Semantic Web

Jan Hladik
mit Material von Birte Glimm, Universität Ulm

Inhalt

Inhalt

Dozent: Jan Hladik

1993–2001 Diplom Informatik, RWTH Aachen
- Diplomarbeit: Tableau-Algorithmus
- Nebenfach: Philosophie

2001–2007 Promotion, TU Dresden
- Dissertation: Tableaus vs. Automaten
- Leitung von Übungsgruppen

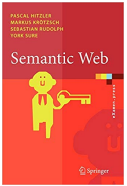
2008–2014 Industrieerfahrung, SAP Research Dresden
- Öffentlich geförderte Forschungsprojekte
- Betreuung von Studenten und Doktoranden

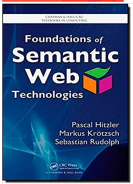
seit 2014 Professor, DHBW Stuttgart
- Logik
- Formale Sprachen und Automaten
- Semantic Web

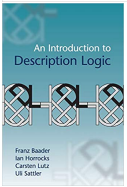
Forschung Semantische Technologien
- Beschreibungslogik
- logische Schlussfolgerungsverfahren
- Semantic Web

Vorlesungsmaterial

- Präsentation, Übungsaufgaben
  - `http://wwwlehre.dhbw-stuttgart.de/~hladik/SW`
- Klausur
  - Dauer: 60 min
  - Hilfsmittel: Formelsammlung 10 Seiten DIN A4
- Literatur
  - Hitzler, Krötzsch, Rudolph, Sure:
    „*Semantic Web — Grundlagen*“, Springer-Verlag, 2008; ca. 40 €
    - deutsch
    - ausführliche Behandlung von XML
    - Übungen / Beispiele
  - Hitzler, Krötzsch, Rudolph:
    „*Foundations of Semantic Web Technologies*“, CRC Press, 2009; ca. 95 €
    - englisch
    - aktualisiert und erweitert: OWL 2.0, Profiles, Rules
  - Baader, Horrocks, Lutz, Sattler:
    An Introduction to Description Logic, Cambridge University Press, 2017; ca. 45 €
    - Grundlagen
    - Schlussfolgerungsverfahren
    - Komplexität







Struktur der Vorlesung

Inhalt

Wissensrepräsentations-Systeme

Brian C. Smith

Wissensrepräsentations-Hypothese (Smith, 1982):

1. Wissen ist in einer Form gespeichert, die für einen äußeren Beobachter verständlich ist.
2. Gespeichertes Wissen spielt wesentliche kausale Rolle im Verhalten des Systems.

ohne 1: Jedes Programm ist WR-System
- Wissen zum Erfüllen der Aufgabe implizit enthalten

ohne 2: Gespeichertes Wissen ist nur Dekoration
- Mind-Map
- Kommentare

Komponenten von WR-Systemen

Wissensbank speichert formalisiertes Wissen (knowledge base, KB)
- WR-Sprache mit Syntax und Semantik

Inferenzmaschine leitet implizites Wissen ab (inference engine, IE)
- Schlussfolgerungsverfahren

Ziele für Wissensrepräsentation:
- Ausdrucksstärke ausreichend für Domäne
- Schlussfolgerungsprobleme (effizient) entscheidbar
- Syntax leicht erlernbar

Logik als WR-System

Bisher behandelte Systeme:

- Aussagen-/ Prädikatenlogik
  - Wissensbank: Formel in Syntax der Aussagen- oder Prädikatenlogik
  - Inferenzmaschine: Erfüllbarkeitstest mit Resolution oder Tableau
- Prolog
  - Wissensbank: Prolog-KB
  - Inferenzmaschine: Prolog-Interpreter mit Horn-Resolution

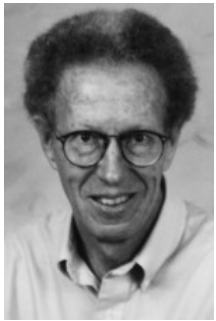
Nachteile:

- Aussagenlogik kann innere Struktur von Aussagen nicht abbilden (zu schwach)
- Erfüllbarkeit für Prädikatenlogik und Prolog ist unentscheidbar (zu stark)
- Syntax ungeeignet für Anwender ohne mathematische Vorkenntnisse

Inhalt

Semantische Netze

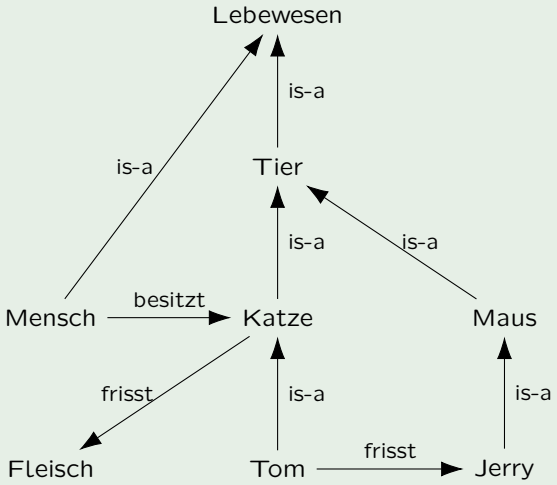
- früher Ansatz zur grafischen Wissensrepräsentation
- entwickelt 1960 von M. Ross Quillian
- WR-Sprache: beschrifteter Graph
  - Knoten: Individuen, Klassen, Eigenschaften
  - Kanten: is-a, has-prop
- Schlussfolgerungsverfahren
  - Vererbung entlang is-a-Kanten
    - ererbte Eigenschaften sind Default
    - Default kann überschrieben werden
  - Spreading Activation: Pfad zwischen Knoten interpretiert als Beziehung zwischen Entitäten („semantischer Abstand“)



M. Ross Quillian

Beispiel: Semantisches Netz

Beispiel 1.1



Schlussfolgerungen

- Tom ist ein Tier
- Tom frisst eine Maus
- Gemeinsamkeiten
  - Katzen und Mäuse sind Tiere
  - Menschen und Mäuse sind Lebewesen

Schwierigkeiten

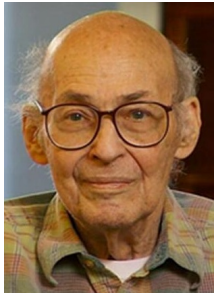
- Besitzt jeder Mensch eine Katze?
- Gehört jede Katze einem Menschen?
- Ist Jerry Fleisch?
- Fressen Katzen
  - nur Fleisch,
  - unter anderem Fleisch,
  - normalerweise Fleisch?
- Was bedeutet „is-a“?

Probleme Semantischer Netze

- Mehrdeutigkeit von is-a
  - Individuum → Klasse: Element
  - Klasse → Klasse: Teilmenge
  - „*What is-a is and isn't*“ (Ron Brachman, 1983)
- Vererbung als Default
  - Jede Eigenschaft kann überschrieben werden
  - „Viereck ist Dreieck mit vier Ecken“
  - keine terminologische Information
  - „*I lied about the trees*“ (Ron Brachman, 1985)
- „Semantischer Abstand“ abhängig von syntaktischen Details
  - Tom → Katze → Tier
    ⇝ Tom ist eng verwandt mit Tier
  - Tom → Perserkatze → Katze → Katzenartige → Raubtier → Wirbeltier → Tier
    ⇝ Tom ist entfernt verwandt mit Tier
- keine formale Semantik!
- Netzwerk bedeutet, was der Ersteller meint
- Konflikte zwischen Benutzer und Programmierer nicht entscheidbar

Frames: Ursprung und Intention

- WR-Formalismus zur Beschreibung von Entitäten
- entwickelt 1975 von Marvin Minsky
- Ziel: Imitation von menschlichem Kategorisieren
- Ablehnung von logischem Schließen
  - „nicht vital und kreativ“
  - „nicht flexibel“
  - keine Möglichkeit, „normalerweise“ auszudrücken
  - „Monotonität“: Einmal ableitbare Folgerungen können nicht durch neue Informationen verhindert werden
  - Trennung von repräsentiertem Wissen und Schlussfolgerungsverfahren „zu radikal“
  - Bestehen auf Korrektheit (Verhinderung falscher Folgerungen) „unglaublich zerstörerisch“
- KI als „Sammlung von Heuristiken“
  - „KI ist, was an KI-Instituten entwickelt wird.“
  - Ähnlich dem Widerstand gegen logische Fundierung von mathematischen Beweisen (frühes 20. Jahrhundert)



Marvin Minsky
(1927-2016)

Frames: Syntax und Schlussfolgerungen

- Syntax:
  - Class Frame beschreibt Klasse von Entitäten
  - Instance Frame beschreibt spezifische Entität
  - Frame enthält slots für bestimmte Attribute
  - Wert eines slots wird filler genannt
  - Defaults für filler sind möglich
  - Sub-frames für Spezialisierung ($\rightsquigarrow$ is-a)
- Schlussfolgerungsverfahren
  - Vererbung: Sub-frames ererben Default-Filler
  - Kriterialität: Wenn ein Instance Frame I für alle Slots eines Class Frame C passende Filler hat, ist I eine Instanz C
  - Matching: Test, ob Instanz eines Frames C_1 auch Instanz eines anderen Frames C_2 ist

Probleme von Frames

- Frame-Inferenzen lassen sich in Prädikatenlogik ausdrücken (Pat Hayes, 1979)
- Wie bei Semantischen Netzen
  - Keine Taxonomie, wenn alle Eigenschaften nur Defaults sind
  - Ohne formale Semantik kein „richtig“ und „falsch“
- KI als Sammlung von Heuristiken
  - keine Wissenschaft, nur Programmiertechnik
  - Frames wichtiger für Objekt-orientiertes Programmieren als für WBS

Probleme früher WR-Systeme

fehlende formale Semantik	unklare Bedeutung des Netzwerks unzuverlässige Schlussfolgerungsverfahren Inkompatibilität
Graph-basierte Algorithmen	Ergebnis abhängig von irrelevanter Information

Konsequenz: Logik-basierte WR-Systeme

- formale Semantik angelehnt an mathematische Logik
- „Semantische Technologien“
- oft: entscheidbare Fragmente der Prädikatenlogik
- oft: einfachere, intuitiv verständliche Syntax, z.B. keine Variablen

Inhalt

Syntax und Semantik

Syntax
- „gemeinsame Ordnung“
- legt zulässige Struktur von Daten fest
- beschreibt, was „wohlgeformte“ Daten sind.

Semantik
- „Bedeutung“
- legt fest, was ein syntaktisch korrekter Satz aussagt
- bestimmt, welche Schlussfolgerungen sich aus ihm ziehen lassen

Beispiel 1.2 (Arithmetik)

$4+) = ($	$3 + 4 = 12$	$3 + 4 = 7$
syntaktisch falsch	syntaktisch richtig	syntaktisch richtig
–	semantisch falsch	semantisch wahr

Begriffsbestimmungen

- Semantische Technologien
  - Teilbereich der KI
  - formale Beschreibung einer Domäne
  - klar definierte Semantik
  - Schlussfolgerungsverfahren
  - Begriff „ST“ seit Ende der 1990er Jahre
  - Technologien seit 1980er Jahren
    - Wissensrepräsentation
    - Wissensbasierte Systeme
    - Expertensysteme
- Semantic Web
  - Tim Berners-Lee et. al., 1994
  - Erweiterung des WWW durch ST
  - Technik: zuerst RDF(S), später OWL
- Linked Data
  - „Semantic Web Done Right“ (TBL, 2006)
  - Schwerpunkt: Vernetzung von Quellen
  - Linked Open Data: frei verfügbar

> *The Semantic Web will bring structure to the meaningful content of Web pages [. . .] A software agent coming to a clinic's Web page will know not just that the page has keywords such as "treatment, medicine, physical, therapy" but also that Dr. Hartman works at this clinic on Mondays, Wednesdays and Fridays and that the script takes a date range in yyyy-mm-dd format and returns appointment times.*
>
> Berners-Lee et al., Scientific American, 2001

Ontologie in der Philosophie

- „Lehre vom Seienden“
  - unvergängliche und unveränderliche Ideen
  - Gegenbegriff: Werdendes
    - Vergängliche materielle Dinge
    - Teilhabe an Ideen
  - Grundlage für Beziehung Klasse $\leftrightarrow$ Instanz
- Begriff beschreibt philosophische Disziplin und existiert nur im Singular
  - die Ontologie, wie „die Physik“
  - nicht „eine Ontologie“ oder „Ontologien“
- Zu finden bei Platon, Aristoteles, Thomas von Aquin, Descartes, Kant, Hegel, Wittgenstein, Heidegger, Quine, . . .

Ontologien in der Informatik

Wissensbank technische Komponente eines WR-Systems
Ontologie repräsentiertes Wissen; Inhalt der Wissensbank (Plural möglich)

> *An ontology is an formal and explicit representation of a shared conceptualisation.*
>
> Tom Gruber, 1993; Willem Borst, 1997

conceptualisation
- The objects, concepts, and other entities that are assumed to exist in some area of interest and the relationships that hold among them.
- An abstract, simplified view of the world that we wish to represent for some purpose. [Genesereth and Nilsson, 1987]

formal
- In formaler Sprache
- verständlich für Maschinen
- keine natürliche Sprache

explicit
- Alles, was gemeint ist, wird auch gesagt
- Bedeutung von Begriffen wird nicht als bekannt vorausgesetzt
- keine intuitive Semantik

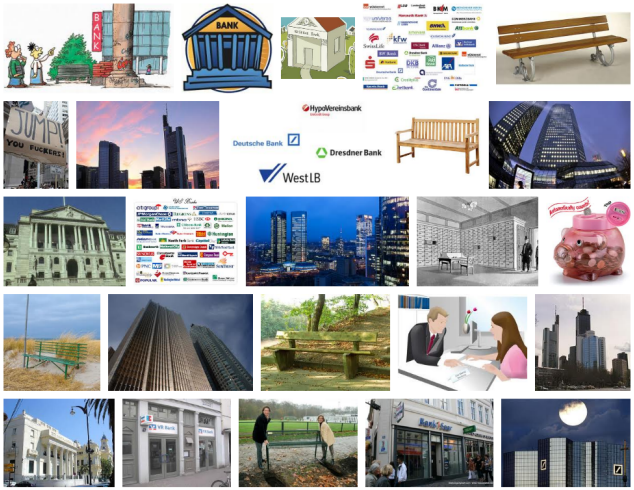
shared
- repräsentiert Sichtweise einer Gemeinschaft, beruht auf Konsens
- kein persönliches Weltbild

Ontologien in der Praxis

- Instanziierung von Klassen durch Individuen
- Klassenhierarchien (Taxonomien, „Vererbung“):
- Binäre Relationen zwischen Individuen: Properties
- Eigenschaften von Relationen (z.B. range, transitive)
- Datentypen (z.B. Zahlen, Strings)
  - Unterschied zu Individuen: Semantik ist vorgegeben
  - `Anna kennt Paul` vs. $5 > 3$
- Logische Ausdrucksmittel: Junktoren, Quantoren
- Klare Semantik

Inhalt

Beispiel 1.3 (Erste 25 Treffer der Schlagwort-Suche nach „Bank“)



- 13x Kreditinstitut
- 4x Möbel
- 3x Logos von Banken
- 3x Menschen und Banken/Bänke (von jeder Sorte, einmal sogar beide)
- 1x Sparschwein („piggy bank“)
- 1x Kommentar zu Banken und Bankmitarbeitern

Problem: Namen von Individuen

Beispiel 1.4 (Erste 27 Treffer der Schlagwort-Suche nach „Franz Baader“)

- 5x Prof. Franz Baader
- 5x Franz Xaver v. Baader (Religionsphilosoph)
- 2x Eishockey-Schiedsrichter Franz Baader
- 2x Franz R. und C. Baader
- 4 Gruppen (eine ohne FB)
- 3 Empfänger von Preisen
- 3 Mitarbeiter
- 2 weitere Franz Baader
- 1x Alexandra Maria Lara: Veranstaltungskalender mit
  - Baader-Meinhof-Komplex
  - franz-arab Disco

Problem: Synonyme

Beispiel 1.5 (Suche nach „Eisenbahn“ und „Zug“)

Eisenbahn

Zug

- Begriffe sind gleichbedeutend
- Jeder Begriff liefert nur Teil der interessanten Links

Problem: Unterschiedliche Aufbereitung von Informationen

Beispiel 1.6 (Persönliche Homepages)

UNIVERSITY OF OXFORD

DEPARTMENT OF
COMPUTER
SCIENCE

Our p

Stumble

You are here: Home › People › Ian Horrocks

Ian Horrocks

Professor Ian Horrocks FRS
Professor of Computer Science
Fellow, Oriel College
ian.horrocks@cs.ox.ac.uk
+44 1865 273939
+44 1865 273839 (fax)
Wolfson Building, Parks Road, Oxford OX1 3QD

- Gleiche Information
- Unterschiedliches Markup
- Unterschiedliche Sprachen

Beispiel 1.7 (Homepage in HTML)

```
<h1>Ian Horrocks</h1>
<table><tr>
  <td class="personImg">
    <img src="IH005-1.jpg" alt="Photo Ian Horrocks"/>
  </td>
  <td>
    <div class="personinfo">
      <div>Professor Ian Horrocks FRS</div>
      <div>Professor of Computer Science</div>
      <div>Fellow, <a href="http://www.oriel.ox.ac.uk">
        Oriel College</a></div>
      <div>ian.horrocks@cs.ox.ac.uk</div>
      <div>+44 1865 273939</div>
      <div>+44 1865 273839 (fax)</div>
    </div>
    <p>Wolfson Building, Parks Road, Oxford OX1 3QD</p>
  </td>
</tr></table>
```

- HTML legt nur Syntax fest
  - Text
  - Layoutinformationen
- Suchmaschine muss Semantik raten
  - Wörter, die wie Namen aussehen
  - Bild in der Nähe eines Namens
  - Zahlen, die wie Telefonnummern aussehen
  - Firmennamen
  - Adressen

Beispiel 1.8 (Automatische Verschlagwortung einer Online-Zeitung)

Gutachten über die Eignung des Marstallgeländes bestellt. Der Verein schlug den Gutachter vor: Yasuhisa Toyota von der Firma Nagata Acoustics in Tokio.

