Grundlagen Semantic Web

Lehrveranstaltung im WS11/12
Seminar für Computerlinguistik
Universität Heidelberg

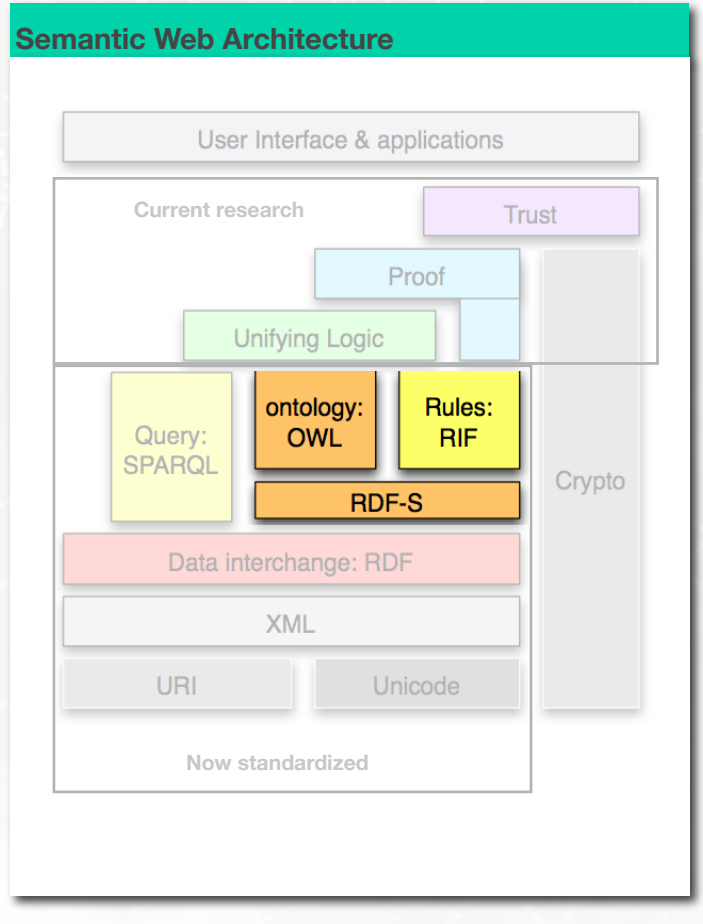
PD Dr. Sebastian Rudolph
Institut AIFB
Karlsruher Institut für Technologie

AIFB

Logic: another thing that penguins aren't very good at.

Logik – Grundlagen

Dr. Sebastian Rudolph

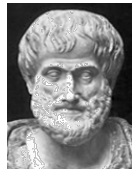


AIFB

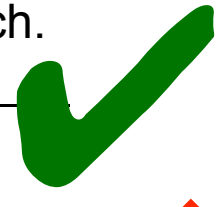
Was ist Logik?

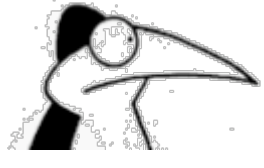
etymologische Herkunft: griechisch λογος
bedeutet „Wort, Rede, Lehre“ (s.a. Faust I…)

- Logik als Argumentation:



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





Alle Pinguine sin[illegible]arz-weiß.
Einige alte TV-Sh[illegible]sind schwarz-weiß.
Einige Pinguine s[illegible]e TV-Shows.

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

Warum formal?

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

Wie funktioniert Logik?

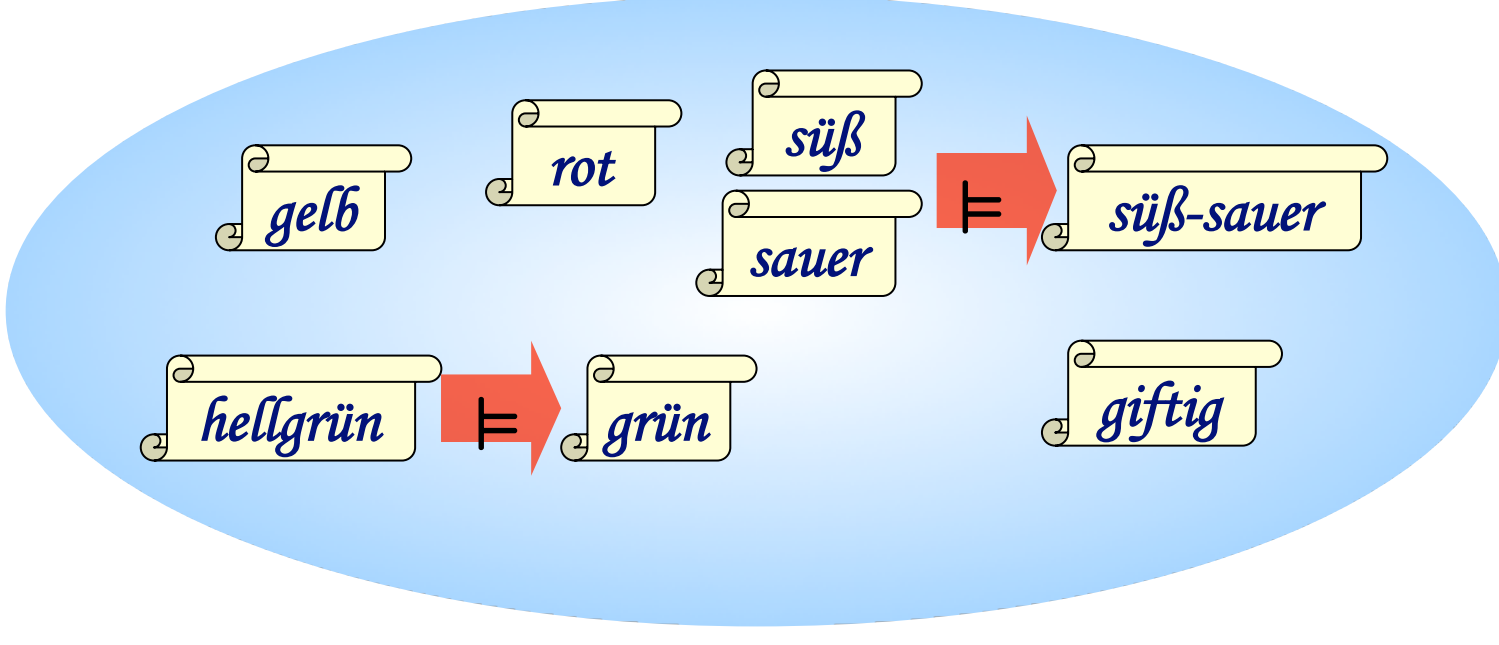
Alle Menschen sind sterblich.
Sokrates ist ein Mensch.
Also ist Sokrates sterblich.

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. σημαντικος – *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 Φ folgt der Satz φ“ oder auch

„ φ ist eine logische Konsequenz aus Φ.“

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

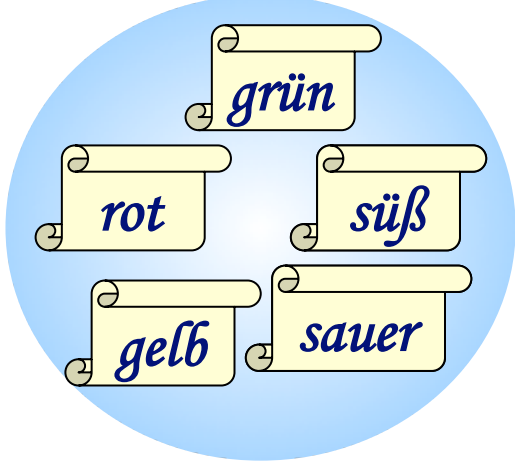
Wie funktioniert Logik? Syntax.

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 φ und ψ Sätze sind, dann auch φ-ψ“

Konstruktor oder *Junktor*

Wie funktioniert Logik? Ausdrucksstärke.

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

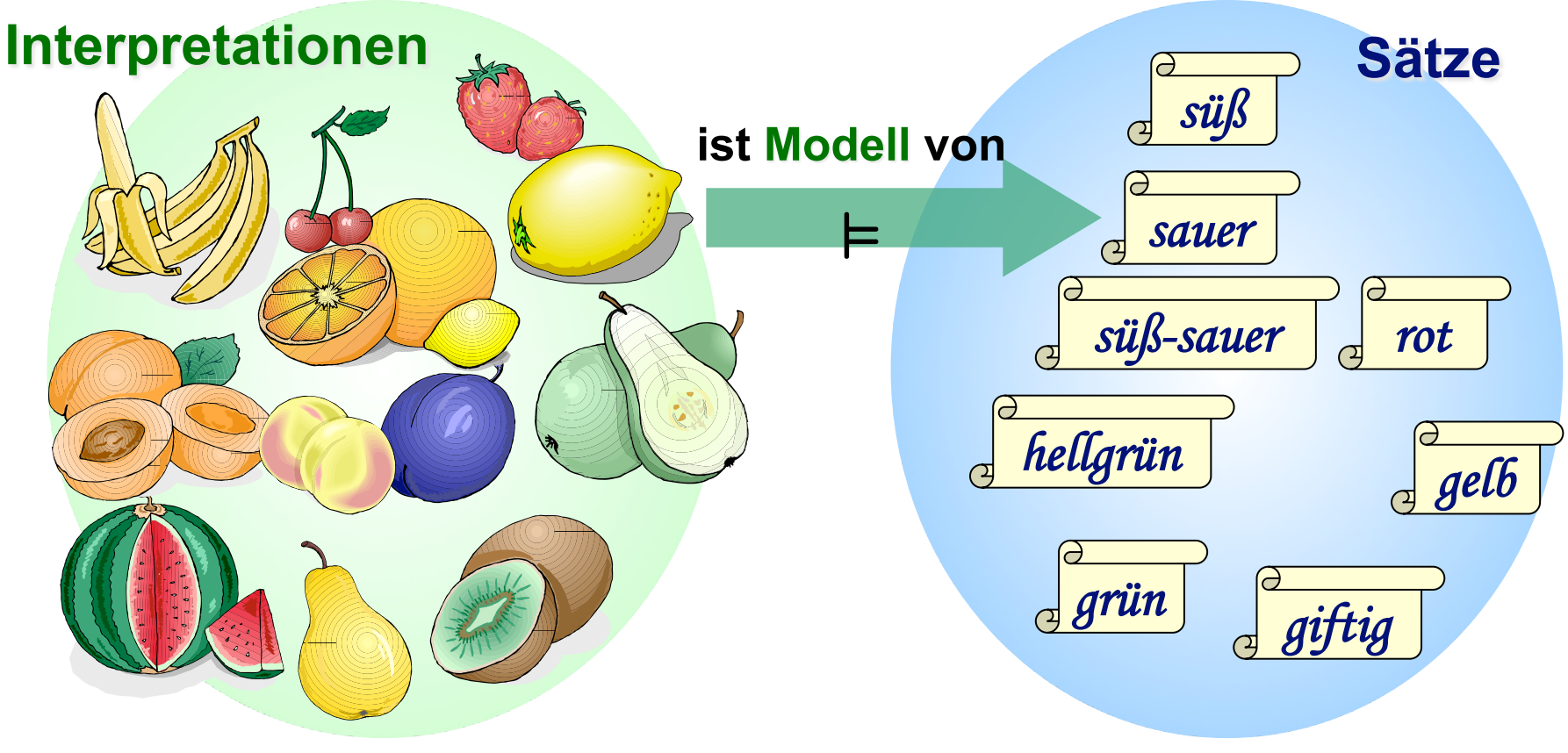
- komfortabler in der Verwendung (verschiedene und komplexe Sachverhalte sind einfach auszudrücken), aber
- schwieriger (meta)mathematisch zu handhaben (Beweisen von Eigenschaften der Logik umständlicher).

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*.

