instanzen führt zu unnötig großen RDF-Dokumenten

# UNTERKLASSEN



- Sinnvoller: einmalige Aussage, dass jedes Lehrbuch auch ein Buch ist, d.h. jede Instanz der Klasse `ex:Lehrbuch` ist automatisch auch eine Instanz der Klasse `ex:Buch`

- realisiert durch die `rdfs:subClassOf`-Property:

```
ex:Lehrbuch rdfs:subClassOf ex:Buch .
```

„Die Klasse der Lehrbücher ist eine *Unterklasse* der Klasse der Bücher.“

# UNTERKLASSEN



- `rdfs:subClassOf`-Property ist reflexiv, d.h. jede Klasse ist Unterklasse von sich selbst, so dass z.B. gilt:

```
ex:Lehrbuch rdfs:subClassOf ex:Lehrbuch .
```

- umgekehrt: Festlegung der Gleichheit zweier Klassen durch gegenseitige Unterklassenbeziehung, etwa:

```
ex:Hospital rdfs:subClassOf ex:Krankenhaus .  
ex:Krankenhaus rdfs:subClassOf ex:Hospital .
```



# KLASSENHIERARCHIEN



- Üblich: nicht nur einzelne Unterklassenbeziehungen sondern ganze *Klassenhierarchien* (auch: *Taxonomien*)  
z.B.:  
ex:Lehrbuch rdfs:subClassOf ex:Buch .  
ex:Buch rdfs:subClassOf ex:Printmedium .  
ex:Zeitschrift rdfs:subClassOf ex:Printmedium .
- in RDFS-Semantik verankert: Transitivität der rdfs:subClassOf-Property, d.h. es folgt automatisch  
ex:Lehrbuch rdfs:subClassOf ex:Printmedium .

# KLASSENHIERARCHIEN



- Klassenhierarchien besonders ausgeprägt etwa in Biologie (z.B. *Klassifikation von Lebewesen*)
- z.B. zoologische Einordnung des modernen Menschen

```

<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:ex="http://www.semantic-web-grundlagen.de/Beispiele#">
  <rdfs:Class rdf:about="&ex;Animalia">
    <rdfs:label xml:lang="de">Tiere</rdfs:label>
  </rdfs:Class>
  <rdfs:Class rdf:about="&ex;Chordata">
    <rdfs:label xml:lang="de">Chordatiere</rdfs:label>
    <rdfs:subClassOf rdfs:resource="&ex;Animalia"/>
  </rdfs:Class>
  <rdfs:Class rdf:about="&ex;Mammalia">
    <rdfs:label xml:lang="de">Säugetiere</rdfs:label>
    <rdfs:subClassOf rdfs:resource="&ex;Chordata"/>
  </rdfs:Class>
  <rdfs:Class rdf:about="&ex;Primates">
    <rdfs:label xml:lang="de">Primaten</rdfs:label>
    <rdfs:subClassOf rdfs:resource="&ex;Mammalia"/>
  </rdfs:Class>
  <rdfs:Class rdf:about="&ex;Hominidae">
    <rdfs:label xml:lang="de">Menschenaffen</rdfs:label>
    <rdfs:subClassOf rdfs:resource="&ex;Primates"/>
  </rdfs:Class>
  <rdfs:Class rdf:about="&ex;Homo">
    <rdfs:label xml:lang="de">Mensch</rdfs:label>
    <rdfs:subClassOf rdfs:resource="&ex;Hominidae"/>
  </rdfs:Class>
  <rdfs:Class rdf:about="&ex;HomoSapiens">
    <rdfs:label xml:lang="de">Moderner Mensch</rdfs:label>
    <rdfs:subClassOf rdfs:resource="&ex;Homo"/>
  </rdfs:Class>
  <ex:HomoSapiens rdf:about="&ex;SebastianRudolph"/>
</rdf:RDF>

```

# KLASSEN

- intuitive Parallele zur Mengenlehre:

`rdf:type` entspricht  $\in$

`rdfs:subClassOf` entspricht  $\subseteq$

- rechtfertigt beispielsweise auch die Reflexivität und Transitivität von `rdfs:subClassOf`

# KLASSEN IN RDF/XML-SYNTAX



- verkürzte Darstellungen bei Angabe von Klasseninstanzen möglich:

```
<ex:HomoSapiens rdf:about="&ex;SebastianRudolph"/>
```

an Stelle von

```
<rdf:Description rdf:about="&ex;SebastianRudolph">
```

```
<rdf:type rdf:resource="&ex;HomoSapiens">
```

```
</rdf:Description>
```

- dementsprechend auch

```
<rdfs:Class rdf:about="&ex;HomoSapiens"/>
```

# VORDEFINIERTE KLASSENBEZEICHNER



- `rdfs:Resource`  
Klasse aller Ressourcen (also sämtliche Elemente des Gegenstandsbereiches)
- `rdf:Property`  
Klasse aller Beziehungen  
(= die Ressourcen, die durch Prädikats-URIs referenziert werden)
- `rdf:List`, `rdf:Seq`, `rdf:Bag`, `rdf:Alt`, `rdfs:Container`  
Klassen verschiedener Arten von Listen
- `rdfs:ContainerMembershipProperty`  
Klasse aller Beziehungen, die eine Enthaltenseinsbeziehung darstellen

# VORDEFINIERTER KLASSENBEZEICHNER



- `rdf:XMLLiteral`  
Klasse aller Werte des vordefinierten Datentyps XMLLiteral
- `rdfs:Literal`  
Klasse aller Literalwerte (enthält also alle Datentypen als Unterklassen)
- `rdfs:Datatype`  
Klasse aller Datentypen (ist also wie `rdfs:Class` eine Klasse von Klassen)
- `rdf:Statement`  
Klasse aller reifizierten Aussagen (s. dort)

# AGENDA



- Motivation
- Klassen und Klassenhierarchien
- **Propertys und Propertyhierarchien**
- Einschränkungen auf Propertys
- offene Listen
- Reifikation
- zusätzliche Informationen in RDFS
- einfache Ontologien

# PROPERTY



- andere Bezeichnungen: Relationen, Beziehungen
- Achtung: Property sind in RDF(S) nicht (wie in OOP) speziellen Klassen zugeordnet
- Property-Bezeichner in Tripeln üblicherweise an Prädikatsstelle
- charakterisieren, auf welche Art zwei Ressourcen zueinander in Beziehung stehen
- mathematisch oft dargestellt als Menge von Paaren:  
verheiratet\_mit = {(Adam,Eva),(Brad,Angelina),...}
- URI wird als Property-Bezeichner gekennzeichnet durch entsprechende Typung:  
`ex:verlegtBei rdf:type rdf:Property .`



# UNTERPROPERTY



- ähnlich zu Unter-/Oberklassen auch Unter-/Oberproperty denkbar und sinnvoll
- Darstellung in RDFS mittels `rdfs:subPropertyOf` z.B.:  
`ex:glücklichVerheiratetMit rdfs:subPropertyOf rdf:verheiratetMit .`
- erlaubt, aus dem Tripel  
`ex:Markus ex:glücklichVerheiratetMit ex:Anja .`  
zu schlussfolgern, dass  
`ex:Markus ex:verheiratetMit ex:Anja .`

# AGENDA



- Motivation
- Klassen und Klassenhierarchien
- Propertys und Propertyhierarchien
- **Einschränkungen auf Propertys**
- offene Listen
- Reifikation
- zusätzliche Informationen in RDFS
- einfache Ontologien

# EINSCHRÄNKUNG VON PROPERTYS



- häufig: Property kann sinnvoll nur ganz bestimmte Ressourcen verbinden, z.B. verbindet `ex:verlegtBei` nur Publikationen mit Verlagen
- d.h. für alle URIs `a`, `b` folgt aus dem Tripel  
`a ex:verlegtBei b .`  
dass auch gilt:  
`a rdf:type ex:Publikation .`  
`b rdf:type ex:Verlag .`
- kann in RDFS direkt kodiert werden:  
`ex:verlegtBei rdfs:domain ex:Publikation .`  
`ex:verlegtBei rdfs:range ex:Verlag .`
- auch zur Angabe von Datentypen für Literale:  
`ex:hatAlter rdfs:range xsd:nonNegativeInteger .`

# EINSCHRÄNKUNG VON PROPERTYS



- Propertyeinschränkungen bieten die einzige Möglichkeit, semantische Zusammenhänge zwischen Propertys und Klassen zu spezifizieren
- Achtung: Propertyeinschränkungen wirken global und konjunktiv, z.B.

`ex:autorVon rdfs:range ex:Kochbuch .`

`ex:autorVon rdfs:range ex:Märchenbuch .`

bedeutet: jede Entität, von der jemand Autor ist, ist **gleichzeitig** Kochbuch und Märchenbuch

- daher: als domain/range immer allgemeinste mögliche Klasse verwenden

# AGENDA

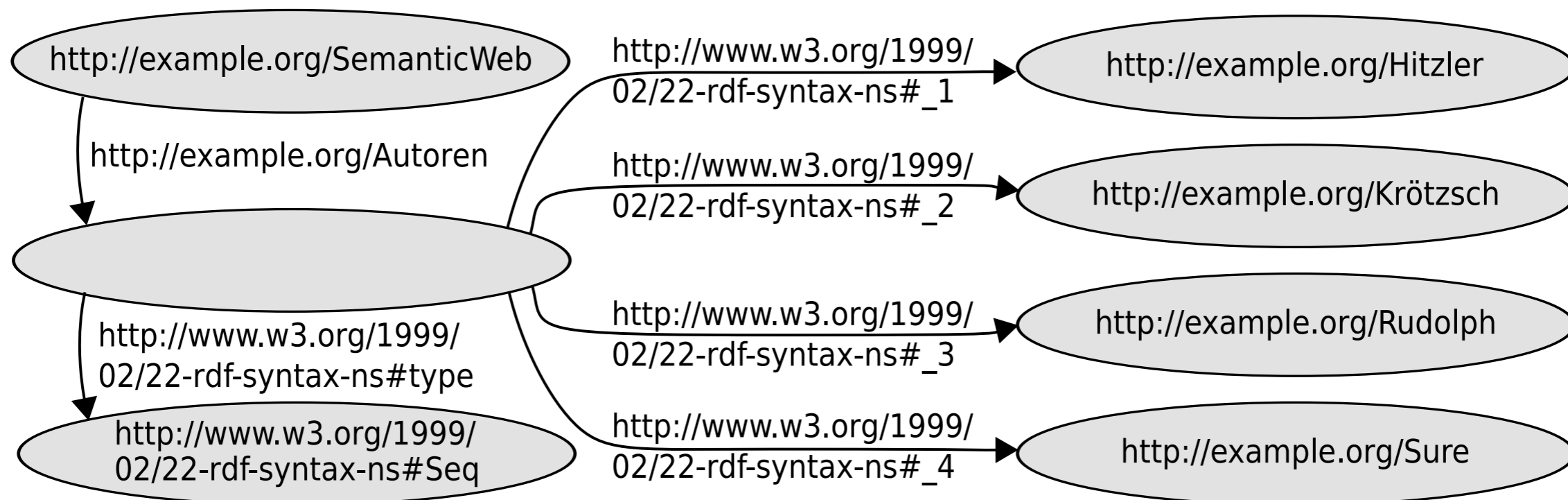


- Motivation
- Klassen und Klassenhierarchien
- Propertys und Propertyhierarchien
- Einschränkungen auf Propertys
- **offene Listen**
- Reifikation
- zusätzliche Informationen in RDFS
- einfache Ontologien

# ARBEIT MIT OFFENEN LISTEN



- zur Erinnerung: offene Listen in RDF:



# ARBEIT MIT OFFENEN LISTEN



- neue Klasse: `rdfs:Container` als Oberklasse von `rdf:Seq`, `rdf:Bag`, `rdf:Alt`
- neue Klasse: `rdfs:ContainerMembershipProperty`  
Elemente sind keine Individuen i.e.S. sondern selbst Property
- intendierte Semantik: jede Property, die aussagt, dass das Subjekt im Objekt enthalten ist, ist Instanz von `rdfs:ContainerMembershipProperty`
- Es gilt also insbesondere  
`rdf:_1 rdf:type rdfs:ContainerMembershipProperty .`  
`rdf:_2 rdf:type rdfs:ContainerMembershipProperty .`  
etc.

# ARBEIT MIT OFFENEN LISTEN



- neue Property: `rdfs:member`  
Oberproperty aller in `rdfs:ContainerMembershipProperty` enthaltenen Property's, also die „universelle Enthaltenseinsrelation“
- damit in RDFS-Semantik verankert: wann immer für eine Property `p` das Tripel  
`p rdfs:type rdfs:ContainerMembershipProperty .`  
gilt, folgt aus dem Tripel  
`a p b .`  
sofort das Tripel  
`a rdfs:member b .`



# AGENDA



- Motivation
- Klassen und Klassenhierarchien
- Propertys und Propertyhierarchien
- Einschränkungen auf Propertys
- offene Listen
- **Reifikation**
- zusätzliche Informationen in RDFS
- einfache Ontologien

# REIFIKATION



- Problematisch in RDF(S): Modellierung von Aussagen über Aussagen (häufig zu erkennen am Wort „dass“), z.B.:  
*„Der Detektiv vermutet, **dass** der Butler den Gärtner ermordet hat.“*
- erster Modellierungsversuch:  
ex:detektiv ex:vermutet "Der Butler hat den Gärtner ermordet." .
  - ungünstig: auf Literal-Objekt kann schlecht in anderen Aussagen Bezug genommen werden (keine URI)
- zweiter Modellierungsversuch:  
ex:detektiv ex:vermutet ex:derButlerHatDenGärtnerErmordet .
  - ungünstig: innere Struktur der dass-Aussage geht verloren

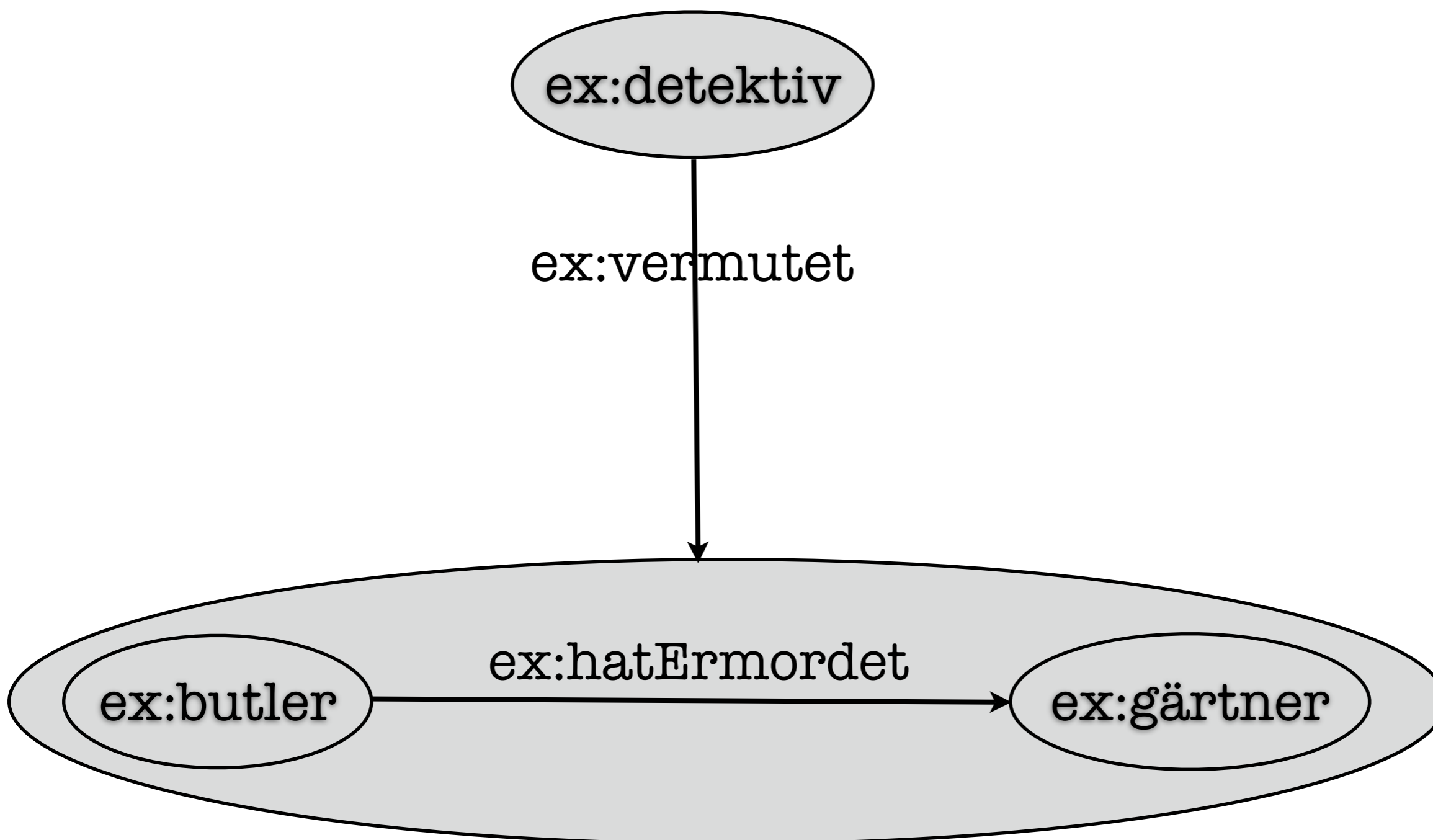
# REIFIKATION



- Problematisch in RDF(S): Modellierung von Aussagen über Aussagen (häufig zu erkennen am Wort „dass“), z.B.:  
*„Der Detektiv vermutet, **dass** der Butler den Gärtner ermordet hat.“*
- einzelne dass-Aussage leicht in RDF modellierbar:  
`ex:butler ex:hatErmordet ex:gärtner .`
- wünschenswert: ganzes RDF-Tripel als Objekt eines anderen Tripels; ist aber kein gültiges RDF

# REIFIKATION

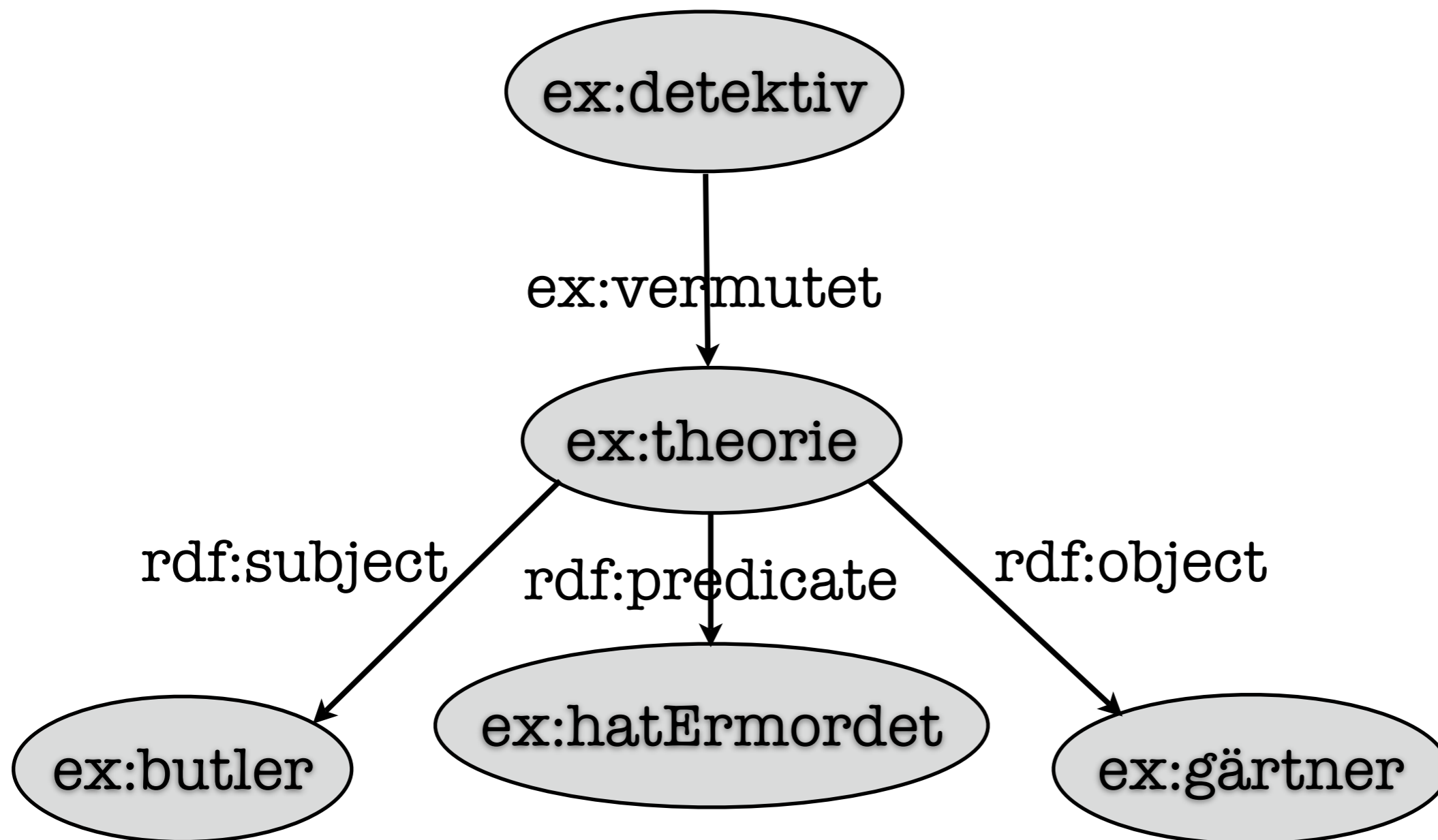
- Lösung (ähnlich wie bei mehrwertigen Beziehungen): Hilfsknoten für die geschachtelte Aussage:



# REIFIKATION



- Lösung (ähnlich wie bei mehrwertigen Beziehungen): Hilfsknoten für die geschachtelte Aussage:



# REIFIKATION

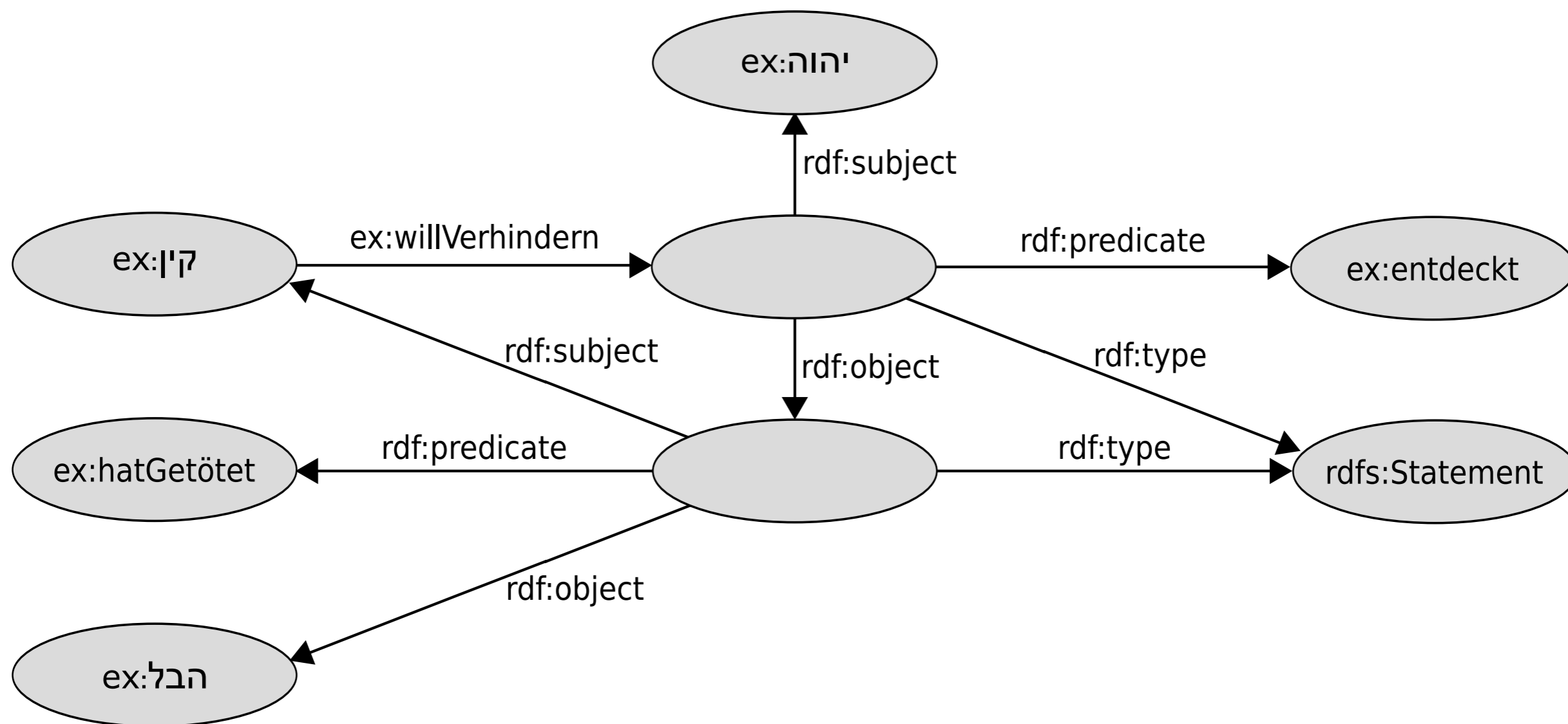


- Achtung: reifiziertes Tripel muss nicht unbedingt gelten (wäre auch nicht immer sinnvoll, z.B. bei Aussagen wie: „Der Detektiv bezweifelt, dass der Butler den Gärtner ermordet hat.“)
- falls dies gewünscht ist, muss das originale (unreifizierte) Tripel dem RDF-Dokument nochmals hinzugefügt werden
- der Klassenbezeichner `rdf:Statement` dient zur Kennzeichnung aller solcher Aussagen-Hilfsknoten
- falls auf eine Aussage nicht (extern) Bezug genommen wird, kann der entsprechende Hilfsknoten ein `bnode` sein

# REIFIKATION



- Übungsaufgabe: noch eine Kriminalgeschichte...



# AGENDA



- Motivation
- Klassen und Klassenhierarchien
- Propertys und Propertyhierarchien
- Einschränkungen auf Propertys
- offene Listen
- Reifikation
- **zusätzliche Informationen in RDFS**
- einfache Ontologien



# ZUSATZINFORMATIONEN



- wie bei Programmiersprachen manchmal Hinzufügen von Kommentaren (ohne Auswirkung auf Semantik) wünschenswert
- Zweck: Erhöhung der Verständlichkeit für menschlichen Nutzer
- es empfiehlt sich (z.B. aus Tool-Kompatibilitätsgründen) auch dieses Wissen als Graph zu repräsentieren
- also: Satz von Property's, die diesem Zweck dienen

# ZUSATZINFORMATIONEN



- `rdfs:label`
  - Property, die einer (beliebigen) Ressource einen alternativen Namen zuweist (Literal)
  - oftmals sind URIs schwer lesbar; zumindest „unhandlich“
  - durch `rdfs:label` zugewiesener Name wird z.B. häufig von Tools bei der graphischen Darstellung verwendet
  - Beispiel (incl. Sprachinformation):

```
<rdfs:Class rdf:about="&ex;Hominidae">
<rdfs:label xml:lang="de">Menschenaffen</rdfs:label>
</rdfs:Class>
```

# ZUSATZINFORMATIONEN



- `rdfs:comment`
  - Property, die einer (beliebigen) Ressource einen umfangreichen Kommentar zuweist (Literal)
  - beinhaltet z.B. natürlichsprachliche Definition einer neu eingeführten Klasse - erleichtert spätere intentionsgemäße Wiederverwendung
- `rdfs:seeAlso`, `rdfs:definedBy`
  - Property, die Ressourcen (URIs!) angeben, die weitere Informationen bzw. eine Definition der Subjekt-Ressource bereitstellen

# ZUSATZINFORMATIONEN



- ## Verwendungsbeispiel

```
:  
  xmlns:wikipedia="http://de.wikipedia.org/wiki/"  
:  
<rdfs:Class rdf:about="&ex;Primates">  
  <rdfs:label xml:lang="de">Primaten</rdfs:label>  
  <rdfs:comment>  
    Eine Säugetierordnung. Primaten zeichnen sich durch ein  
    hochentwickeltes Gehirn aus. Sie besiedeln hauptsächlich  
    die wärmeren Erdregionen.  
    Die Bezeichnung Primates (lat. "Herrentiere") stammt von  
    Carl von Linné.  
  </rdfs:comment>  
  <rdfs:seeAlso rdf:resource="&wikipedia;Primaten"/>  
  <rdfs:subClassOf rdf:resource="&ex;Mammalia"/>  
</rdfs:Class>
```

# AGENDA



- Motivation
- Klassen und Klassenhierarchien
- Propertys und Propertyhierarchien
- Einschränkungen auf Propertys
- offene Listen
- Reifikation
- zusätzliche Informationen in RDFS
- **einfache Ontologien**

# EINFACHE ONTOLOGIEN



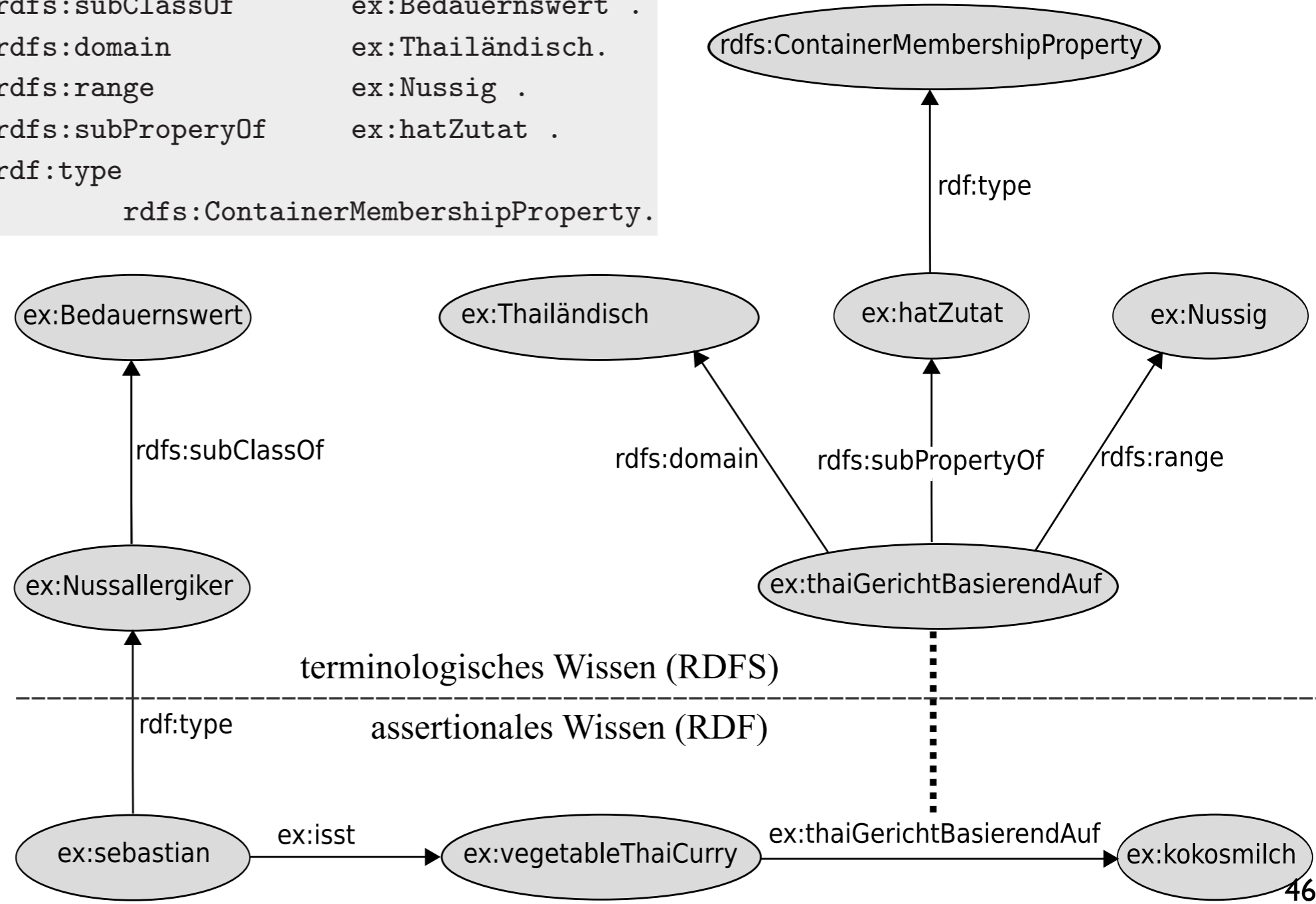
- mit den durch RDFS bereitgestellten Sprachmitteln können bestimmte Gegenstandsbereiche bereits in wichtigen Aspekten semantisch erfasst werden
- auf der Basis der speziellen Semantik von RDFS kann schon ein gewisses Maß impliziten Wissens geschlussfolgert werden
- mithin stellt RDFS eine (wenn auch noch vergleichsweise wenig ausdrucksstarke) Ontologiesprache dar

# EINFACHE ONTOLOGIEN - BEISPIEL

```

ex:VegetableThaiCurry    ex:ThaigerichtBasierendAuf    ex:Kokosmilch .
ex:Sebastian              rdf:type                                ex:Nussallergiker .
ex:Sebastian              ex:isst                                ex:VegetableTaiCurry .

ex:Nussallergiker         rdfs:subClassOf            ex:Bedauernswert .
ex:ThaigerichtBasierendAuf rdfs:domain                ex:Thailändisch .
ex:ThaigerichtBasierendAuf rdfs:range                 ex:Nussig .
ex:ThaigerichtBasierendAuf rdfs:subPropertyOf        ex:hatZutat .
ex:hatZutat               rdf:type                                rdfs:ContainerMembershipProperty .
    
```

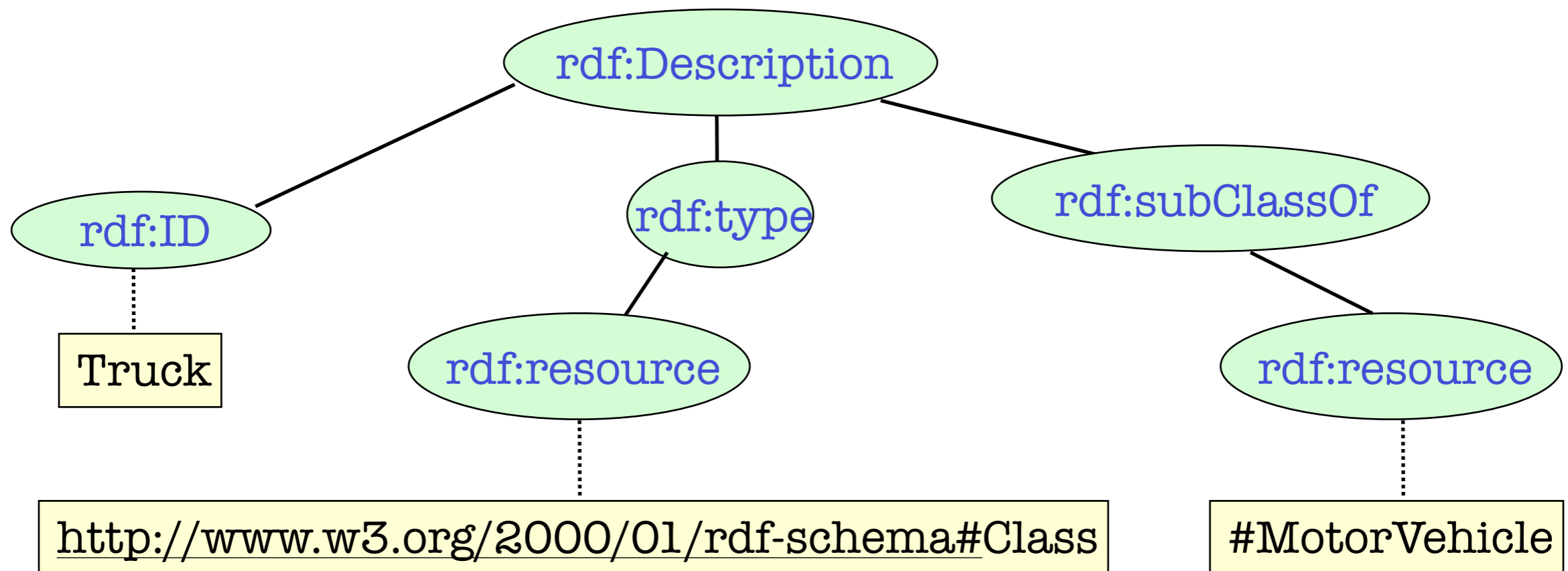


# 1 DOKUMENT - 3 INTERPRETATIONEN



```
<rdf:Description rdf:ID="Truck">  
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>  
  <rdfs:subClassOf rdf:resource="#MotorVehicle"/>  
</rdf:Description>
```

- Interpretation als XML:





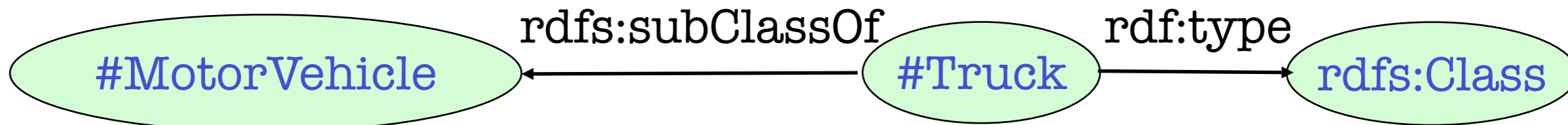
# 1 DOKUMENT - 3 INTERPRETATIONEN



```
<rdf:Description rdf:ID="Truck">  
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>  
  <rdfs:subClassOf rdf:resource="#MotorVehicle"/>  
</rdf:Description>
```

- Interpretation als RDF:
  - Anderes Datenmodell
  - `rdf:Description`, `rdf:ID` und `rdf:resource` haben eine festgelegte Bedeutung

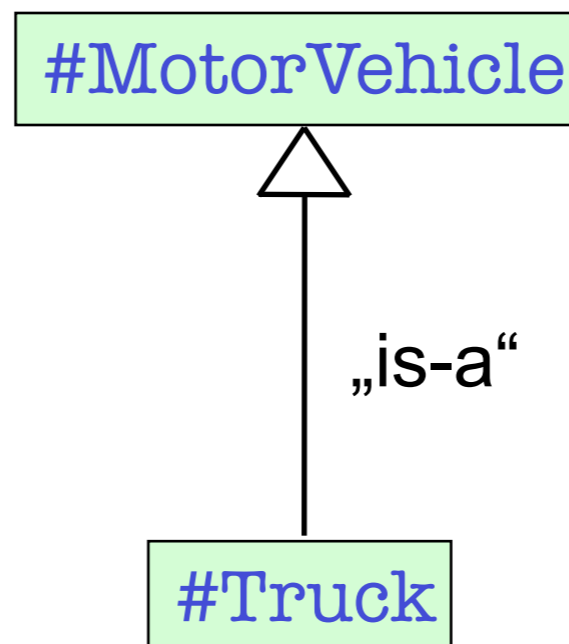
subject	predicate	object
1. #Truck	rdf:type	rdfs:Class
2. #Truck	rdfs:subClassOf	#MotorVehicle



# I DOKUMENT - 3 INTERPRETATIONEN

```
<rdf:Description rdf:ID="Truck">  
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>  
  <rdfs:subClassOf rdf:resource="#MotorVehicle"/>  
</rdf:Description>
```

- Interpretation als RDF Schema
  - Wieder anderes Datenmodell
  - `rdf:type` und `rdfs:subClassOf` werden speziell interpretiert



# LOGIK – GRUNDLAGEN

Dr. Sebastian Rudolph

Einleitung und Ausblick

XML und URIs

Einführung in RDF

RDF Schema

Logik - Grundlagen

Semantik von RDF(S)

OWL - Syntax und Intuition

OWL - Semantik und Reasoning

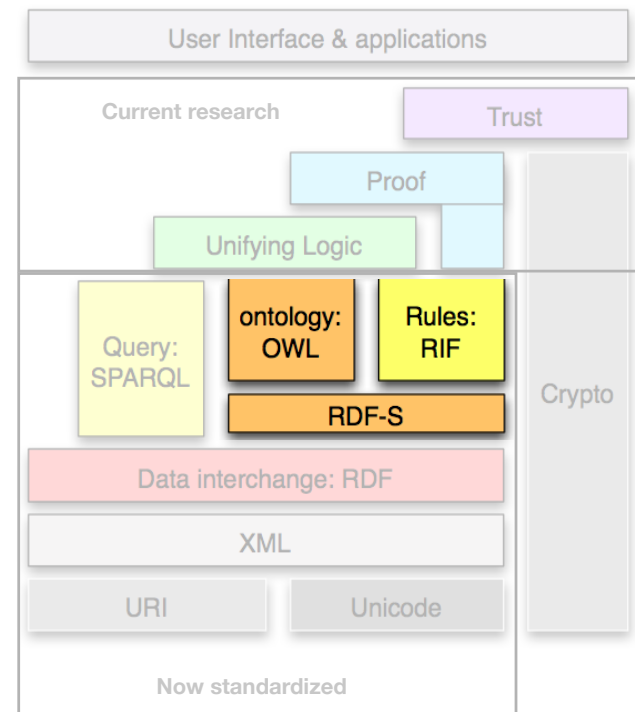
SPARQL - Syntax und Intuition

Semantik von SPARQL und konjunktive Anfragen

OWL 1.1 - Syntax und Semantik

Semantic Web und Regeln

## Semantic Web Architecture



# Was ist Logik?



etymologische Herkunft: griechisch  $\lambda\omicron\gamma\omicron\varsigma$   
bedeutet „Wort, Rede, Lehre“ (s.a. Faust I...)