Am 11. Mai 2010 unterrichtete Kunstminister Wolfgang Heubisch von der FDP das Kabinett und die Öffentlichkeit über die Ergebnisse des Gutachtens. Die

Alle Nachrichten und Informationen der F.A.Z. zum Thema **Toyota Motor**

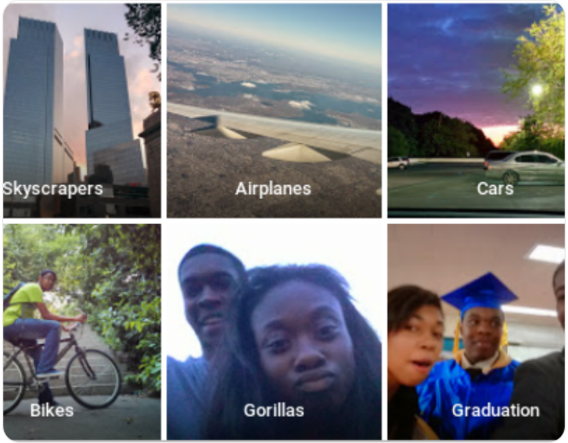
Alle Nachrichten und Informationen der F.A.Z. zum Thema **FDP**

- Automatische Erkennung von Schlagwörtern kann klappen . . .
- . . . muss aber nicht.

Problem: Bilderkennung

Beispiel 1.9 (Automatische Verschlagwortung von Fotos)

2015: Problem bei Google Photos



2017: Problem gelöst

Hmm… not seeing this clearly yet.

What can Google Lens do?

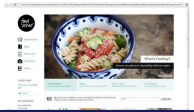
Problem: Bedeutung von Links

Beispiel 1.10 (Blog „Interactive Web Design“ verlinkt gute und schlechte Webseiten)

Examples of Good Web Design
Luckily, there are also many good website designs, some examples of which are:

1. Tinkering Monkey (http://www.tinkeringmonkey.com/)
This is an online shop managed through WordPress that sells simple wooden goods for everyday use. Everything is made in the garage-turned-woodworking-studio of Mike Cheung. The website is simple and easy to navigate, loads quickly, provides a lot of information about each product, and allows users to choose their own logo to place on the product. It also provides a section for do-it-yourself activities with detailed pictures. It even contains a blog and a live video stream of the woodworking studio. The choice of the colours is excellent and the text is quite clear.

2. Food Sense (http://foodsense.is/)
This is a website for healthy food recipes. The page layout is nice, the text is clear and navigation is easy. The photos are crisp and the pages download quickly with excellent use of white space. It allows users to contribute recipes to the website, and has a blog and a search engine.



Examples of Bad Web Design
There are many examples of bad website design, some of which are:

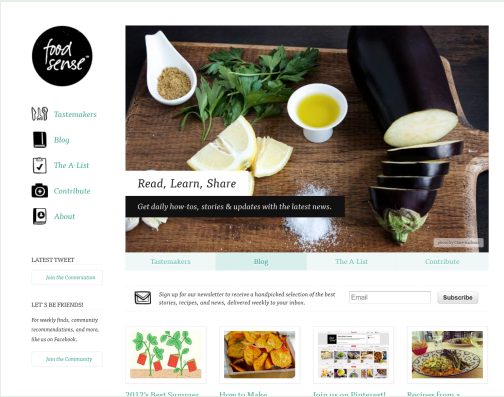
1. Penny Juice (http://www.pennyjuice.com/htmlversion/whoispj.htm)
The colours are awful with no apparent reason for using all those colours. The text is too small and hard to read with those colours in the background. The front page is merely testimonials from customers about the juice this website sells, with a few links in a small-size text that is hard to read because of the colours. The rest of the pages again use those too many colours.

2. After Life (http://heaven.internetarchaeology.org/heaven.html#bottom)
This website is nothing but animated pictures that keep scrolling upwards without any control from the user (and may malfunction in Internet Explorer). There are no links or any type of navigation whatsoever. What is the purpose of this website?!

3. Great Dreams (http://www.greatdreams.com/)
This is a clear example of bad combination of the background colour and text colour which makes the text hard to read, especially when its size is too small. The page is too long

Was der Webcrawler sieht

Interactive Web Design



- Pagerank wird für beide Seiten erhöht.

Probleme im Web of (Hyper-)Text

- Von Menschen für Menschen gemacht
  - Nicht verständlich für Maschinen
  - Räumliche Nähe als Maßstab für Enge der Beziehung
- Suche nur nach Text, nicht nach Entitäten
  - Synonyme
  - Homonyme
- Heterogenität auf verschiedenen Ebenen:
  - Zeichenkodierung (z.B. ASCII vs. Unicode)
  - verwendete natürliche Sprachen
  - Anordnung von Information auf Webseiten
  - Links sagen nichts über Art der Beziehung aus
- Kein automatisches Schlussfolgern (Reasoning)
  - Ableitung von implizitem Wissen, ggf. unter Einbeziehung externer Quellen
  - erfordert formal-logische Methoden

Anforderungen an ein intelligenteres Web

Was fehlt

- Disambiguierung von Homonymen
- Bedeutung von Links (👍 oder 👎 oder . . .)
- Beziehung zwischen einzelnen Elementen (z.B. Foto und Text)

Was gebraucht wird: Semantik

- Global eindeutige Bezeichner für Entitäten
- Methode, um Beziehungen zwischen Entitäten auszudrücken
  - mittels festgelegten und global eindeutigen Vokabulars
  - verständlich für Menschen und Maschinen

Lösungsansätze

a posteriori Verwendung von KI-Methoden zur Auswertung bestehender unstrukturierter Informationen im Web

- Verlagerung des Problems: Interpretation wird durch KI statt Suchmaschine durchgeführt
- Schlussfolgern unsicher

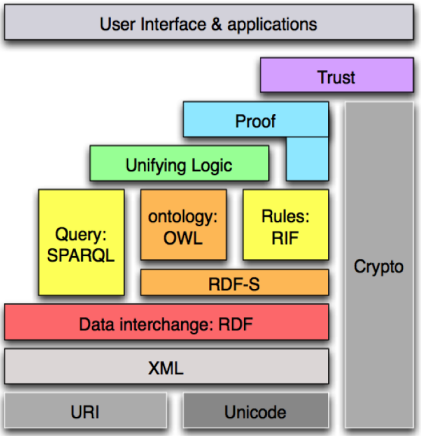
a priori Strukturierung der Web-Informationen zur Erleichterung der automatisierten Auswertung

Anforderungen

- offene Standards zur Beschreibung von Informationen
  - klar definiert
  - flexibel
  - erweiterbar
- Methoden zur Gewinnung von Informationen aus solchen Beschreibungen

Inhalt

Standards: Der Semantic Web Layer Cake



W3C Semantic Web

1994	Erste öffentliche Präsentation der Idee „Semantic Web“
1998	Beginn der Standardisierung des Datenmodells (RDF) und einer ersten Ontologiesprache (RDFS) beim W3C
2000	Beginn von großen Forschungsprojekten zu Ontologien in den USA und Europa (DAML & OntoKnowledge)
2002	Beginn der Standardisierung einer neuen Ontogiesprache (OWL) basierend auf den Forschungsergebnissen
2004	Abschluss der Standards für das Datenmodell (RDF) und die Ontologiesprache (OWL)
2008	Standard für die Anfragesprache (SPARQL)
2009	Erweiterung von OWL auf OWL 2.0
2010	Standard für Regeln: Rule Interchange Format (RIF)
2013	Erweiterung von SPARQL auf SPARQL 1.1
2014	Erweiterung von RDFS auf RDFS 1.1

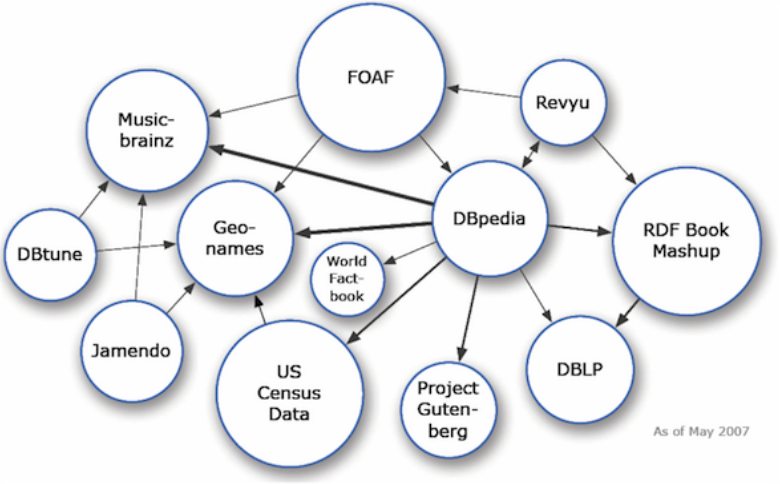
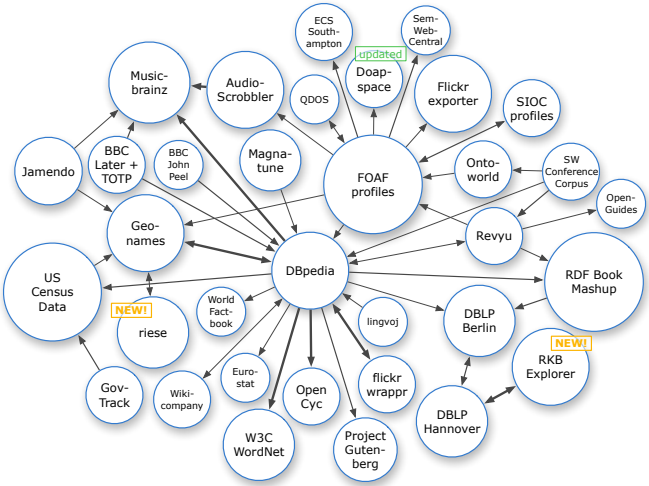
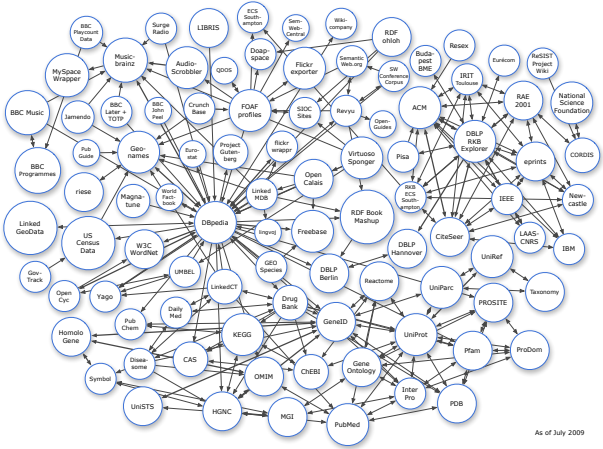
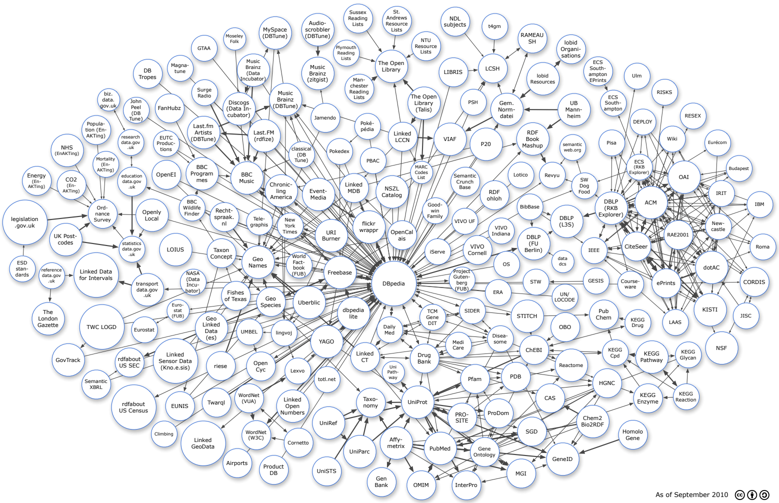


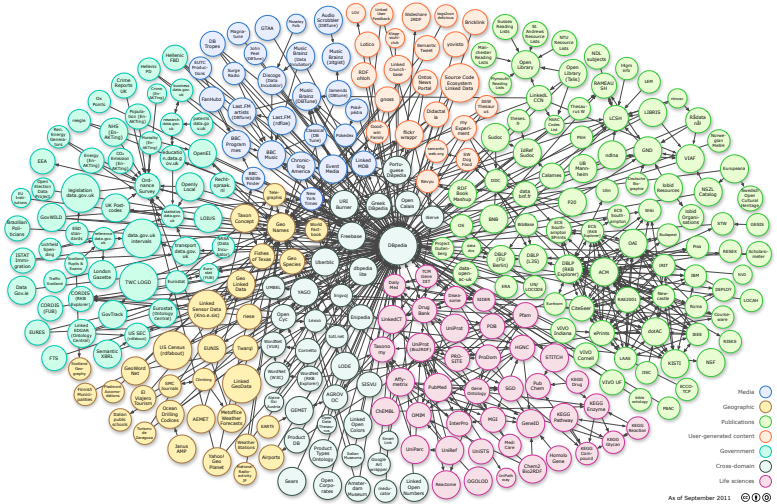
Diagramme der Linking Open Data Cloud von Max Schmachtenberg, Christian Bizer, Anja Jentzsch and Richard Cyganiak. http://lod-cloud.net/


ECS South-ampton
Sem-Web-Central
updated
Doap-space
Flickr exporter
SIOC profiles
Music-brainz
Audio-Scrobbler
QDOS
Jamendo
BBC Later + TOTP
BBC John Peel
Magna-tune
FOAF profiles
Onto-world
SW Conference Corpus
Open-Guides
Geo-names
Revyu
DBpedia
RDF Book Mashup
US Census Data
NEW!
riese
World Fact-book
lingvoj
DBLP Berlin
NEW!
RKB Explorer
Euro-stat
flickr wrappr
Gov-Track
Wiki-company
Open Cyc
DBLP Hannover
W3C WordNet
Project Guten-berg

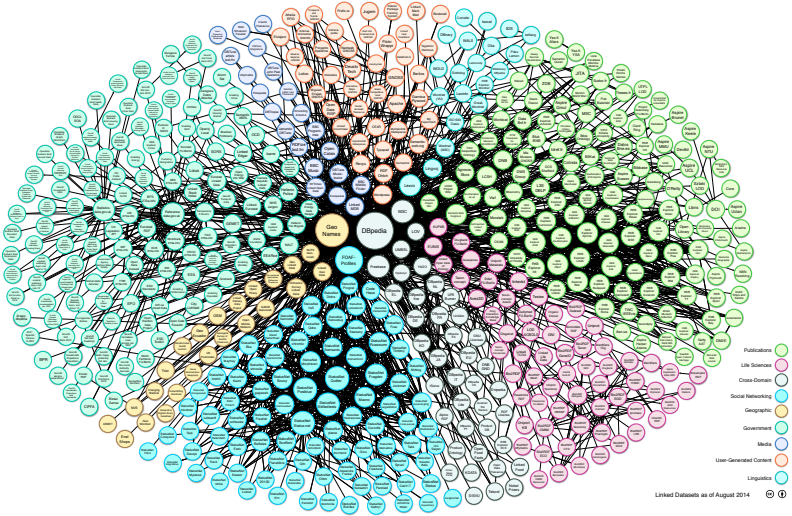

BBC Playcount Data
Surge Radio
LIBRIS
ECS South-ampton
Sem-Web-Centre
Wiki-company
RDF ohloh
Resex
Budapest BME
Eunicom
ReSIST Project Wiki
MySpace Wrapper
Music-brainz
Audio-Scrobbler
QDOS
Doap-space
Flickr exporter
Semantic Web.org
SW Conference Corpus
IRIT Toulouse
RAE 2001
National Science Foundation
BBC Music
Jamendo
BBC Later + TOTP
BBC John Peel
Crunch Base
FOAF profiles
S3OC Sites
Revyu
Open-Guides
ACM
DBLP RKB Explorer
CORDIS
Pub Guide
Geo-names
Euro-stat
Project Guten-berg
flickr wrappr
Virtuoso Sponger
Pisa
eprints
BBC Programmes
riese
Magna-tune
World Fact-book
Linked MDB
Open Calais
SKB ECS South-ampton
IEEE
New-castle
Linked GeoData
US Census Data
W3C WordNet
DBpedia
lingvoj
Freebase
RDF Book Mashup
CiteSeer
LAAS CNRS
IBM
Gov-Track
UMBEL
GEO Species
DBLP Berlin
DBLP Hannover
UniRef
Open Cyc
Yago
LinkedCT
Drug Bank
Reactome
UniParc
Taxonomy
PROSITE
Daily Med
Pub Chem
GeneID
KEGG
UniProt
Homolo Gene
Disease-some
CAS
OMIM
ChEBI
Gene Ontology
Pfam
ProDom
Symbol
UniSTS
HGNC
MGI
PubMed
Inter Pro
PDB
As of July 2009

Semantic Web 22. 9. 2010




Media
Geographic
Publications
User-generated content
Government
Cross-domain
Life sciences
As of September 2011

Semantic Web 30. 6. 2014



Semantic Web: Software

Triple Stores Wissensbanken für Ontologien

- Jena
- Sesame
- Virtuoso (kommerziell)

Editoren Erstellung von Ontologien

- Protege (Schwerpunkt OWL)

Anfrage-Tools Finden von Antworten

- Sesame 2: SPARQL-Query-Engine

Integrations-Tools Anbindung von existierenden Systemen

- D2RQ: Semantisches Liften von Datenbanken
- XLwrap: Spreadsheets

Zusammenfassung: Semantische Technologien

- reine Schlagwortsuche in vielerlei Hinsicht unbefriedigend
  - Synonyme
  - Homonyme
  - HTML beschreibt nur Layout, keine Bedeutung
  - nur explizit vorhandene Information können gefunden werden
  - Links zwischen Quellen wenig aussagekräftig
- Semantische Technologien
  - Festlegung der Bedeutung von Aussagen
  - formale Semantik entscheidend für praktischen Nutzen
- Semantic Web
  - Sammlung von W3C-Standards
  - ermöglicht
    - Veröffentlichung
    - gemeinsame Nutzung
    - Vernetzung
    - Ableitung von implizitem Wissen

    von Informationen mit klar definierter Bedeutung

Inhalt

Inhalt

Linked Data

The Semantic Web done right

Berners-Lee, 2006

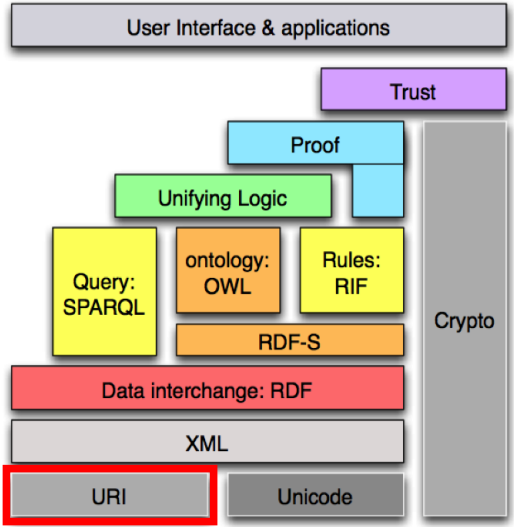
1. Use URIs as names for things.
2. Use HTTP URIs so that people can look up those names.
3. When someone looks up a URI, provide useful information, using the standards (RDF*, SPARQL).
4. Include links to other URIs so that they can discover more things.