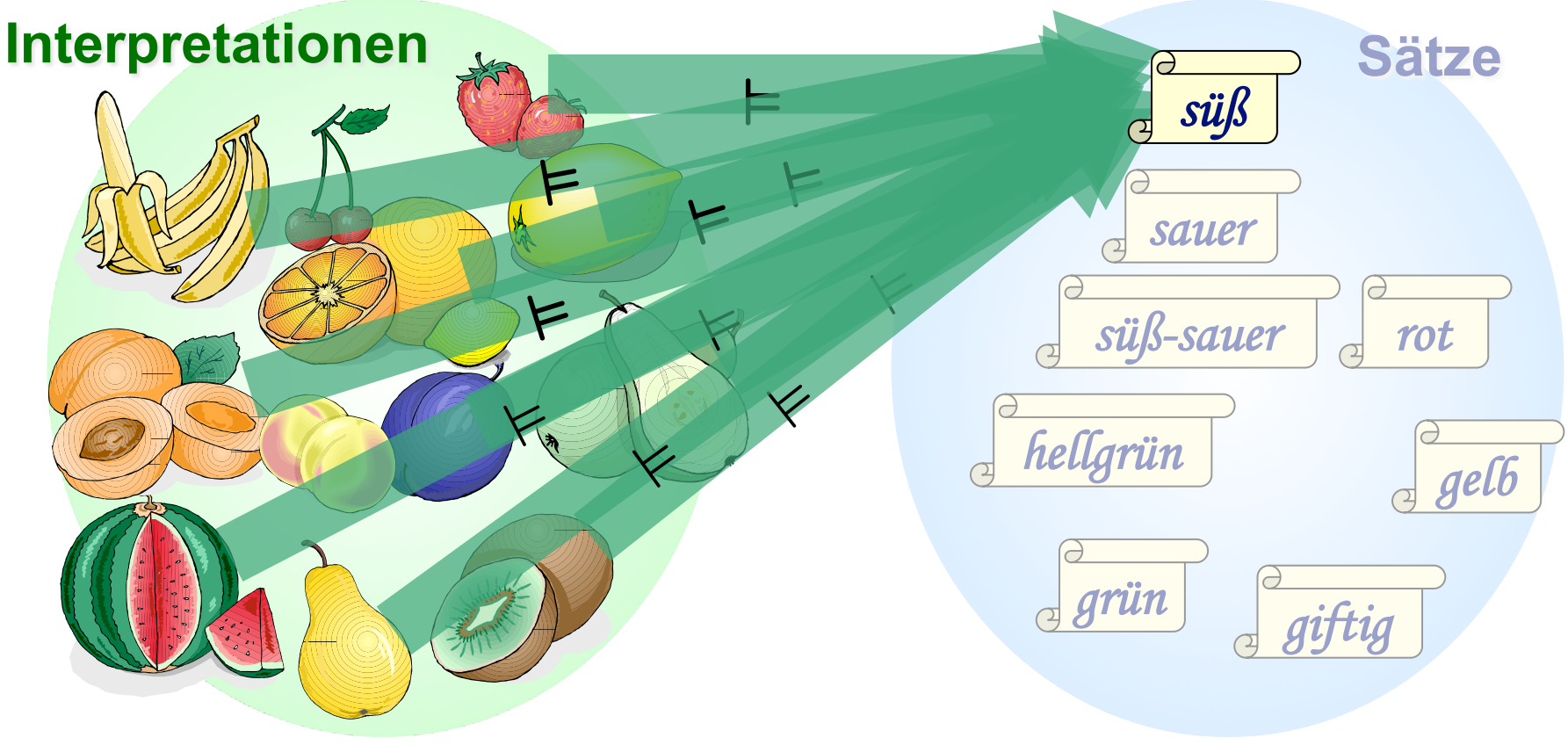
Wie funktioniert Logik? - Modelltheorie

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



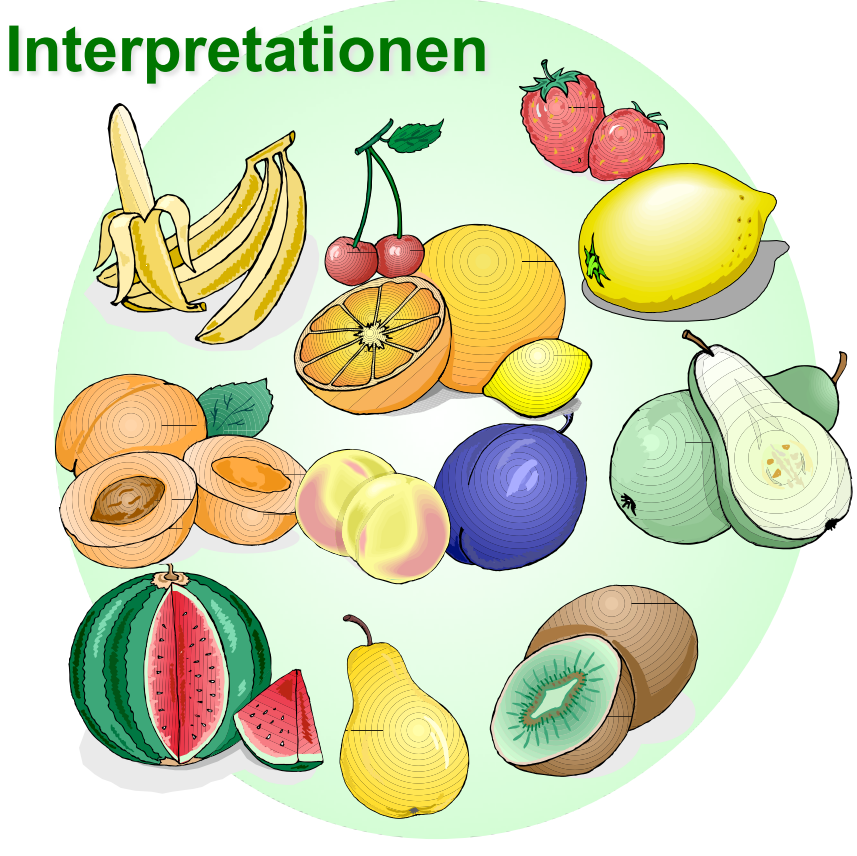
Wie funktioniert Logik? - Modelltheorie

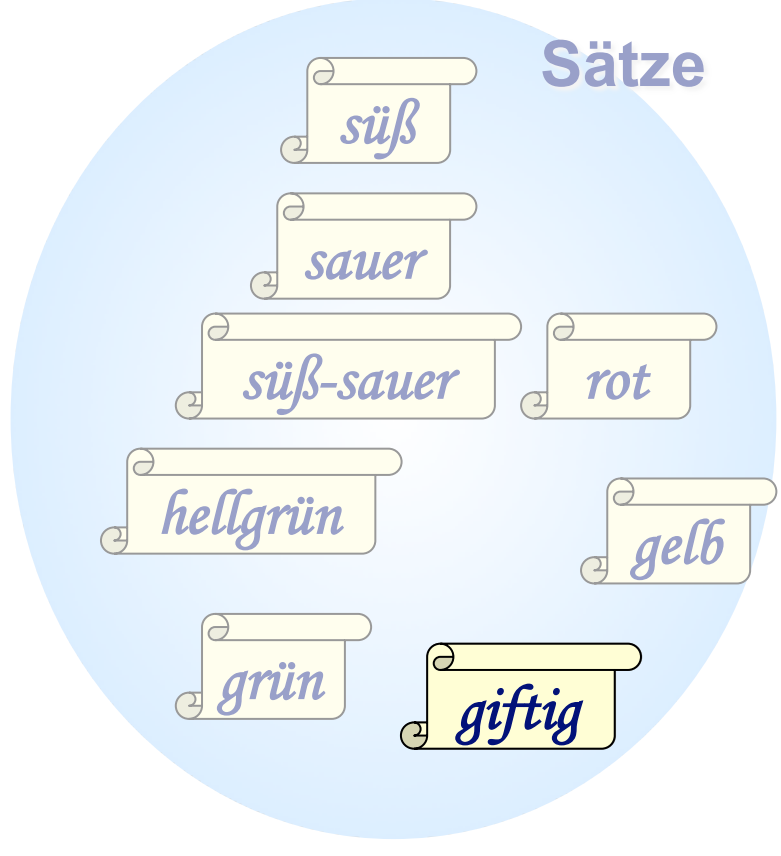
Sätze, für die **jede** Interpretation ein Modell ist, heißen *allgemeingültig* oder *Tautologien* (grch. ταυτολογία).



Wie funktioniert Logik? - Modelltheorie

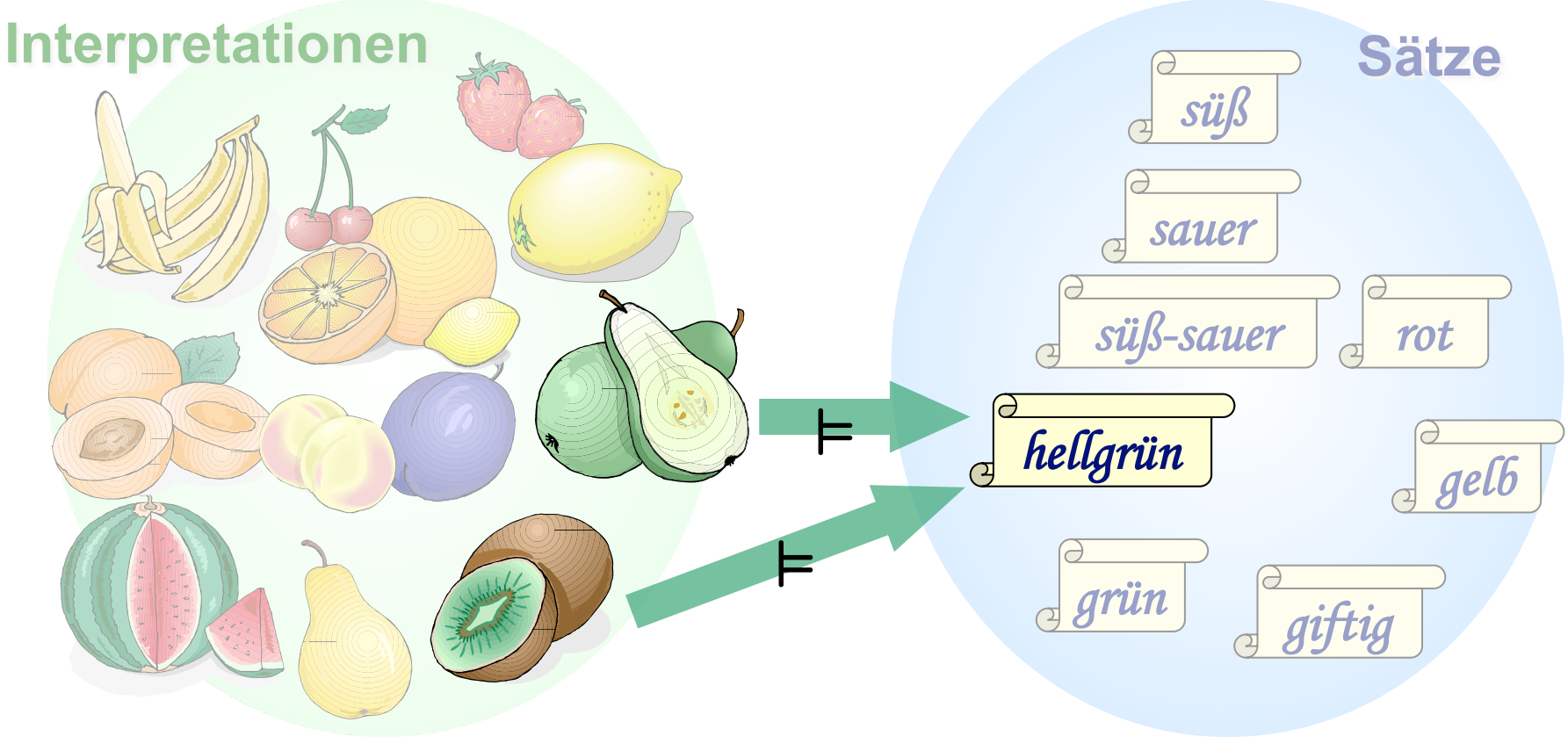
Sätze, für die **keine** Interpretation ein Modell ist, heißen *widersprüchlich* oder *unerfüllbar*.