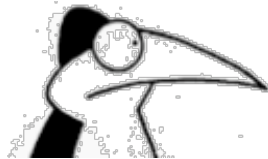
- Logik als Argumentation:



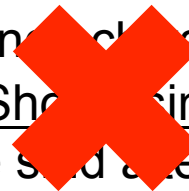
Alle Menschen sind sterblich.  
Sokrates ist ein Mensch.  
Also ist Sokrates sterblich.



Warum?



Alle Pinguine sind schwarz-weiß.  
Einige alte TV-Shows sind schwarz-weiß.  
Einige Pinguine sind alte TV-Shows.



- Definition für diese Vorlesung:  
*Logik ist die Lehre vom formal korrekten Schließen.*

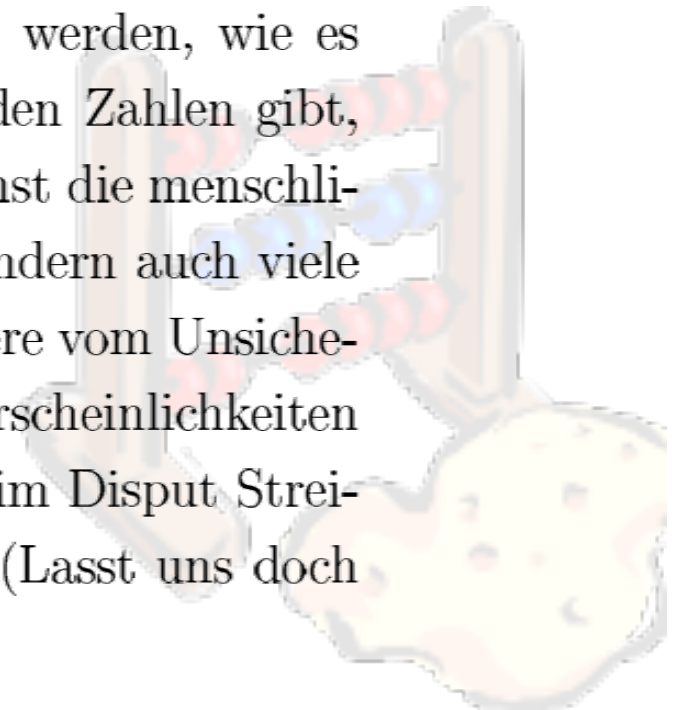
# Warum formal?

AIFB 



Automatisierbarkeit! Eine  
„Rechenmaschine“ für Logik!!  
G. W. Leibniz (1646-1716):

„alle menschlichen Schlussfolgerungen müssten auf irgendeine mit Zeichen arbeitende Rechnungsart zurückgeführt werden, wie es sie in der Algebra und Kombinatorik und mit den Zahlen gibt, wodurch nicht nur mit einer unzweifelhaften Kunst die menschliche Erfindungsgabe gefördert werden könnte, sondern auch viele Streitigkeiten beendet werden könnten, das Sichere vom Unsicheren unterschieden und selbst die Grade der Wahrscheinlichkeiten abgeschätzt werden könnten, da ja der eine der im Disput Streitenden zum anderen sagen könnte: *Calculemus* (Lasst uns doch nachrechnen).“



# Grundbegriffe der Logik



Interpretation  
Modell  
Erfüllbarkeit  
Ableitungsregel  
Folgerung  
Term  
Proposition  
Satz  
Domäne  
Atom  
Formel  
Entscheidbarkeit  
Deduktionskalkül  
Individuum  
Syntax  
Semantik  
Diskursuniversum  
Tautologie  
Modelltheorie  
Widerspruch

# Wie funktioniert Logik?



Alle Menschen sind sterblich.

Sokrates ist ein Mensch.

Also ist Sokrates sterblich.

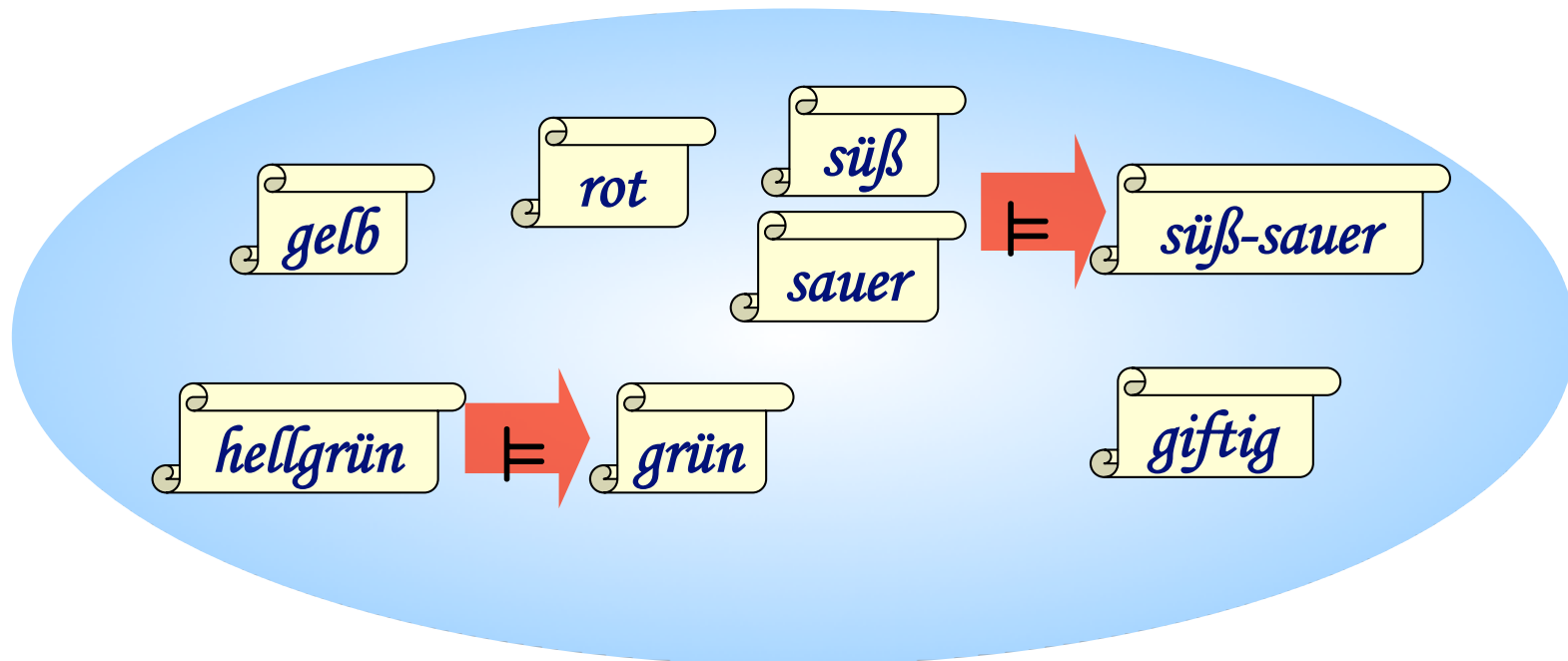
*Logik ist die Lehre vom formal korrekten Schließen.*

- Was schließen wir *woraus*?
- Beschreibende Grundelemente der Logik nennen wir **Sätze**.

# Wie funktioniert Logik? Sätze und Schlussfolgerungen



Jede Logik besteht aus einer Menge von **Sätzen** zusammen mit einer **Schlussfolgerungsrelation** (entailment relation). Letztere liefert die Semantik (grch. σημαντικός – *zum Zeichen gehörend*).





# Folgerung und Äquivalenz von Sätzen



Formal:  $L := (S, \models)$  mit  $\models \in 2^S \times S$

Dabei bedeutet für

- ⇒ eine Menge  $\Phi \subseteq S$  von Sätzen und
- ⇒ einen Satz  $\varphi \in S$

$$\Phi \models \varphi$$

„Aus den Sätzen  $\Phi$  folgt der Satz  $\varphi$ “ oder auch

„ $\varphi$  ist eine logische Konsequenz aus  $\Phi$ .“

Gilt für zwei Sätze  $\varphi$  und  $\psi$ , dass sowohl  $\{\varphi\} \models \psi$  als auch  $\{\psi\} \models \varphi$ , dann sind diese Sätze (*logisch äquivalent*) und man schreibt auch  $\psi \equiv \varphi$ .

# Wie funktioniert Logik? Syntax.

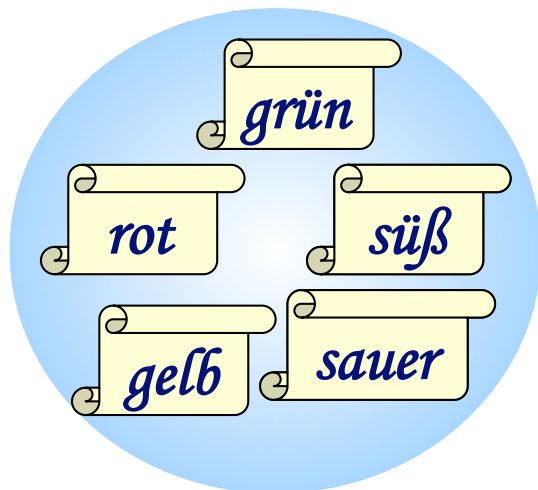


Syntax (von grch. συνταξις – *Zusammenstellung*, *Satzbau*) erschließt sich über die Frage

Was ist ein „richtiger“ Satz? D.h. wie wird die Menge der Sätze einer Logik definiert?

Nutzung von „Erzeugungsregeln“ zur Definition (Konstruktion) von wohlgeformten Sätzen, z.B.:

## Grundelemente:



**Syntax-Regel:** „Wenn  $\varphi$  und  $\psi$  Sätze sind, dann auch  $\varphi$ - $\psi$ “



*Konstruktor oder Junktor*

# Wie funktioniert Logik? Ausdrucksstärke.



Tradeoff: Logiken mit vielen Ausdrucksmitteln (Konstruktoren/Junktoren) sind:

- komfortabler in der Verwendung (verschiedene und komplexe Sachverhalte sind einfach auszudrücken), aber
- schwieriger (meta)mathematisch zu handhaben (Beweisen von Eigenschaften der Logik umständlicher).

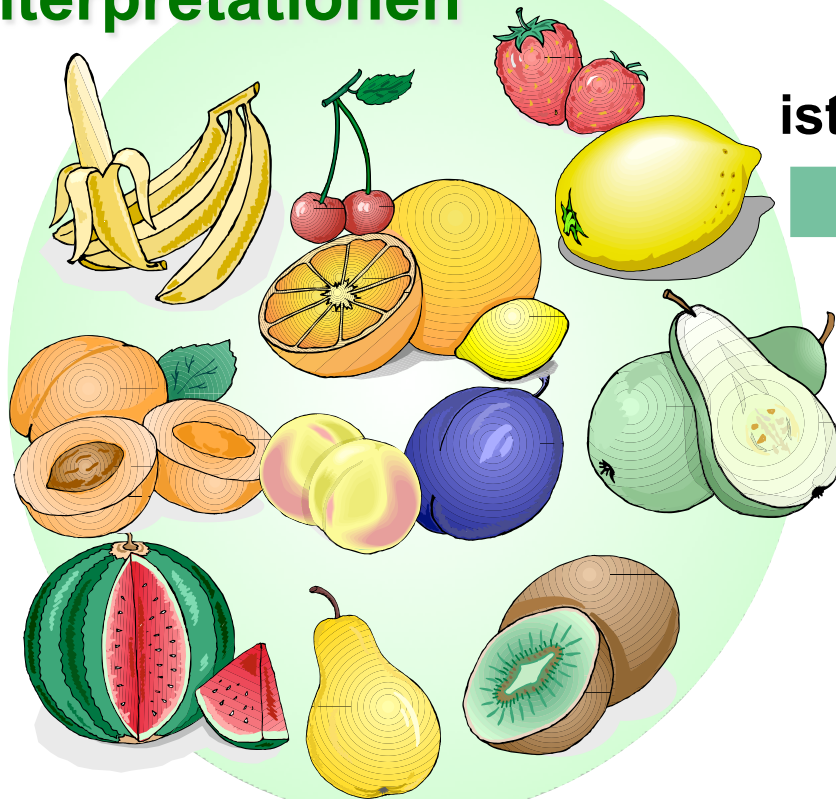
Möglicher Ausweg: Einschränkung der Sätze auf Teilmenge, die für jeden Satz der Logik einen logisch äquivalenten Vertreter enthält (vgl. Normalformen, minimale Junktorenmengen...) und Definition der anderen Sätze/Junktoren als „syntactic sugar“.

Wird eine Logik über dieses Maß hinaus eingeschränkt, erhält man ein *Fragment* der ursprünglichen Logik mit geringerer *Ausdrucksstärke*.

# Wie funktioniert Logik? - Modelltheorie

AIFB  Eine Möglichkeit, die **Schlussfolgerungsrelation** zu definieren besteht über **Interpretationen** bzw. **Modelle**.

## Interpretationen



ist Modell von

$\models$

## Sätze

*süß*

*sauer*

*süß-sauer*

*rot*

*hellgrün*

*gelb*

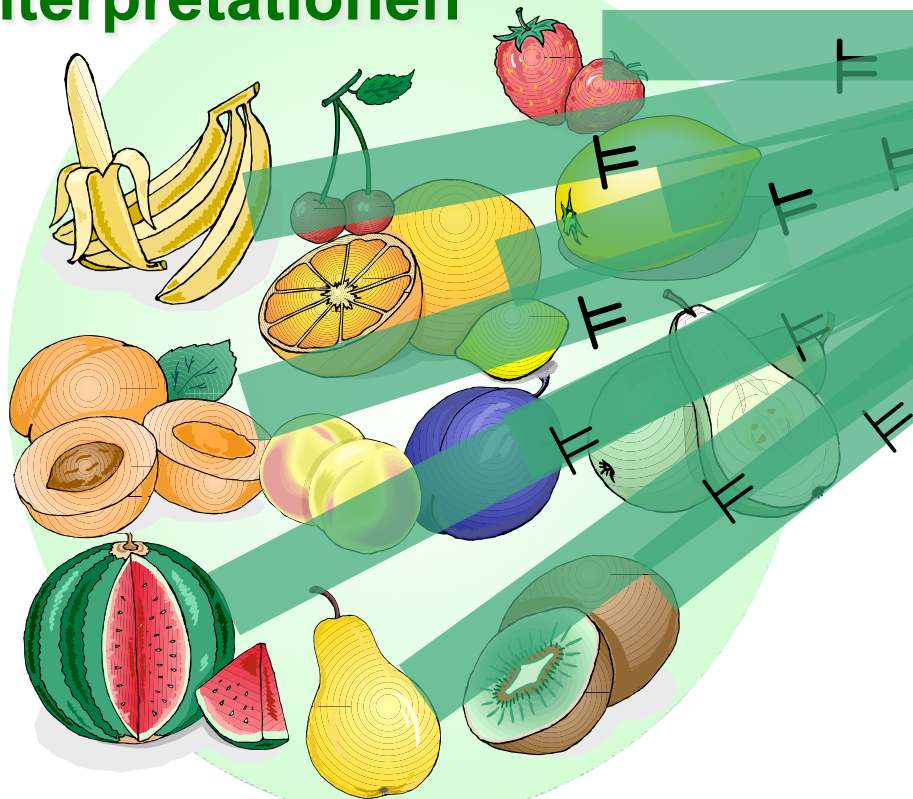
*grün*

*giftig*

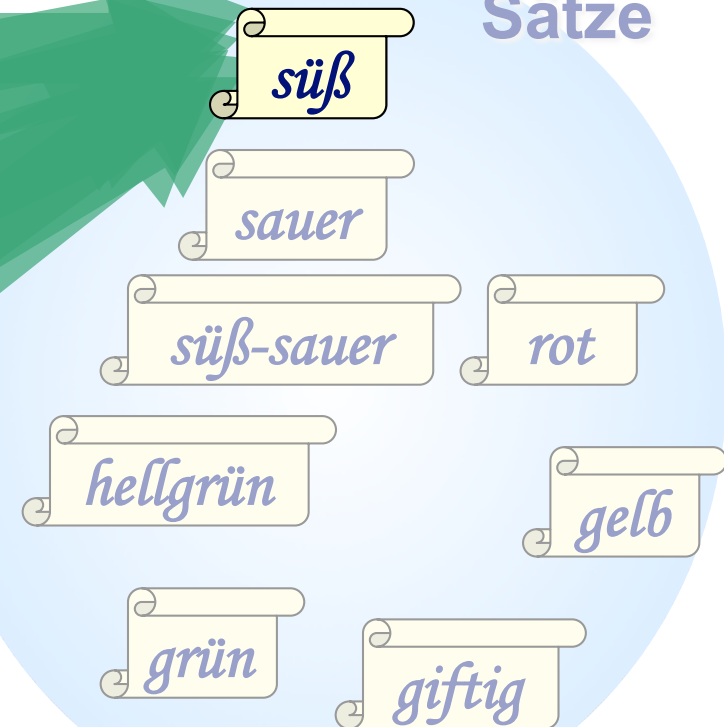
# Wie funktioniert Logik? - Modelltheorie

**AIFB**  Sätze, für die **jede** Interpretation ein Modell ist, heißen *allgemeingültig* oder *Tautologien* (grch. ταυτολογία).


**Interpretationen**



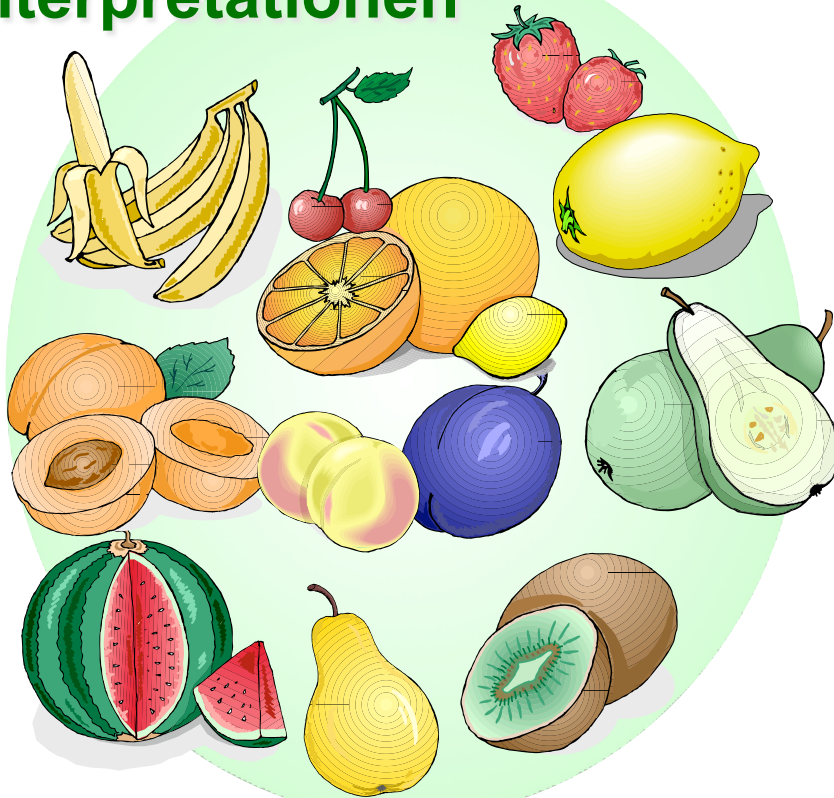
**Sätze**



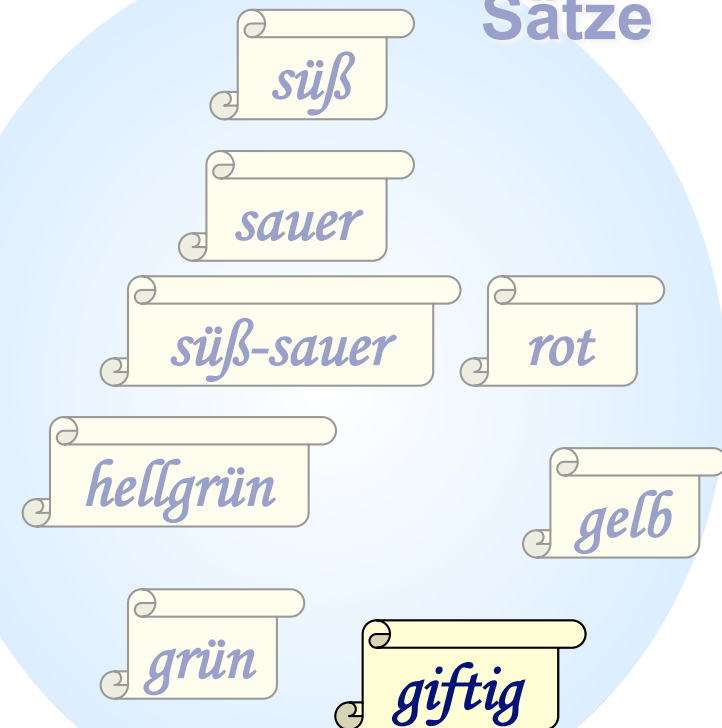
# Wie funktioniert Logik? - Modelltheorie

AIFB  Sätze, für die **keine** Interpretation ein Modell ist, heißen *widersprüchlich* oder *unerfüllbar*.