The real power of the Semantic Web will be realized when people create many programs that collect Web content from diverse sources, process the information and exchange the results with other programs. [. . .] The Semantic Web promotes this synergy: even agents that were not expressly designed to work together can transfer data among themselves when the data come with semantics.

Berners-Lee, Hendler, and Lassila; Scientific American, 2001

Inhalt

URIs im Schichtenmodell



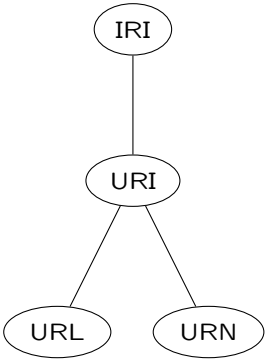
URIs

Uniform Resource Identifier (IETF RFC 3986)

- „compact sequence of characters that identifies an abstract or physical resource“
- 5 Komponenten:

scheme	`http`, `ftp`, `tel`, `mailto`, `isbn` ...
authority	`example.org:21`, `www.dhbw-stuttgart.de` (optional)
path	`/themen/hochschule/duales-studium`
	`jan.hladik@dhbw-stuttgart.de`
query	`?user=jan&from=home` (optional)
fragment	`#intro` (optional)

- URL(ocator): Ressource kann elektronisch übertragen werden
  - Webseiten
  - PDF-Dokumente
- URN(ame): Ressource kann elektronisch identifiziert werden
  - Menschen
  - Bücher
- I(nternationalised)RI: erlaubt Unicode



Warum URIs?

1. Use URIs as names for things.
   - eindeutige Identifizierung von Entitäten
   - Vermeidung von Doppeldeutigkeiten

Beispiel 2.1 (Disambiguierung)

```
http://informatik.de/Franz_Baader
http://philosophie.de/Franz_von_Baader
http://eishockey.de/Franz_Baader
```

oder

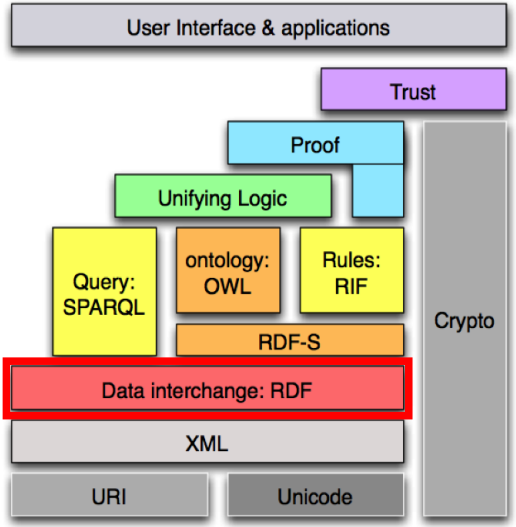
```
http://example.org/Bank_(Geldinstitut)
http://example.org/Bank_(Moebel)
```

Warum HTTP-URIs?

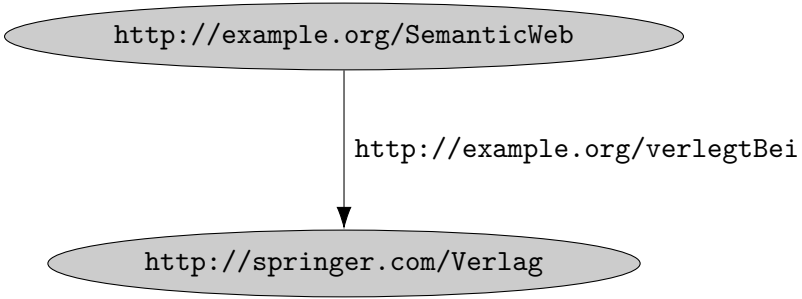
2. Use HTTP URIs so that people can look up those names.
   - Disambiguierung ist nur der erste Schritt
   - Welcher URI ist der, den ich suche?
   - HTTP-URIs liefern Datei beim Aufruf
     - Beschreibung der vom URI referenzierten Entität
     - idealerweise: verständlich für Menschen und Computer
   - Verwendung anderer Schemas (`mailto`, `isbn`), wenn sinnvoll

Inhalt

Einführung in RDF



3. When someone looks up a URI, provide useful information, using the standards (RDF*, SPARQL).



Inhalt

Allgemeines zu RDF

- „Resource Description Framework“
- W3C Recommendation (`http://www.w3.org/RDF`); aktuelle Version: 1.1 (2014)
- Datenmodell
  - ursprünglich: zur Angabe von Metadaten für Web-Ressourcen
  - später: allgemeines Format zur Formalisierung von Wissen
  - kodiert strukturierte Informationen
  - universelles, maschinenlesbares Austauschformat
  - Graph-basiert

Knoten in RDF-Graphen

- URI
  - zur eindeutigen Referenzierung einer Ressource
  - Darstellung: Ellipse
- Literal
  - z. B. Zahl, Datum, String
  - Darstellung: Rechteck
- Leerer Knoten
  - erlaubt Existenzaussage über Entität, ohne sie zu benennen
  - ähnlich Existenzquantor in Prädikatenlogik
  - Darstellung: leere Ellipse

Literale

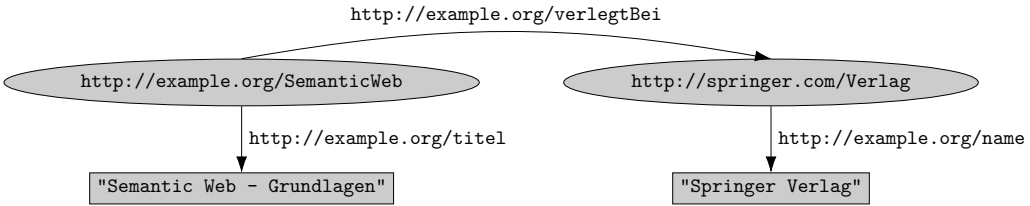
- Zur Repräsentation von Datenwerten
- Darstellung als Zeichenketten
- Interpretation erfolgt durch Datentyp (default: String)

URIs

- können alles denkbare repräsentieren
- haben keine Bedeutung a priori
- erhalten Bedeutung durch Ontologie
  `http://example.org/Tom`
  kann Mensch oder Katze bezeichnen

Literale

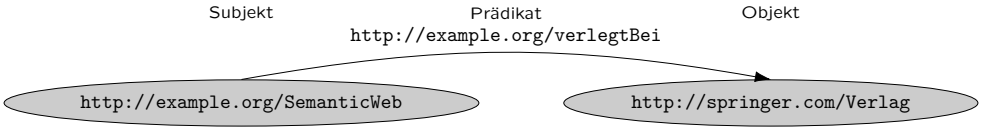
- haben eine feste Bedeutung
- unabhängig von der Ontologie
- Bedeutung von „12“ ist eindeutig



RDF-Tripel

RDF-Graph ist Menge von Tripeln

RDF-Tripel besteht aus Subjekt (Startknoten) – Prädikat (Kante) – Objekt (Zielknoten)



- Angelehnt an linguistische Kategorien
- Erlaubte Belegungen
  - Subjekt URI (oder leerer Knoten)
  - Prädikat URI
  - Objekt URI oder Literal (oder leerer Knoten)

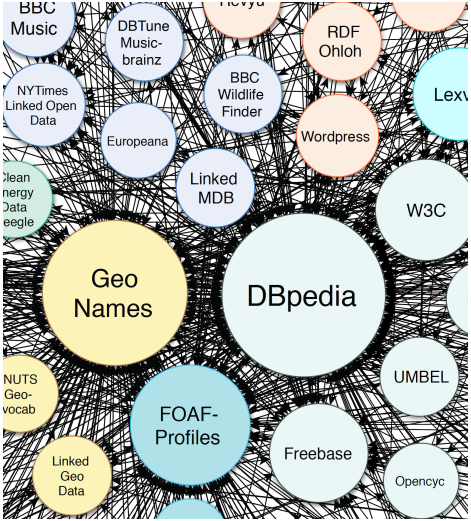
Wiederholung: Linked-Data-Prinzipien

1. Use URIs as names for things.
2. Use HTTP URIs so that people can look up those names.
3. When someone looks up a URI, provide useful information, using the standards (RDF*, SPARQL).
4. Include links to other URIs so that they can discover more things.

Verwende auch URIs aus anderen Ontologien in Tripeln

- Wenn man über bekannte Ressourcen spricht:
  - nicht das Rad neu erfinden (benötigten Teil in eigener Ontologie nachbauen)
  - sondern existierendes Wissen verwenden
- Vorteile
  - Informationen wird „automatisch“ aktualisiert
  - Benutzern stehen alle verfügbaren Informationen zur Verfügung, nicht nur die aus Sicht des Entwicklers nützlichen
  - Verteilung für Benutzer transparent

Rückblick: LOD-Cloud



- Größe des Kreises:
  Anzahl der Tripel
  Größte: $> 1.000.000.000$
- Breite des Pfeils $A \rightarrow B$:
  Anzahl der URIs aus B, die in A verwendet werden
  Breiteste: > 100.000

Inhalt

Einfache Syntax für RDF

XML	gut zu parsen, aber schwer zu lesen
Notation 3	(N3) umfangreicher Formalismus
N-Triples	Teil von N3
Terse Triple Language	(TTL / „Turtle“) Erweiterung von N-Triples (Abkürzungen)

Syntax in TTL:

- URIs in spitzen Klammern
- Literale in Anführungszeichen
- Tripel durch Punkt abgeschlossen
- Leerzeichen und Zeilenumbrüche außerhalb von Bezeichnern werden ignoriert

TTL-Syntax: Präfixe

Beispiel 2.2 (Tripel in TTL-Notation)

```
<http://example.org/SemanticWeb> <http://example.org/verlegtBei>
   <http://springer.com/Verlag> .
<http://example.org/SemanticWeb> <http://example.org/titel>
   "Semantic Web - Grundlagen" .
<http://springer.com/Verlag> <http://example.org/name> "Springer Verlag" .
```

Beispiel 2.3 (Abkürzungen in TTL)

```
@prefix ex: <http://example.org/> .
@prefix springer: <http://springer.com/> .
@prefix : <http://jan.org/ontology/> .

ex:SemanticWeb ex:verlegtBei springer:Verlag .
ex:SemanticWeb ex:titel "Semantic Web - Grundlagen" .
springer:Verlag ex:name "Springer Verlag" .
:jan :vorname "Jan".
```

RDF-Vokabular: `@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .`

TTL-Syntax: Abkürzungen

Mehrere Tripel mit gleichem Subjekt kann man zusammenfassen:

Beispiel 2.4

```
@prefix ex: <http://example.org/> .
@prefix springer: <http://springer.com/> .

 ex:SemanticWeb   ex:verlegtBei  springer:Verlag ;
                  ex:titel       "Semantic Web - Grundlagen" .
 springer:Verlag  ex:name        "Springer Verlag" .
```

Ebenso Tripel mit gleichem Subjekt und Prädikat:

Beispiel 2.5

```
@prefix ex: <http://example.org/> .

 ex:SemanticWeb  ex:autor  ex:Hitzler, ex:Krötzsch,
                           ex:Rudolph, ex:Sure ;
                 ex:titel  "Semantic Web - Grundlagen" .
```

Übung: TTL-Syntax

Übung 2.6

Wir nehmen an, dass diese Vorlesung den URI `http://dhbw.de/SW` hat,
Jan Hladik den URI `http://example.org/JanHladik`.

Modellieren Sie in TTL und grafisch, dass Jan Hladik den Namen (`http://example.org/name`) „Jan Hladik“ hat, die Vorlesung Semantic Web hält (`http://example.org/haelt`), die außerdem den Namen „Semantic Web“ hat. Verwenden Sie, wo möglich, eine abkürzende Schreibweise.

Inhalt

Literale und Datentypen

- Bisher: Literale untypisiert, wie Strings behandelt
  - "02" $\neq$ "2"
  - "100" $<$ "11"
- Typisierung erlaubt besseren (semantischen) Umgang mit Werten

Datentyp

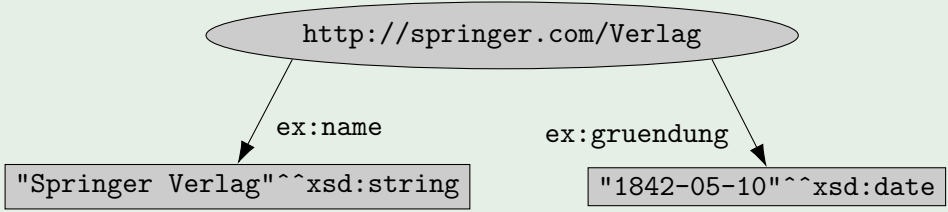
- Gruppe von gleichartigen Literalen
- referenziert durch URI $\rightsquigarrow$ neue Datentypen möglich
- Präfix für XML-Schema-Datatypen: `xsd: <http://www.w3.org/2001/XMLSchema#>`
- verpflichtend
  - `xsd:string` Zeichenketten (default für untypisierte Literale)
  - `rdf:langString` Zeichenketten mit Sprachangabe
- standardisiert
  - `xsd:integer` Ganzzahlen
  - `xsd:decimal` Dezimalzahlen
  - `xsd:boolean` {`true`, `false`}
  - `xsd:date` Datum im Format `jjjj-mm-tt`
  - `xsd:time` Zeit im Format `hh:mm:ss.sss...`

Datentypen in RDF

TTL-Syntax: `"Datenwert"^^Datentyp-URI`

Beispiel 2.7

Graph:



TTL:

```
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix ex: <http://example.org/> .
<http://springer.com/Verlag> ex:name "Springer Verlag"^^xsd:string ;
                             ex:gruendung "1842-05-10"^^xsd:date .
```

Sprachangaben für Literale

- TTL-Syntax: `"Literal"@Language-Tag`
- Language-Tags spezifiziert in
  - ISO 639 (`https://www.iso.org/iso-639-language-codes.html`)
  - IETF BCP 47 (`https://tools.ietf.org/html/bcp47`)
- Sprachinformationen durch Codes, z. B. `de`, `en`, `en-gb`
- Datentyp: `rdf:langString`; Angabe optional

Beispiel 2.8 (Sprachinformationen in TTL)

```
<http://springer.com/Verlag> ex:name "Springer Verlag"@de,
                                     "Springer Science+Business Media"@en .
```

Inhalt

Leere Knoten

URI repräsentiert abstrakte oder physische Ressource

Literal repräsentiert sich selbst

Leerer Knoten repräsentiert Ressource, die nicht benannt werden kann oder soll

- Englisch: bnode („blank node“)
- hat keinen URI, sondern Identifier (ID) mit Präfix „_:“
- kann als Existenzaussage gelesen werden

Beispiel 2.9 („John liebt irgendjemanden.“)

```
ex:John ex:liebt _:id1 .
```

Beispiel 2.10 („Irgendjemand liebt Mary.“)

```
_:id1 ex:liebt ex:Mary .
```

Vorsicht: „John steht in irgendeiner Beziehung zu Mary.“

```
ex:John _:id1 ex:Mary .
```

Bnodes sind als Prädikate unzulässig!