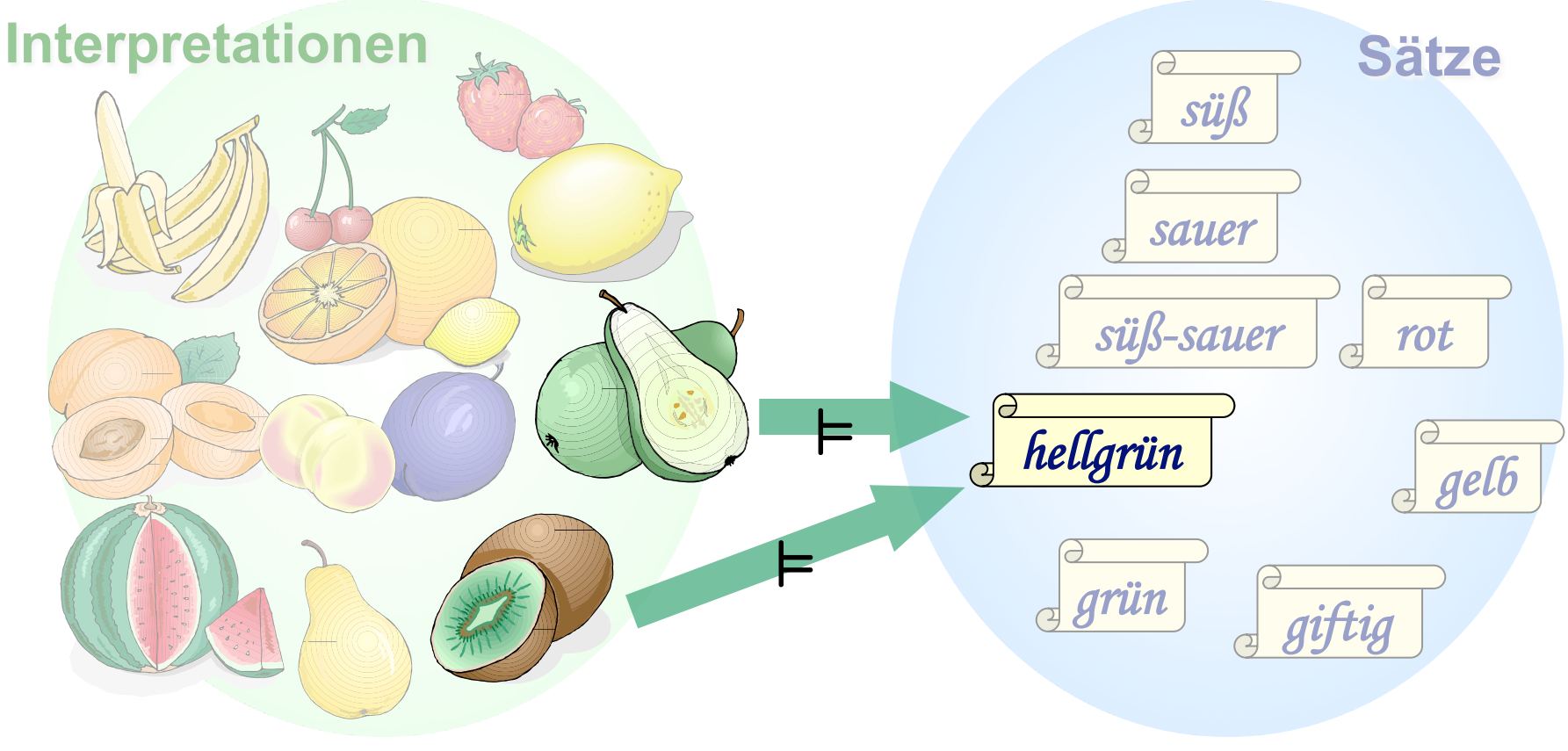
Wie funktioniert Logik? - Modelltheorie

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



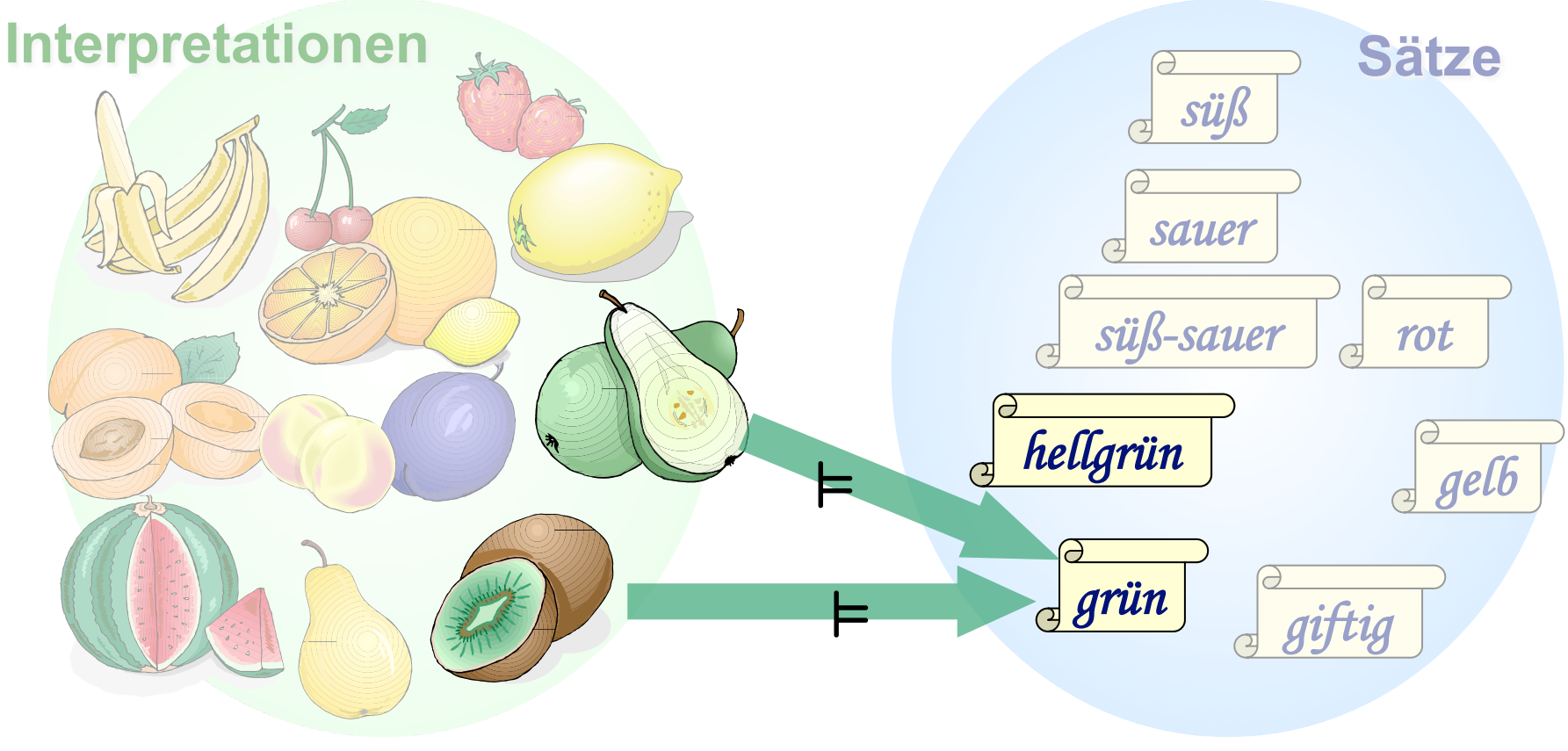
Wie funktioniert Logik? - Modelltheorie

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



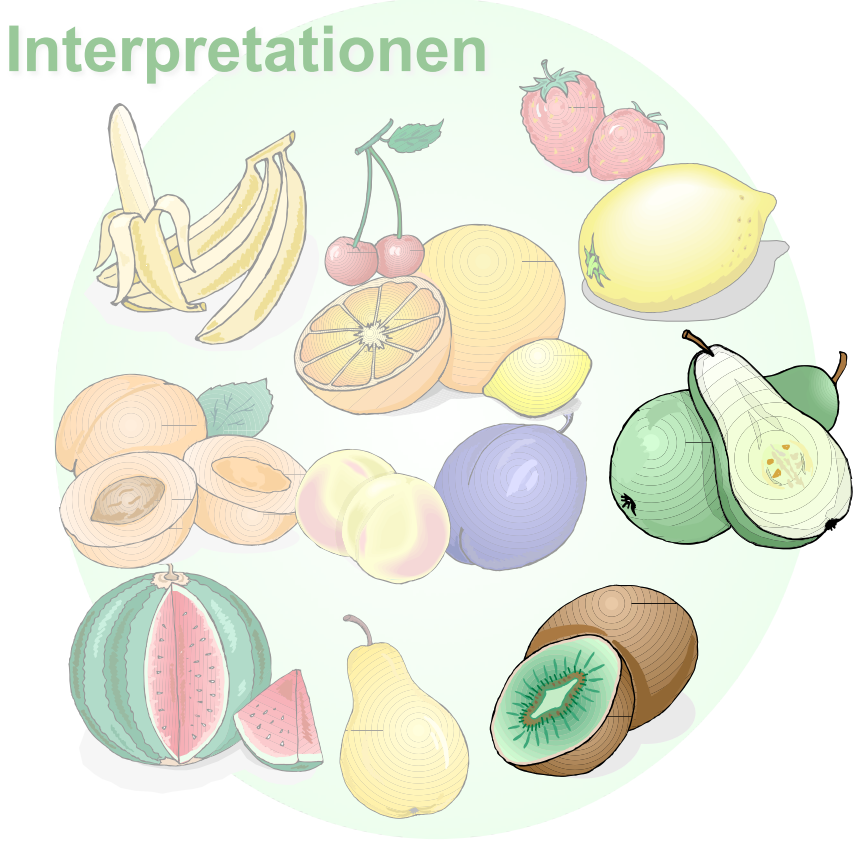
Wie funktioniert Logik? - Modelltheorie

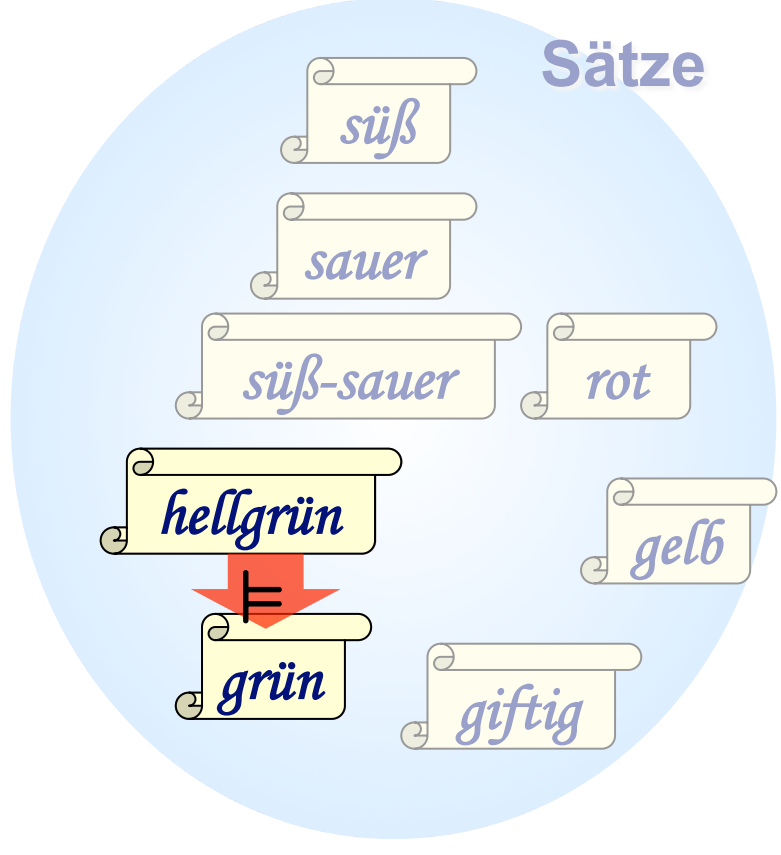
Eine Möglichkeit, die Schlussfolgerungsrelation zu definieren besteht über Interpretationen bzw. Modelle.



Wie funktioniert Logik? - Modelltheorie

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



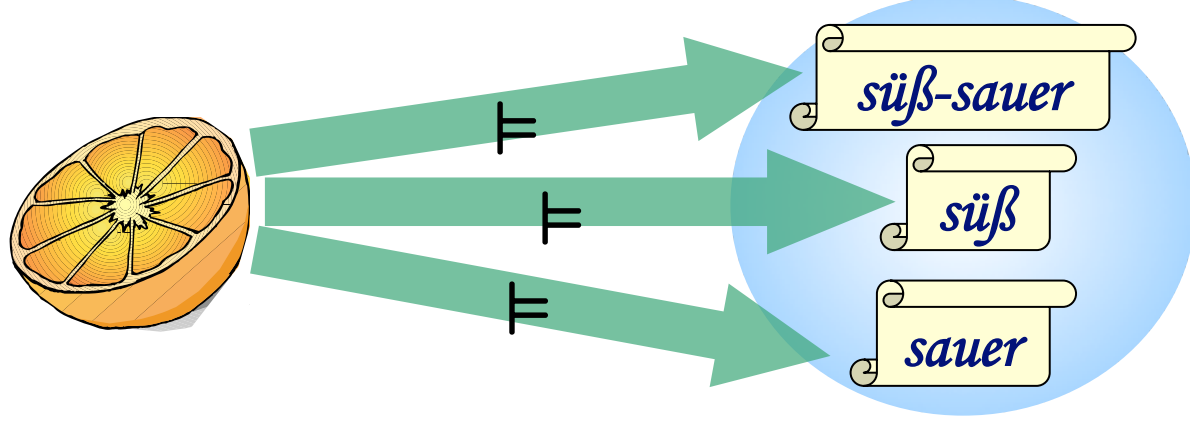


Wie funktioniert Logik?
Semantik entlang der Syntax

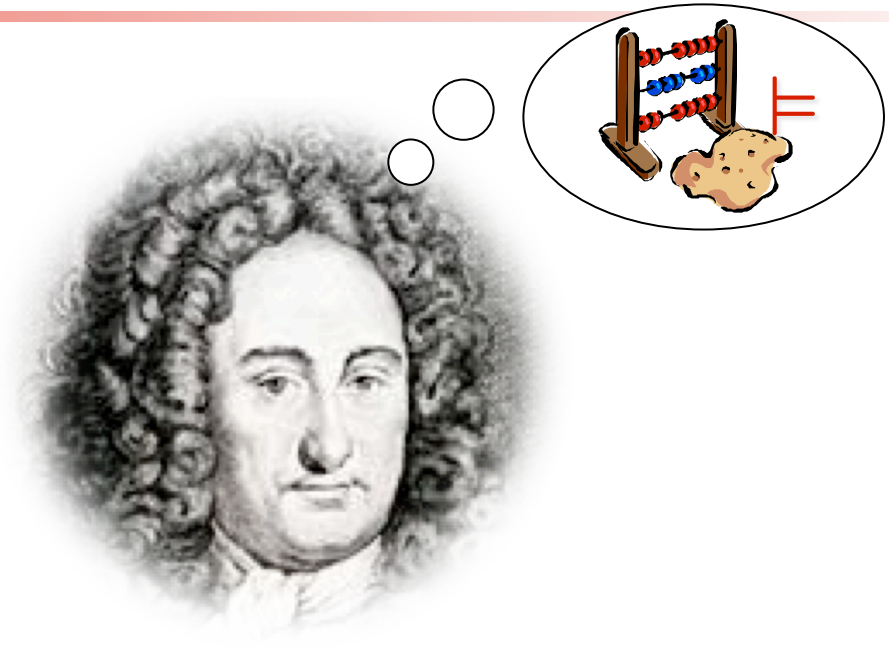
Häufiges Prinzip bei Definition von Interpretationen:

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

Semantik-Regel: „Die Modelle von φ-ψ sind genau die Interpretationen, die Modelle sowohl von φ als auch von ψ sind."