## Interpretationen



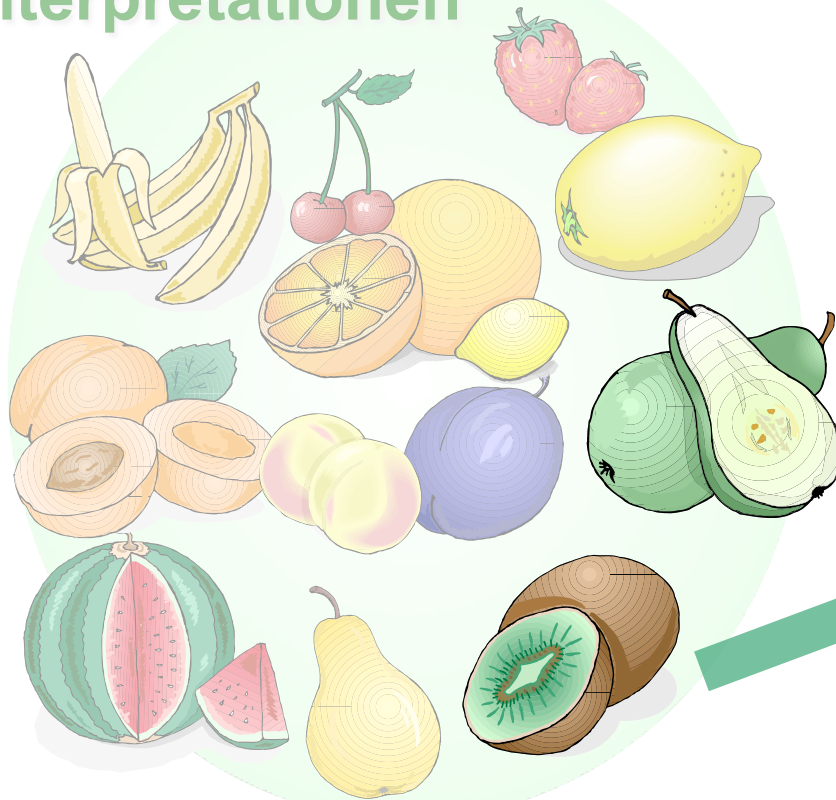
## Sätze



# Wie funktioniert Logik? - Modelltheorie

**AIFB**  Sätze, die (mindestens) ein Modell haben, heißen *erfüllbar*.

## Interpretationen

 $\models$  $\models$ 

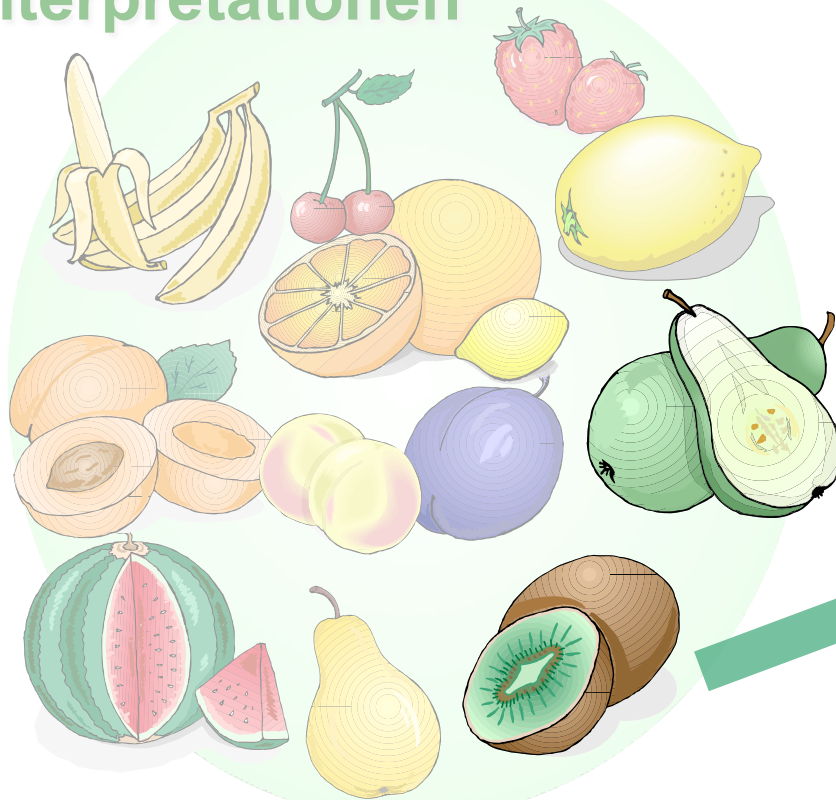
## Sätze

*süß**sauer**süß-sauer**rot**hellgrün**gelb**grün**giftig*

# Wie funktioniert Logik? - Modelltheorie

AIFB  Eine Möglichkeit, die **Schlussfolgerungsrelation** zu definieren besteht über **Interpretationen** bzw. **Modelle**.

## Interpretationen

 $\models$  $\models$ 

## Sätze

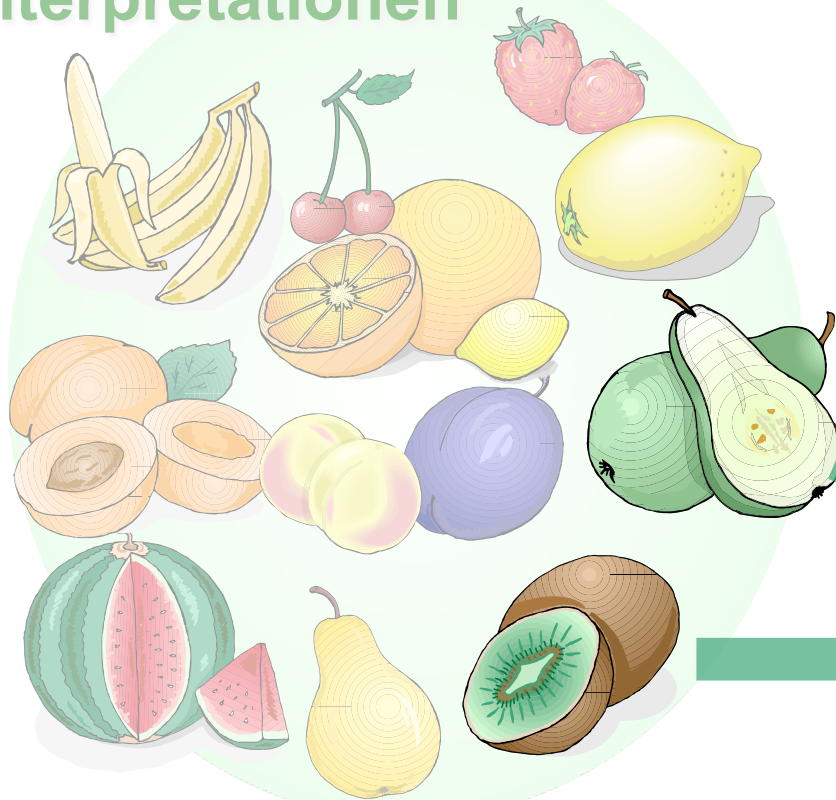
*süß**sauer**süß-sauer**rot**hellgrün**gelb**grün**giftig*



# Wie funktioniert Logik? - Modelltheorie

AIFB  Eine Möglichkeit, die **Schlussfolgerungsrelation** zu definieren besteht über **Interpretationen** bzw. **Modelle**.

## Interpretationen



## Sätze



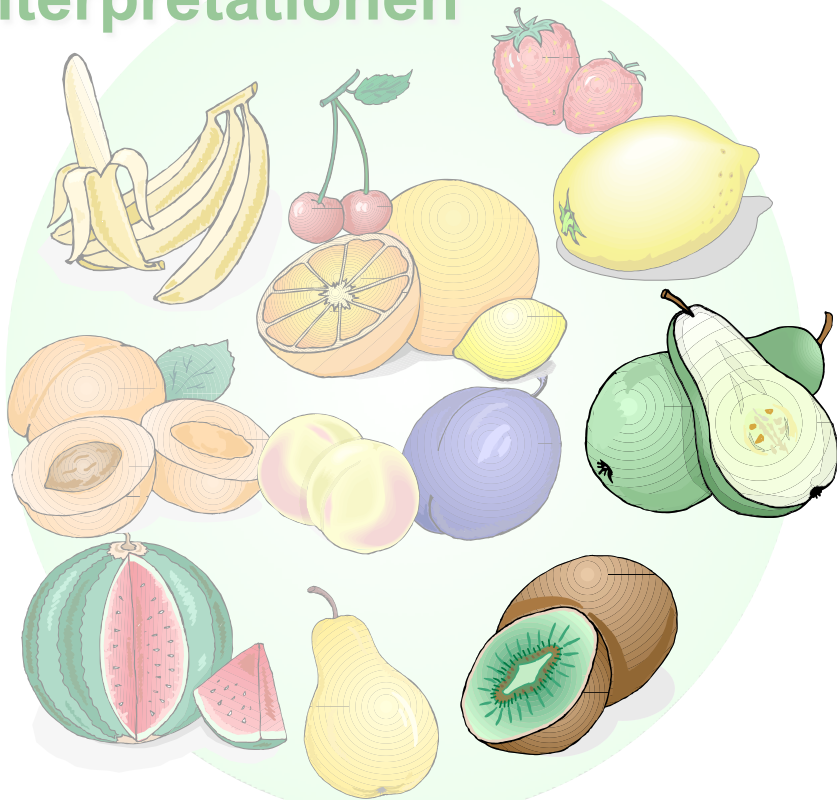
$\models$

$\models$

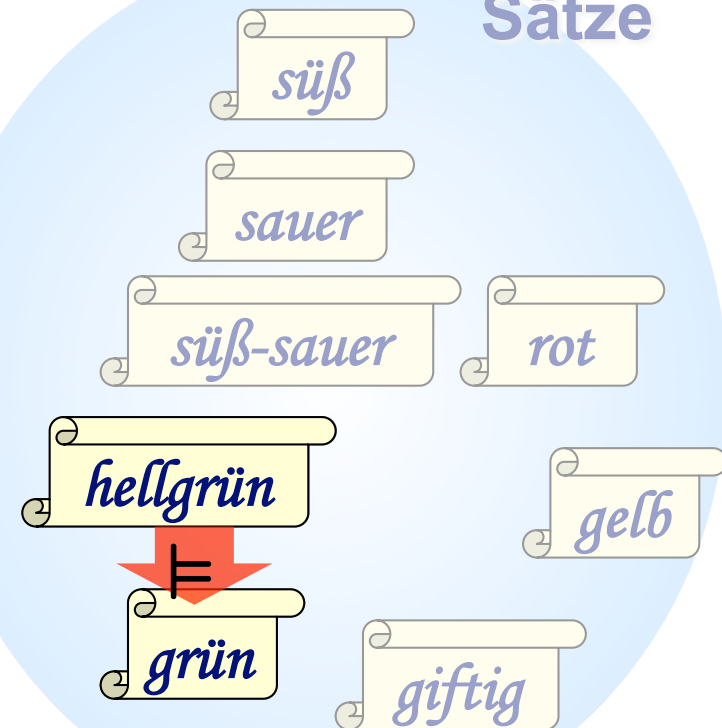
# Wie funktioniert Logik? - Modelltheorie

AIFB  Eine Möglichkeit, die **Schlussfolgerungsrelation** zu definieren besteht über **Interpretationen** bzw. **Modelle**.

## Interpretationen



## Sätze



# Wie funktioniert Logik?

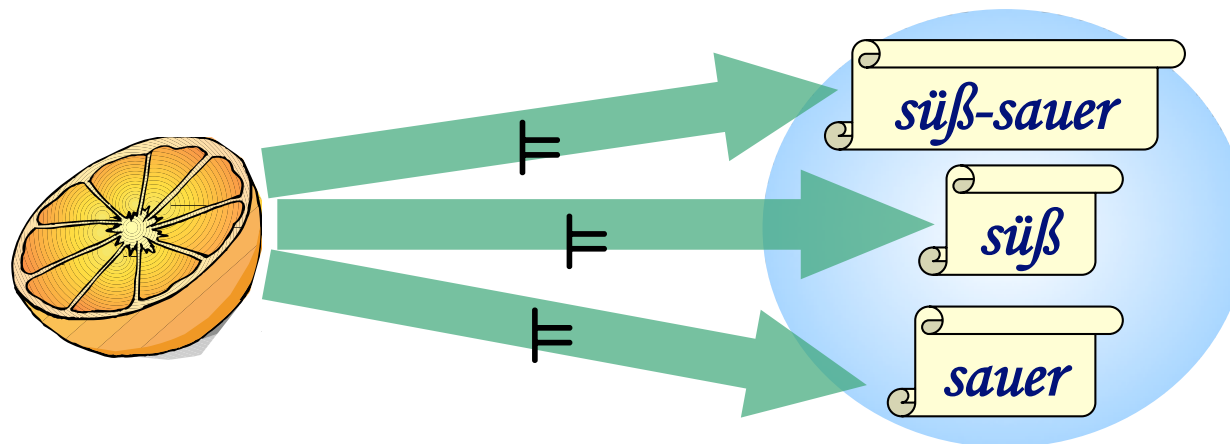
## Semantik entlang der Syntax



Häufiges Prinzip bei Definition von Interpretationen:

- Interpretation von Grundelementen wird festgelegt
- Interpretation von zusammengesetzten (konstruierten) Sätzen wird auf die Interpretation der Teile zurückgeführt, z.B.:

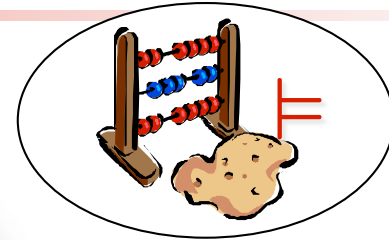
**Semantik-Regel:** „Die Modelle von  $\varphi$ - $\psi$  sind genau die Interpretationen, die Modelle sowohl von  $\varphi$  als auch von  $\psi$  sind.“



# Beweistheorie

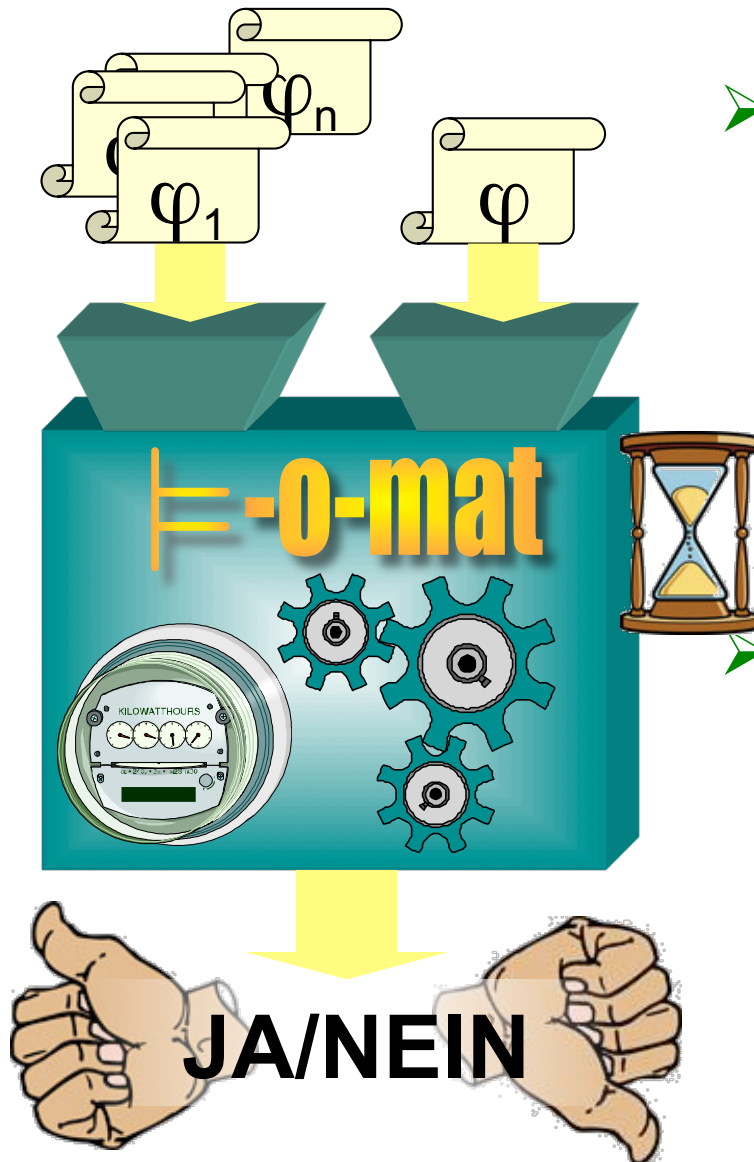


- Zurück zu Leibniz:  
Rechenmaschine  
für Logik



- Aber: Möglichkeit, direkt mit allen möglichen Interpretationen zu arbeiten, oft eingeschränkt
- Daher: Versuch, Schlussfolgerungsrelation durch rein syntaktische Verfahren zu beschreiben/berechnen

# Entscheidungsverfahren/Entscheidbarkeit

AIFB 

➤ Entscheidungsalgorithmus:

⇒ input: Menge  $\{\varphi_1, \dots, \varphi_n\}$  von Sätzen und Satz  $\varphi$

⇒ terminiert nach endlicher Zeit

⇒ output:

✧ „Ja“, falls  $\{\varphi_1, \dots, \varphi_n\} \models \varphi$

✧ „Nein“ sonst

➤ Gibt es einen solchen Algorithmus für eine Logik, dann nennt man sie *entscheidbar*.

# Aufzählungsverfahren/Semientscheidbarkeit

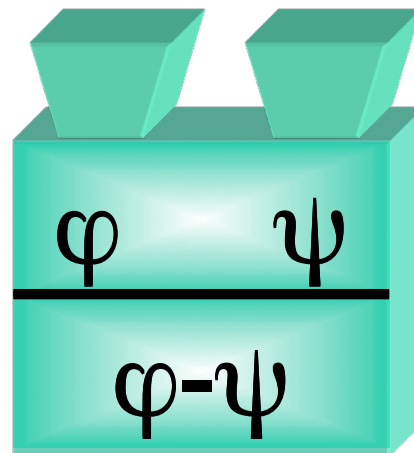
AIFB 

# Deduktionskalkül

## AIFB

- kann gesehen werden als spezielle Form eines Aufzählungsverfahrens
- besteht aus *Ableitungsregeln*, z.B.:

$\{\varphi, \psi, \omega, \dots\}$



# Deduktionskalkül



Ein Satz  $\varphi$  ist aus einer Menge  $\Phi$  von Sätzen *ableitbar* (geschrieben:  $\Phi \vdash \varphi$ ), wenn sich  $\varphi$  durch wiederholtes Anwenden der Ableitungsregeln eines Deduktionskalküls aus  $\Phi$  „erzeugen“ lässt.