Bnodes in TTL-Notation

- mit Präfix „_:“

Beispiel 2.11 („Jemand, der 28 Jahre alt ist und in Stuttgart geboren ist, kennt jemanden, der Mary heißt und in New York wohnt.“)

```
_:id1 ex:alter "28"^^xsd:integer ;
      ex:geburtsort ex:Stuttgart ;
      ex:kennt _:id2 .
_:id2 ex:name "Mary"^^xsd:string ;
      ex:wohnort ex:NewYork .
```

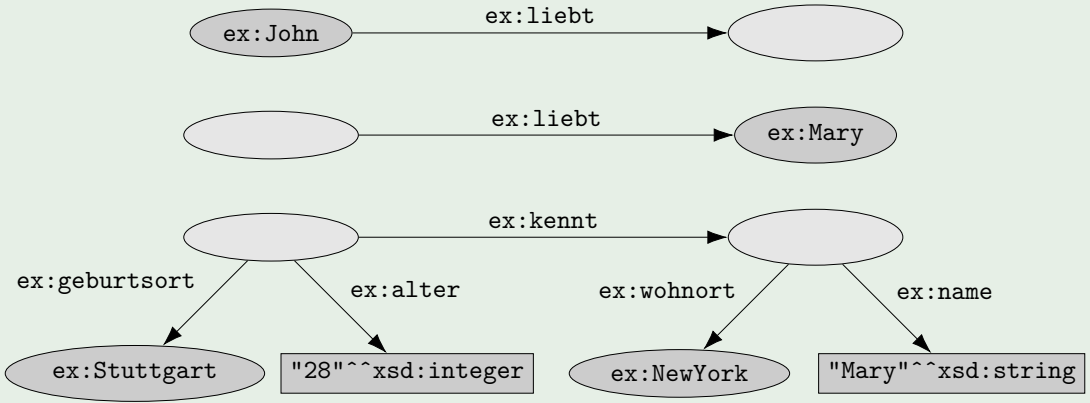
- mit eckigen Klammern [ ]
  - vermeidet Notwendigkeit, einen Namen zu vergeben
  - als Subjekt oder Objekt möglich
  - zusätzliche Tripel mit Bnode als Subjekt können zwischen Klammern angegeben werden

Beispiel 2.12

```
[ ex:alter "28"^^xsd:integer ;
  ex:geburtsort ex:Stuttgart ] ex:kennt [ ex:name "Mary"^^xsd:string ;
                                         ex:wohnort ex:NewYork ] .
```

Darstellung von Bnodes im RDF-Graphen

Beispiel 2.13



Äquivalente Darstellungen von Bnodes

Beispiel 2.14

```
[ ex:name "John" ]
  ex:age "28" .
```

„Jemand, der John heißt, ist 28 Jahre alt.“

Beispiel 2.15

```
[ ex:age "28" ]
  ex:name "John" .
```

„Jemand, der 28 Jahre alt ist, heißt John.“

Beispiel 2.16

```
[ ] ex:name "John" ;
    ex:age "28" .
```

„Jemand heißt John und ist 28 Jahre alt.“

Vorsicht: Kein Tripel!

```
[ ex:name "John" ;
  ex:age "28" ] .
```

„Jemand, der John heißt und 28 Jahre alt ist, . . .“

Inhalt

Mehrwertige Beziehungen

„Für die Zubereitung von Chutney benötigt man

- 300g Mango,
- einen Teelöffel Cayennepfeffer,
- ...“

Beispiel 2.17 (Turtle Chutney 1)

```
@prefix ex: <http://example.org/> .
ex:Chutney ex:zutat "300g Mango",
                    "1TL Cayennepfeffer",
                    ...
```

Nicht zufriedenstellend:

- Zutaten samt Menge als Zeichenkette
- Suche nach Rezepten, die Mango beinhalten, so nicht möglich

Mehrwertige Beziehungen

„Für die Zubereitung von Chutney benötigt man

- 300g Mango,
- einen Teelöffel Cayennepfeffer,
- ...“

Beispiel 2.18 (Turtle Chutney 2)

```
@prefix ex: <http://example.org/> .
ex:Chutney ex:zutat ex:Mango;
           ex:menge "300g";
           ex:zutat ex:Cayennepfeffer;
           ex:menge "1TL";
           ...
```

Überhaupt nicht zufriedenstellend:

- keine eindeutige Zuordnung von konkreter Zutat und Menge mehr möglich

Mehrwertige Beziehungen

„Für die Zubereitung von Chutney benötigt man

- 300g Mango,
- einen Teelöffel Cayennepfeffer,
- . . . “

Beispiel 2.19 (Turtle Chutney 3)

```
@prefix ex: <http://example.org/> .
ex:Chutney         ex:zutat ex:Mango;
                   ex:zutat ex:Cayennepfeffer.
ex:Mango           ex:menge "300g".
ex:Cayennepfeffer ex:menge "1TL".
```

Ebenfalls nicht zufriedenstellend:

- Menge wird zu Eigenschaft der Zutat
- scheitert bei mehreren Rezepten mit gleicher Zutat

Mehrwertige Beziehungen: Grundlagen

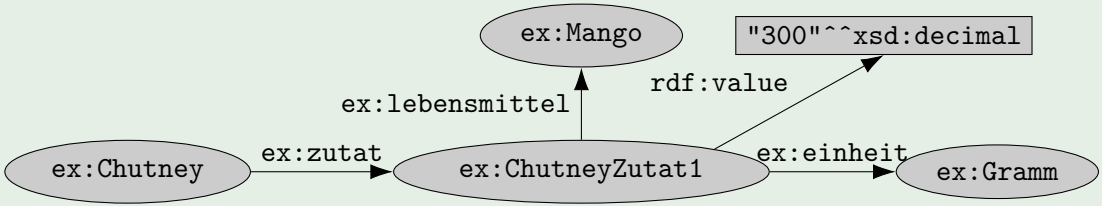
- Problem: Zutat-Angabe ist eine echte dreiwertige (ternäre) Beziehung
  - in Prädikatenlogik: dreistellige Relation
    Zutat(chutney, mango, 300g), Zutat(chutney, cayennepfeffer, 1TL)
  - in Datenbank:

Gericht	Zutat	Menge
Chutney	Mango	300g
Chutney	Cayennepfeffer	1TL

- direkte Darstellung in RDF nicht möglich
- Lösung: Verwendung von Hilfsknoten

Beispiel 2.20 (Hilfsknoten in RDF)

- Als Graph

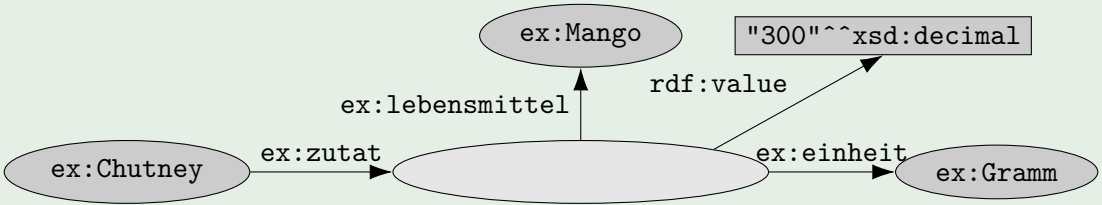


- in TTL-Syntax

```
ex:Chutney        ex:zutat ex:ChutneyZutat1 .
ex:ChutneyZutat1  ex:lebensmittel ex:Mango ;
                  rdf:value "300"^^xsd:decimal ;
                  ex:einheit ex:Gramm .
```

`rdf:value` ordnet Knoten Zahlenwerte zu

Beispiel 2.21 (Bnode)



Leere Knoten in TTL

Beispiel 2.22 (TTL-Syntax)

```
ex:Chutney ex:zutat _:id1 .
_:id1 ex:lebensmittel ex:Mango ;
      rdf:value "300"^^xsd:decimal ;
      ex:einheit ex:Gramm .
```

Beispiel 2.23 (Verkürzte TTL-Schreibweise)

```
ex:Chutney ex:zutat
    [ ex:lebensmittel ex:Mango ;
      rdf:value "300"^^xsd:decimal ;
      ex:einheit ex:Gramm ] ,
    [ ex:lebensmittel ex:Cayennepfeffer ;
      rdf:value "1"^^xsd:decimal ;
      ex:einheit ex:Teeloeffel ] .
```

Online-Tools zu RDF-Syntax

- `https://www.w3.org/RDF/Validator/`
  - „offizielle“ Seite
  - Syntax-Check und Visualisierung
  - Eingabeformat nur XML
- `http://www.ldf.fi/service/rdf-grapher`
  - Visualisierung von RDF-Graphen
  - verschiedene Syntaxformate (XML, TTL, JSON)
  - verschiedene Grafikformate (PNG, EPS, PDF)
- `https://www.easyrdf.org/converter`
  - Konvertierung zwischen Syntaxformaten, z.B. TTL nach XML

Übung: Bnodes

Übung 2.24

1. Modellieren Sie Ihren eigenen dreiseitigen Ausbildungsvertrag der DHBW (Student, DHBW, Ausbildungsbetrieb) mit Hilfe eines Bnodes.
2. Modellieren Sie in TTL:
   - `ex:MeinAuto` hat die folgenden Bestandteile (`ex:hatTeil`):
     - 4 Räder (`ex:Rad`)
     - 2 Türen (`ex:Tür`)
     - 1 Motor (`ex:Motor`)
   - der Motor hat die Teile
     - 1 Motorblock
     - 4 Zylinder
     - 8 Ventile
3. Visualisieren Sie die Ergebnisse mit dem „RDF Grapher“.

Inhalt

Listen

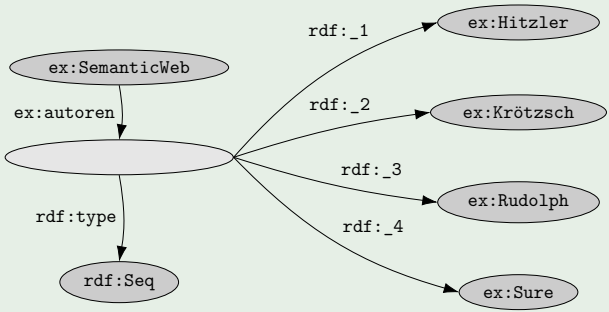
- Allgemeine Datenstrukturen zur Aufzählung von Ressourcen
  - Reihenfolge relevant $\rightsquigarrow$ Folgen
  - z. B. Autoren eines Buches
- Zwei Varianten

  Container offene Liste
  Hinzufügen von neuen Einträgen möglich
  Typ wird Wurzelknoten via `rdf:type` zugewiesen
  - `rdf:Seq` geordnete Liste (Folge)
  - `rdf:Bag` ungeordnete Liste (Menge)
  - `rdf:Alt` liste alternativer Möglichkeiten zur Auswahl

  Collections geschlossene Liste
  Hinzufügen von neuen Einträgen nicht möglich

Container (Offene Listen)

Beispiel 2.25

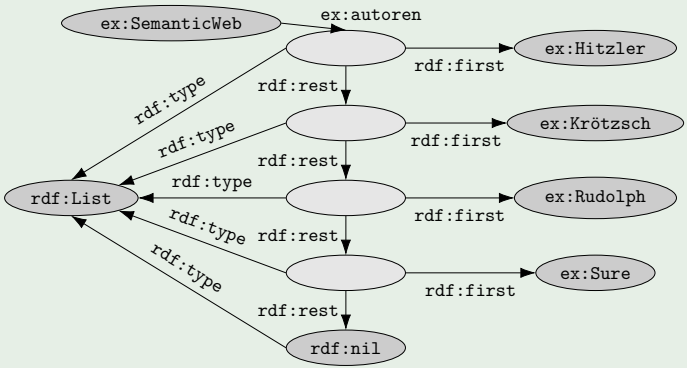



```
ex:SemanticWeb ex:autoren
    [ rdf:type rdf:Seq ;
      rdf:_1 ex:Hitzler;
      rdf:_2 ex:Krötzsch;
      rdf:_3 ex:Rudolph;
      rdf:_4 ex:Sure ] .
```

- `rdf:_n`: n-tes Element
- Abkürzung: `rdf:li`
  - vergibt nächstes freies Prädikat
  - abhängig von Reihenfolge im Dokument
  - sinnvoll, wenn Reihenfolge im Dokument feststeht oder irrelevant ist

Collections (Geschlossene Listen)

Beispiel 2.26




```
ex:SemanticWeb ex:autoren
    ( ex:Hitzler ex:Krötzsch
      ex:Rudolph ex:Sure ) .
```

Idee: Verkettete Liste (rekursive Datenstruktur)

- first Kopfelement
- rest Restliste (kann leer sein)

Übung: RDF-Syntax

Übung 2.27

Entscheiden Sie, ob die folgenden Aussagen wahr oder falsch sind:

1. URIs können für beliebige Satzteile (Subjekt/Prädikat/Objekt) eines Tripels stehen.
2. Leere Knoten können für beliebige Satzteile stehen.
3. Literale können für beliebige Satzteile stehen.
4. Zwei leere Knoten mit unterschiedlicher ID können für die gleiche Ressource stehen.
5. Zwei unterschiedliche URIs können für die gleiche Ressource stehen.
6. Kommen in mehreren RDF-Dokumenten leere Knoten mit der selben ID vor, dann müssen sie für dieselbe Ressource stehen.
7. Kommen in mehreren RDF-Dokumenten die gleichen URIs vor, dann müssen sie für dieselbe Ressource stehen.
8. Zwei unterschiedliche Literale können für den gleichen Wert stehen.
9. Zwei Literale unterschiedlichen Datentyps können für den gleichen Wert stehen.

RDF-Vokabular mit vorgegebener Bedeutung

- Klassen

`rdf:Property`	Klasse aller Properties (URIs, die als Prädikate fungieren können)
`rdf:Seq`, `rdf:Bag`, `rdf:Alt`	Klassen verschiedener Arten von Containern
`rdf:List`	Klasse aller Collections

- Properties

`rdf:type`	Instanz-Beziehung
`rdf:value`	Zahlenwerte
`rdf:li`, `rdf:_1`, `rdf:_2`, ...	für Container
`rdf:first`, `rdf:rest`	für Collections

- Sonstige URIs

`rdf:nil`	leere Collection

Zusammenfassung: RDF

Fähigkeiten

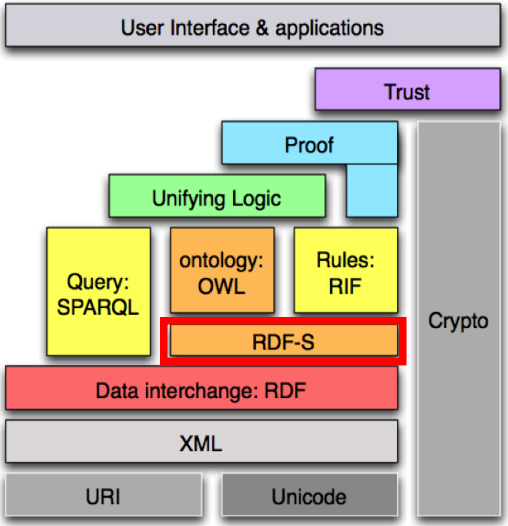
- Universeller Standard für Speicherung und Austausch von Daten
- Leicht zu verstehendes graph-basiertes Datenmodell
- Unterstützung für global eindeutige Vokabulare

Grenzen

- Schwerpunkt auf Faktenwissen (Individuen)
  - „`Hamlet` hat den Autor Shakespeare“
  - „Mein Auto hat einen Motor“
- Kaum Möglichkeiten zur Kodierung von Schemawissen
  - „Nur Menschen sind Autoren.“
  - „Bücher und Zeitschriften sind Veröffentlichungen.“
- Intuitive Semantik: Bedeutung der Vokabulare bleibt im Kopf der Anwender
  - Bedeutung von `ex:titel` ist nicht im RDF-Graph enthalten
  - Anwender muss Bedeutung kennen und URI richtig benutzen

Inhalt

RDF Schema

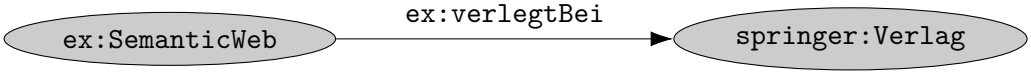

User Interface & applications
Trust
Proof
Unifying Logic
Query: SPARQL
ontology: OWL
Rules: RIF
RDF-S
Crypto
Data interchange: RDF
XML
URI
Unicode

Inhalt

Grenzen von RDF

RDF bietet:

- universelle Möglichkeit zur Kodierung von faktischen Daten im Web:



- Aussagen über einzelne Ressourcen (Individuen) und deren Beziehungen

Wünschenswert:

- Aussagen über Klassen (Mengen von Individuen)
  - Verlage
  - Organisationen
  - Menschen
- Spezifikation von Zusammenhängen zwischen Individuen, Klassen und Beziehungen
  - „Verlage sind Organisationen.“
  - „Etwas, das einen Autor hat, ist eine Veröffentlichung.“
- bessere Repräsentation der Semantik der Domäne
- In Datenbank-Terminologie: Schemawissen

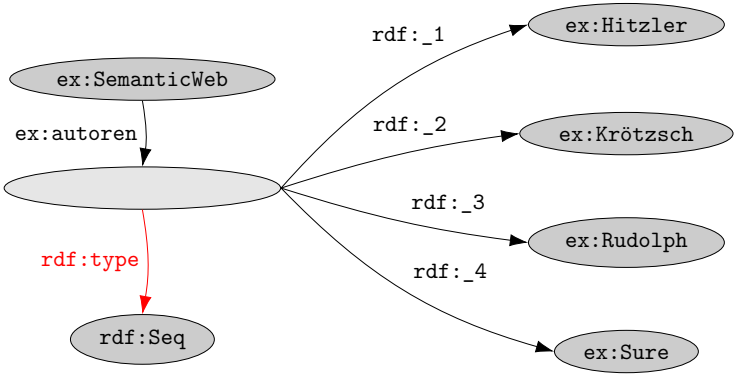
RDFS

- Teil der W3C Recommendation zu RDF
- Namensraum `rdfs: http://www.w3.org/2000/01/rdf-schema#`
- Ermöglicht Spezifikation von schematischem (terminologischem) Wissen
- Spezielles RDF-Vokabular
  - jedes RDFS-Dokument ist ein RDF-Dokument
- Vokabular ist generisch
  - erlaubt die Spezifikation der Semantik beliebiger RDF-Vokabulare
  - nicht themengebunden wie z. B. FOAF
- Software mit RDFS-Unterstützung interpretiert mit RDFS definierte Vokabulare korrekt
- Funktionalität macht RDFS zu einer Ontologiesprache (für lightweight Ontologies)
  - eingeschränkte Ausdrucksmöglichkeiten
  - keine Widersprüche modellierbar (z.B. keine Negation)

Inhalt

Klassen und Instanzen

- Typisierung von Knoten in RDF für Listen:



- Prädikat `rdf:type` weist dem Subjekt das Objekt als Typ zu
- Subjekt ist Instanz der Klasse, die das Objekt bezeichnet

Klassen und Instanzen

Beispiel 2.28 (Klassen und Instanzen)

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

- Charakterisiert `ex:SemanticWeb` als Instanz der Klasse `ex:Lehrbuch`
- „Semantic Web – Grundlagen ist ein Lehrbuch.“
- Klassenzugehörigkeit ist nicht exklusiv, z. B. ist gleichzeitig möglich:
  `ex:SemanticWeb rdf:type ex:Informativ .`

- keine strikte Trennung zwischen Individuen- und Klassenbezeichnern
  - auch in der Praxis oft Ermessensfrage
- Kennzeichnung eines URI als Klassenbezeichner möglich durch `rdfs:Class`
  - `ex:Lehrbuch rdf:type rdfs:Class .`
- `rdfs:Class` ist „Klasse aller Klassen“
  - `rdfs:Class` enthält sich selbst
  - `rdfs:Class rdf:type rdfs:Class .` gilt immer.