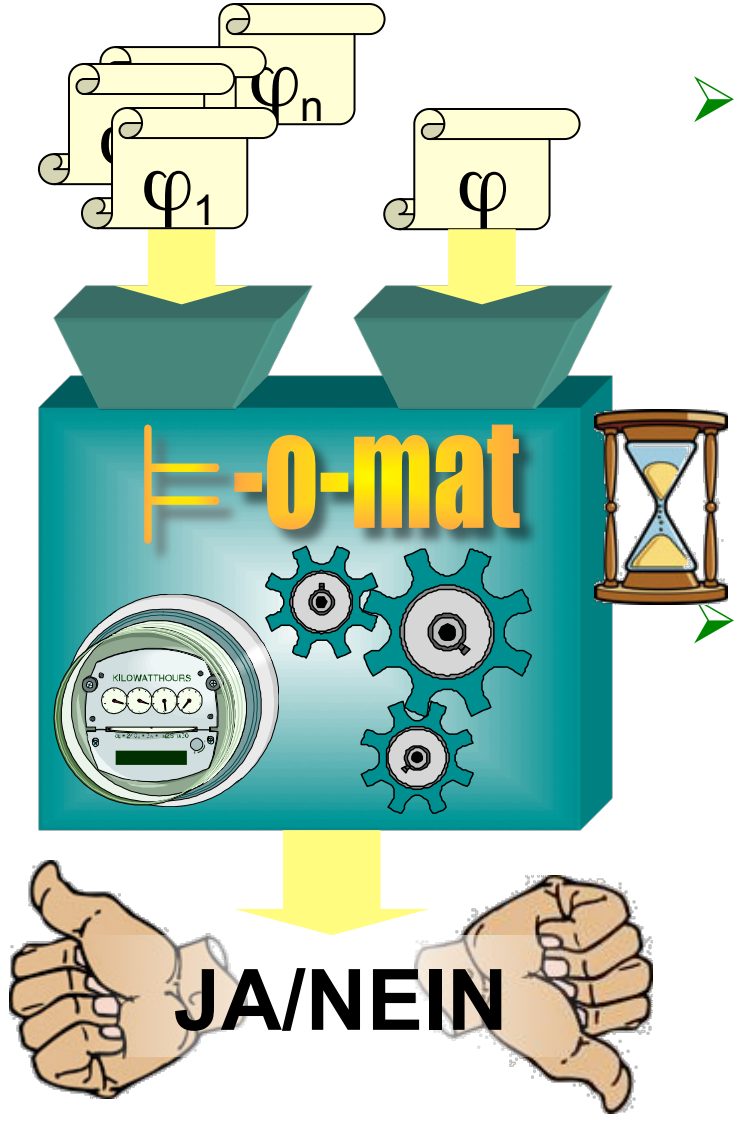
Beweistheorie



- Zurück zu Leibniz: Rechenmaschine für Logik

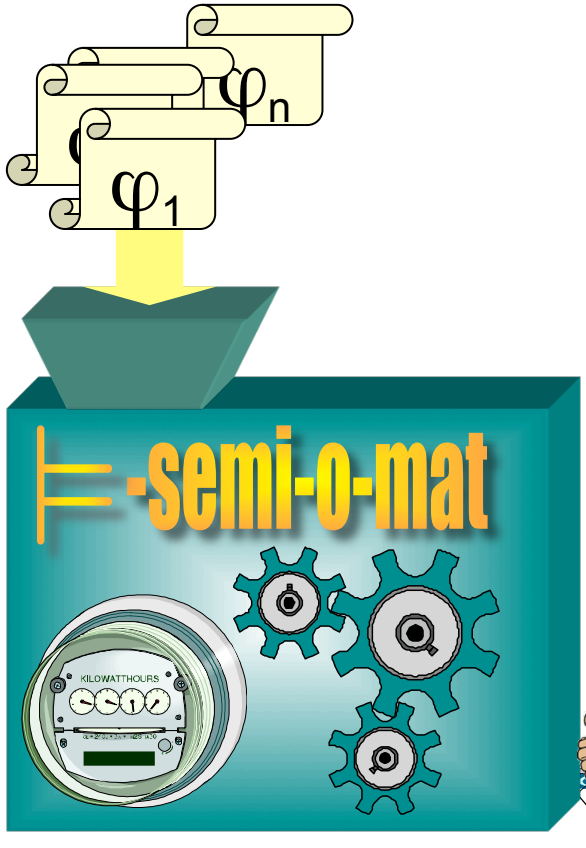
- Aber: Möglichkeit, direkt mit allen möglichen Interpretationen zu arbeiten, oft eingeschränkt
- Daher: Versuch, Schlussfolgerungsrelation durch rein syntaktische Verfahren zu beschreiben/berechnen

Entscheidungsverfahren/Entscheidbarkeit

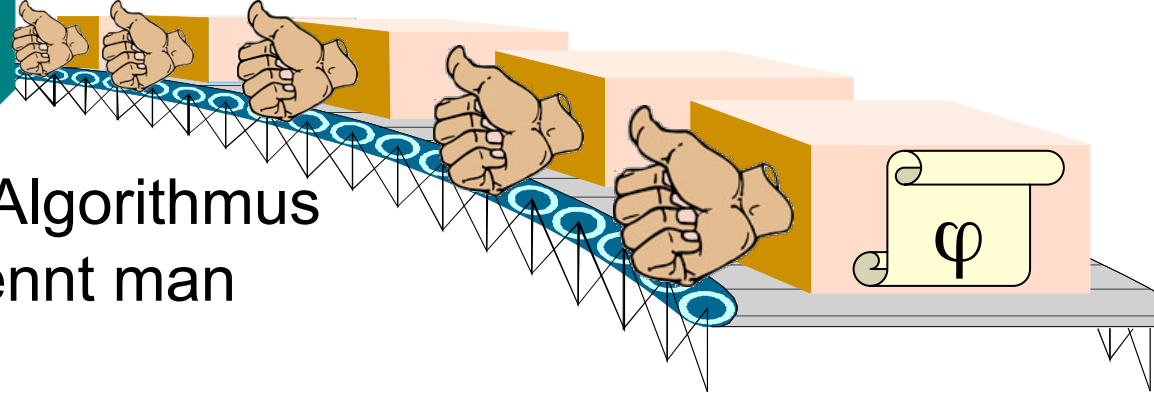


- Entscheidungsalgorithmus:
  - input: Menge $\{\varphi_1, \ldots, \varphi_n\}$ von Sätzen und Satz φ
  - terminiert nach endlicher Zeit
  - output:
    - „Ja“, falls $\{\varphi_1, \ldots, \varphi_n\} \models \varphi$
    - „Nein“ sonst
- Gibt es einen solchen Algorithmus für eine Logik, dann nennt man sie *entscheidbar.*

Aufzählungsverfahren/Semientscheidbarkeit



- Aufzählungsverfahren:
  - input: Sätze $\{\varphi_1, \ldots, \varphi_n\}$
  - output: Sätze φ, für die gilt $\{\varphi_1, \ldots, \varphi_n\} \models \varphi$
  - jeder solche Satz wird (irgendwann) ausgegeben

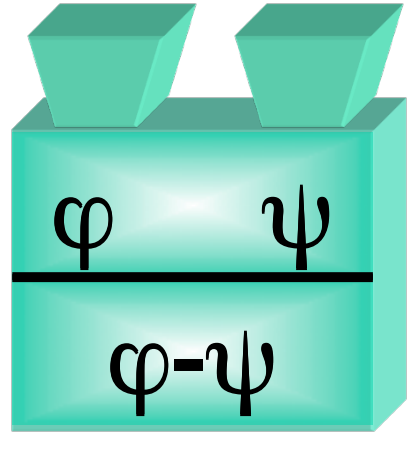


- Gibt es einen solchen Algorithmus für eine Logik, dann nennt man sie *semientscheidbar.*

Deduktionskalkül

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

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



Deduktionskalkül

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

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

Aussagenlogik

- 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,…, p_1,p_2,…)
- Können als natürlichsprachliche Aussagen gedacht werden: „Es regnet.“…

Aussagenlogik – Syntax

- Erzeugungsregeln für Sätze:
  - alle atomaren Propositionen sind Sätze (p, q,…)
  - ist φ ein Satz, dann auch $\neg\varphi$
  - sind φ und ψ 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 trotzem oft lesbarer…

Aussagenlogik – Syntax

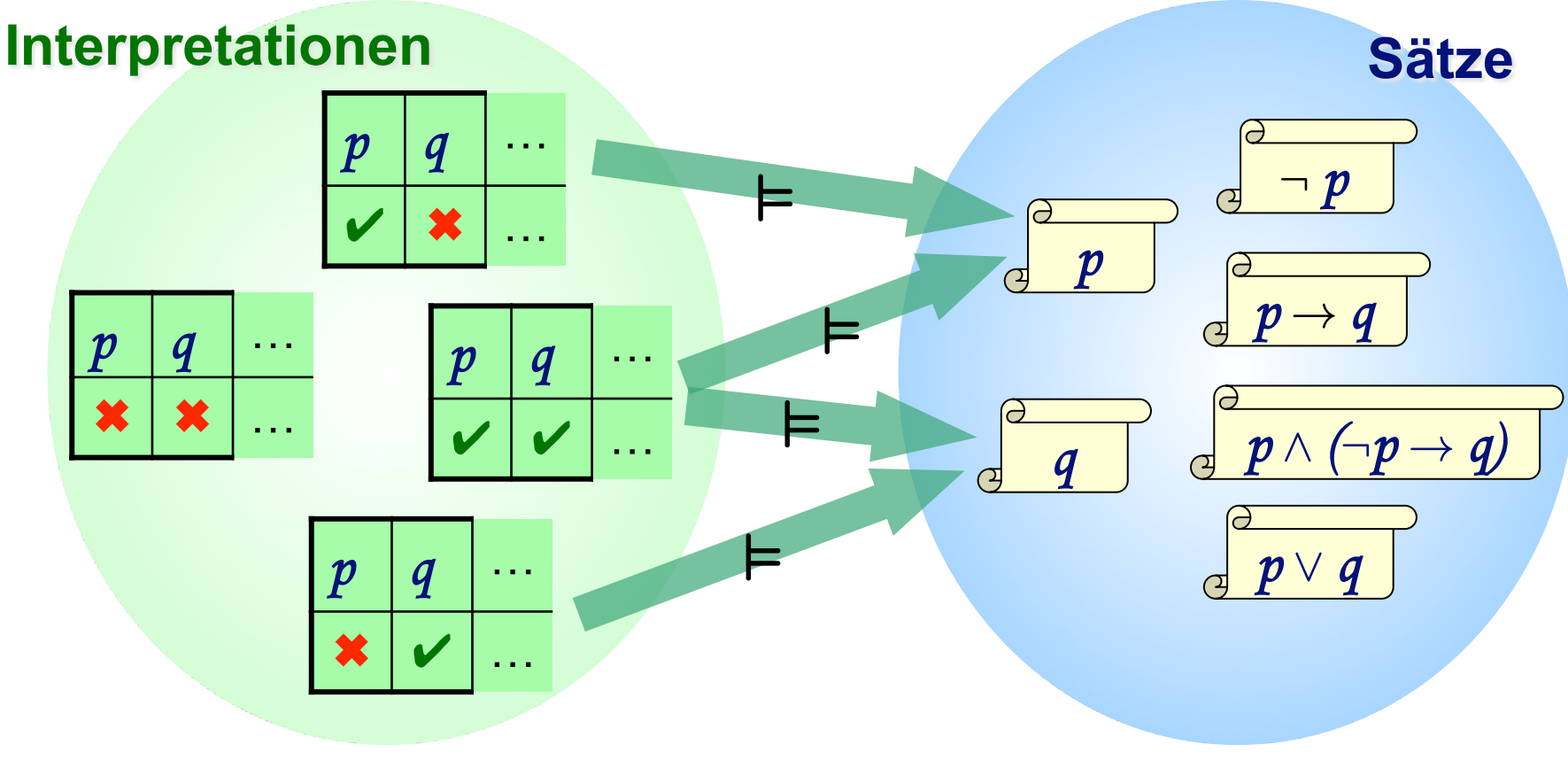
Junktor	*Name*	*Intuitive Bedeutung*
$\neg$	Negation	„nicht“
$\wedge$	Konjunktion	„und“
$\vee$	Disjunktion	„oder“
$\rightarrow$	Implikation	„wenn – dann“
$\leftrightarrow$	Äquivalenz	„genau dann, wenn“

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?