Deduktionskalkül ist *korrekt* (engl. *sound*), wenn aus  $\Phi \vdash \varphi$  immer  $\Phi \models \varphi$  folgt, d.h. alle ableitbaren Schlüsse auch wirklich logisch folgen.

Deduktionskalkül ist *vollständig* (engl. *complete*), wenn aus  $\Phi \models \varphi$  immer  $\Phi \vdash \varphi$  folgt, d.h. alle logischen Konsequenzen auch abgeleitet werden können.

In einem korrekten und vollständigen Deduktionskalkül gilt:

$$\models = \vdash$$

und man kann es als Aufzählungsverfahren verwenden.  
Achtung! Es gibt Logiken, für die nachweislich kein solches Deduktionskalkül existiert (Gödel 1931).





## Weitere interessante Eigenschaften von Logiken:

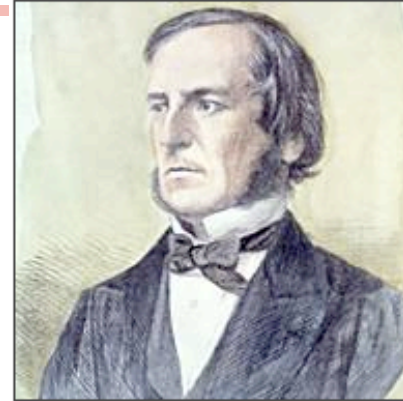
AIFB 

- Monotonie
- Kompaktheit
- Algorithmische Komplexität für Entscheidungsverfahren
- ...und jede Menge anderes...

# Aussagenlogik

## AIFB

- auch: *propositionale* Logik  
*boolesche* Logik
- schon bei den Stoikern voll  
ausgearbeitete Junktorenlogik
- George Boole (1815 – 1864)  
„*An Investigation of the Laws of Thought*“ (1854)
- syntaktische Grundelemente:  
atomare Sätze / Propositionen / Aussagen  
( $p, q, \dots, p_1, p_2, \dots$ )
- Können als natürlichsprachliche Aussagen gedacht  
werden: „Es regnet.“ ...





- Erzeugungsregeln für Sätze:
  - ⇒ alle atomaren Propositionen sind Sätze ( $p, q, \dots$ )
  - ⇒ ist  $\varphi$  ein Satz, dann auch  $\neg\varphi$
  - ⇒ sind  $\varphi$  und  $\psi$  Sätze, dann auch  $(\varphi \wedge \psi)$ ,  $(\varphi \vee \psi)$ ,  $(\varphi \rightarrow \psi)$  und  $(\varphi \leftrightarrow \psi)$
- Klammern können ggf. weggelassen werden;  
Präzedenzen (bei uns):  $\neg$  vor  $\wedge, \vee$  vor  $\rightarrow, \leftrightarrow$ .
- Zusätzliche Klammern machen es trotzdem oft lesbarer...

# Aussagenlogik – Syntax



<i>Junktor</i>	<i>Name</i>	<i>Intuitive Bedeutung</i>
$\neg$	Negation	„nicht“
$\wedge$	Konjunktion	„und“
$\vee$	Disjunktion	„oder“
$\rightarrow$	Implikation	„wenn – dann“
$\leftrightarrow$	Äquivalenz	„genau dann, wenn“

## *Einfache Aussagen*

## *Modellierung*

Es regnet.

r

Die Straße wird nass.

n

Die Sonne ist grün

g

## *Zusammengesetzte Aussagen*

## *Modellierung*

Wenn es regnet, dann wird die Straße nass.

$r \rightarrow n$

Wenn es regnet, und die Straße nicht nass wird,  
dann ist die Sonne grün.

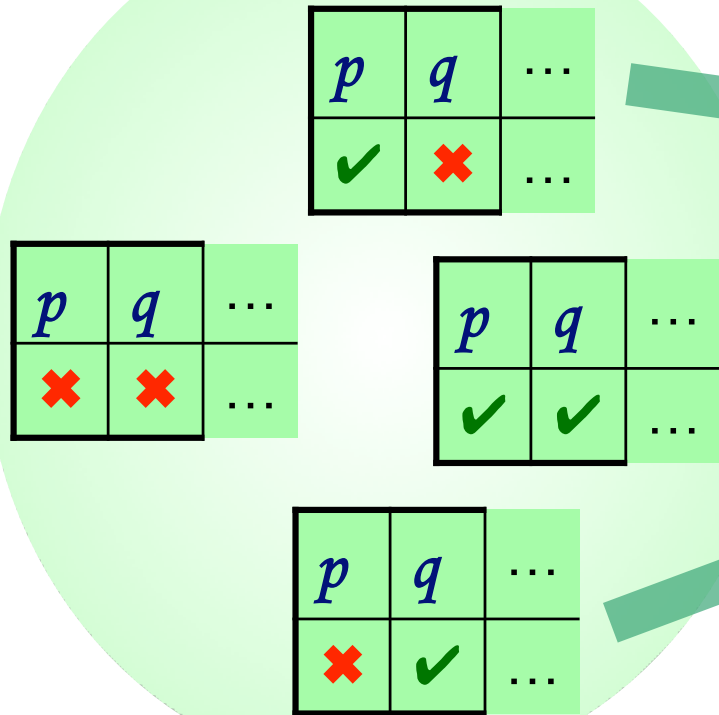
$(r \wedge \neg n) \rightarrow g$

# Aussagenlogik – Modelltheoretische Semantik

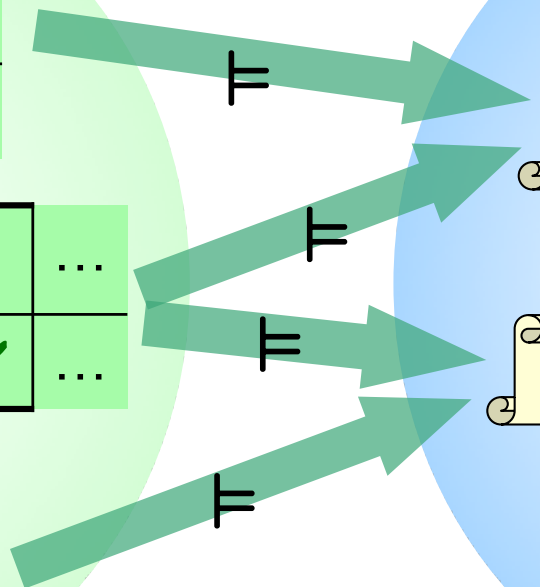
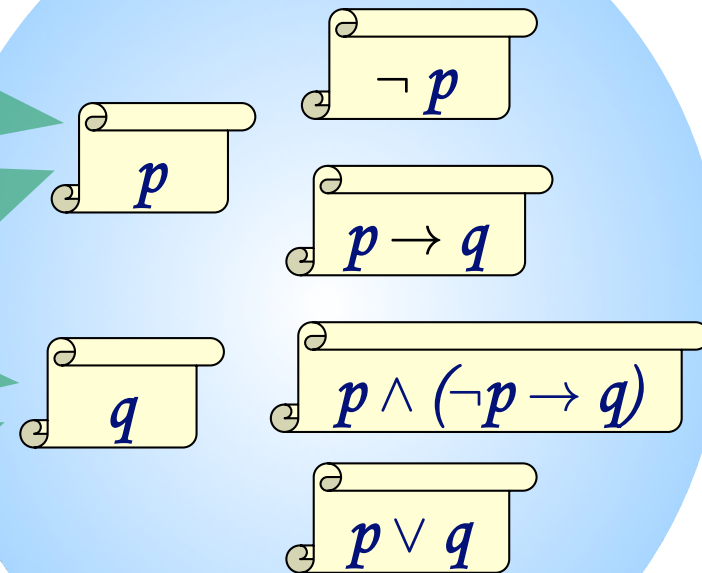


Was sind die Modelle der Aussagenlogik?

## Interpretationen



## Sätze





- Formal: Interpretationen  $I$  sind Abbildungen von der Menge der atomaren Propositionen in die Menge {wahr, falsch}, d.h. jeder dieser Propositionen  $p$  wird ein Wahrheitswert  $WW_I(p)$  zugeordnet.
- Daraus bestimmt man Modelle für zusammengesetzte Sätze über

## Semantik-Regeln

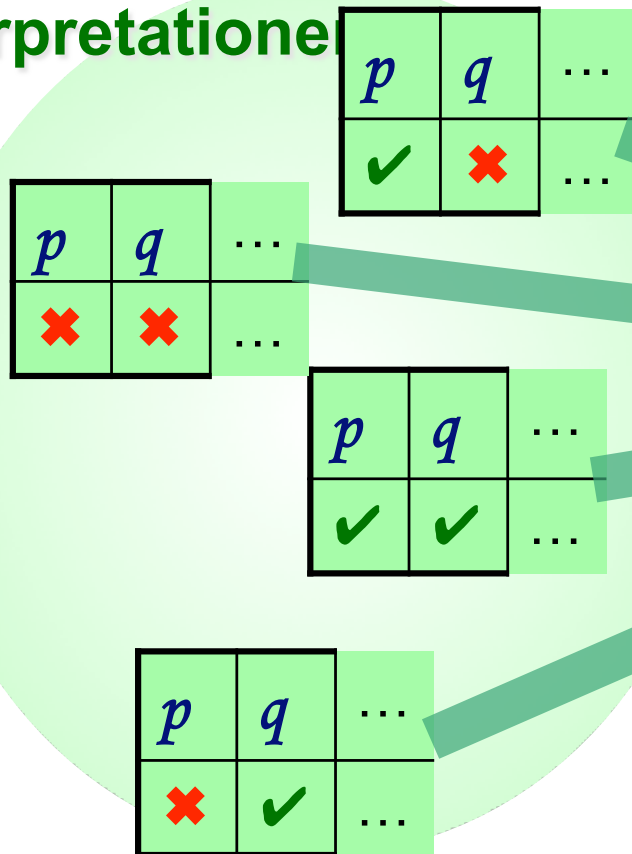
- ⇒  $I$  Modell von  $\neg\varphi$  genau dann, wenn  $I$  **kein** Modell von  $\varphi$
- ⇒  $I$  Modell von  $(\varphi \wedge \psi)$  genau dann, wenn  $I$  Modell von  $\varphi$  **und** von  $\psi$
- ⇒  $I$  Modell von  $(\varphi \vee \psi)$  genau dann, wenn  $I$  Modell von  $\varphi$  **oder** von  $\psi$
- ⇒  $I$  Modell von  $(\varphi \rightarrow \psi)$  genau dann, wenn  $I$  **kein** Modell von  $\varphi$  **oder**  $I$  Modell von  $\psi$
- ⇒  $I$  Modell von  $(\varphi \leftrightarrow \psi)$  genau dann, wenn  $I$  Modell für **jeden oder keinen** der beiden Sätze ist.

# Aussagenlogik – Modelltheoretische Semantik

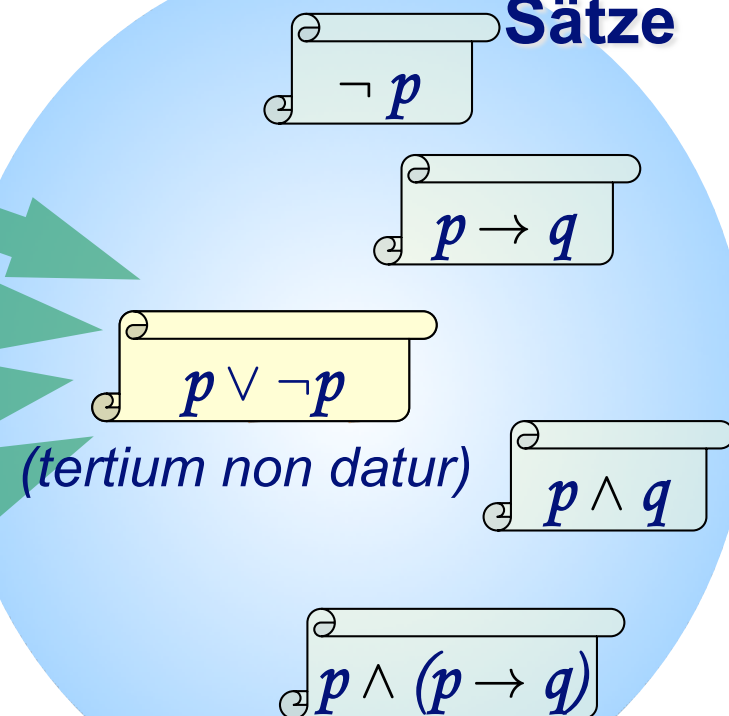
AIFB 

Beispiel für Tautologie in der Aussagenlogik.

Interpretationen



Sätze

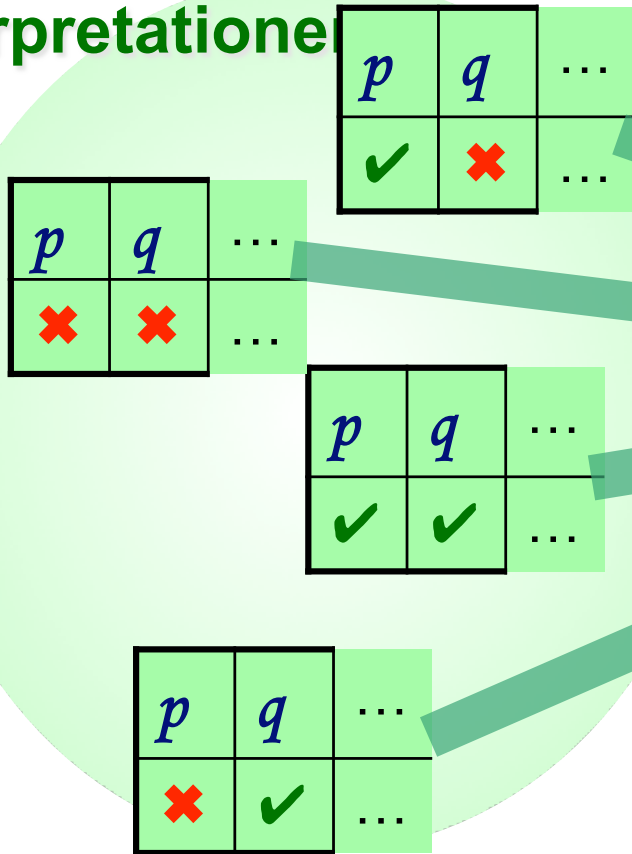


# Aussagenlogik – Modelltheoretische Semantik

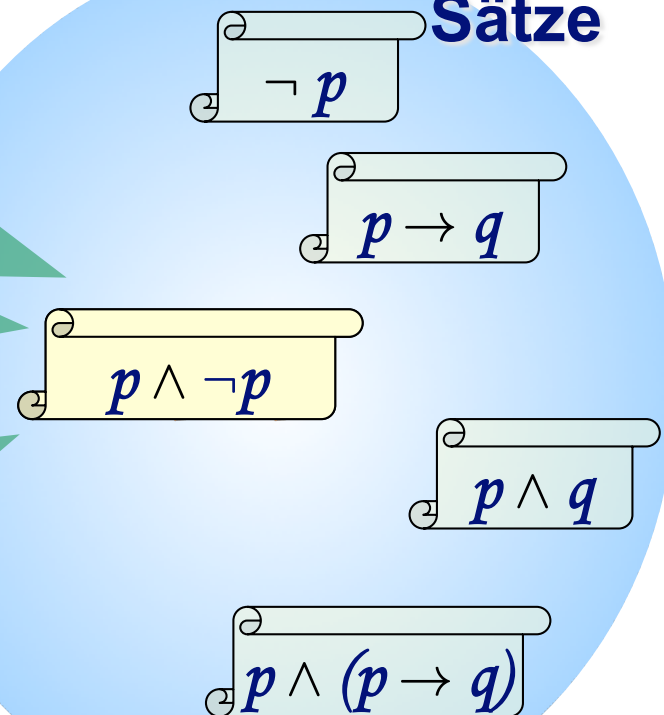
AIFB 

Beispiel für Kontradiktion in der Aussagenlogik.

Interpretationen



Sätze





# Aussagenlogik – einige logische Äquivalenzen



$$\varphi \wedge \psi \equiv \psi \wedge \varphi$$

$$\varphi \vee \psi \equiv \psi \vee \varphi$$

$$\varphi \wedge (\psi \wedge \omega) \equiv (\varphi \wedge \psi) \wedge \omega$$

$$\varphi \vee (\psi \vee \omega) \equiv (\varphi \vee \psi) \vee \omega$$

$$\varphi \wedge \varphi \equiv \varphi$$

$$\varphi \vee \varphi \equiv \varphi$$

$$\varphi \wedge (\psi \vee \varphi) \equiv \varphi$$

$$\varphi \vee (\psi \wedge \varphi) \equiv \varphi$$

$$\varphi \rightarrow \psi \equiv \neg \varphi \vee \psi$$

$$\varphi \leftrightarrow \psi \equiv (\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi)$$

$$\neg(\varphi \wedge \psi) \equiv \neg\varphi \vee \neg\psi$$

$$\neg(\varphi \vee \psi) \equiv \neg\varphi \wedge \neg\psi$$

$$\neg\neg\varphi \equiv \varphi$$

$$\varphi \vee (\psi \wedge \omega) \equiv (\varphi \vee \psi) \wedge (\varphi \vee \omega)$$

$$\varphi \wedge (\psi \vee \omega) \equiv (\varphi \wedge \psi) \vee (\varphi \wedge \omega)$$

# Aussagenlogik – Normalformen & vollständige Junktoren



aus diesen Äquivalenzen folgt:

- zu jeder Formel gibt es eine logisch äquivalente Formel, die nur die Junktoren  $\wedge$  und  $\neg$  enthält.
- zu jeder Formel gibt es eine Formel in konjunktiver Normalform, d.h.
  - ⇒ nur einfache Negation direkt vor atomaren Propositionen (sog. Literale)
  - ⇒ Formel ist Konjunktion von Disjunktionen von Literalen
  - ⇒ Bsp.:  $(p \vee \neg q \vee r \vee \neg s) \wedge (\neg p \vee q \vee s) \wedge (q \vee \neg r \vee s)$

# Aussagenlogik – Entscheidungsalgorithmus



- Aussagenlogik ist entscheidbar
- nützliche Eigenschaft dabei:  
 $\{\varphi_1, \dots, \varphi_n\} \models \varphi$  gilt genau dann, wenn  $(\varphi_1 \wedge \dots \wedge \varphi_n) \rightarrow \varphi$  eine Tautologie ist
- Entscheidung, ob Satz Tautologie ist, über Wahrheitstabelle
- im Prinzip: Überprüfung aller Interpretationen (nur die Wahrheitswerte der vorkommenden atomaren Propositionen fallen ins Gewicht)

# Aussagenlogik – Entscheidungsalgorithmus



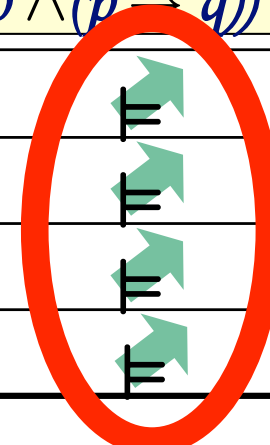
Modus Ponens:

$$\{ \text{p}, \text{p} \rightarrow \text{q} \} \models \text{q}$$

$$\models (\text{p} \wedge (\text{p} \rightarrow \text{q})) \rightarrow \text{q}$$



<i>p</i>	<i>q</i>	...	<i>p</i> → <i>q</i>	<i>p</i> ∧ ( <i>p</i> → <i>q</i> )	( <i>p</i> ∧ ( <i>p</i> → <i>q</i> )) → <i>q</i>
×	×	...	⇒	⊘	⇒
×	✓	...	⇒	⊘	⇒
✓	×	...	⊘	⊘	⇒
✓	✓	...	⇒	⇒	⇒



# SEMANTIK VON RDF(S)

Dr. Sebastian Rudolph

Einleitung und Ausblick

XML und URIs

Einführung in RDF

RDF Schema

Logik - Grundlagen

Semantik von RDF(S)