Beispiel 2.29 (Jedes Lehrbuch ist ein Buch)

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

Problem Suche nach Instanzen von `ex:Buch` liefert kein Resultat

Ansatz 1 `ex:SemanticWeb rdf:type ex:Buch .`

- Löst das Problem nur für einen URI `ex:SemanticWeb`
- Hinzufügen für alle Instanzen führt zu unnötig großen RDF-Graphen

Ansatz 2 Allgemeine Aussage, dass jedes Lehrbuch auch ein Buch ist

- jede Instanz von `ex:Lehrbuch` ist automatisch auch Instanz von `ex:Buch`
- Property `rdfs:subClassOf`

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

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

Unterklassen und Klassenhierarchien

- `rdfs:subClassOf` ist reflexiv
  - jede Klasse ist Unterklasse von sich selbst
  - `ex:Lehrbuch rdfs:subClassOf ex:Lehrbuch .`
- Festlegung der Gleichheit zweier Klassen durch gegenseitige Unterklassenbeziehung
  - `ex:Hospital rdfs:subClassOf ex:Krankenhaus .`
  - `ex:Krankenhaus rdfs:subClassOf ex:Hospital .`
- `rdfs:subClassOf` ermöglicht komplexe Klassenhierarchien (Taxonomien)
  - `ex:Lehrbuch rdfs:subClassOf ex:Buch .`
  - `ex:Buch rdfs:subClassOf ex:Printmedium .`
  - `ex:Zeitschrift rdfs:subClassOf ex:Printmedium .`
- Transitivität von `rdfs:subClassOf` in RDFS-Semantik verankert
  - es folgt automatisch
    `ex:Lehrbuch rdfs:subClassOf ex:Printmedium .`

Beispiel 2.30 (Zoologische Einordnung des modernen Menschen)

```
@prefix ex: <http://example.org/> .

ex:Animalia rdf:type rdfs:Class .
ex:Vertebrata rdfs:subClassOf ex:Animalia .
ex:Insecta rdfs:subClassOf ex:Animalia .
ex:Mammalia rdfs:subClassOf ex:Vertebrata .
ex:Aves rdfs:subClassOf ex:Vertebrata .
ex:Primates rdfs:subClassOf ex:Mammalia .
ex:Carnivora rdfs:subClassOf ex:Mammalia .
ex:Hominidae rdfs:subClassOf ex:Primates .
ex:Homo rdfs:subClassOf ex:Hominidae .
ex:HomoSapiens rdfs:subClassOf ex:Homo .
```

Klassen und Mengen

- Intuitive Parallele zur Mengenlehre:

`rdf:type`	entspricht	$\in$
`rdfs:subClassOf`	entspricht	$\subseteq$

- Rechtfertigt auch Reflexivität und Transitivität von `rdfs:subClassOf`
- zwei der Bedeutungen von „is-a“ in Semantischen Netzen
  - Abkürzung `a` für `rdf:type`
  - `ex:Animalia a rdfs:Class .`

Inhalt

Properties

- charakterisieren Beziehung zwischen zwei Ressourcen
- Mathematisch: Binäre Relation (Menge von Paaren)
  Mag = {(Peter, Paul), (Anna, Carla), . . .}
- Property-Bezeichner in Tripeln üblicherweise an Prädikatsstelle
  - Ausnahme: RDF(S)-Terminologie
  - `ex:verlegtBei rdf:type rdf:Property .`

Unterproperties

- Ähnlich zu Unter-/Oberklassen sind auch Unter-/Oberproperties möglich
- Darstellung in RDFS mittels `rdfs:subPropertyOf`

Beispiel 2.31

```
ex:mag rdfs:subPropertyOf ex:kennt .
ex:Markus ex:mag ex:Anja .
```

impliziert

```
ex:Markus ex:kennt ex:Anja .
```

Inhalt

Bereichs-Einschränkungen für Properties

Festlegung von Definitions- und Zielbereiche mit `rdfs:domain` und `rdfs:range`

Beispiel 2.32 (Bereichs-Einschränkungen)

- `ex:verlegtBei` verbindet nur Publikationen mit Verlagen
- Codierung in RDFS:
  ```
  ex:verlegtBei rdfs:domain ex:Publikation .
  ex:verlegtBei rdfs:range ex:Verlag .
  ```
- Aus `ex:SemanticWeb ex:verlegtBei springer:Verlag .` folgt:
  ```
  ex:SemanticWeb rdf:type ex:Publikation .
  springer:Verlag rdf:type ex:Verlag .
  ```
- Auch zur Angabe von Datentypen für Literale:
  `ex:alter rdfs:range xsd:nonNegativeInteger .`

Einschränkung von Properties

- Properties sind in RDF(S) nicht (wie in OOP) speziellen Klassen zugeodnet
  - nicht notwendig
  - nicht immer sinnvoll („Menschen oder Organisationen besitzen Dinge“)
- Propertyeinschränkungen wirken konjunktiv

```
ex:besitzt rdfs:domain ex:Mensch .
ex:besitzt rdfs:domain ex:Organisation .
```

  bedeutet: jedes Individuum, das etwas besitzt, ist gleichzeitig Mensch und Organisation
- Daher: Immer allgemeinste mögliche Klasse als domain/range
- Wenn es keine eindeutige Klasse gibt: Keine Einschränkung angeben

Übung: Klassen und Schemawissen

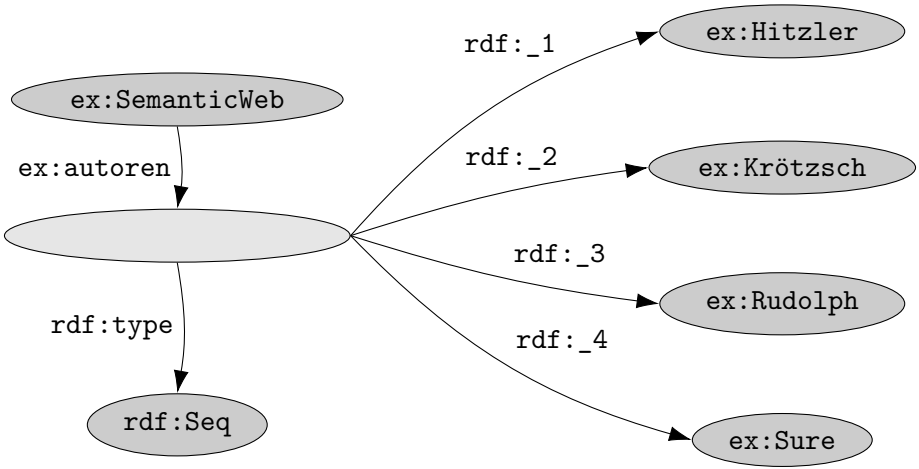
Übung 2.33

Entscheiden Sie, ob sich die folgenden Aussagen mittels RDF(S) zufriedenstellend modellieren lassen und geben Sie gegebenenfalls entsprechende RDF(S)-Spezifikationen an.

1. Jede Pizza ist ein Lebensmittel.
2. Pizzen haben immer mindestens zwei Beläge.
3. Pizza Margherita ist eine Pizza.
4. Jede Pizza hat Tomatensoße als Belag.
5. Alles, was mit irgendetwas belegt ist, ist eine Pizza.
6. Pizza Margherita hat keinen Belag aus Fleisch.

Inhalt

Arbeit mit offenen Listen



Arbeit mit offenen Listen

- `rdfs:Container`
  Oberklasse von `rdf:Seq`, `rdf:Bag`, `rdf:Alt`
- `rdfs:ContainerMembershipProperty`
  Instanzen sind keine Individuen, sondern selbst Properties
- Intendierte Semantik: jede Property, die aussagt, dass das Objekt im Subjekt enthalten ist, ist Instanz von `rdfs:ContainerMembershipProperty`
- Es gilt also insbesondere
  `rdf:_1 rdf:type rdfs:ContainerMembershipProperty .`
  `rdf:_2 rdf:type rdfs:ContainerMembershipProperty .`
  etc.
- `rdfs:member`: „universelle Enthaltenseinsrelation“
  Oberproperty aller ContainerMembershipProperties
- `ex:teil rdf:type rdfs:ContainerMembershipProperty .`
  `ex:Auto ex:teil ex:Motor .`

  impliziert

  `ex:Auto rdfs:member ex:Motor .`

Inhalt

Annotationen

- Ziel: Hinzufügen von Anmerkungen
  - erhöht der Verständlichkeit für menschliche Nutzer
  - ähnlich Kommentaren in Programmiersprachen
- Zwecks einheitlicher Syntax wird auch dieses Wissen im RDF-Graph repräsentiert
- Properties rdfs:label und rdfs:comment

Die Property rdfs:label

- Property, die einem Knoten einen alternativen Namen zuweist (Literal)
- Oftmals sind URIs schwer lesbar; zumindest unhandlich
- Bedeutung sollte unabhängig von URIs sein
- durch `rdfs:label` zugewiesener Name wird z.B. häufig von Tools bei der grafischen Darstellung verwendet

Beispiel 2.34 (rdfs:label)

```
ex:Hominidae rdf:type rdfs:Class ;
             rdfs:label "Menschenaffen"@de,
                        "great apes"@en .
```

Weitere Kommentar-Properties

`rdfs:comment`

- Property, die einem Knoten einen Kommentar zuweist (Literal)
  - typisch: länger als `rdfs:label`, mehrere Sätze
- Beinhaltet z.B. natürlichsprachliche Definition einer neu eingeführten Klasse
- erleichtert spätere intentionsgemäße Wiederverwendung

`rdfs:isDefinedBy`

- Gibt URI an, durch die ein URI definiert wurde
- ähnlich Quellenangabe

`rdfs:seeAlso`

- ist Oberproperty von `rdfs:isDefinedBy`
- unspezifischer Link zu weiterführenden „verwandten“ URIs
- ähnlich HTML-Link: keine Semantik
- wann immer möglich: spezifischere Property verwenden

Beispiel: rdfs:label und rdfs:comment

Beispiel 2.35 (Verwendung von RDFS-Properties)

```
@prefix wikipedia: <http://de.wikipedia.org/wiki/> .
@prefix ex:        <http://example.org/> .

ex:Primates rdf:type rdfs:Class ;
            rdfs:label "Primaten"@de ;
            rdfs:comment "Eine Säugetierordnung. Primaten zeichnen sich
                durch ein hochentwickeltes Gehirn aus. Sie besiedeln
                hauptsächlich die wärmeren Erdregionen. Die Bezeichnung
                Primates (lat. für Herrentiere) stammt von
            Carl von Linné."@de ;
            rdfs:seeAlso wikipedia:Primaten ;
            rdfs:subClassOf ex:Mammalia .
```

Übung: Nutzung von RDFS-Labels und -Comments

Übung 2.36

Die FOAF-Ontologie (`http://xmlns.com/foaf/spec/`) beschreibt ein Vokabular für Informationen über Menschen. Eine TTL-Version finden Sie auf

`http://wwwlehre.dhbw-stuttgart.de/~hladik/SW/Ontologien/foaf.ttl`

Versuchen Sie, anhand der RDFS-Labels und -Comments die intendierte Verwendung des Vokabulars zu verstehen.
Erstellen Sie dann unter Verwendung des FOAF-Vokabulars eine TTL-Datei mit Ihren persönlichen Informationen.

FOAF-File von Tim Berners-Lee

Timothy Berners-Lee

rdf:type	foaf:Person contact:Male
rdfs:label	Tim Berners-Lee
foaf:name	Timothy Berners-Lee
foaf:nick	TimBL timbl
foaf:img	
foaf:mbox	timbl@w3.org
foaf:title	Sir
foaf:account	http://en.wikipedia.org/wiki/User:Timbl http://www.reddit.com/user/timbl/ http://twitter.com/timberners_lee http://identi.ca/timbl
foaf:phone	tel:+1-(617)-253-5702
foaf:homepage	http://www.w3.org/People/Berners-Lee/
foaf:family_name	Berners-Lee
foaf:givenname	Timothy

RDFS-Vokabular mit vorgegebener Bedeutung

- Klassen

`rdfs:Resource`	Klasse aller Ressourcen (sämtliche Elemente der Domäne)
`rdfs:Class`	Klasse aller Klassen
`rdfs:Container`	Oberklasse von `rdf:Seq`, `rdf:Bag` und `rdf:Alt`
`rdfs:ContainerMembershipProperty`	Klasse aller Properties, die eine Enthaltenseinsbeziehung darstellen, z. B. Container, Collections
`rdfs:Literal`	Klasse aller Literalwerte (Oberklasse aller Datentypen)
`rdfs:Datatype`	Klasse aller Datentypen

- Properties

`rdfs:subClassOf`	Unterklassenbeziehung
`rdfs:subPropertyOf`	Unterpropertybeziehung
`rdfs:domain`	Definitionsbereich
`rdfs:range`	Zielbereich
`rdfs:member`	Oberproperty aller ContainerMembershipProperties
`rdfs:label`, `rdfs:comment`, `rdfs:isDefinedBy`, `rdfs:seeAlso`	Annotationen

Inhalt

Einfache Ontologien mit RDFS

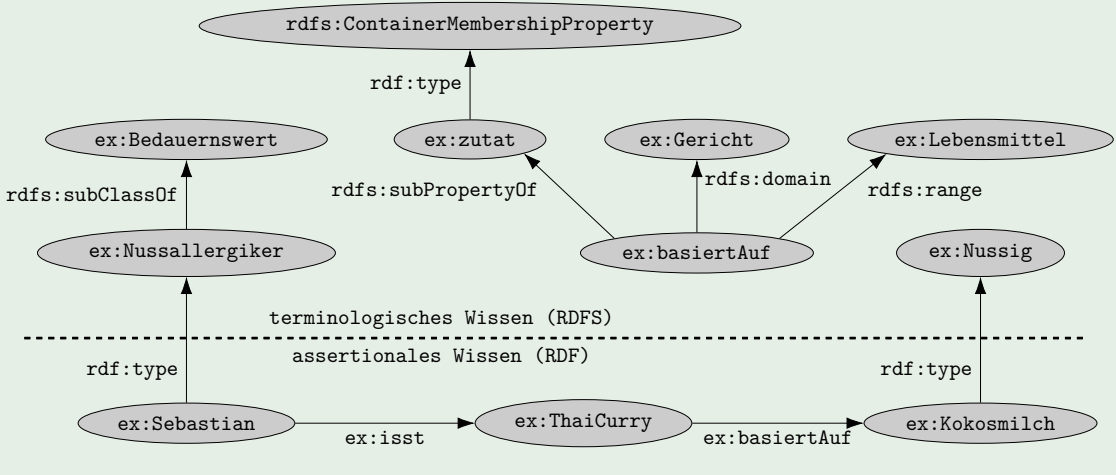
RDFS

- ermöglicht die semantische Beschreibung von wesentlichen Aspekten der Domäne
- ermöglicht das Ableiten von implizitem Wissen entsprechend der RDFS-Semantik
- ist eine (vergleichsweise wenig ausdrucksstarke) Ontologiesprache

> *A little semantics goes a long way.*
>
> James Hendler, 1998

James Hendler
(*1957)

Beispiel 2.37



Drei Typen von Informationen in RDFS-Graphen

Beispiel 2.37 (Fortsetzung)

```
ex:Lebensmittel       rdf:type                rdfs:Class .
ex:Gericht            rdf:type                rdfs:Class .
ex:isst               rdf:type                rdf:Property .
ex:Nussallergiker     rdfs:subClassOf         ex:Bedauernswert .
ex:basiertAuf         rdfs:domain             ex:Gericht .
ex:basiertAuf         rdfs:range              ex:Lebensmittel .
ex:basiertAuf         rdfs:subPropertyOf      ex:zutat .
ex:zutat              rdf:type                rdfs:ContainerMembershipProperty .
ex:Sebastian          rdf:type                ex:Nussallergiker .
ex:Sebastian          ex:isst                 ex:ThaiCurry .
ex:ThaiCurry          ex:basiertAuf           ex:Kokosmilch .
```

1. Vokabular-Definitionen
2. Terminologisches Wissen über Klassen und Properties sowie deren Zusammenhänge
3. Assertionales Wissen über Individuen, deren Beziehungen zueinander und zu Klassen

Verschiedene Arten von Informationen in RDF-Graphen

- Alle 3 Arten von Informationen werden als RDF-Tripel formalisiert
- Wichtige Unterschiede
  - in Tripeln über assertionales Wissen dürfen Properties nur Prädikate sein
  - in Tripeln über terminologisches Wissen oder Vokabulare auch Subjekte und Objekte
- In der Praxis: Aufteilung auf mehrere Graphen

Zusammenfassung: RDFS

- RDF-Vokabular mit spezieller Semantik
- Terminologisches Wissen
  - Beziehungen zwischen Klassen
  - Beziehungen zwischen Properties
  - Definitions- und Zielbereiche von Properties
- Anmerkungen mit `rdfs:label` und `rdfs:comment`
- lightweight ontology language

Inhalt

Warum formale Semantik?