Aussagenlogik – Modelltheoretische Semantik

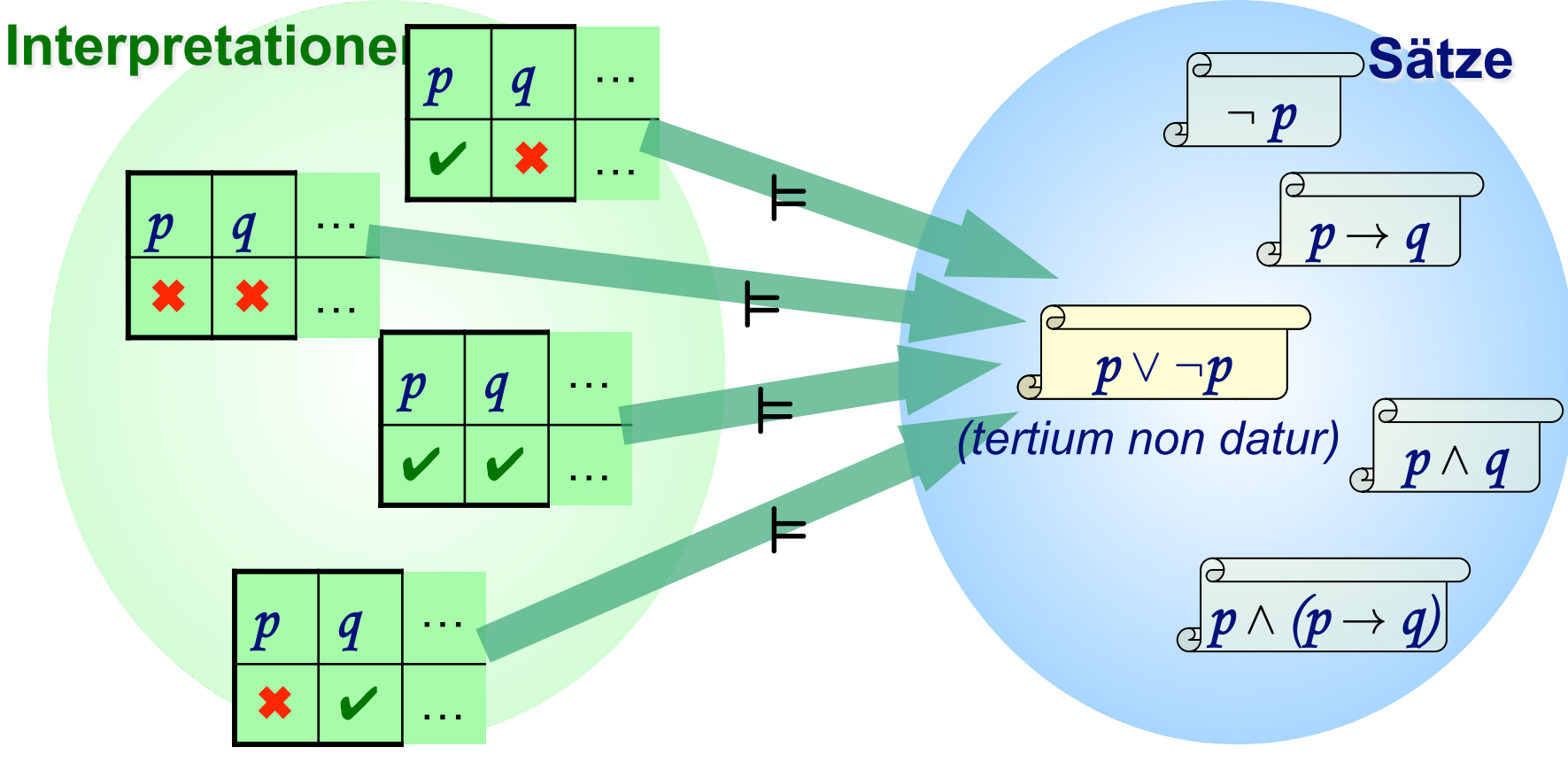
- Formal: Interpretationen $\mathcal{I}$ sind Abbildungen von der Menge der atomaren Propositionen in die Menge {wahr, falsch}, d.h. jeder dieser Propositionen *p* wird ein Wahrheitswert $WW_{\mathcal{I}}(p)$ zugeordnet.
- Daraus bestimmt man Modelle für zusammengesetzte Sätze über

Semantik-Regeln

- ⇨ $\mathcal{I}$ Modell von ¬φ genau dann, wenn $\mathcal{I}$ **kein** Modell von φ
- ⇨ $\mathcal{I}$ Modell von (φ ∧ ψ) genau dann, wenn $\mathcal{I}$ Modell von φ **und** von ψ
- ⇨ $\mathcal{I}$ Modell von (φ ∨ ψ) genau dann, wenn $\mathcal{I}$ Modell von φ **oder** von ψ
- ⇨ $\mathcal{I}$ Modell von (φ → ψ) genau dann, wenn $\mathcal{I}$ **kein** Modell von φ **oder** $\mathcal{I}$ Modell von ψ
- ⇨ $\mathcal{I}$ Modell von (φ ↔ ψ) genau dann, wenn $\mathcal{I}$ Modell für **jeden oder keinen** der beiden Sätze ist.

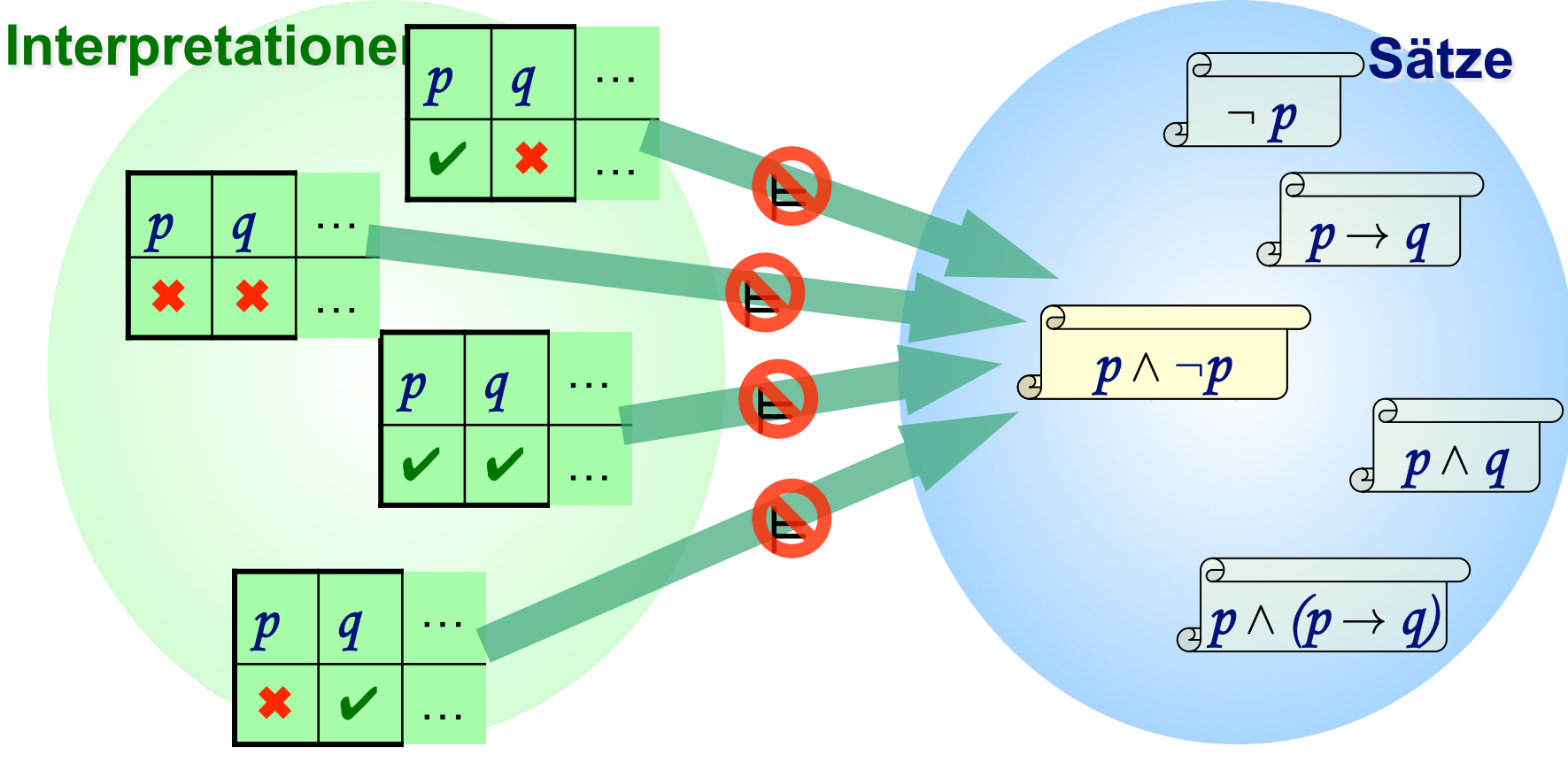
Aussagenlogik – Modelltheoretische Semantik

Beispiel für Tautologie in der Aussagenlogik.



Aussagenlogik – Modelltheoretische Semantik

Beispiel für Kontradiktion in der Aussagenlogik.



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

Aussagenlogik – Entscheidungsalgorithmus

Modus Ponens:

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

$$\models (p \wedge (p \rightarrow q)) \rightarrow q$$

p	q	…	$p \rightarrow q$	$p \wedge (p \rightarrow q)$	$(p \wedge (p \rightarrow q)) \rightarrow q$
✖	✖	…	⊨	⊭	⊨
✖	✔	…	⊨	⊭	⊨
✔	✖	…	⊭	⊭	⊨
✔	✔	…	⊨	⊨	⊨

Grundlagen Semantic Web

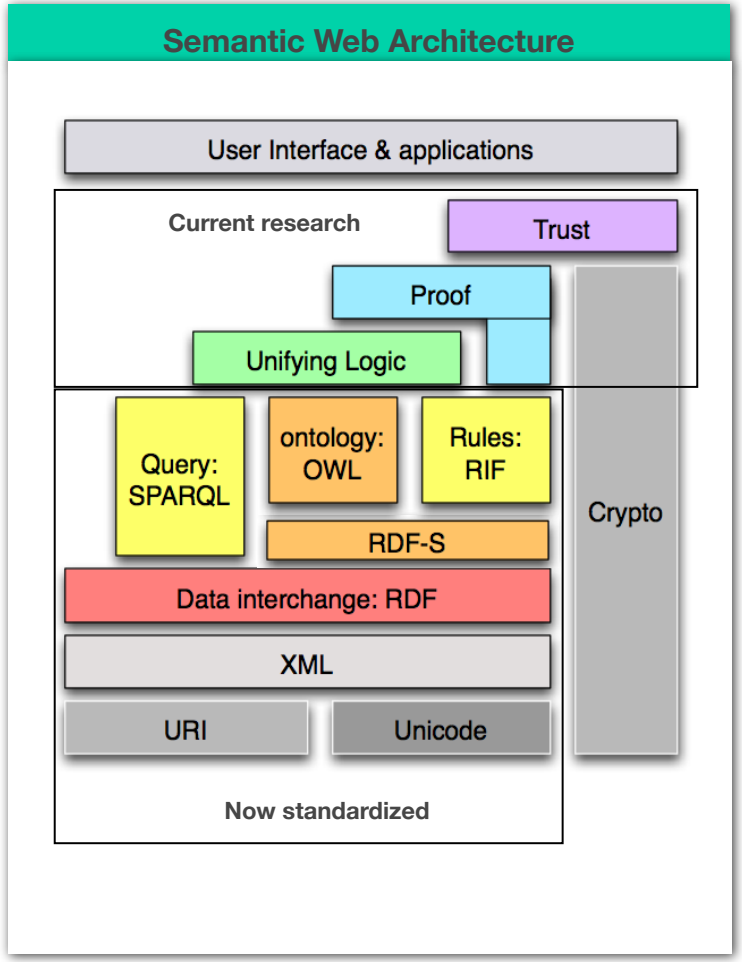
Lehrveranstaltung im WS09/10
Seminar für Computerlinguistik
Universität Heidelberg