OWL - Syntax und Intuition

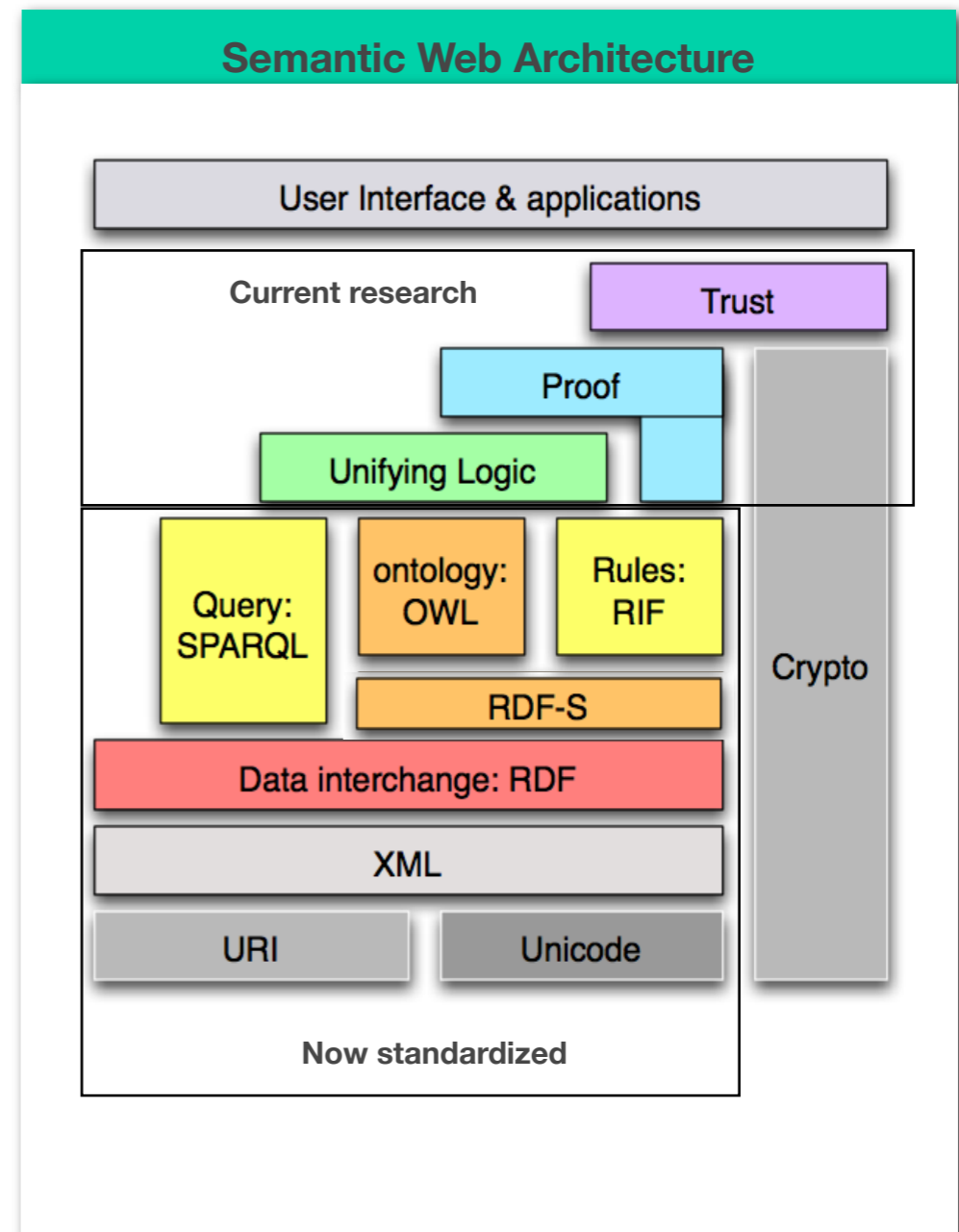
OWL - Semantik und Reasoning

SPARQL - Syntax und Intuition

Semantik von SPARQL und konjunktive Anfragen

OWL 1.1 - Syntax und Semantik

Semantic Web und Regeln



# SEMANTIK VON RDF(S)

Dr. Sebastian Rudolph

Einleitung und Ausblick

XML und URIs

Einführung in RDF

RDF Schema

Logik - Grundlagen

**Semantik von RDF(S)**

OWL - Syntax und Intuition

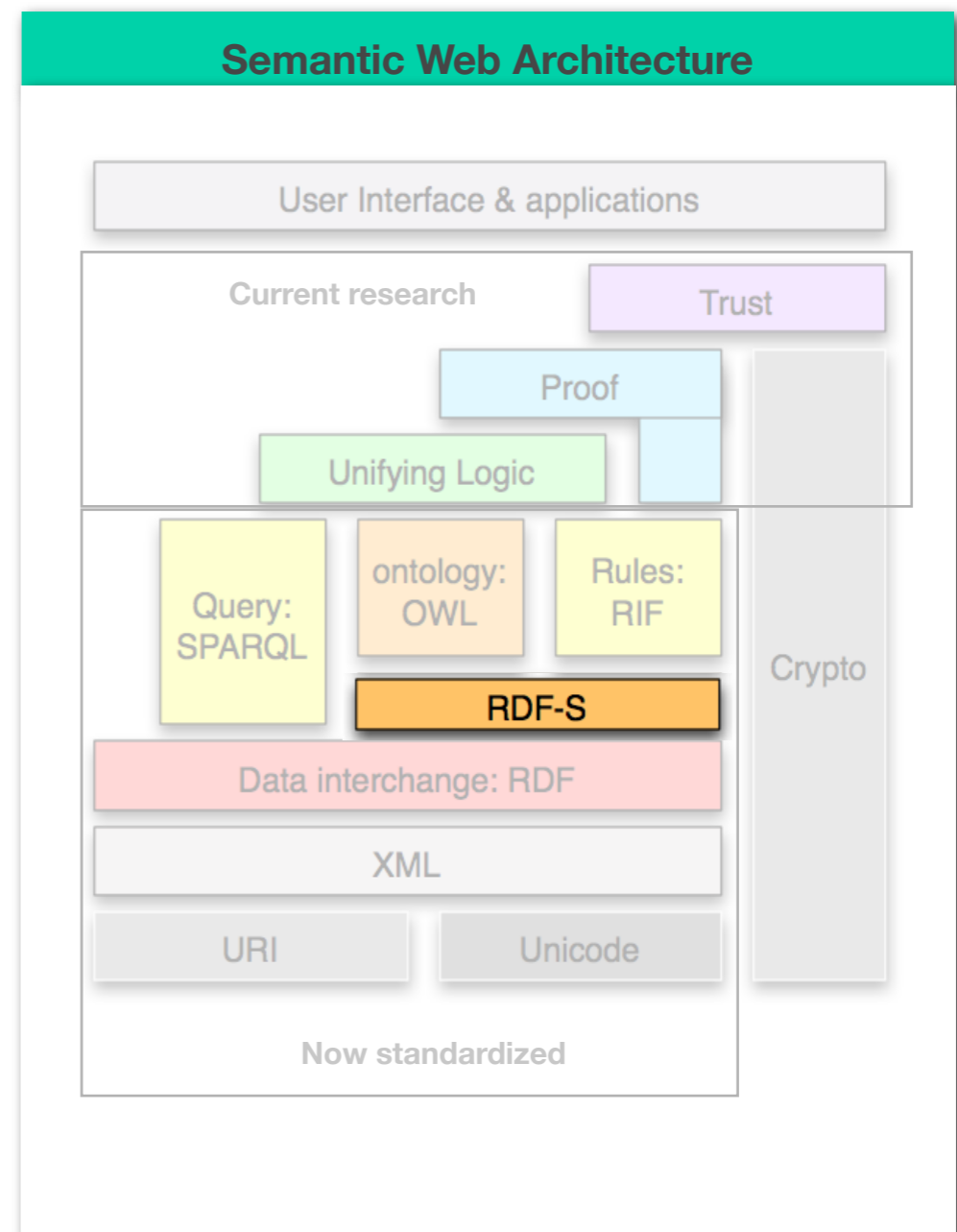
OWL - Semantik und Reasoning

SPARQL - Syntax und Intuition

Semantik von SPARQL und konjunktive Anfragen

OWL 1.1 - Syntax und Semantik

Semantic Web und Regeln



# AGENDA



- Motivation
- Vorbetrachtungen
- einfache Folgerung
- RDF-Folgerung
- RDFS-Folgerung
- Unzulänglichkeiten von RDF(S)

# AGENDA



- Motivation
- Vorbetrachtungen
- einfache Folgerung
- RDF-Folgerung
- RDFS-Folgerung
- Unzulänglichkeiten von RDF(S)



# WARUM FORMALE SEMANTIK?

- nach Einführung von RDFS Kritik von Tool-Herstellern: verschiedene Tools - Inkompatibilitäten (trotz Spezifikation)
- z.B. bei triple stores:
  - gleiches RDF-Dokument
  - gleiche SPARQL-Anfrage
  - verschiedene Antworten
- daher: modelltheoretische Semantik für RDF(S)

# AGENDA



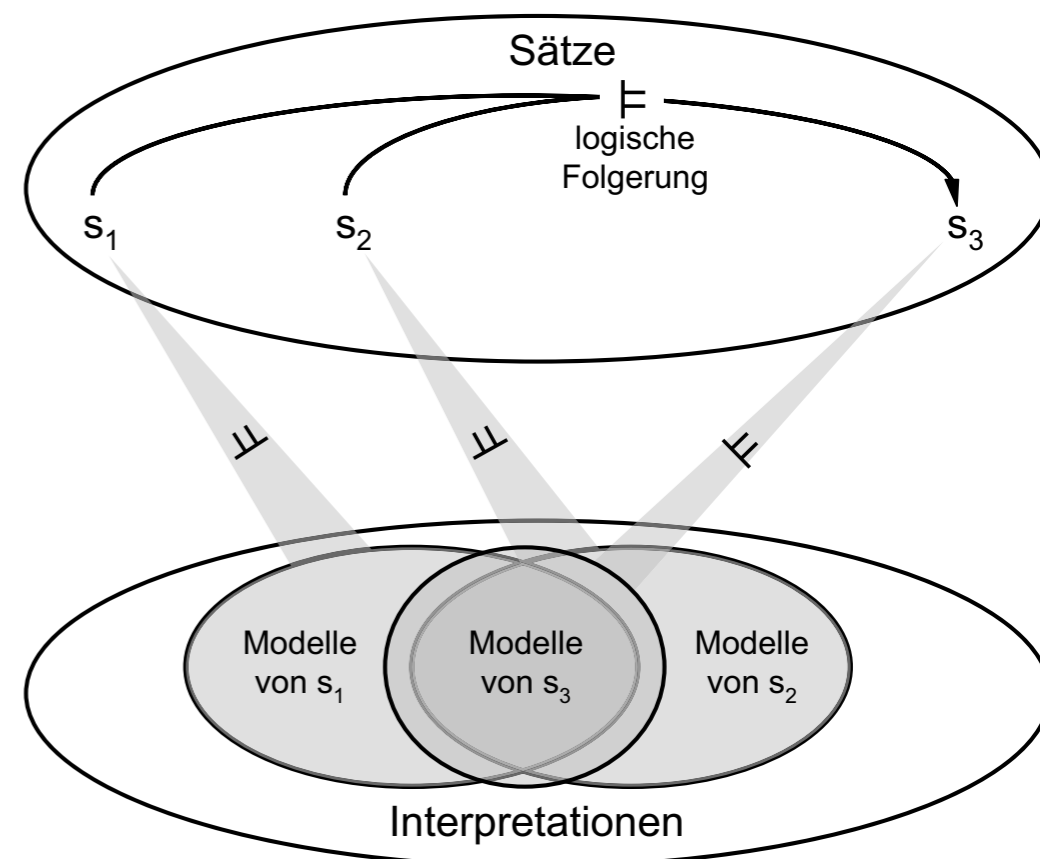
- Motivation
- Vorbetrachtungen
- einfache Folgerung
- RDF-Folgerung
- RDFS-Folgerung
- Unzulänglichkeiten von RDF(S)

# WAS IST DIE SYNTAX?

- also: was sind die Sätze in RDF(S)?
  - Grundelemente (Vokabular  $V$ ): URIs, bnodes und Literale (sind selbst keine Sätze)
  - jedes Tripel  
 $(s,p,o) \in$   
 $(\text{URI} \cup \text{bnode}) \times \text{URI} \times (\text{URI} \cup \text{bnode} \cup \text{Literal})$   
ist ein Satz
  - jede endliche Menge von Tripeln (genannt Graph)  
ist ein Satz

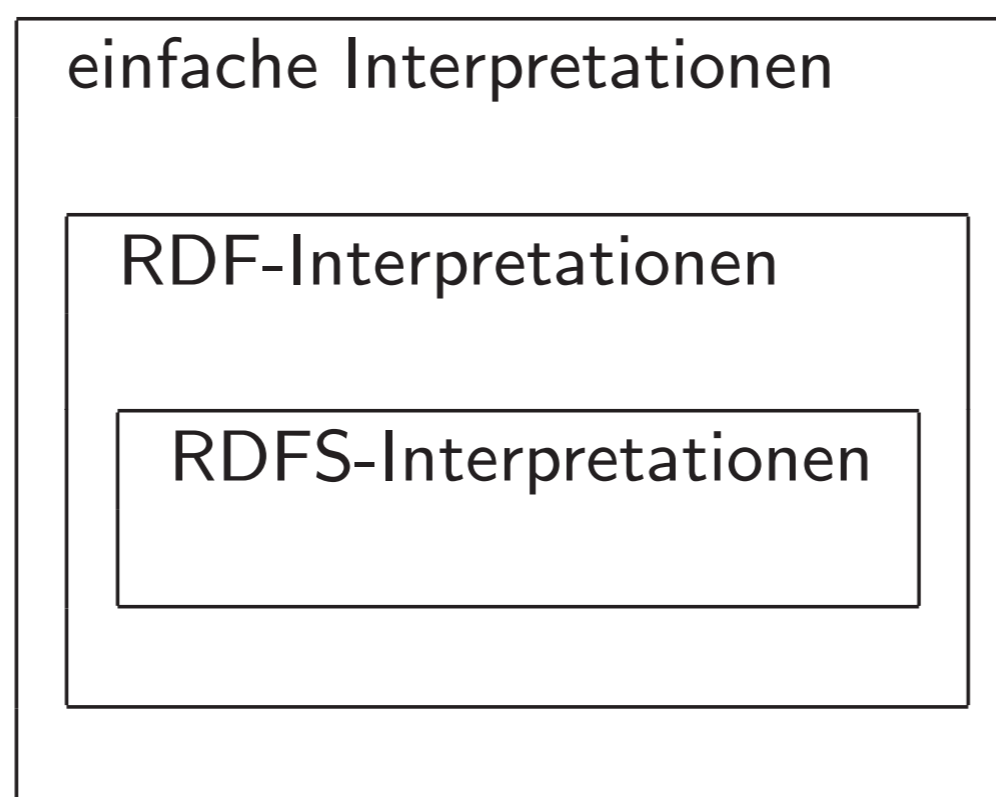
# WAS IST DIE SEMANTIK?

- Konsequenzrelation, die sagt, wann ein RDF(S)-Graph  $G'$  aus einem RDF(S)-Graphen  $G$  folgt, d.h.  $G \models G'$
- Modelltheoretische Semantik: wir definieren Menge von Interpretationen und legen fest, wann eine Interpretation Modell eines Graphen ist



# WAS IST DIE SEMANTIK?

- Vorgehen schrittweise:



- je eingeschränkter die Interpretationen umso stärker die Folgerungsrelation

# AGENDA



- Motivation
- Vorbetrachtungen
- einfache Folgerung
- RDF-Folgerung
- RDFS-Folgerung
- Unzulänglichkeiten von RDF(S)

# SEMANTIK DER EINFACHEN FOLGERUNG



- einfache Interpretation:

Wir definieren also: Eine *einfache Interpretation*  $\mathcal{I}$  für ein Vokabular  $V$  besteht aus

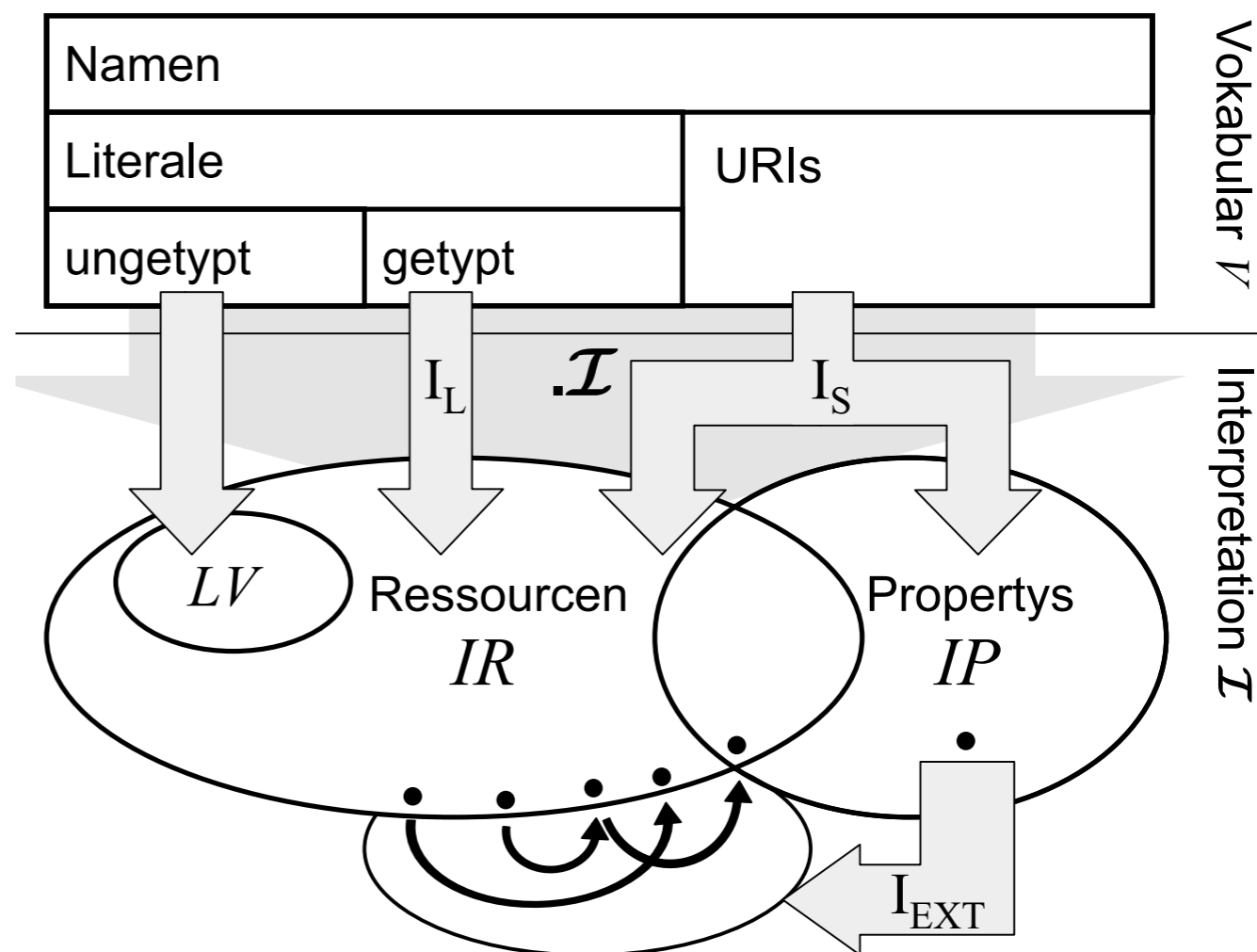
- $IR$ , einer nichtleeren Menge von *Ressourcen*, auch genannt Domäne oder (Diskurs-)Universum von  $\mathcal{I}$ ,
- $IP$ , der Menge der *Propertys* von  $\mathcal{I}$
- $I_{\text{EXT}}$ , einer Funktion, welche jeder Property eine Menge von Paaren aus  $IR$  zuordnet, also  $I_{\text{EXT}} : IP \rightarrow 2^{IR \times IR}$ , dabei nennt man  $I_{\text{EXT}}(p)$  auch die *Extension* der Property  $p$ ,
- $I_S$ , einer Funktion, welche URIs aus  $V$  in die Vereinigung der Mengen  $IR$  und  $IP$  abbildet, also  $I_S : V \rightarrow IR \cup IP$ ,
- $I_L$ , einer Funktion von den getypten Literalen aus  $V$  in die Menge  $IR$  der Ressourcen und
- $LV$  einer speziellen Teilmenge von  $IR$ , genannt Menge der Literalwerte, die (mindestens) alle ungetypten Literale aus  $V$  enthält.

# SEMANTIK DER EINFACHEN FOLGERUNG



- jedes ungetypte Literal " $a$ " wird auf  $a$  abgebildet:  $(\text{"}a\text{"})^{\mathcal{I}} = a$ ,
- jedes ungetypte Literal mit Sprachangabe " $a$ "@ $t$  wird auf das Paar  $\langle a, t \rangle$  abgebildet:  $(\text{"}a\text{"}@t)^{\mathcal{I}} = \langle a, t \rangle$ ,
- jedes getypte Literal  $l$  wird auf  $I_L(l)$  abgebildet:  $l^{\mathcal{I}} = I_L(l)$  und
- jede URI  $u$  wird auf  $I_S(u)$  abgebildet:  $u^{\mathcal{I}} = I_S(u)$ .