- Nach Einführung von RDFS Kritik von Tool-Herstellern:
  Inkompatibilitäten zwischen verschiedenen Tools (trotz Spezifikation)
- Z. B. bei Triple Stores:
  - Gleiches RDF-Dokument
  - Gleiche Anfrage
  - Verschiedene Antworten
- Daher: modelltheoretische Semantik für RDF(S)
- Nachträgliche Ergänzung $\rightsquigarrow$ aufwändige Konstruktion

RDFS-Syntax

- Vokabular V: URIs und Literale
  - bezeichnen Entitäten, keine Sätze
  - entsprechen Konstanten in Prädikatenlogik
- Bnodes
  - bezeichnen unbestimmte Entitäten
  - entsprechen Variablen in Prädikatenlogik
- 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
  - entsprechen Atomen in Prädikatenlogik
- Jeder Graph (endliche Menge von Tripeln) ist ein Satz
  - Konjunktion der Tripel
  - entsprechen komplexen Formeln in Prädikatenlogik

Schlussfolgerungsprobleme

Erfüllbarkeit uninteressant: (Fast) jeder Graph ist erfüllbar

- Keine Negation $\rightsquigarrow$ keine Widersprüche ($A \wedge \neg A$)
- Ausnahme: Widersprüche bei Datentypen
  - `"a"^^xsd:integer`
  - `"7777-88-99"^^xsd:date`
  - `ex:alter rdfs:range xsd:nonNegativeInteger .`
    `ex:Maria ex:alter "-5" .`

Gültigkeit uninteressant: Kein Graph ist gültig

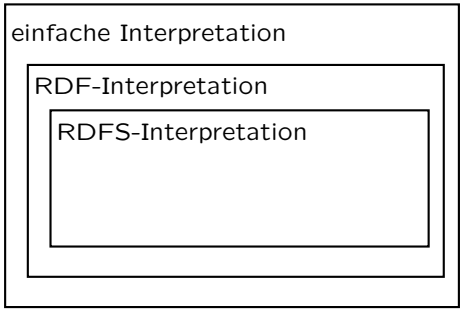
- Keine Negation $\rightsquigarrow$ keine Tautologien ($A \vee \neg A$)

Folgerung Folgt aus einem Graphen G_1 ein Graph G_2 ?

- Ist jedes Modell von G_1 auch Modell von G_2?

Was ist die Semantik?

- Vorgehen schrittweise:



- Je stärker eingeschränkt die Interpretationen
  - um so stärker die Folgerungsrelation
  - um so mehr Tripel sind ableitbar

Inhalt

Definition 2.38 (Einfache Interpretation)

Ein Vokabular $V = (S, L)$ besteht aus einer Menge S von URIs und einer Menge L von Literalen.
Eine einfache Interpretation $\mathcal{I}$ für V besteht aus

- IR, einer nichtleeren Menge von Ressourcen,
- IP, einer Menge von Properties,
- $I_S : S \to IR \cup IP$, einer Funktion von den URIs aus V in die Vereinigung von IR und IP,
- $I_L : L \to IR$, einer Funktion von den Literalen aus V nach IR, und
- $I_{EXT} : IP \to 2^{IR \times IR}$, einer Funktion, die jeder Property aus IP eine Menge von Paaren aus IR zuordnet.

- IR wird Domäne oder Universum von $\mathcal{I}$ genannt
- $I_{EXT}(p)$ wird Extension der Property p genannt

Interpretation von Literalen

Definition 2.39 (Datentyp)

Ein RDF-Datentyp D besteht aus

- lexikalischer Bereich $L(D)$: Menge der zulässigen Strings
- Wertebereich $V(D)$: Zielmenge
- Abbildung $L2V(D)$ von L auf V

URIs I_S kann URIs auf beliebige Elemente von $IR \cup IP$ abbilden

Literale für I_L ist die Interpretationsfunktion durch den Datentyp vorgegeben

- $I_L(l\hat{}\hat{}d) = (L2V(d))(l)$
- $I_L(l\hat{}\hat{}d)$ ist undefiniert, wenn $l \notin L(d)$ gilt

Beispiel 2.40 (Datentyp `xsd:integer`)

- $L(\texttt{xsd:integer}) = (+|-|\varepsilon) \cdot [0..9]^+$
- $V(\texttt{xsd:integer}) = \mathbb{Z}$
- $L2V(\texttt{xsd:integer}) = \{0 \mapsto 0_{dec}; 01 \mapsto 1_{dec}; -0050 \mapsto -50_{dec}, \ldots\}$

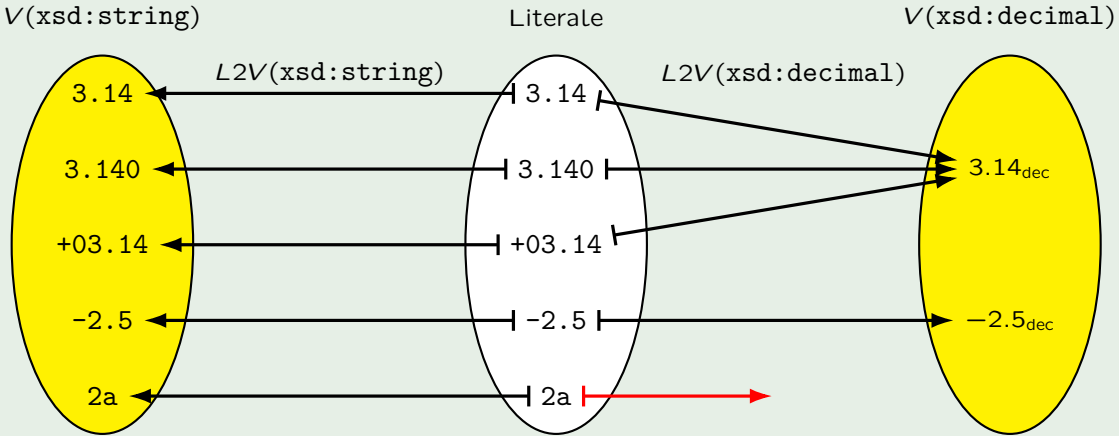
Beispiel 2.41 (Datentyp `xsd:string`)

- $L(\texttt{xsd:string})$ ist die Menge (fast) aller Unicode-Strings
- $V(\texttt{xsd:string}) = L(\texttt{xsd:string})$
- $L2V(\texttt{xsd:string}) = s \mapsto s$

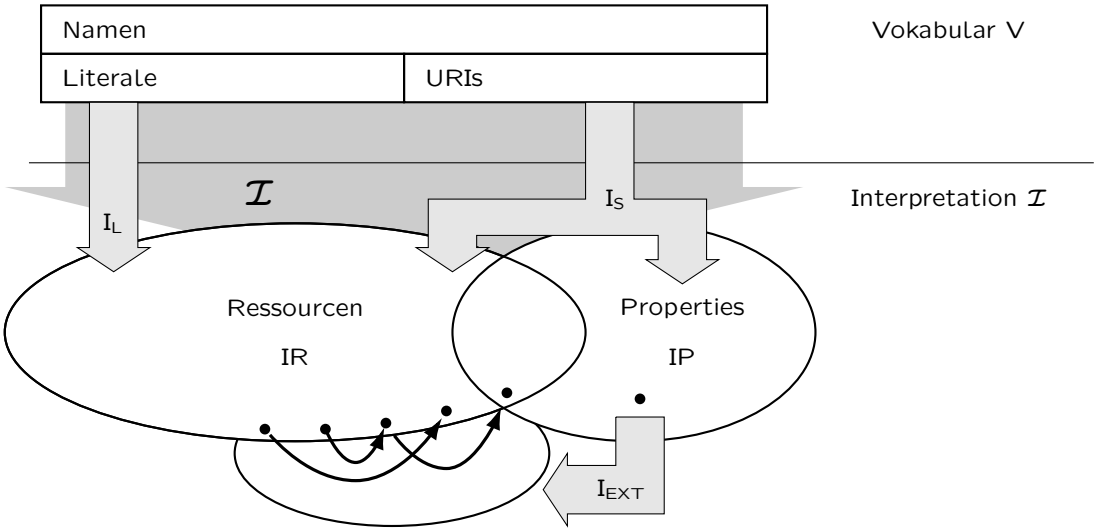
Beispiel 2.42 (Datentyp `rdf:langString`)

- $L(\texttt{rdf:langString}) = \{s@t \mid s \in L(\texttt{xsd:string}) \wedge t \text{ ist ein Sprach-Tag}\}$
- $V(\texttt{rdf:langString}) = \{(s,t) \mid s \in L(\texttt{xsd:string}) \wedge t \text{ ist ein Sprach-Tag}\}$
- $L2V(\texttt{rdf:langString}) = s@t \mapsto (s,t)$

Beispiel 2.43 (xsd:string und xsd:decimal)



Schematische Darstellung der einfachen Interpretation

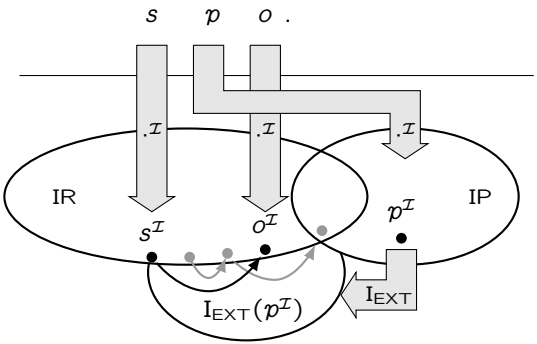


Definition 2.44 (Modell)

Eine Interpretation $\mathcal{I}$ ist Modell des Tripels $s\ p\ o\ .$, wenn

- s, p und o in V enthalten sind und
- $(s^{\mathcal{I}}, o^{\mathcal{I}}) \in \mathrm{I}_{\mathrm{EXT}}(p^{\mathcal{I}})$ gilt.

$\mathcal{I}$ ist Modell des Graphen G, wenn sie Modell jedes Tripels in G ist.



Subjekt/Objekt nach IR abgebildet

Prädikat nach IP abgebildet

Übung: Semantik der einfachen Interpretation

Übung 2.45

Entscheiden Sie, welche der Interpretationen $\mathcal{I}_1, \mathcal{I}_2$ Modelle des Tripels

$$\texttt{ex:s} \quad \texttt{ex:p} \quad \texttt{ex:o} \ .$$

mit Vokabular $V = \{\texttt{ex:s}, \texttt{ex:p}, \texttt{ex:o}\}$ sind:

$$\begin{aligned} \mathcal{I}_1: \quad \mathrm{IR} &= \{a, c\} \\ \mathrm{IP} &= \{b\} \\ \mathrm{I_S} &= \{\texttt{ex:s} \mapsto a, \\ &\quad\ \texttt{ex:p} \mapsto b, \\ &\quad\ \texttt{ex:o} \mapsto c\} \\ \mathrm{I_{EXT}} &= \{b \mapsto \{(a, c)\}\} \end{aligned}$$

$$\begin{aligned} \mathcal{I}_2: \quad \mathrm{IR} &= \{d\} \\ \mathrm{IP} &= \{d\} \\ \mathrm{I_S} &= \{\texttt{ex:s} \mapsto d, \\ &\quad\ \texttt{ex:p} \mapsto d, \\ &\quad\ \texttt{ex:o} \mapsto d\} \\ \mathrm{I_{EXT}} &= \{d \mapsto \{(d, d)\}\} \end{aligned}$$

Interpretation von Bnodes

- Funktion A bildet Bnodes des Graphen auf Elemente von IR ab
- Für eine Interpretation $\mathcal{I}$ sei $\mathcal{I} + A$ wie $\mathcal{I}$, wobei zusätzlich für jeden Bnode `_:id` gilt:

$$(\texttt{\_:id})^{\mathcal{I}+A} = A(\texttt{\_:id})$$

- 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

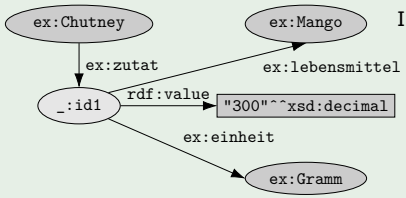
Anschaulich:

- Bnodes werden genau wie URIs und Literale auf Ressourcen abgebildet.
- $\mathcal{I} \models G$ gilt, wenn
  - man jeden Bnode in G auf eine Ressource abbilden kann,
  - so dass alle Tripel wahr werden.

Einfache Interpretation: Beispiel

Beispiel 2.46

Graph G:



Vokabular $V = \{\texttt{ex:Chutney}, \texttt{ex:Mango}, \texttt{ex:Gramm}, 300, \texttt{ex:zutat}, \texttt{ex:lebensmittel}, \texttt{ex:einheit}, \texttt{rdf:value}\}$

Interpretation $\mathcal{I}$:

$$\begin{aligned} \mathrm{IR} &= \{c, m, b, g, 300\} \\ \mathrm{IP} &= \{z, l, v, e\} \\ \mathrm{I_{EXT}} &= \{z \mapsto \{(c,b)\}, \\ &\quad\ l \mapsto \{(b,m)\}, \\ &\quad\ v \mapsto \{(b,300)\}, \\ &\quad\ e \mapsto \{(b,g)\}\} \end{aligned} \qquad \begin{aligned} \mathrm{I_S} &= \{\texttt{ex:Chutney} \mapsto c \\ &\quad\ \texttt{ex:Mango} \mapsto m, \\ &\quad\ \texttt{ex:zutat} \mapsto z, \\ &\quad\ \texttt{ex:lebensmittel} \mapsto l, \\ &\quad\ \texttt{rdf:value} \mapsto v \\ &\quad\ \texttt{ex:einheit} \mapsto e, \} \\ \mathrm{I_L} &= \{300 \mapsto 300\} \end{aligned}$$

Ist $\mathcal{I}$ ein Modell von G?

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

$$\begin{aligned} (\texttt{ex:Chutney}^{\mathcal{I}+A}, \texttt{\_:id1}^{\mathcal{I}+A}) &= (c,b) \quad \in \mathrm{I_{EXT}}(z) = \mathrm{I_{EXT}}(\texttt{ex:zutat}^{\mathcal{I}+A}) \\ (\texttt{\_:id1}^{\mathcal{I}+A}, \texttt{ex:Mango}^{\mathcal{I}+A}) &= (b,m) \quad \in \mathrm{I_{EXT}}(l) = \mathrm{I_{EXT}}(\texttt{ex:lebensmittel}^{\mathcal{I}+A}) \\ (\texttt{\_:id1}^{\mathcal{I}+A}, \texttt{"300"}^{\mathcal{I}+A}) &= (b,300) \in \mathrm{I_{EXT}}(v) = \mathrm{I_{EXT}}(\texttt{rdf:value}^{\mathcal{I}+A}) \\ (\texttt{\_:id1}^{\mathcal{I}+A}, \texttt{ex:Gramm}^{\mathcal{I}+A}) &= (b,g) \quad \in \mathrm{I_{EXT}}(e) = \mathrm{I_{EXT}}(\texttt{ex:einheit}^{\mathcal{I}+A}) \end{aligned}$$

- Also ist $\mathcal{I}$ Modell von G.

Einfache Folgerung

Definition 2.47

Aus einem RDF-Graphen G_1 folgt einfach ein RDF-Graph G_2 ($G_1 \models_S G_2$), wenn jedes Modell von G_1 auch Modell von G_2 ist.

Entscheidungsalgorithmus für einfache Folgerung

- Resolution / Tableaus nicht einsetzbar
  - entscheiden Erfüllbarkeit
  - Reduktion von Folgerung auf Erfüllbarkeit verwendet Negation
    $\varphi \models \psi$ gdw. $\varphi \wedge \neg\psi$ unerfüllbar
- Deshalb: Ableitungsregeln

$$\frac{\text{[Prämisse 1]} \qquad \text{[Prämisse 2]}}{\text{[Konklusion]}}\ \text{[Name der Regel]}$$

Einfache Folgerung: Ableitungsregeln

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

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

`a`	URI
`u`	URI oder Bnode
`x`	URI, Bnode oder Literal
`_:n`	Bnode

Bedingung für Anwendung: `_:n` ist neu, d. h. nicht bereits anderem URI / Literal zugeordnet

Anschaulich:

- Jeder Subjekt- oder Objekt-Knoten kann zu einem Bnode abgeschwächt werden.
- Umkehrung der Skolemisierung

Satz 2.48

Aus einem Graphen G_1 folgt einfach ein Graph G_2 genau dann, wenn G_1 mithilfe der Regeln se1 und se2 zu einem Graphen G_1' ergänzt werden kann, so dass G_2 in G_1' enthalten ist.

Beispiel 2.49 (Einfache Folgerung)

G_1

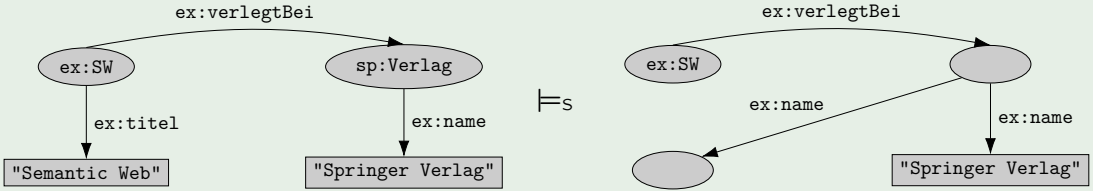
```
ex:SW ex:verlegtBei sp:Verlag .
ex:SW ex:titel "Semantic Web" .
sp:Verlag ex:name "Springer Verlag" .
```

G_1'

```
ex:SW ex:verlegtBei sp:Verlag .
ex:SW ex:titel "Semantic Web" .
sp:Verlag ex:name "Springer Verlag" .
ex:SW ex:verlegtBei _:n1 .
 _:n1 ex:name "Springer Verlag" .
 _:n1 ex:name _:n2 .
```

G_2

```
ex:SW ex:verlegtBei _:n1 .
_:n1 ex:name "Springer Verlag" .
_:n1 ex:name _:n2 .
```



Übung: Einfache Folgerung

Übung 2.50

Gegeben sei der Graph G_1:

```
ex:JohnDoe  foaf:firstName  "John";
            foaf:lastName   "Doe";
            ex:worksFor     ex:ACME .
```

Welche der Graphen G_2, G_3, G_4, G_5 folgen einfach aus G_1?
Geben Sie ggf. die Ableitungsschritte an.

G_2:
```
ex:JohnDoe  ex:worksFor     _:n21 ;
            ex:worksFor     _:n22 .
```

G_3:
```
ex:JohnDoe  _:n3            "John" .
```

G_4:
```
ex:JohnDoe  foaf:firstName  _:n4 ;
            foaf:lastName   _:n4 .
```

G_5:
```
_:n51       foaf:firstName  _:n52 ;
            ex:worksFor     ex:ACME .
```

Inhalt

RDF-Interpretationen

- stellen höhere Anforderungen als einfache Interpretationen
- Literale müssen richtigen Datentyp haben
- RDF-Vokabular V_{RDF} muss „richtig“ behandelt werden
  - richtig: entsprechend intendierter Semantik

- `rdf:type`
- `rdf:Property`

- `rdf:List`
- `rdf:Seq`
- `rdf:Bag`
- `rdf:Alt`

- `rdf:first`
- `rdf:rest`
- `rdf:nil`
- `rdf:_1`,
- `rdf:_2`,
- ...