Dr. Sebastian Rudolph
Institut AIFB
Universität Karlsruhe

AIFB

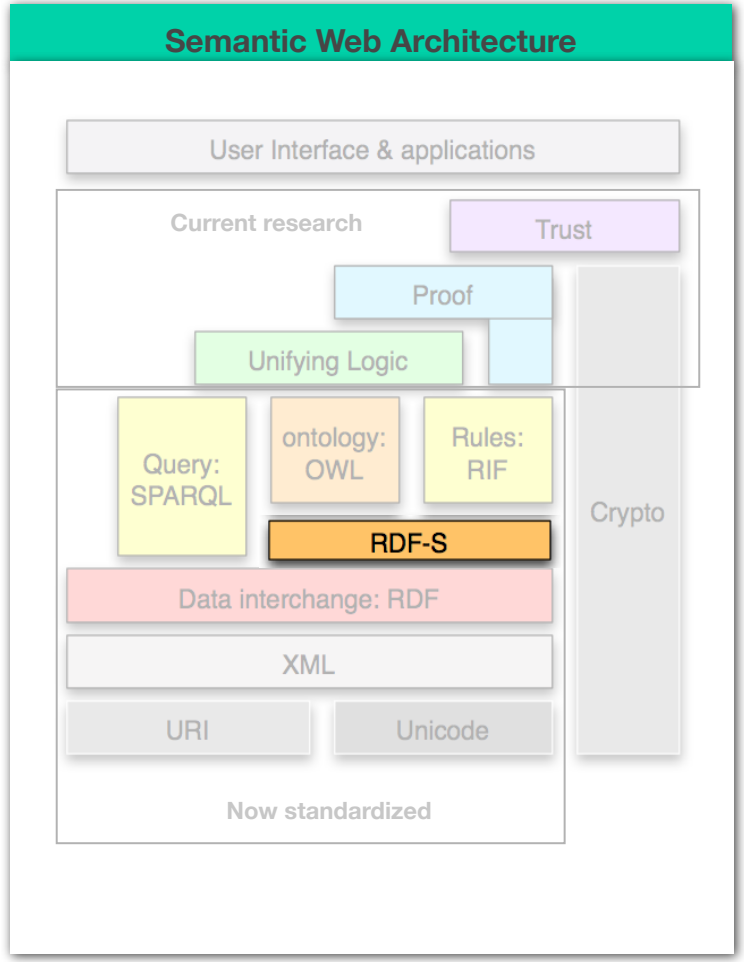
Semantik von RDF(S)

Dr. Sebastian Rudolph



Semantik von RDF(S)

Dr. Sebastian Rudolph



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?

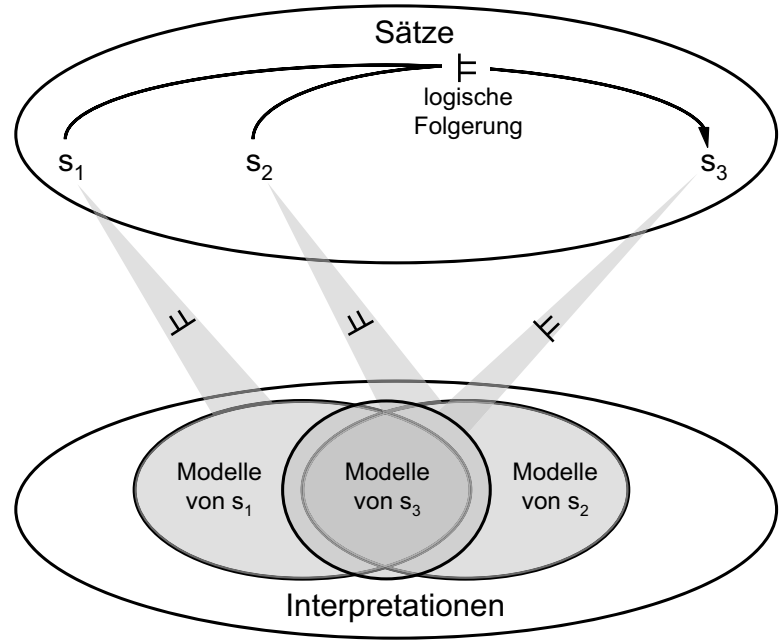
- also: was sind die Sätze in RDF(S)?
  - Grundelemente (Vokabular V): URIs, bnodes und Literale (sind selbst keine Sätze)
  - jedes Tripel

    $(s,p,o) \in$
    $(URI \cup bnode) \times URI \times (URI \cup bnode \cup Literal)$

    ist ein Satz
  - jede endliche Menge von Tripeln (genannt Graph) ist ein Satz

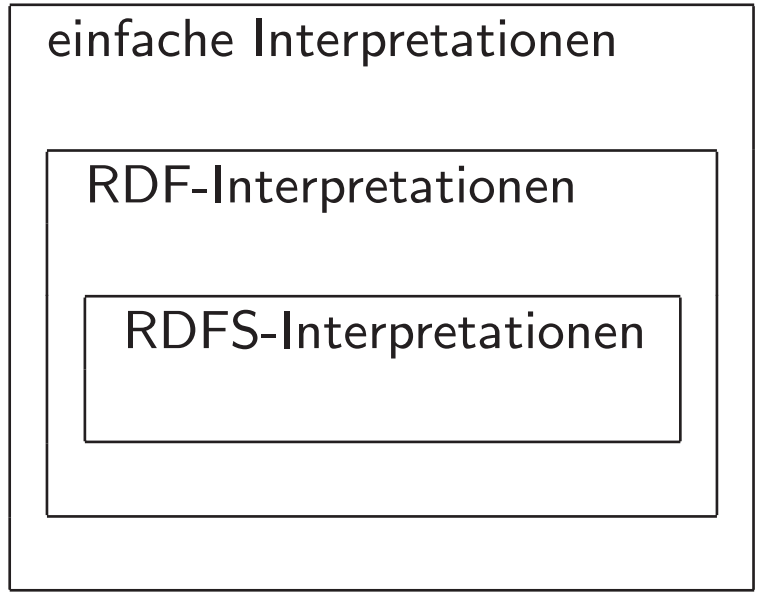
Was ist die Semantik?

- Konsequenzrelation, die sagt, wann ein RDF(S)-Graph G´ aus einem RDF(S)-Graphen G folgt, d.h. G ⊨ 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:

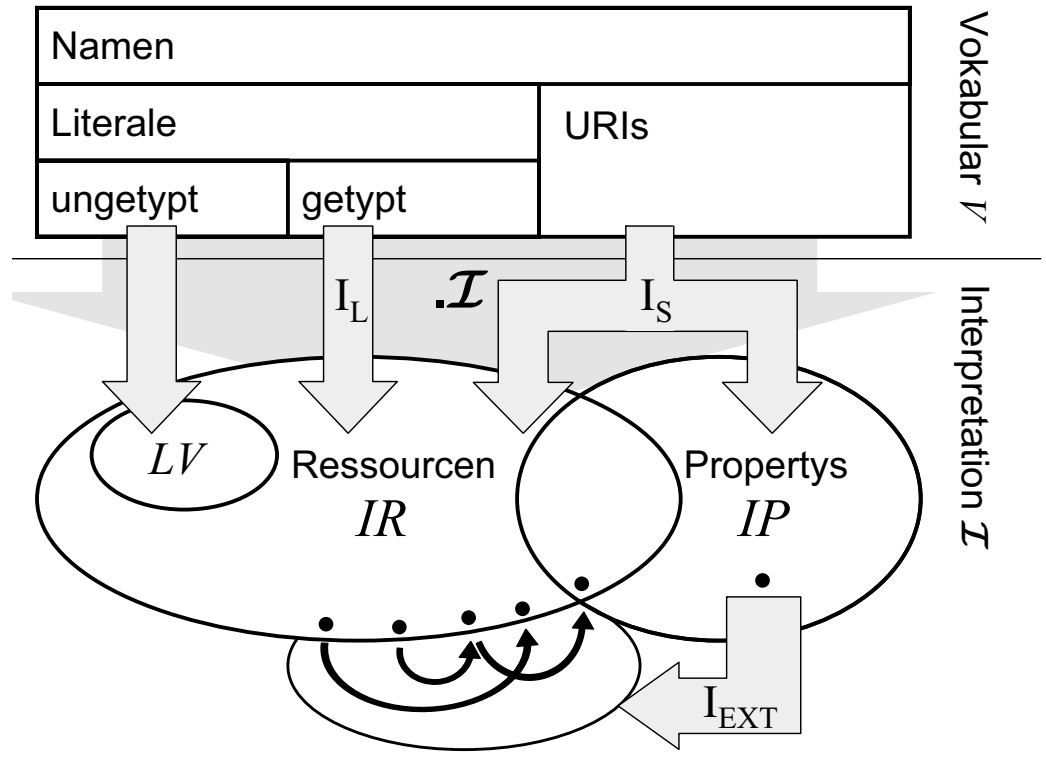
Wir definieren also: Eine *einfache Interpretation* $\mathcal{I}$ für ein Vokabular V besteht aus

- IR, einer nichtleeren Menge von *Ressourcen*, auch genannt Domäne oder (Diskurs-)Universum von $\mathcal{I}$,
- IP, der Menge der *Propertys* von $\mathcal{I}$
- $\mathrm{I_{EXT}}$, einer Funktion, welche jeder Property eine Menge von Paaren aus IR zuordnet, also $\mathrm{I_{EXT}} : IP \rightarrow 2^{IR \times IR}$, dabei nennt man $\mathrm{I_{EXT}}(p)$ auch die *Extension* der Property p,
- $\mathrm{I_S}$, einer Funktion, welche URIs aus V in die Vereinigung der Mengen IR und IP abbildet, also $\mathrm{I_S} : V \rightarrow IR \cup IP$,
- $\mathrm{I_L}$, 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: $(\texttt{"}a\texttt{"})^{\mathcal{I}} = a$,
- jedes ungetypte Literal mit Sprachangabe `"a"@t` wird auf das Paar $\langle a, t \rangle$ abgebildet: $(\texttt{"}a\texttt{"@}t)^{\mathcal{I}} = \langle a, t \rangle$,
- jedes getypte Literal l wird auf $\mathrm{I_L}(l)$ abgebildet: $l^{\mathcal{I}} = \mathrm{I_L}(l)$ und
- jede URI u wird auf $\mathrm{I_S}(u)$ abgebildet: $u^{\mathcal{I}} = \mathrm{I_S}(u)$.

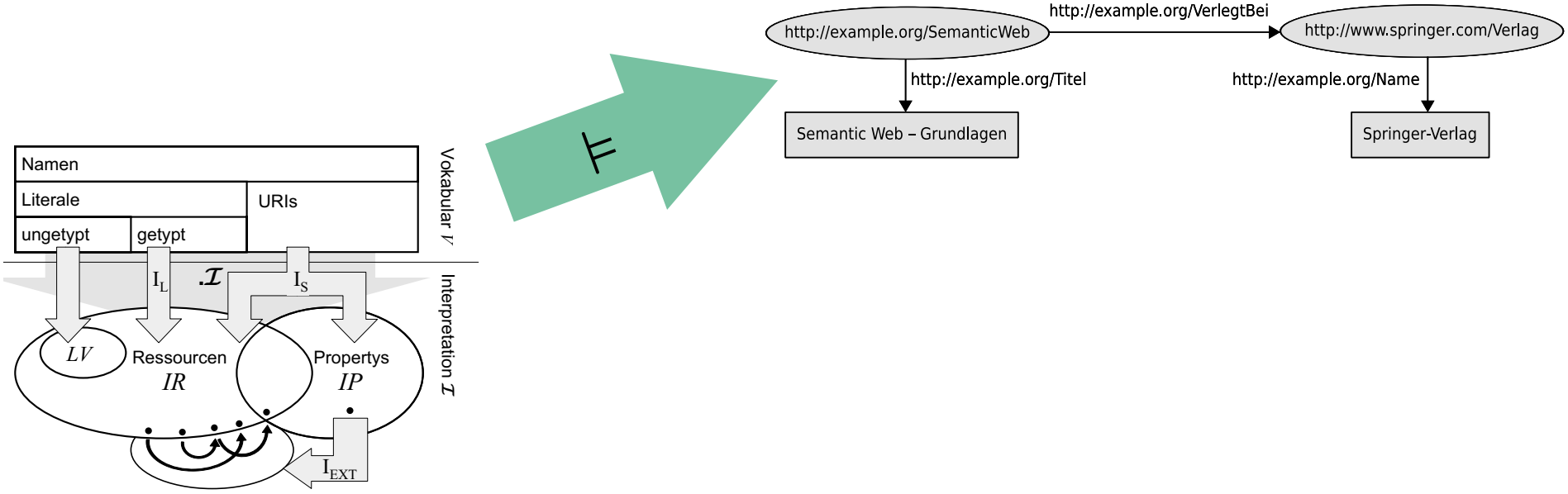
- Interpretation (schematisch):