- Interpretation (schematisch):

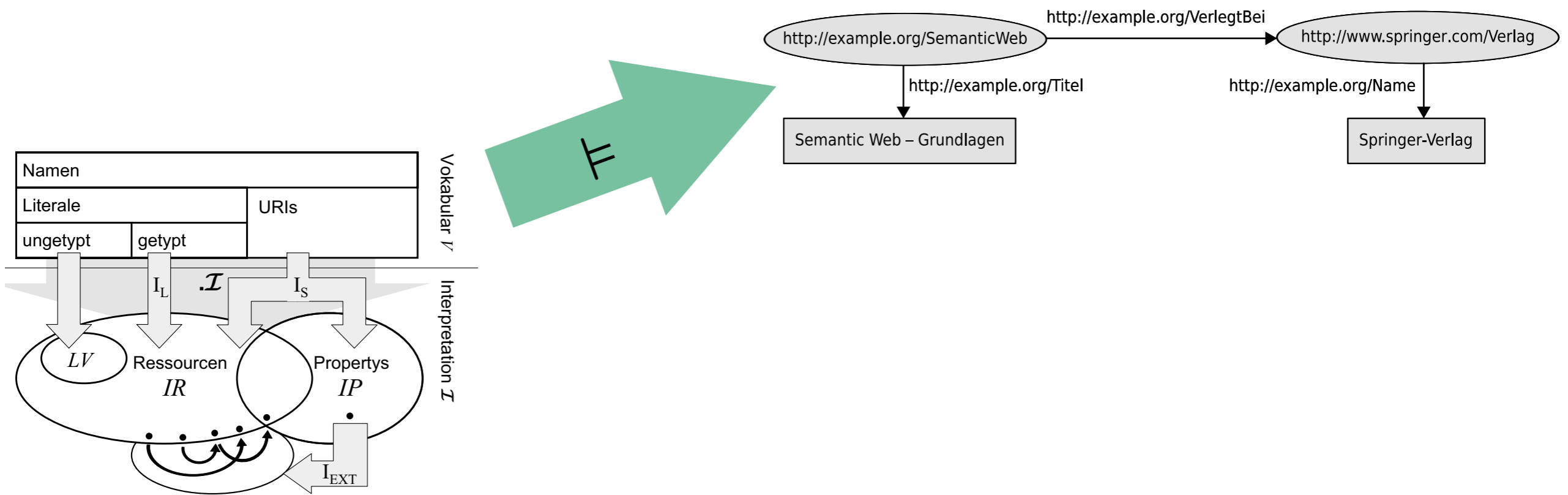




# SEMANTIK DER EINFACHEN FOLGERUNG



- Frage: Wann ist eine gegebene Interpretation Modell eines Graphen?

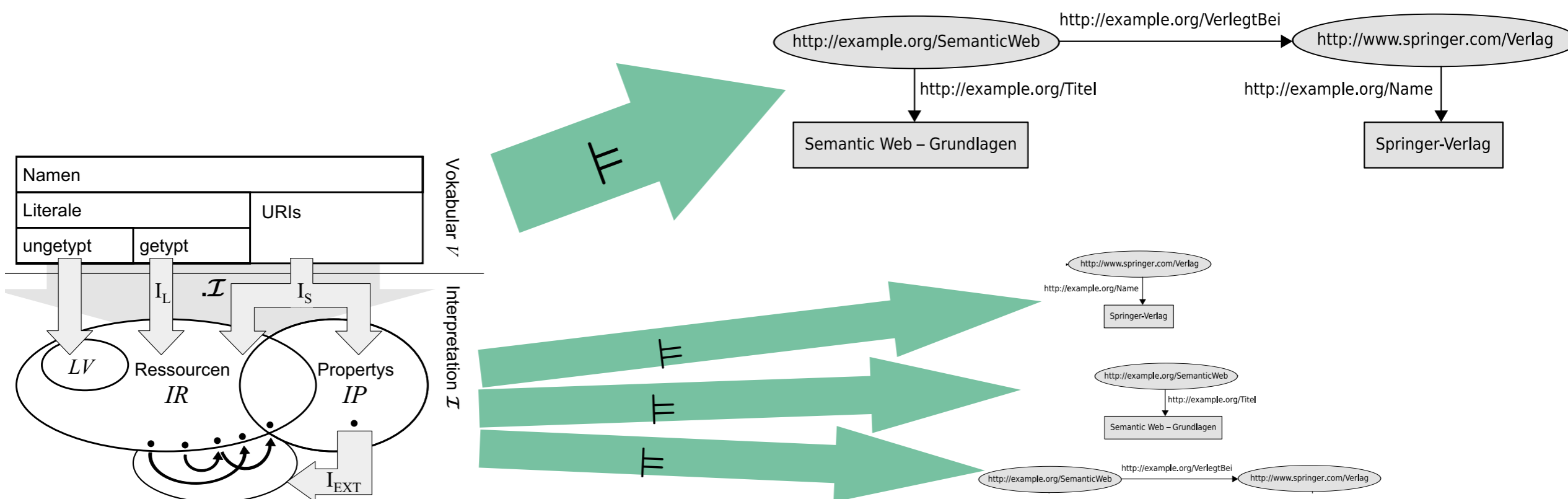


- ...wenn sie Modell jedes Tripels des Graphen ist!

# SEMANTIK DER EINFACHEN FOLGERUNG



- Frage: Wann ist eine gegebene Interpretation Modell eines Graphen?

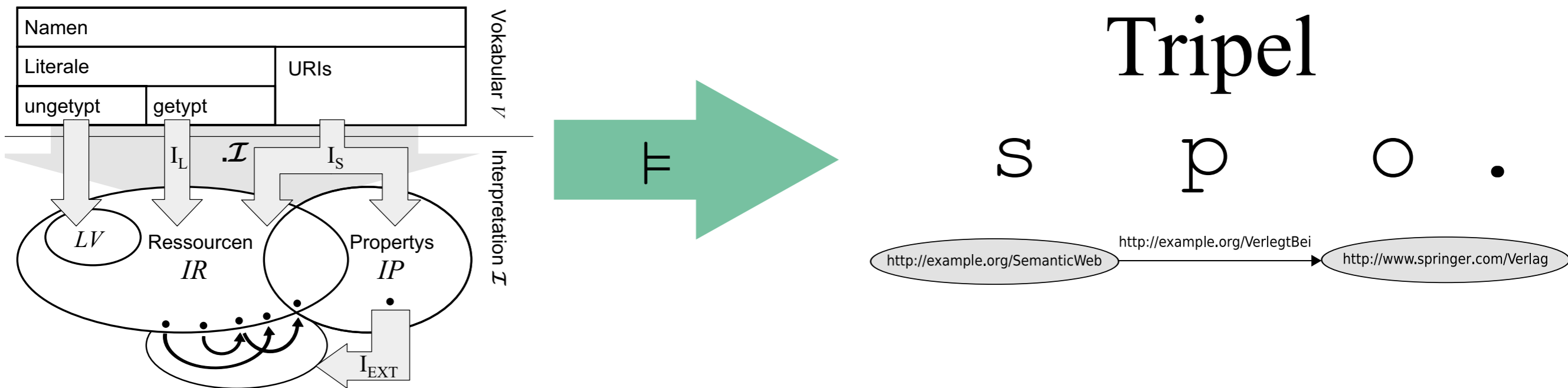


- ...wenn sie Modell jedes Tripels des Graphen ist!

# SEMANTIK DER EINFACHEN FOLGERUNG



- Frage: Wann ist eine gegebene Interpretation Modell eines Tripels?

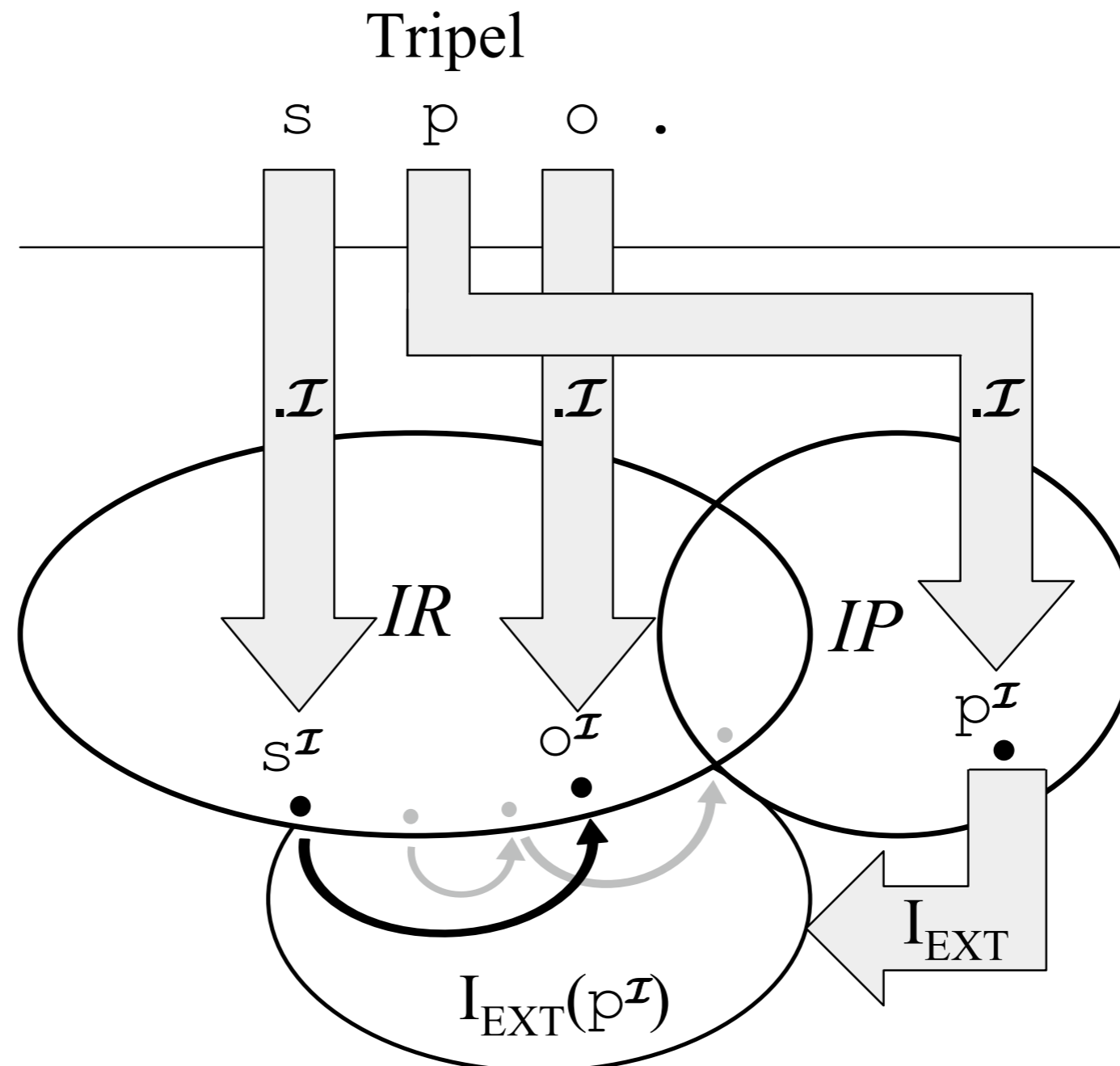


- ...wenn Subjekt, Prädikat und Objekt in  $V$  enthalten sind und außerdem:

$$\langle s^{\mathcal{I}}, o^{\mathcal{I}} \rangle \in I_{EXT}(p^{\mathcal{I}})$$

# SEMANTIK DER EINFACHEN FOLGERUNG

- schematisch:



# SEMANTIK DER EINFACHEN FOLGERUNG

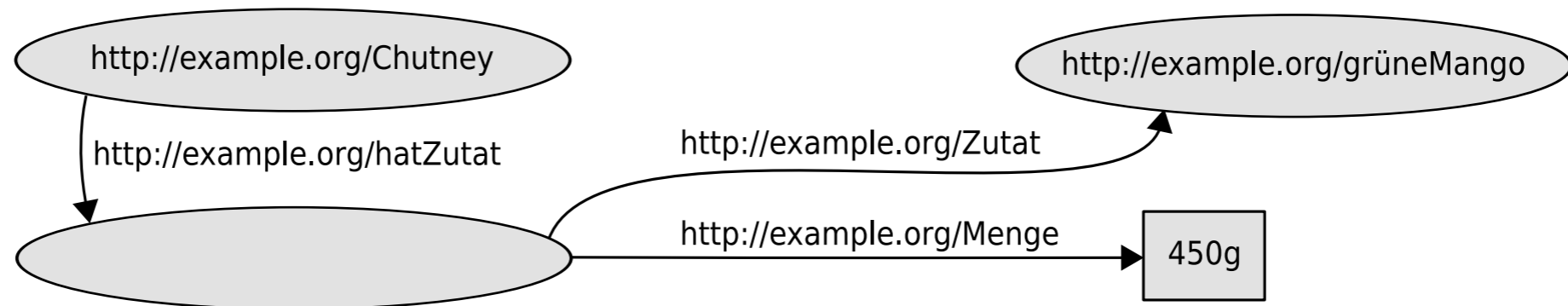


- ...ups, wir haben die bnodes vergessen!
- wird nachgeholt: sei  $A$  Funktion, die alle bnodes auf Elemente von  $\mathbb{R}$  abbildet
- für eine Interpretation  $\mathcal{I}$ , sei  $\mathcal{I}+A$  wie  $\mathcal{I}$ , wobei zusätzlich für jeden bnode  $b$  gilt  $b^{\mathcal{I}+A} = A(b)$
- eine Interpretation  $\mathcal{I}$  ist nun Modell eines RDF-Graphen  $G$ , wenn es ein  $A$  gibt, so dass alle Tripel bezüglich  $\mathcal{I}+A$  wahr werden

# EINFACHE INTERPRETATION: BEISPIEL



- gegeben Graph  $G$ :



und Interpretation  $I$ :

$$IR = \{\chi, \nu, \tau, \nu, \epsilon, \iota, 450g\}$$

$$IP = \{\tau, \nu, \iota\}$$

$$LV = \{450g\}$$

$$I_{EXT} = \tau \mapsto \{\langle \chi, \epsilon \rangle\}$$

$$\nu \mapsto \{\langle \epsilon, \nu \rangle\}$$

$$\iota \mapsto \{\langle \epsilon, 450g \rangle\}$$

$$I_S = \text{ex:Chutney} \mapsto \chi$$

$$\text{ex:grüneMango} \mapsto \nu$$

$$\text{ex:hatZutat} \mapsto \tau$$

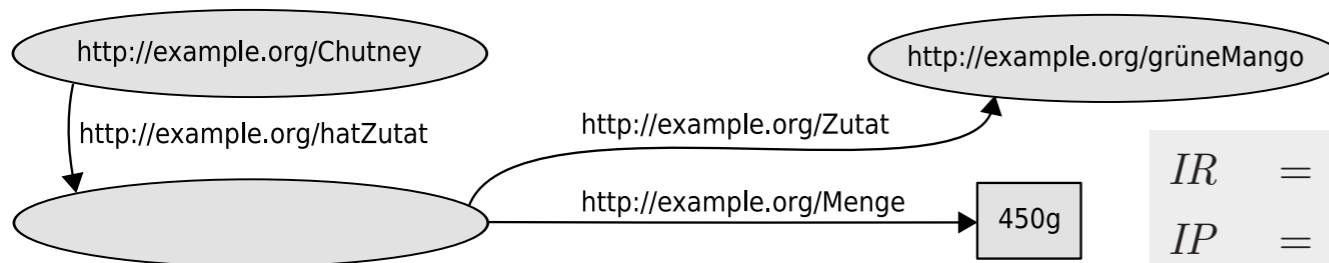
$$\text{ex:Zutat} \mapsto \nu$$

$$\text{ex:Menge} \mapsto \iota$$

$I_L$  ist die „leere Funktion“, da es keine getypten Literale gibt.

- ...ist  $I$  ein Modell von  $G$ ?

# EINFACHE INTERPRETATION: BEISPIEL



$$IR = \{\chi, \nu, \tau, \nu, \epsilon, \iota, 450g\}$$

$$IP = \{\tau, \nu, \iota\}$$

$$LV = \{450g\}$$

$$I_{EXT} = \tau \mapsto \{\langle \chi, \epsilon \rangle\}$$

$$\nu \mapsto \{\langle \epsilon, \nu \rangle\}$$

$$\iota \mapsto \{\langle \epsilon, 450g \rangle\}$$

$$I_S = \text{ex:Chutney} \mapsto \chi$$

$$\text{ex:grüneMango} \mapsto \nu$$

$$\text{ex:hatZutat} \mapsto \tau$$

$$\text{ex:Zutat} \mapsto \nu$$

$$\text{ex:Menge} \mapsto \iota$$

$I_L$  ist die „leere Funktion“, da es keine getypten Literale gibt.

- wählt man  $A : \_ : id1 \mapsto \epsilon$ ,  
dann ergibt sich

$$\begin{aligned} \langle \text{ex:Chutney}^{\mathcal{I}+A}, \_ : id1^{\mathcal{I}+A} \rangle &= \langle \chi, \epsilon \rangle \in I_{EXT}(\tau) = I_{EXT}(\text{ex:hatZutat}^{\mathcal{I}+A}) \\ \langle \_ : id1^{\mathcal{I}+A}, \text{ex:grüneMango}^{\mathcal{I}+A} \rangle &= \langle \epsilon, \nu \rangle \in I_{EXT}(\nu) = I_{EXT}(\text{ex:Zutat}^{\mathcal{I}+A}) \\ \langle \_ : id1^{\mathcal{I}+A}, "450g"^{\mathcal{I}+A} \rangle &= \langle \epsilon, 450g \rangle \in I_{EXT}(\iota) = I_{EXT}(\text{ex:Menge}^{\mathcal{I}+A}) \end{aligned}$$

- also ist  $\mathcal{I}$  Modell von  $G$

# EINFACHE FOLGERUNG



- Definition der einfachen Interpretation legt (modelltheoretisch) einfache Folgerung für RDF-Graphen fest
- Frage: wie lässt sich diese (abstrakt definierte) Semantik im Sinne des automatischen Schlussfolgerns umsetzen
- Antwort: Ableitungsregeln



# EINFACHE FOLGERUNG

- Ableitungsregeln für einfache Folgerung:

$$\frac{u \quad a \quad x \quad .}{u \quad a \quad \_ : n \quad .} \text{ se1}$$

$$\frac{u \quad a \quad x \quad .}{\_ : n \quad a \quad x \quad .} \text{ se2}$$

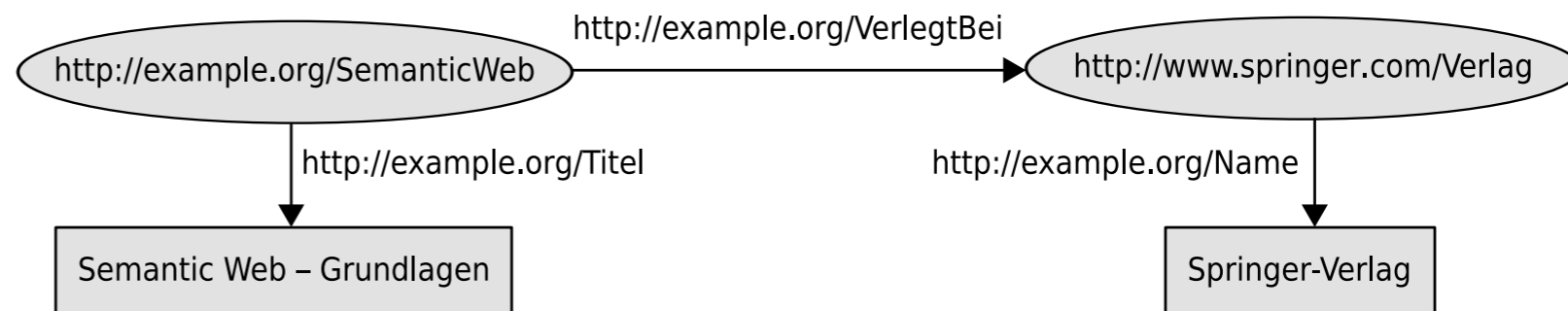
- Bedingung für Anwendung: leerer Knoten nicht bereits anderer URI / anderem Literal zugeordnet

# EINFACHE FOLGERUNG

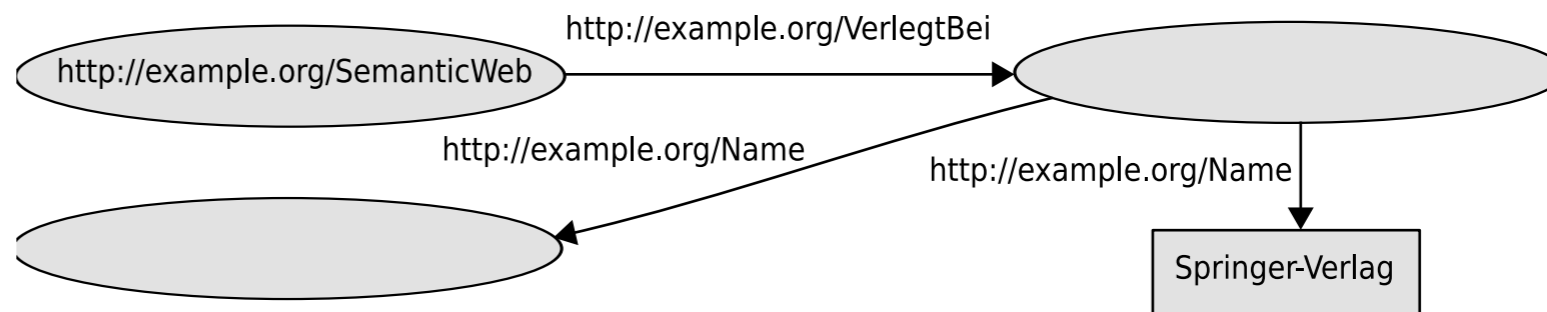


- Satz: Ein Graph  $G_2$  folgt einfach aus einem Graphen  $G_1$ , wenn  $G_1$  mithilfe der Regeln  $se_1$  und  $se_2$  zu einem Graphen  $G_1'$  ergänzt werden kann, so dass  $G_2$  in  $G_1'$  enthalten ist.