Definition 2.51 (RDF-Interpretation)

Eine RDF-Interpretation $\mathcal{I}$ für ein Vokabular V ist eine einfache Interpretation für das Vokabular $V \cup V_{RDF}$, die zusätzlich folgende Bedingungen erfüllt:

1. Für jeden Datentyp D gilt: $(x, D^{\mathcal{I}}) \in I_{EXT}(\texttt{rdf:type}^{\mathcal{I}})$ gdw. $x \in V(D)$.
   Ein Domänenelement x ist vom Datentyp D genau dann, wenn x im Wertebereich von D liegt.

$$\frac{\texttt{u a l\^\^d .}}{\texttt{u a \_:n .}\quad \texttt{\_:n rdf:type d .}}\ \text{rdfD1}$$

a	URI
u	URI oder Bnode
l	Literal
d	Datentyp-URI
_:n	neuer Bnode

RDF-Interpretation: Properties

2. $a \in \mathrm{IP}$ genau dann, wenn $(a, \texttt{rdf:Property}^{\mathcal{I}}) \in \mathrm{I}_{\mathrm{EXT}}(\texttt{rdf:type}^{\mathcal{I}})$.
   Jedes Prädikat ist vom Typ `rdf:Property`.

$$\frac{\texttt{u a x .}}{\texttt{a rdf:type rdf:Property .}}\ \text{rdfD2}$$

`a`	URI
`u`	URI oder Bnode
`x`	URI, Bnode oder Literal

- Wenn `a` als Prädikat verwendet wird, ist es über `rdf:type` mit `rdf:Property` verbunden.
- Dies impliziert $\mathrm{IP} \subseteq \mathrm{IR}$.

RDF-Interpretation: Axiomatische Tripel

3 $\mathcal{I}$ ist Modell der folgenden axiomatischen Tripel:

```
rdf:type    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:type   rdf:Property .
rdf:nil     rdf:type   rdf:List .
```

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

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

Definition 2.52 (RDF-Folgerung)

Aus einem Graphen G_1 RDF-folgt ein Graph G_2 ($G_1 \models_{\text{RDF}} G_2$), wenn für jede RDF-Interpretation $\mathcal{I}$ gilt: Ist $\mathcal{I}$ Modell von G_1, dann ist $\mathcal{I}$ auch Modell von G_2.

Satz 2.53

Aus G_1 RDF-folgt G_2 genau dann, wenn es einen Graphen G_1' gibt, so dass

- aus G_1 der Graph G_1' via rdfD1, rdfD2 und rdfax hergeleitet werden kann und
- aus G_1' der Graph G_2 einfach folgt.

Beachte: zweistufiger Folgerungsprozess $G_1 \overset{\text{rdf}}{\rightsquigarrow} G_1' \overset{\text{se}}{\rightsquigarrow} G_2$

Inhalt

RDFS-Interpretationen

- stellen höhere Anforderungen als RDF-Interpretationen
- RDFS-Vokabular V_{RDFS} muss „richtig“ behandelt werden

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

RDFS-Interpretation: Hilfsdefinitionen

Erinnerung: Eine einfache Interpretation $\mathcal{I}$ für ein Vokabular V besteht aus

- IR, einer nicht leeren Menge von Ressourcen,
- $\mathrm{I}_{\mathrm{EXT}}$, einer Funktion, die jeder Property eine Menge von Paaren aus IR zuordnet, also $\mathrm{I}_{\mathrm{EXT}} : \mathrm{IP} \to 2^{\mathrm{IR} \times \mathrm{IR}}$, ...

Definition 2.54 ($\mathrm{I}_{\mathrm{CEXT}}$, IC)

- Für eine gegebene Interpretation $\mathcal{I}$ bildet die Funktion $\mathrm{I}_{\mathrm{CEXT}} : \mathrm{IR} \to 2^{\mathrm{IR}}$ jede Ressource auf die Menge der Ressourcen ab, für die (x, y) in $\mathrm{I}_{\mathrm{EXT}}(\texttt{rdf:type}^{\mathcal{I}})$ enthalten ist. $\mathrm{I}_{\mathrm{CEXT}}(y)$ nennt man auch die (Klassen-)Extension von y.
- IC ist die Extension der speziellen URI `rdfs:Class`, also: $\mathrm{IC} = \mathrm{I}_{\mathrm{CEXT}}(\texttt{rdfs:Class}^{\mathcal{I}})$.

Sowohl $\mathrm{I}_{\mathrm{CEXT}}$ als auch IC sind durch $\cdot^{\mathcal{I}}$ sowie $\mathrm{I}_{\mathrm{EXT}}$ bereits eindeutig festgelegt.

Definition 2.55 (RDFS-Interpretation)

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

1. Für jeden Datentyp d gilt: $d^{\mathcal{I}} \in I_{CEXT}(\texttt{rdfs:Datatype}^{\mathcal{I}})$.
   Jeder Datentyp ist vom Typ `rdfs:Datatype`.

$$\frac{\texttt{u a l\^\^d .}}{\texttt{d rdf:type rdfs:Datatype .}}$$ rdfs1 Datentypen

`a`	URI
`u`	URI oder Bnode
`l`	Literal
`d`	Datentyp-URI

RDFS-Interpretation: `rdfs:domain` und `rdfs:range`

2. Aus $(a, x) \in \mathrm{I}_{\mathrm{EXT}}(\texttt{rdfs:domain}^{\mathcal{I}})$ und $(u, y) \in \mathrm{I}_{\mathrm{EXT}}(a)$ folgt $u \in \mathrm{I}_{\mathrm{CEXT}}(x)$.
   Hat die Property a den Definitionsbereich x und gehört die Ressource u zum Definitionsbereich von a, dann ist u vom Typ x.
3. Aus $(a, x) \in \mathrm{I}_{\mathrm{EXT}}(\texttt{rdfs:range}^{\mathcal{I}})$ und $(u, v) \in \mathrm{I}_{\mathrm{EXT}}(a)$ folgt $v \in \mathrm{I}_{\mathrm{CEXT}}(x)$.
   Hat die Property a den Wertebereich x und gehört die Ressource v zum Wertebereich von a, dann ist v vom Typ x.

$$\frac{\texttt{a rdfs:domain x .} \quad \texttt{u a y .}}{\texttt{u rdf:type x .}}$$ rdfs2 Definitionsbereiche von Properties

$$\frac{\texttt{a rdfs:range x .} \quad \texttt{u a v .}}{\texttt{v rdf:type x .}}$$ rdfs3 Wertebereiche von Properties

`a`	URI
`u, v`	URI oder Bnode
`x, y`	URI, Bnode oder Literal

RDFS-Interpretation: `rdfs:Resource`

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

4. $IR = I_{CEXT}(\texttt{rdfs:Resource}^{\mathcal{I}})$
   Jedes Domänenelement ist vom Typ `rdfs:Resource`.

$$\frac{\texttt{u a x .}}{\texttt{u rdf:type rdfs:Resource .}}$$ rdfs4a — Das Subjekt jedes Tripels ist eine Ressource

$$\frac{\texttt{u a v .}}{\texttt{v rdf:type rdfs:Resource .}}$$ rdfs4b — Ein Objekt, das kein Literal ist, ist eine Ressource

a	URI
u, v	URI oder Bnode
x	URI, Bnode oder Literal

RDFS-Interpretation: `rdfs:subPropertyOf`

5. Die Relation $I_{EXT}(\texttt{rdfs:subPropertyOf}^{\mathcal{I}})$ ist transitiv.
   Eine Unterproperty einer Unterproperty von x ist auch eine Unterproperty von x.
6. Die Relation $I_{EXT}(\texttt{rdfs:subPropertyOf}^{\mathcal{I}})$ ist reflexiv.
   Jede Property ist ihre eigene Unterproperty.
7. $(a, b) \in I_{EXT}(\texttt{rdfs:subPropertyOf}^{\mathcal{I}})$ impliziert $I_{EXT}(a) \subseteq I_{EXT}(b)$.
   Ist a Unterproperty von b, ist die Extension von a eine Teilmenge der Extension von b.

$$\frac{\texttt{u rdfs:subPropertyOf v .} \quad \texttt{v rdfs:subPropertyOf x .}}{\texttt{u rdfs:subPropertyOf x .}}$$ rdfs5 Transitivität

$$\frac{\texttt{u rdf:type rdf:Property .}}{\texttt{u rdfs:subPropertyOf u .}}$$ rdfs6 Reflexivität

$$\frac{\texttt{a rdfs:subPropertyOf b .} \quad \texttt{u a y .}}{\texttt{u b y .}}$$ rdfs7 Unterproperties und Instanzen

a, b	URI
u, v	URI oder Bnode
x, y	URI, Bnode oder Literal

RDFS-Interpretation: Klassen und Ressourcen

8 Aus $u \in \mathrm{IC}$ folgt $(u, \texttt{rdfs:Resource}^{\mathcal{I}}) \in \mathrm{I}_{\mathrm{EXT}}(\texttt{rdfs:subClassOf}^{\mathcal{I}})$.
Jede Klasse ist Unterklasse von `rdfs:Resource`.

$$\frac{\texttt{u rdf:type rdfs:Class .}}{\texttt{u rdfs:subClassOf rdfs:Resource .}}$$ rdfs8 Klassen enthalten nur Ressourcen

`u` URI oder Bnode

RDFS-Interpretation: `rdfs:subClassOf`

9. $(u, x) \in \mathrm{I}_{\mathrm{EXT}}(\texttt{rdfs:subClassOf}^{\mathcal{I}})$ impliziert $\mathrm{I}_{\mathrm{CEXT}}(u) \subseteq \mathrm{I}_{\mathrm{CEXT}}(x)$.
   Ist u Unterklasse von x, ist jede Instanz von u ist auch Instanz von x.
10. Die Relation $\mathrm{I}_{\mathrm{EXT}}(\texttt{rdfs:subClassOf}^{\mathcal{I}})$ ist reflexiv.
    Jede Klasse ist ihre eigene Unterklasse.
11. Die Relation $\mathrm{I}_{\mathrm{EXT}}(\texttt{rdfs:subClassOf}^{\mathcal{I}})$ ist transitiv.
    Eine Unterklasse einer Unterklasse von x ist auch eine Unterklasse von x.

$$\frac{\texttt{u rdfs:subClassOf x .} \quad \texttt{v rdf:type u .}}{\texttt{v rdf:type x .}}$$ rdfs9 — Unterklassen und Instanzen

$$\frac{\texttt{u rdf:type rdfs:Class .}}{\texttt{u rdfs:subClassOf u .}}$$ rdfs10 — Reflexivität

$$\frac{\texttt{u rdfs:subClassOf v .} \quad \texttt{v rdfs:subClassOf x .}}{\texttt{u rdfs:subClassOf x .}}$$ rdfs11 — Transitivität

u, v	URI oder Bnode
x	URI, Bnode oder Literal

RDFS-Interpretation: Weitere Anforderungen

12 Aus $u \in I_{CEXT}(\texttt{rdfs:ContainerMembershipProperty}^{\mathcal{I}})$ folgt $(u, \texttt{rdfs:member}^{\mathcal{I}}) \in I_{EXT}(\texttt{rdfs:subPropertyOf}^{\mathcal{I}})$.
Jede Property vom Typ `rdfs:ContainerMembershipProperty` *ist Unterproperty von* `rdfs:member`.

13 Aus $u \in I_{CEXT}(\texttt{rdfs:Datatype}^{\mathcal{I}})$ folgt $(u, \texttt{rdfs:Literal}^{\mathcal{I}}) \in I_{EXT}(\texttt{rdfs:subClassOf}^{\mathcal{I}})$.
Jede Klasse vom Typ `rdfs:Datatype` *ist eine Unterklasse der Klasse aller Literalwerte (*`rdfs:Literal`*).*

$$\frac{\texttt{u rdf:type rdfs:ContainerMembershipProperty .}}{\texttt{u rdfs:subPropertyOf rdfs:member .}}\ \text{rdfs12}$$

CMPs und `rdfs:member`

$$\frac{\texttt{u rdf:type rdfs:Datatype .}}{\texttt{u rdfs:subClassOf rdfs:Literal .}}\ \text{rdfs13}$$

Datentypen und `rdfs:Literal`

`u` URI oder Bnode

RDFS-Interpretation: Axiomatische Tripel

14 Axiomatische Tripel können immer abgeleitet werden.

$$\frac{}{\texttt{u a x .}}\ \text{rdfsax}$$

Jedes axiomatische Tripel "u a x ." kann immer abgeleitet werden

```
rdf:type             rdfs:domain          rdfs:Resource .
rdf:type             rdfs:range           rdfs:Class .
rdfs:domain          rdfs:domain          rdf:Property .
rdfs:domain          rdfs:range           rdfs:Class .
rdfs:range           rdfs:domain          rdf:Property .
rdfs:range           rdfs:range           rdfs:Class .
rdfs:subPropertyOf   rdfs:domain          rdf:Property .
rdfs:subPropertyOf   rdfs:range           rdf:Property .
rdfs:subClassOf      rdfs:domain          rdfs:Class .
rdfs:subClassOf      rdfs:range           rdfs:Class .
rdfs:Datatype        rdfs:subClassOf      rdfs:Class .
```

RDFS: Axiomatische Tripel für Annotationen

```
rdfs:label          rdfs:domain          rdfs:Resource .
rdfs:label          rdfs:range           rdfs:Literal .

rdfs:comment        rdfs:domain          rdfs:Resource .
rdfs:comment        rdfs:range           rdfs:Literal .

rdfs:isDefinedBy    rdfs:domain          rdfs:Resource .
rdfs:isDefinedBy    rdfs:range           rdfs:Resource .
rdfs:isDefinedBy    rdfs:subPropertyOf   rdfs:seeAlso .

rdfs:seeAlso        rdfs:domain          rdfs:Resource .
rdfs:seeAlso        rdfs:range           rdfs:Resource .

rdf:value           rdfs:domain          rdfs:Resource .
rdf:value           rdfs:range           rdfs:Resource .
```

RDFS: Axiomatische Tripel für Listen

```
rdf:first                rdfs:domain                rdf:List .
rdf:first                rdfs:range                 rdfs:Resource .
rdf:rest                 rdfs:domain                rdf:List .
rdf:rest                 rdfs:range                 rdf:List .

   rdf:Alt                  rdfs:subClassOf         rdfs:Container .
   rdf:Bag                  rdfs:subClassOf         rdfs:Container .
   rdf:Seq                  rdfs:subClassOf         rdfs:Container .
   rdfs:ContainerMembershipProperty
                            rdfs:subClassOf         rdf:Property .
   rdfs:member              rdfs:domain             rdfs:Resource .
   rdfs:member              rdfs:range              rdfs:Resource .
   rdf:_1                   rdf:type                rdfs:ContainerMembershipProperty .
   rdf:_1                   rdfs:domain             rdfs:Resource .
   rdf:_1                   rdfs:range              rdfs:Resource .
   rdf:_2                   rdf:type                rdfs:ContainerMembershipProperty .
   ...
```

Definition 2.56 (RDFS-Folgerung)

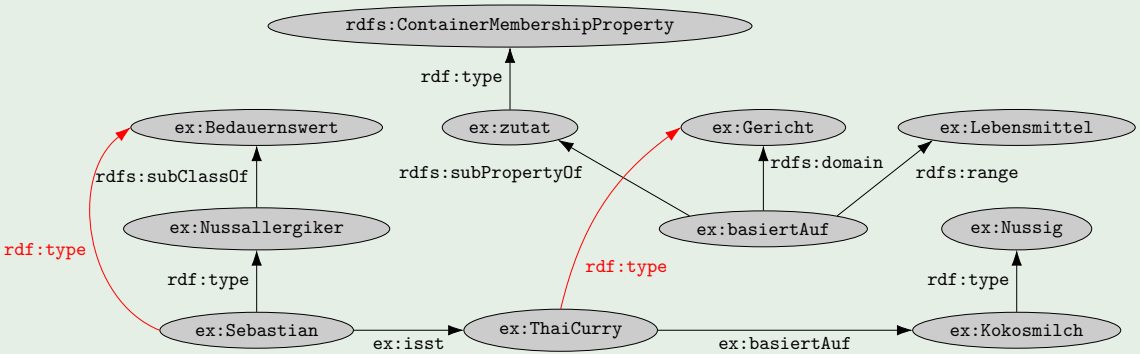
Aus einem Graphen G_1 RDFS-folgt ein Graph G_2 ($G_1 \models_{\text{RDFS}} G_2$), wenn für jede RDFS-Interpretation $\mathcal{I}$ gilt: Ist $\mathcal{I}$ Modell von G_1, dann ist $\mathcal{I}$ auch Modell von G_2.

Satz 2.57

Aus G_1 RDFS-folgt G_2 genau dann, wenn es einen Graphen G_1' gibt, so dass

- aus G_1 der Graph G_1' via rdfD1, rdfD2, rdfax, rdfs1 – rdfs13 und rdfsax hergeleitet werden kann und
- aus G_1' der Graph G_2 einfach folgt.