Semantik der einfachen Folgerung

AIFB

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

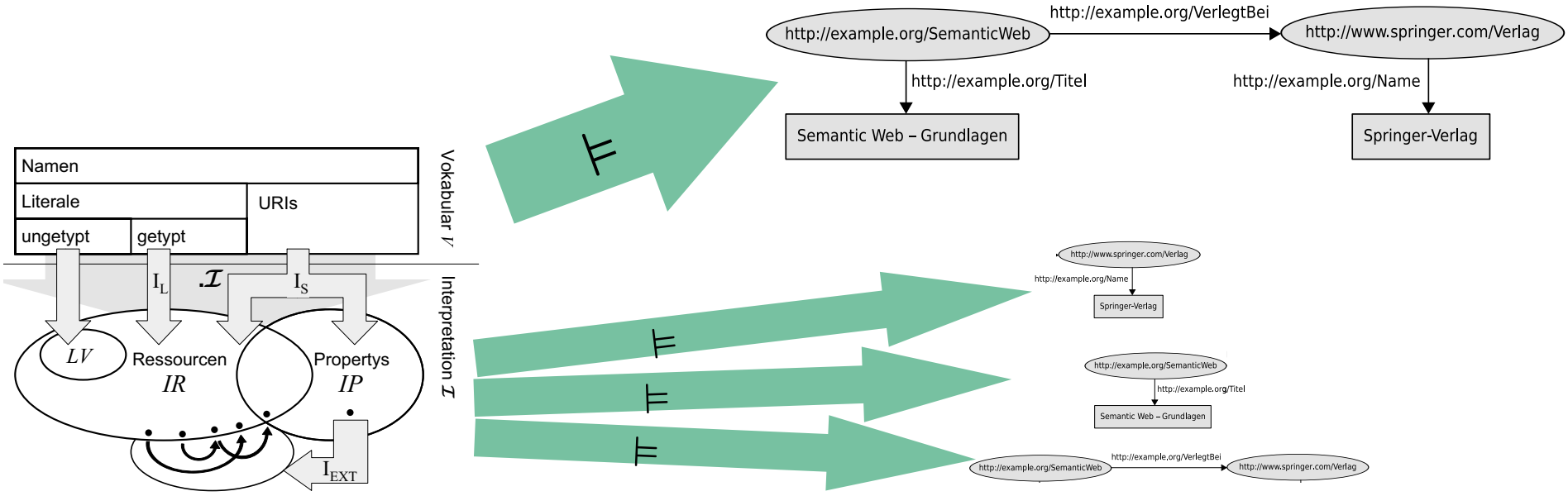


- ...wenn sie Modell jedes Tripels des Graphen ist!

Semantik der einfachen Folgerung

AIFB

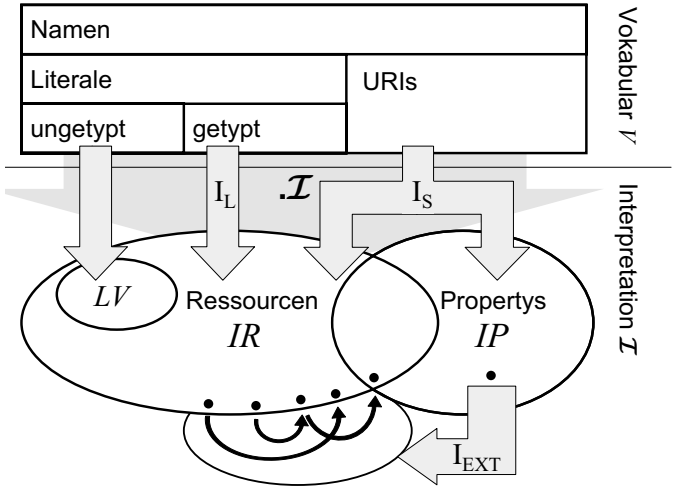
- Frage: Wann ist eine gegebene Interpretation Modell eines Graphen?

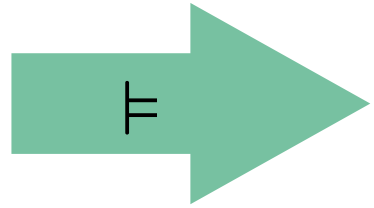


- ...wenn sie Modell jedes Tripels des Graphen ist!

Semantik der einfachen Folgerung

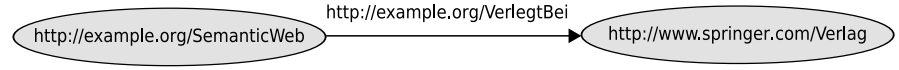
- Frage: Wann ist eine gegebene Interpretation Modell eines Tripels?





Tripel

s p o .

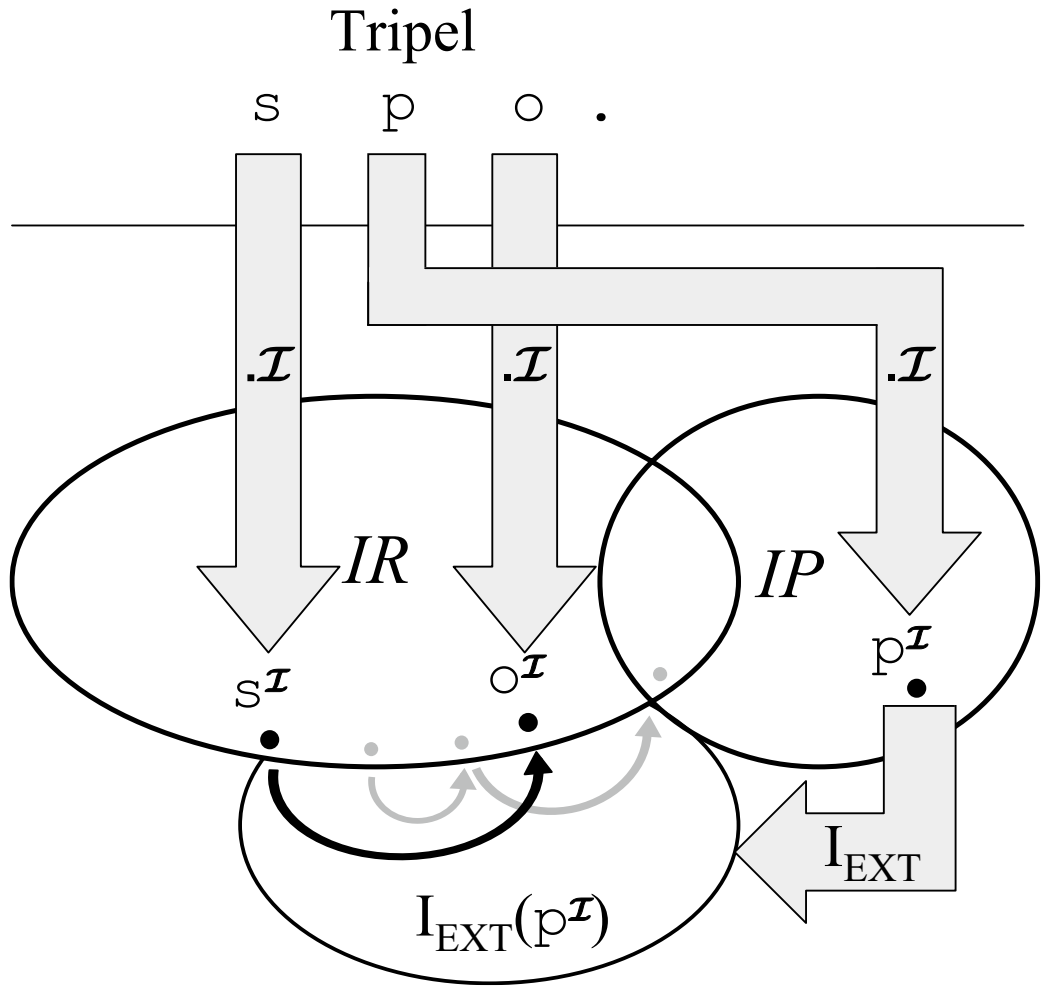


- ...wenn Subjekt, Prädikat und Objekt in V enthalten sind und außerdem:

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

Semantik der einfachen Folgerung

- schematisch:

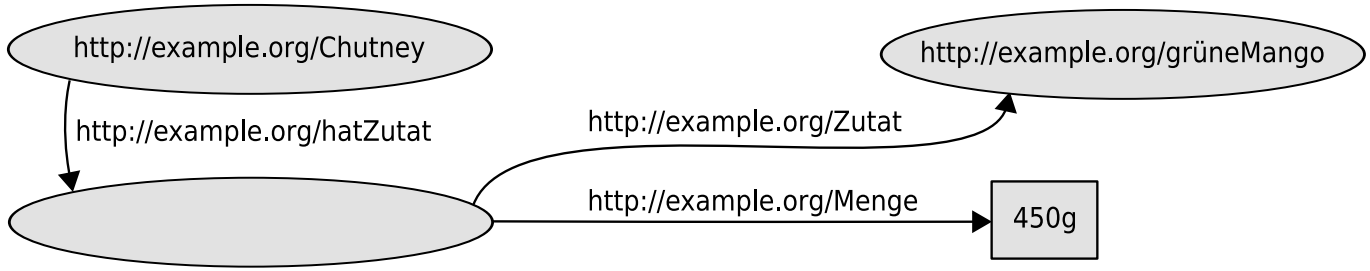


Semantik der einfachen Folgerung

- ...ups, wir haben die bnodes vergessen!
- wird nachgeholt: sei A Funktion, die alle bnodes auf Elemente von IR 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 $\mathcal{I}$:

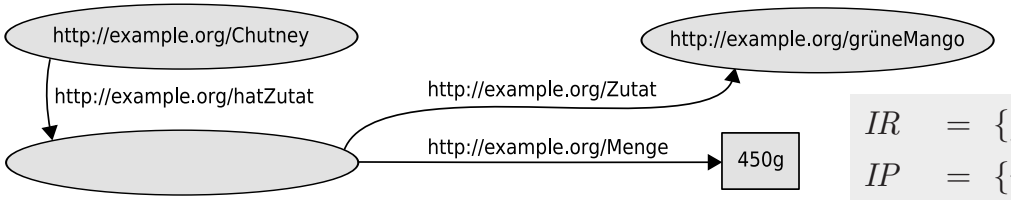
$$
\begin{aligned}
IR &= \{\chi, \upsilon, \tau, \nu, \epsilon, \iota, 450g\} \\
IP &= \{\tau, \nu, \iota\} \\
LV &= \{450g\} \\
\mathrm{I_{EXT}} &= \tau \mapsto \{\langle \chi, \epsilon \rangle\} \\
&\quad\ \nu \mapsto \{\langle \epsilon, \upsilon \rangle\} \\
&\quad\ \iota \mapsto \{\langle \epsilon, 450g \rangle\}
\end{aligned}
\qquad
\begin{aligned}
\mathrm{I_S} = \ & \texttt{ex:Chutney} && \mapsto \chi \\
& \texttt{ex:grüneMango} && \mapsto \upsilon \\
& \texttt{ex:hatZutat} && \mapsto \tau \\
& \texttt{ex:Zutat} && \mapsto \nu \\
& \texttt{ex:Menge} && \mapsto \iota
\end{aligned}
$$

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

- ...ist $\mathcal{I}$ ein Modell von *G*?

Einfache Interpretation: Beispiel

AIFB



$$
\begin{aligned}
IR &= \{\chi, \upsilon, \tau, \nu, \epsilon, \iota, 450g\} \\
IP &= \{\tau, \nu, \iota\} \\
LV &= \{450g\} \\
\mathrm{I_{EXT}} &= \tau \mapsto \{\langle \chi, \epsilon \rangle\} \\
&\quad\ \nu \mapsto \{\langle \epsilon, \upsilon \rangle\} \\
&\quad\ \iota \mapsto \{\langle \epsilon, 450g \rangle\}
\end{aligned}
\qquad
\begin{aligned}
\mathrm{I_S} = \ & \texttt{ex:Chutney} \mapsto \chi \\
& \texttt{ex:grüneMango} \mapsto \upsilon \\
& \texttt{ex:hatZutat} \mapsto \tau \\
& \texttt{ex:Zutat} \mapsto \nu \\
& \texttt{ex:Menge} \mapsto \iota \\
\mathrm{I_L} \quad & \text{ist die „leere Funktion“, da es keine getypten Literale gibt.}
\end{aligned}
$$

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

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

- 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:

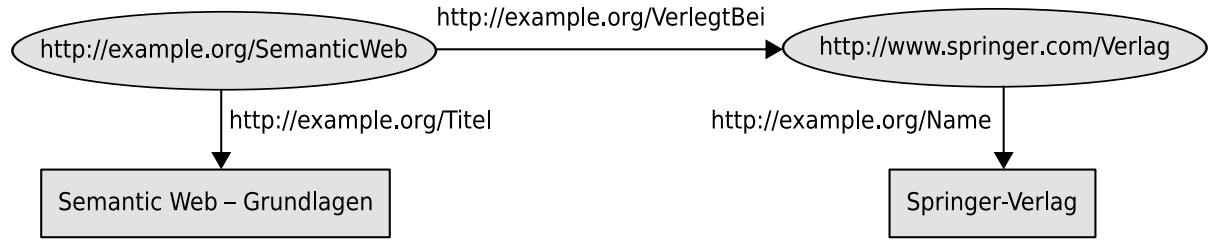
$$\frac{u \;\; a \;\; x \;\; .}{u \;\; a \;\; \_:n \;\; .}\ \text{se1}$$