- Bsp.: aus



folgt ein-  
fach



# AGENDA



- Motivation
- Vorbetrachtungen
- einfache Folgerung
- **RDF-Folgerung**
- RDFS-Folgerung
- Unzulänglichkeiten von RDF(S)

# RDF-INTERPRETATIONEN

einfache Interpretationen

RDF-Interpretationen

RDFS-Interpretationen

- ...RDF-Interpretationen sind spezielle einfache Interpretationen, wobei für die URIs des RDF-Vokabulars

```
rdf:type rdf:Property rdf:XMLLiteral rdf:nil
```

```
rdf:List rdf:Statement rdf:subject rdf:predicate rdf:object
```

```
rdf:first rdf:rest rdf:Seq rdf:Bag rdf:Alt
```

```
rdf:_1 rdf:_2 ...
```

zusätzliche Forderungen gestellt werden, die die intendierte Semantik der RDF-Bezeichner realisieren:

# RDF-INTERPRETATIONEN



Eine *RDF-Interpretation* für ein Vokabular  $V$  ist nun eine einfache Interpretation für das Vokabular  $V \cup V_{\text{RDF}}$ , welche zusätzlich folgende Bedingungen erfüllt:

- $x \in IP$  genau dann, wenn  $\langle x, \text{rdf:Property}^{\mathcal{I}} \rangle \in I_{\text{EXT}}(\text{rdf:type}^{\mathcal{I}})$ .  
 $x$  ist eine Property genau dann, wenn es mit der durch `rdf:Property` bezeichneten Ressource über die `rdf:type`-Property verbunden ist (dies führt auch automatisch dazu, dass für jede RDF-Interpretation  $IP \subseteq IR$  gilt).
- wenn `"s"^^rdf:XMLLiteral` in  $V$  enthalten und  $s$  ein wohlgeformtes XML-Literal ist, dann
  - $I_L(\text{"s"^^rdf:XMLLiteral})$  ist der XML-Wert<sup>1</sup> von  $s$ ;
  - $I_L(\text{"s"^^rdf:XMLLiteral}) \in LV$ ;
  - $\langle I_L(\text{"s"^^rdf:XMLLiteral}), \text{rdf:XMLLiteral}^{\mathcal{I}} \rangle \in I_{\text{EXT}}(\text{rdf:type}^{\mathcal{I}})$
- wenn `"s"^^rdf:XMLLiteral` in  $V$  enthalten und  $s$  ein *nicht* wohlgeformtes XML-Literal ist, dann
  - $I_L(\text{"s"^^rdf:XMLLiteral}) \notin LV$  und
  - $\langle I_L(\text{"s"^^rdf:XMLLiteral}), \text{rdf:XMLLiteral}^{\mathcal{I}} \rangle \notin I_{\text{EXT}}(\text{rdf:type}^{\mathcal{I}})$ .

# RDF-INTERPRETATIONEN



- zusätzliche Forderung: jede RDF-Interpretation muss Modell der folgenden, „axiomatischen“ Tripel sein:

```

rdf:type          rdf:type          rdf:Property .
rdf:subject       rdf:type          rdf:Property .
rdf:predicate     rdf:type          rdf:Property .
rdf:object        rdf:type          rdf:Property .
rdf:first         rdf:type          rdf:Property .
rdf:rest          rdf:type          rdf:Property .
rdf:value         rdf:type          rdf:Property .
rdf:_1            rdf:type          rdf:Property .
rdf:_2            rdf:type          rdf:Property .
...
rdf:nil           rdf:type          rdf>List .

```

# RDF-FOLGERUNGEN



- automatische Folgerungen werden wieder über Ableitungsregeln realisiert:

$$\frac{}{u \ a \ x} \text{rdfax}$$

jedes axiomatische Tripel „u a x.“ kann immer abgeleitet werden

$$\frac{u \ a \ l \ .}{u \ a \ \_ : n \ .} \text{lg}$$

Literals dürfen durch nicht anderweitig gebundene bnodes ersetzt werden

$$\frac{u \ a \ y \ .}{a \ \text{rdf:type} \ \text{rdf:Property} \ .} \text{rdf1}$$

für jedes Tripelprädikat kann abgeleitet werden dass es eine Entität aus der Klasse der Properties ist

$$\frac{u \ a \ l \ .}{\_ : n \ \text{rdf:type} \ \text{rdf:XMLLiteral}} \text{rdf2}$$

wenn  $\_ : n$  durch lg dem wohlgeformten XML-Literal l zugewiesen wurde

# RDF-FOLGERUNG



- Satz: Ein Graph  $G_2$  RDF-folgt aus einem Graphen  $G_1$ , wenn es einen Graphen  $G_1'$  gibt, so dass
  - $G_1'$  aus  $G_1$  via  $lg$ ,  $rdf1$ ,  $rdf2$  und  $rdfax$  hergeleitet werden kann und
  - $G_2$  aus  $G_1'$  einfach folgt.
- Beachte: zweistufiger Folgerungsprozess.



# AGENDA



- Motivation
- Vorbetrachtungen
- einfache Folgerung
- RDF-Folgerung
- **RDFS-Folgerung**
- Unzulänglichkeiten von RDF(S)

# RDFS-INTERPRETATIONEN

einfache Interpretationen

RDF-Interpretationen

RDFS-Interpretationen



- ...RDFS-Interpretationen sind spezielle RDF-Interpretationen, wobei für die URIs des RDFS-Vokabulars

```
rdfs:domain rdfs:range rdfs:Resource rdfs:Literal rdfs:Datatype  
rdfs:Class rdfs:subClassOf rdfs:subPropertyOf rdfs:member  
rdfs:Container rdfs:ContainerMembershipProperty rdfs:comment  
rdfs:seeAlso rdfs:isDefinedBy rdfs:label
```

zusätzliche Forderungen gestellt werden,  
die die intendierte Semantik der RDF-  
Bezeichner realisieren:

# RDFS-INTERPRETATIONEN



Eine *RDFS-Interpretation* für ein Vokabular  $V$  ist eine RDF-Interpretation des Vokabulars  $V \cup V_{\text{RDFS}}$ , welche zusätzlich die folgenden Kriterien erfüllt:

- $IR = I_{\text{CEXT}}(\text{rdfs:Resource}^{\mathcal{I}})$   
Jede Ressource ist vom Typ `rdfs:Resource`.
- $LV = I_{\text{CEXT}}(\text{rdfs:Literal}^{\mathcal{I}})$   
Jedes ungetypte und jedes wohlgeformte getypte Literal ist vom Typ `rdfs:Literal`.
- Wenn  $\langle x, y \rangle \in I_{\text{EXT}}(\text{rdfs:domain}^{\mathcal{I}})$  und  $\langle u, v \rangle \in I_{\text{EXT}}(x)$ ,  
dann  $u \in I_{\text{CEXT}}(y)$ .  
Ist  $x$  mit  $y$  durch die Property `rdfs:domain` verbunden und verbindet die Property  $x$  die Ressourcen  $u$  und  $v$ , dann ist  $u$  vom Typ  $y$ .
- Wenn  $\langle x, y \rangle \in I_{\text{EXT}}(\text{rdfs:range}^{\mathcal{I}})$  und  $\langle u, v \rangle \in I_{\text{EXT}}(x)$ ,  
dann  $v \in I_{\text{CEXT}}(y)$ .  
Ist  $x$  mit  $y$  durch die Property `rdfs:range` verbunden und verbindet die Property  $x$  die Ressourcen  $u$  und  $v$ , dann ist  $v$  vom Typ  $y$ .
- $I_{\text{EXT}}(\text{rdfs:subPropertyOf}^{\mathcal{I}})$  ist reflexiv und transitiv auf  $IP$ .  
Die `rdfs:subPropertyOf`-Property verbindet jede Property mit sich selbst.  
Darüber hinaus gilt: Verbindet `rdfs:subPropertyOf` die Property  $x$  mit Property  $y$  und außerdem  $y$  mit der Property  $z$ , so verbindet `rdfs:subPropertyOf` auch  $x$  direkt mit  $z$ .

# RDFS-INTERPRETATIONEN



- Wenn  $\langle x, y \rangle \in I_{\text{EXT}}(\text{rdfs:subPropertyOf}^{\mathcal{I}})$ ,  
dann  $x, y \in IP$  und  $I_{\text{EXT}}(x) \subseteq I_{\text{EXT}}(y)$ .  
Wird  $x$  mit  $y$  durch `rdfs:subPropertyOf` verbunden, dann sind sowohl  $x$  als auch  $y$  Property's und jedes in der Extension von  $x$  enthaltene Ressourcenpaar ist auch in der Extension von  $y$  enthalten.
- Wenn  $x \in IC$ ,  
dann  $\langle x, \text{rdfs:Resource}^{\mathcal{I}} \rangle \in I_{\text{EXT}}(\text{rdfs:subClassOf}^{\mathcal{I}})$ .  
Bezeichnet  $x$  eine Klasse, dann muss es eine Unterklasse der Klasse aller Ressourcen sein, d.h., das Paar aus  $x$  und `rdfs:Resource` ist in der Extension von `rdfs:subClassOf`.
- Wenn  $\langle x, y \rangle \in I_{\text{EXT}}(\text{rdfs:subClassOf}^{\mathcal{I}})$ ,  
dann  $x, y \in IC$  und  $I_{\text{CEXT}}(x) \subseteq I_{\text{CEXT}}(y)$ .  
Stehen  $x$  und  $y$  in der `rdfs:subClassOf`-Beziehung, sind sowohl  $x$  als auch  $y$  Klassen und die (Klassen-)Extension von  $x$  ist Teilmenge der (Klassen-)Extension von  $y$ .
- $I_{\text{EXT}}(\text{rdfs:subClassOf}^{\mathcal{I}})$  ist reflexiv und transitiv auf  $IC$ .  
Die `rdfs:subClassOf`-Property verbindet jede Klasse mit sich selbst. Darüber hinaus folgt, wann immer diese Property Klasse  $x$  mit Klasse  $y$  und Klasse  $y$  mit Klasse  $z$  verbindet, dass sie  $x$  auch direkt mit  $z$  verbindet.

# RDFS-INTERPRETATIONEN



- Wenn  $x \in I_{\text{CEXT}}(\text{rdfs:ContainerMembershipProperty}^{\mathcal{I}})$ ,  
dann  $\langle x, \text{rdfs:member}^{\mathcal{I}} \rangle \in I_{\text{EXT}}(\text{rdfs:subPropertyOf}^{\mathcal{I}})$ .  
Ist  $x$  eine Property vom Typ `rdfs:ContainerMembershipProperty`,  
so steht sie in der `rdfs:subPropertyOf`-Beziehung zur `rdfs:memberProperty`.
- Wenn  $x \in I_{\text{CEXT}}(\text{rdfs:Datatype}^{\mathcal{I}})$ ,  
dann  $\langle x, \text{rdfs:Literal}^{\mathcal{I}} \rangle \in I_{\text{EXT}}(\text{rdfs:subClassOf}^{\mathcal{I}})$   
Ist ein  $x$  als Element der Klasse `rdfs:Datatype` „getypt“, dann muss  
dieses auch eine Unterklasse der Klasse aller Literalwerte (bezeichnet  
mit `rdfs:Literal`) sein.

- ...dazu kommen dann noch jede Menge  
weitere axiomatische Tripel:

# RDFS-INTERPRETATIONEN



rdf:type	rdfs:domain	rdfs:Resource .	rdfs:ContainerMembershipProperty		
rdfs:domain	rdfs:domain	rdf:Property .	rdfs:subClassOf	rdf:Property .	
rdfs:range	rdfs:domain	rdf:Property .	rdf:Alt	rdfs:subClassOf	rdfs:Container .
rdfs:subPropertyOf	rdfs:domain	rdf:Property .	rdf:Bag	rdfs:subClassOf	rdfs:Container .
rdfs:subClassOf	rdfs:domain	rdfs:Class .	rdf:Seq	rdfs:subClassOf	rdfs:Container .
rdf:subject	rdfs:domain	rdf:Statement .			
rdf:predicate	rdfs:domain	rdf:Statement .	rdfs:isDefinedBy	rdfs:subPropertyOf	rdfs:seeAlso .
rdf:object	rdfs:domain	rdf:Statement .			
rdfs:member	rdfs:domain	rdfs:Resource .	rdf:XMLLiteral	rdf:type	rdfs:Datatype .
rdf:first	rdfs:domain	rdf:List .	rdf:XMLLiteral	rdfs:subClassOf	rdfs:Literal .
rdf:rest	rdfs:domain	rdf:List .	rdfs:Datatype	rdfs:subClassOf	rdfs:Class .
rdfs:seeAlso	rdfs:domain	rdfs:Resource .			
rdfs:isDefinedBy	rdfs:domain	rdfs:Resource .	rdf:_1	rdf:type	
rdfs:comment	rdfs:domain	rdfs:Resource .		rdfs:ContainerMembershipProperty	
rdfs:label	rdfs:domain	rdfs:Resource .	rdf:_1	rdfs:domain	rdfs:Resource .
rdf:value	rdfs:domain	rdfs:Resource .	rdf:_1	rdfs:range	rdfs:Resource .
			rdf:_2	rdf:type	
rdf:type	rdfs:range	rdfs:Class .		rdfs:ContainerMembershipProperty	
rdfs:domain	rdfs:range	rdfs:Class .	rdf:_2	rdfs:domain	rdfs:Resource .
rdfs:range	rdfs:range	rdfs:Class .	rdf:_2	rdfs:range	rdfs:Resource .
rdfs:subPropertyOf	rdfs:range	rdf:Property .	...		
rdfs:subClassOf	rdfs:range	rdfs:Class .			
rdf:subject	rdfs:range	rdfs:Resource .			
rdf:predicate	rdfs:range	rdfs:Resource .			
rdf:object	rdfs:range	rdfs:Resource .			
rdfs:member	rdfs:range	rdfs:Resource .			
rdf:first	rdfs:range	rdfs:Resource .			
rdf:rest	rdfs:range	rdf:List .			
rdfs:seeAlso	rdfs:range	rdfs:Resource .			
rdfs:isDefinedBy	rdfs:range	rdfs:Resource .			
rdfs:comment	rdfs:range	rdfs:Literal .			
rdfs:label	rdfs:range	rdfs:Literal .			
rdf:value	rdfs:range	rdfs:Resource .			

# RDFS-FOLGERUNG



- automatische Folgerungen werden wieder über Ableitungsregeln realisiert:

$$\frac{}{u \ a \ x} \text{ rdfsax} \qquad \frac{u \ \text{rdfs:subPropertyOf} \ v \ . \quad v \ \text{rdfs:subPropertyOf} \ x \ .}{u \ \text{rdfs:subPropertyOf} \ x \ .} \text{ rdfs5} \qquad \frac{u \ \text{rdf:type} \ \text{rdfs:ContainerMembershipProperty} \ .}{u \ \text{rdfs:subPropertyOf} \ \text{rdfs:member} \ .} \text{ rdfs12}$$

$$\frac{u \ a \ \_ :n \ .}{u \ a \ l \ .} \text{ gl} \qquad \frac{u \ \text{rdf:type} \ \text{rdf:Property} \ .}{u \ \text{rdfs:subPropertyOf} \ u \ .} \text{ rdfs6} \qquad \frac{u \ \text{rdf:type} \ \text{rdfs:Datatype} \ .}{u \ \text{rdfs:subClassOf} \ \text{rdfs:Literal} \ .} \text{ rdfs13}$$

$$\frac{u \ a \ l \ .}{\_ :n \ \text{rdf:type} \ \text{rdfs:Literal} \ .} \text{ rdfs1} \qquad \frac{a \ \text{rdfs:subPropertyOf} \ b \ . \quad u \ a \ y \ .}{u \ b \ y \ .} \text{ rdfs7}$$

$$\frac{a \ \text{rdfs:domain} \ x \ . \quad u \ a \ y \ .}{u \ \text{rdf:type} \ x \ .} \text{ rdfs2} \qquad \frac{u \ \text{rdf:type} \ \text{rdfs:Class} \ .}{u \ \text{rdfs:subClassOf} \ \text{rdfs:Resource} \ .} \text{ rdfs8}$$

$$\frac{a \ \text{rdfs:range} \ x \ . \quad u \ a \ v \ .}{v \ \text{rdf:type} \ x \ .} \text{ rdfs3} \qquad \frac{u \ \text{rdfs:subClassOf} \ x \ . \quad v \ \text{rdf:type} \ u \ .}{v \ \text{rdf:type} \ x \ .} \text{ rdfs9}$$

$$\frac{u \ a \ x \ .}{u \ \text{rdf:type} \ \text{rdfs:Resource} \ .} \text{ rdfs4a} \qquad \frac{u \ \text{rdf:type} \ \text{rdfs:Class} \ .}{u \ \text{rdfs:subClassOf} \ u \ .} \text{ rdfs10}$$

$$\frac{u \ a \ v \ .}{v \ \text{rdf:type} \ \text{rdfs:Resource} \ .} \text{ rdfs4b} \qquad \frac{u \ \text{rdfs:subClassOf} \ v \ . \quad v \ \text{rdfs:subClassOf} \ x \ .}{u \ \text{rdfs:subClassOf} \ x \ .} \text{ rdfs11}$$

# RDFS-FOLGERUNG



- wichtige Definition: XML-Clash

```
ex:hatSmiley          rdfs:range      rdf:Literal .  
ex:böseBemerkung     ex:hatSmiley    ">:->"^^XMLLiteral .
```

- tritt auf, wenn einem Knoten vom Typ `rdf:Literal` ein nicht-wohlgeformter Literalwert zugewiesen werden muss.



# RDFS-FOLGERUNG



- Satz: Ein Graph RDFS-folgt aus  $G_1$  genau dann, wenn es einen Graphen  $G_1'$  gibt, der durch Anwendung der Regeln lg, gl, rdfsax, rdf1, rdf2, rdfs1 – rdfs13 und rdfsax aus  $G_1$  folgt, so dass
  - $G_2$  aus  $G_1'$  einfach folgt oder
  - $G_1'$  einen XML-Clash enthält.

# AGENDA



- Motivation
- Vorbetrachtungen
- einfache Folgerung
- RDF-Folgerung
- RDFS-Folgerung
- Unzulänglichkeiten von RDF(S)

# WAS KANN RDF(S) NICHT?



- bestimmte (vernünftig) scheinende Folgerungen können nicht RDFS-gefolgert werden, z.B.

```
ex:sprichtMit    rdfs:domain    ex:Homo .
ex:Homo         rdfs:subClassOf ex:Primates .
```

impliziert

```
ex:sprichtMit    rdfs:domain    ex:Primates .
```

- mögliche Lösung: noch stärkere, „extensionale“, Semantik
- keine Möglichkeit, Negation auszudrücken