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

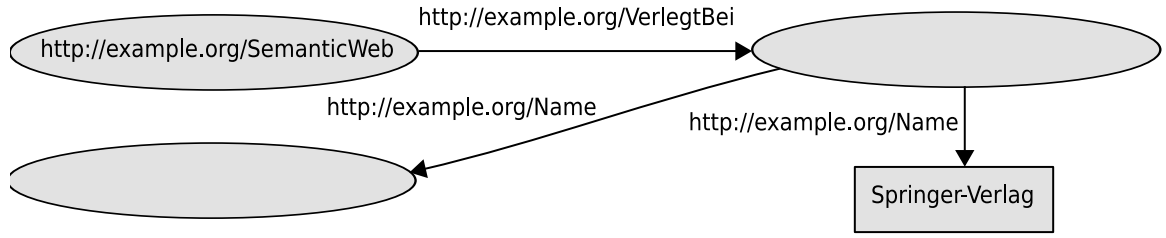
- Bedingung für Anwendung: leerer Knoten nicht bereits anderer URI / anderem Literal zugeordnet

Einfache Folgerung

- Satz: Ein Graph G_2 folgt einfach aus einem Graphen G_1, wenn G_1 mithilfe der Regeln se1 und se2 zu einem Gaphen G_1' ergänzt werden kann, so dass G_2 in G_1' enthalten ist.
- Bsp.: aus



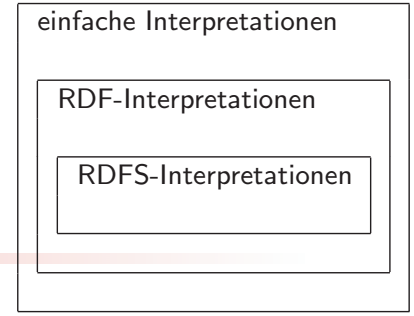
folgt einfach



Agenda

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

RDF-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 $\langle x, \texttt{rdf:Property}^{\mathcal{I}} \rangle \in \text{I}_{\text{EXT}}(\texttt{rdf:type}^{\mathcal{I}})$.
  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}(`"`*s*`"^^rdf:XMLLiteral`) ist der XML-Wert[1] von *s*;
  - I_{L}(`"`*s*`"^^rdf:XMLLiteral`) $\in LV$;
  - $\langle \text{I}_{\text{L}}(\texttt{"}s\texttt{"\^\^rdf:XMLLiteral}), \texttt{rdf:XMLLiteral}^{\mathcal{I}} \rangle \in \text{I}_{\text{EXT}}(\texttt{rdf:type}^{\mathcal{I}})$
- wenn `"`*s*`"^^rdf:XMLLiteral` in V enthalten und *s* ein *nicht* wohlgeformtes XML-Literal ist, dann
  - I_{L}(`"`*s*`"^^rdf:XMLLiteral`) $\notin LV$ und
  - $\langle \text{I}_{\text{L}}(\texttt{"}s\texttt{"\^\^rdf:XMLLiteral}), \texttt{rdf:XMLLiteral}^{\mathcal{I}} \rangle \notin \text{I}_{\text{EXT}}(\texttt{rdf:type}^{\mathcal{I}})$.

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

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

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

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

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

RDF-Folgerung

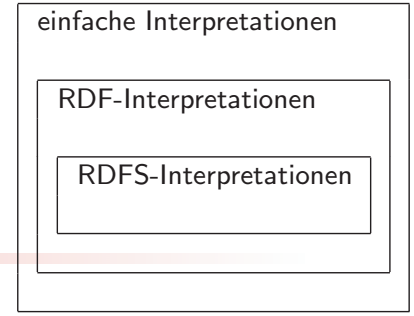
- Satz: Ein Graph G_2 RDF-folgt aus einem Graphen G_1, wenn es einen Graphen G_1' gibt, so dass
  - G_1' aus G_1 via lg, rdf1, rdf2 und rdfax hergeleitet werden kann und
  - G_2 aus G_1' einfach folgt.

- Beachte: zweistufiger Folgerungsprozess.

Agenda

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

RDFS-Interpretationen



- ...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:

- $IR = \text{I}_{\text{CEXT}}(\texttt{rdfs:Resource}^{\mathcal{I}})$
  Jede Ressource ist vom Typ `rdfs:Resource`.
- $LV = \text{I}_{\text{CEXT}}(\texttt{rdfs:Literal}^{\mathcal{I}})$
  Jedes ungetypte und jedes wohlgeformte getypte Literal ist vom Typ `rdfs:Literal`.
- Wenn $\langle x, y\rangle \in \text{I}_{\text{EXT}}(\texttt{rdfs:domain}^{\mathcal{I}})$ und $\langle u, v\rangle \in \text{I}_{\text{EXT}}(x)$, dann $u \in \text{I}_{\text{CEXT}}(y)$.
  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 $\langle x, y\rangle \in \text{I}_{\text{EXT}}(\texttt{rdfs:range}^{\mathcal{I}})$ und $\langle u, v\rangle \in \text{I}_{\text{EXT}}(x)$, dann $v \in \text{I}_{\text{CEXT}}(y)$.
  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.
- $\text{I}_{\text{EXT}}(\texttt{rdfs:subPropertyOf}^{\mathcal{I}})$ 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 $\langle x, y \rangle \in \mathrm{I}_{\mathrm{EXT}}(\texttt{rdfs:subPropertyOf}^{\mathcal{I}})$,
  dann $x, y \in IP$ und $\mathrm{I}_{\mathrm{EXT}}(x) \subseteq \mathrm{I}_{\mathrm{EXT}}(y)$.
  Wird x mit y durch `rdfs:subPropertyOf` verbunden, dann sind sowohl x als auch y Propertys und jedes in der Extension von x enthaltene Ressourcenpaar ist auch in der Extension von y enthalten.
- Wenn $x \in IC$,
  dann $\langle x, \texttt{rdfs:Resource}^{\mathcal{I}} \rangle \in \mathrm{I}_{\mathrm{EXT}}(\texttt{rdfs:subClassOf}^{\mathcal{I}})$.
  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 $\langle x, y \rangle \in \mathrm{I}_{\mathrm{EXT}}(\texttt{rdfs:subClassOf}^{\mathcal{I}})$,
  dann $x, y \in IC$ und $\mathrm{I}_{\mathrm{CEXT}}(x) \subseteq \mathrm{I}_{\mathrm{CEXT}}(y)$.
  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.
- $\mathrm{I}_{\mathrm{EXT}}(\texttt{rdfs:subClassOf}^{\mathcal{I}})$ 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 $x \in \mathrm{I}_{\mathrm{CEXT}}(\texttt{rdfs:ContainerMembershipProperty}^{\mathcal{I}})$,
  dann $\langle x, \texttt{rdfs:member}^{\mathcal{I}} \rangle \in \mathrm{I}_{\mathrm{EXT}}(\texttt{rdfs:subPropertyOf}^{\mathcal{I}})$.
  Ist x eine Property vom Typ `rdfs:ContainerMembershipProperty`, so steht sie in der `rdfs:subPropertyOf`-Beziehung zur `rdfs:member`-Property.
- Wenn $x \in \mathrm{I}_{\mathrm{CEXT}}(\texttt{rdfs:Datatype}^{\mathcal{I}})$,
  dann $\langle x, \texttt{rdfs:Literal}^{\mathcal{I}} \rangle \in \mathrm{I}_{\mathrm{EXT}}(\texttt{rdfs:subClassOf}^{\mathcal{I}})$
  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

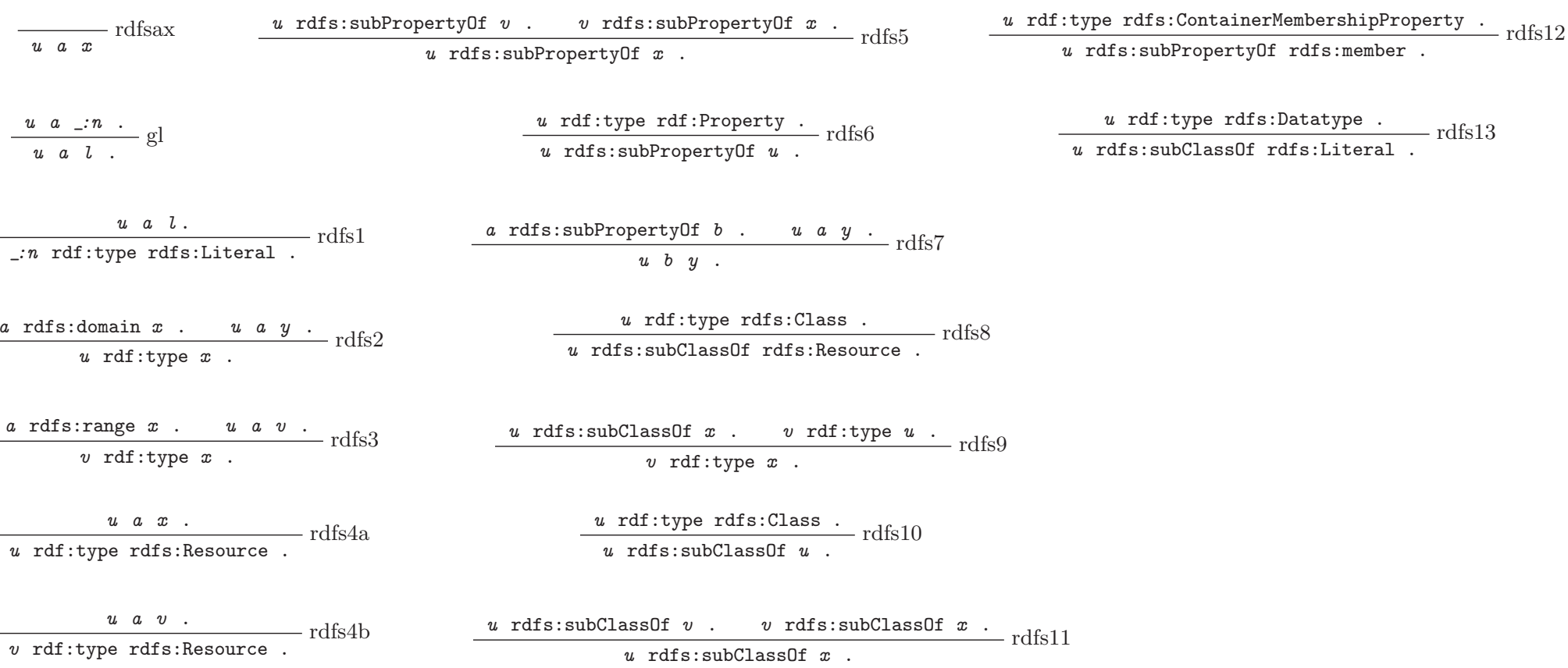

```
rdf:type              rdfs:domain          rdfs:Resource .
rdfs:domain           rdfs:domain          rdf:Property .
rdfs:range            rdfs:domain          rdf:Property .
rdfs:subPropertyOf    rdfs:domain          rdf:Property .
rdfs:subClassOf       rdfs:domain          rdfs:Class .
rdf:subject           rdfs:domain          rdf:Statement .
rdf:predicate         rdfs:domain          rdf:Statement .
rdf:object            rdfs:domain          rdf:Statement .
rdfs:member           rdfs:domain          rdfs:Resource .
rdf:first             rdfs:domain          rdf:List .
rdf:rest              rdfs:domain          rdf:List .
rdfs:seeAlso          rdfs:domain          rdfs:Resource .
rdfs:isDefinedBy      rdfs:domain          rdfs:Resource .
rdfs:comment          rdfs:domain          rdfs:Resource .
rdfs:label            rdfs:domain          rdfs:Resource .
rdf:value             rdfs:domain          rdfs:Resource .

rdf:type              rdfs:range           rdfs:Class .
rdfs:domain           rdfs:range           rdfs:Class .
rdfs:range            rdfs:range           rdfs:Class .
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:ContainerMembershipProperty
                      rdfs:subClassOf      rdf:Property .
rdf:Alt               rdfs:subClassOf      rdfs:Container .
rdf:Bag               rdfs:subClassOf      rdfs:Container .
rdf:Seq               rdfs:subClassOf      rdfs:Container .

rdfs:isDefinedBy      rdfs:subPropertyOf   rdfs:seeAlso .

rdf:XMLLiteral        rdf:type             rdfs:Datatype .
rdf:XMLLiteral        rdfs:subClassOf      rdfs:Literal .
rdfs:Datatype         rdfs:subClassOf      rdfs:Class .

rdf:_1                rdf:type
                         rdfs:ContainerMembershipProperty .
rdf:_1                rdfs:domain          rdfs:Resource .
rdf:_1                rdfs:range           rdfs:Resource .
rdf:_2                rdf:type
                         rdfs:ContainerMembershipProperty .
rdf:_2                rdfs:domain          rdfs:Resource .
rdf:_2                rdfs:range           rdfs:Resource .
...
```

RDFS-Folgerung

AIFB

- automatische Folgerungen werden wieder über Ableitungsregeln realisiert:



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, rdfax, 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