Beispiel 2.58 (Thai-Gericht)



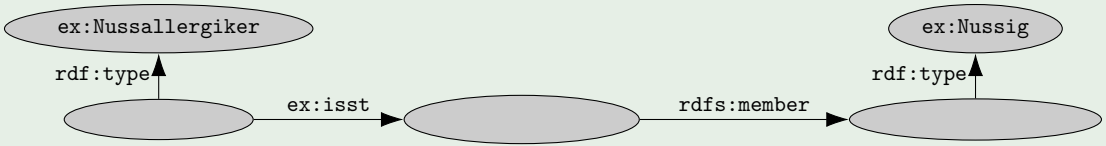
$$\frac{\texttt{ex:NA rdfs:subClassOf ex:Bedauernswert .} \quad \texttt{ex:Sebastian rdf:type ex:NA .}}{\texttt{ex:Sebastian rdf:type ex:Bedauernswert .}}\ \text{rdfs9}$$

$$\frac{\texttt{ex:basiertAuf rdfs:domain ex:Gericht .} \quad \texttt{ex:ThaiCurry ex:basiertAuf ex:Kokosmilch .}}{\texttt{ex:ThaiCurry rdf:type ex:Gericht .}}\ \text{rdfs2}$$

Beispiel: RDFS-Folgerung (2)

Beispiel 2.59 (Unerwünschte Situation)

„Ein Nussallergiker isst etwas, das etwas Nussiges enthält.“



$$\frac{\texttt{ex:zutat rdf:type rdfs:ContainerMembershipProperty .}}{\texttt{ex:zutat rdfs:subPropertyOf rdfs:member .}}\ \text{rdfs12}$$

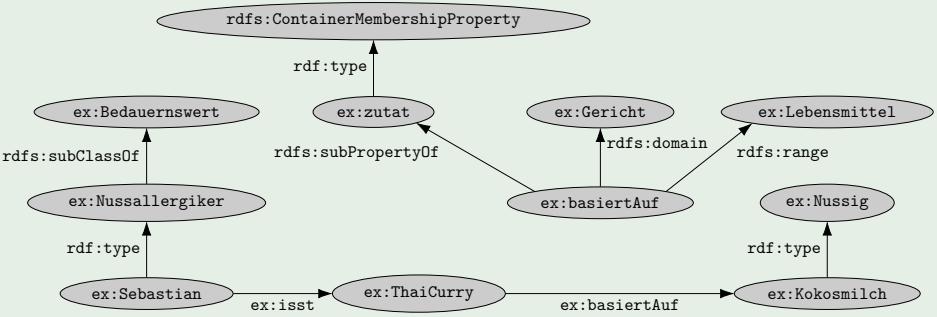
$$\frac{\texttt{ex:basiertAuf rdfs:subPropertyOf ex:zutat .} \quad \texttt{ex:zutat rdfs:subPropertyOf rdfs:member .}}{\texttt{ex:basiertAuf rdfs:subPropertyOf rdfs:member .}}\ \text{rdfs5}$$

$$\frac{\texttt{ex:basiertAuf rdfs:subPropertyOf rdfs:member .} \quad \texttt{ex:ThaiCurry ex:basiertAuf ex:Kokosmilch .}}{\texttt{ex:ThaiCurry rdfs:member ex:Kokosmilch .}}\ \text{rdfs7}$$

$\rightsquigarrow$ Aus dem „Thai-Gericht“-Graphen RDFS-folgt der „unerwünschte“ Graph.

Übung: RDFS-Folgerung (1)

Übung 2.60



Nennen Sie für den Thai-Gericht-Graphen

1. ein Tripel, das einfach folgt,
2. ein Tripel, das RDF-folgt, aber nicht einfach folgt,
3. ein Tripel, das RDFS-folgt, aber nicht RDF-folgt.

Übung: RDFS-Folgerung (2)

Übung 2.61

Gegeben sei der RDF-Graph G:

a. `ex:Klaus ex:freund ex:Anna .`
b. `ex:freund rdfs:subPropertyOf ex:mag .`
c. `ex:freund rdfs:domain ex:Mensch .`
d. `ex:freund rdfs:range ex:Mensch .`
e. `ex:Mann rdfs:subClassOf ex:Mensch .`
f. `ex:besitzt rdfs:domain ex:Mensch .`
g. `ex:besitzt rdfs:range ex:Objekt .`
h. `ex:Anna ex:besitzt ex:AnnasHaus .`
i. `ex:AnnasHaus rdf:type ex:Haus .`
j. `ex:mag rdfs:subPropertyOf ex:kennt .`

Geben Sie für jedes der folgenden Tripel an, ob es aus G unter der RDFS-Semantik folgt. Falls das Tripel folgt, geben Sie eine Ableitung an.

1. `ex:Klaus rdf:type ex:Mann .`
2. `ex:Anna rdf:type ex:Mensch .`
3. `ex:Klaus ex:mag ex:Anna .`
4. `ex:Anna ex:freund ex:Klaus .`
5. `_:n rdf:type ex:Objekt .`
   (`_:n` ist ein neuer bnode.)
6. `ex:Haus rdfs:subClassOf ex:Objekt .`
7. `ex:freund rdfs:subPropertyOf ex:kennt`

Übung: RDFS-Folgerung (3)

Übung 2.62

Der leere Graph enthält keine Tripel (entspricht also der leeren Menge).
Zeigen Sie durch Ableitungen, dass die folgenden Tripel aus dem leeren Graphen RDFS-folgen:

1. `rdfs:Resource rdf:type rdfs:Class .`
2. `rdfs:Class rdf:type rdfs:Class .`
3. `rdfs:Literal rdf:type rdfs:Class .`
4. `rdf:Property rdf:type rdfs:Class .`
5. `rdf:Property rdfs:subClassOf rdfs:Resource .`
6. `rdfs:domain rdf:type rdf:Property .`
7. `rdfs:label rdf:type rdf:Property .`

Zusammenfassung: Schlussfolgerungsverfahren

Wesentliche Fähigkeiten der einzelnen Folgerungsmechanismen

- Einfache Folgerung
  - Abschwächen von Knoten zu Bnodes
  - „John belegt die Vorlesung Semantic Web“
    $\rightsquigarrow$ „Jemand belegt Semantic Web“ oder „John belegt etwas“
- RDF-Folgerung
  - Behandlung von Properties
    - jedes Prädikat ist eine Property $\rightsquigarrow$ Jede Property ist eine Ressource
    - RDF-Standard-Properties `rdf:type`, `rdf:first`, `rdf:_1`, ...
- RDFS-Folgerung
  - `rdfs:domain` und `rdfs:range` werden realisiert
  - `rdfs:subClassOf` und `rdfs:subPropertyOf` werden realisiert
  - Axiome für RDFS-Standard-Vokabular:
    `rdfs:domain`/`rdfs:range`/`rdfs:subClassOf`/...-Beziehungen

Grenzen von RDFS: Folgerungsverfahren

- Manche intuitive Folgerungen können nicht RDFS-gefolgert werden.

  Aus

  ```
  ex:sprichtMit  rdfs:domain      ex:Mensch .
  ex:Mensch      rdfs:subClassOf  ex:Lebewesen .
  ```

  folgt nicht

  ```
  ex:sprichtMit  rdfs:domain  ex:Lebewesen .
  ```

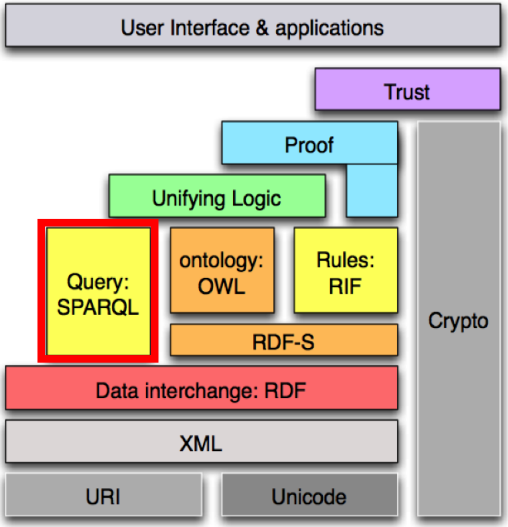
- Mögliche Lösung:
  - noch stärkere Semantik
  - entsprechende Schlussfolgerungsverfahren
  - $\rightsquigarrow$ aufwändigere Algorithmen

Grenzen von RDFS: Ausdrucksstärke

- Keine Möglichkeit, Negation auszudrücken
  - Kein Widerspruch in
    ```
    ex:John  rdf:type  ex:Male .
    ex:John  rdf:type  ex:Female .
    ```
- Allgemein: (Fast) keine Möglichkeit für Inkonsistenzen
  - Vorteil: Falsche Information beeinflusst nicht andere Informationen
  - Nachteil: Keine automatische Fehlererkennung
- Keine Möglichkeit, auszudrücken, dass verschiedene URIs dieselbe Ressource bezeichnen
  - `ex:Stuttgart` ist dasselbe wie `dbpedia:Stuttgart`

Inhalt

SPARQL im Schichtenmodell



SPARQL Protocol And RDF Query Language

- W3C-Spezifikation seit 2008
- Erweiterung auf SPARQL 1.1 im Jahr 2013
  - Aggregatfunktionen
  - Property Paths
  - Negation
- Große praktische Bedeutung

Teile der SPARQL-Spezifikation

- Anfragesprache: Zulässige Anfragen und deren Bedeutung
- Ergebnisformat: Darstellung von Ergebnissen in XML
- Anfrageprotokoll: Übermittlung von Anfragen und Ergebnissen

Inhalt

Beispiel 3.1 (SPARQL-Anfrage)

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?mbox
WHERE { ?x foaf:name ?name .
        ?x foaf:mbox ?mbox }
```

- Die Bedingung der WHERE-Klausel heißt *Anfragemuster oder Muster (query pattern)*
- Ein Tripel mit Variablen heißt *Tripel-Muster (triple pattern)*
- Variablen beginnen mit „?“
- Eine Menge von Tripel-Mustern heißt *einfaches Graph-Muster (basic graph pattern, BGP)*
- Abgekürzte IRIs sind möglich (`PREFIX`)
- Anfrageergebnis für die selektierten Variablen (`SELECT`)

Beispiel 3.2

Muster:

```
{  ?x foaf:name ?name .
   ?x foaf:mbox ?mbox .  }
```

Graph:

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix ex: <http://example.org/> .

ex:Jan foaf:name "Jan Hladik" ;
       foaf:mbox "jan.hladik@dhbw-stuttgart.de" ;
       foaf:icqChatID "hladik" .
ex:Seb foaf:name "Sebastian Rudolph" ;
       foaf:mbox <mailto:rudolph@kit.edu> .
ex:Pascal foaf:name "Pascal Hitzler" ;
          foaf:aimChatID "phi" .
```

BGP-Matching-Ergebnis:

x	name	mbox
ex:Jan	"Jan Hladik"	"jan.hladik@dhbw-stuttgart.de"
ex:Seb	"Sebastian Rudolph"	<mailto:rudolph@kit.edu>

Matching- und Anfrage-Ergebnis

Zurückgegeben werden nur die Ergebnisse für Variablen im SELECT-Ausdruck.

Beispiel 3.3

Anfrage:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?mbox
WHERE { ?x foaf:name ?name .
        ?x foaf:mbox ?mbox . }
```

BGP-Matching-Ergebnis:

x	name	mbox
ex:Jan	"Jan Hladik"	"jan.hladik@dhbw-stuttgart.de"
ex:Seb	"Sebastian Rudolph"	<mailto:rudolph@kit.edu>

Anfrage-Ergebnis:

name	mbox
"Jan Hladik"	"jan.hladik@dhbw-stuttgart.de"
"Sebastian Rudolph"	<mailto:rudolph@kit.edu>

Definition 3.4 (Einfaches Graph-Muster, BGP)

Ein Einfaches Graph-Muster (BGP) ist eine in {, } eingeschlossene Menge von RDF-Tripeln in TTL-Syntax, wobei

- Variablen anstelle von Subjekt, Prädikat oder Objekt zulässig sind,
- Variablen durch „? “ gekennzeichnet werden,
- TTL-Abkürzungen (mit , und ;) erlaubt sind,
- der letzte Punkt vor der schließenden Klammer optional ist.

Übung: DBpedia und SPARQL

Übung 3.5

Der SPARQL-Endpoint der DBpedia hat den URL
`http://dbpedia.org/sparql`

Finden Sie heraus, welche Ober- und Unterklassen die Klasse „Person“ in der DBpedia hat. (URI: `dbo:Person` mit Präfix `dbo: http://dbpedia.org/ontology/`)

Können Sie auch feststellen, für welche Properties diese Klasse als Domain und Range definiert ist?

Inhalt

Gruppierte Graph-Muster

Einfache Graph-Muster können durch $\{\ldots\}$ gruppiert werden:

Beispiel 3.6

```
PREFIX ex:  <http://example.org/>
SELECT ?titel ?autor
WHERE
      { { ?buch ex:verlegtBei <http://springer.com> .
          ?buch ex:titel ?titel .
        }
        { }
        ?buch ex:autor ?autor .
      }
```

Sinnvoll erst bei Verwendung zusätzlicher Konstruktoren

Optionale Muster

Das Schlüsselwort OPTIONAL erlaubt die Angabe optionaler Teile eines Musters.

Beispiel 3.7

```
{ ?buch ex:verlegtBei <http://springer.com> .
  OPTIONAL { ?buch ex:titel ?titel . }
  OPTIONAL { ?buch ex:autor ?autor . }
}
```

⇝ Teile eines Anfrageergebnisses können ungebunden sein:

buch	titel	autor
<http://example.org/buch1>	"Titel1"	<http://example.org/autor1>
<http://example.org/buch2>	"Titel2"	
<http://example.org/buch3>	"Titel3"	_:a
<http://example.org/buch4>		_:a
<http://example.org/buch5>		

Alternative Muster

Das Schlüsselwort UNION erlaubt die Angabe alternativer Teile eines Musters

Beispiel 3.8

```
{ ?buch ex:verlegtBei <http://springer.com> .
  { ?buch ex:autor ?autor . } UNION
  { ?buch ex:herausgeber ?autor . }
}
```

- gleiche Variablennamen in beiden Teilen von UNION beeinflussen sich nicht
- Ergebnis: Vereinigung der Ergebnisse mit einer der beiden Bedingungen
- ähnlich logischer Disjunktion

Übung: Installation von Jena-Fuseki

Übung 3.9

Installieren Sie den Jena-Fuseki-Server:

1. Laden Sie das Paket von
   `https://jena.apache.org/download/index.cgi`
2. Starten Sie den Server mit
   `[./]fuseki-server -update -mem /ds`
3. Testen Sie im Browser (Firefox) mit dem folgenden URL, ob der Server läuft:
   `http://localhost:3030/`
4. Fügen Sie mit „add data“ den folgenden RDF-Graphen hinzu:
   `http://wwwlehre.dhbw-stuttgart.de/~hladik/SW/Ontologien/books.ttl`

Übung: Graph-Muster

Übung 3.10

Der Graph `books.ttl` enthält die folgenden Tripel:

```
@prefix dc10:  <http://purl.org/dc/elements/1.0/> .
@prefix dc11:  <http://purl.org/dc/elements/1.1/> .
@prefix ex: <http://example.org/> .
_:a  dc10:title     "SPARQL Query Tutorial" .
_:a  dc10:creator   "Alice" .
_:b  dc11:title     "SPARQL Protocol Tutorial" .
_:b  dc11:creator   "Bob" .
_:b  ex:level "beginner" .
```

Schreiben Sie eine Anfrage, die Titel (`dc10:title` oder `dc11:title`) und, wo vorhanden, die Stufe (`ex:level`) aus den oben gegebenen Daten selektiert. Testen Sie Ihre Anfrage mit Hilfe des Fuseki-Servers.

Kombination von Optionen und Alternativen

Wie sind Kombinationen von `OPTIONAL` und `UNION` zu verstehen?

Beispiel 3.11 (Muster mit Option und Vereinigung)

```
{ ?buch ex:verlegtBei  <http://springer.com> .
  { ?buch ex:autor ?autor . } UNION
  { ?buch ex:herausgeber ?autor . } OPTIONAL
  { ?autor foaf:name ?name . } }
```

- Vereinigung zweier Muster mit angefügtem optionalem Muster ✓

Beispiel 3.12 (Äquivalentes Muster mit expliziter Klammerung)

```
{ ?buch ex:verlegtBei <http://springer.com> .
  { { ?buch ex:autor ?autor . } UNION
    { ?buch ex:herausgeber ?autor . }
  }
  OPTIONAL { ?autor foaf:name ?name . } }
```

Regeln für OPTIONAL und UNION

- OPTIONAL bezieht sich auf genau ein gruppiertes Muster rechts davon.
- OPTIONAL und UNION sind gleichwertig und linksassoziativ.
  - beziehen sich auf jeweils alle links davon stehenden Ausdrücke
- Wenn anderer Gültigkeitsbereich gewünscht: Gruppieren mit { ... }

Beispiel 3.13

```
{ {s1 p1 o1} OPTIONAL {s2 p2 o2} UNION {s3 p3 o3} OPTIONAL {s4 p4 o4}
  OPTIONAL {s5 p5 o5}
}
```

bedeutet:

Beispiel 3.14 (Äquivalentes Muster mit expliziter Klammerung)

```
{ { { { {s1 p1 o1} OPTIONAL {s2 p2 o2}
      } UNION {s3 p3 o3}
    } OPTIONAL {s4 p4 o4}
  } OPTIONAL {s5 p5 o5}
}
```

Inhalt

Wozu Filter?

Viele Anfragen sind auch mit komplexen Graph-Mustern nicht möglich:

- „Welche Menschen sind zwischen 18 und 23 Jahre alt?“
- „Der Nachname welcher Menschen enthält einen Bindestrich?“
- „Welche Literale in deutscher Sprache sind in der Ontologie enthalten?“

⇝ Filter als allgemeiner Mechanismus für solche Ausdrucksmittel

Beispiel 3.15 (Filter)

```
PREFIX ex: <http://example.org/>
SELECT ?buch WHERE
{ ?buch ex:verlegtBei <http://springer.com> .
  ?buch ex:preis ?preis
  FILTER (?preis < 35)
}
```

- Schlüsselwort `FILTER`, gefolgt von Filterausdruck in Klammern
- Filterbedingungen liefern Wahrheitswerte (und evtl. auch Fehler)

Filterfunktionen: Vergleiche und Arithmetik

Vergleichsoperatoren: `<`, `=`, `>`, `<=`, `>=`, `!=`

- Vergleich von Datenliteralen gemäß der jeweils natürlichen Reihenfolge
- Unterstützung für
  - numerische Datentypen,
  - `xsd:dateTime`,
  - `xsd:string` (alphabetische Ordnung),
  - `xsd:boolean` ($1 > 0$)
- Für andere Typen und sonstige RDF-Elemente nur `=` und `!=` verfügbar
- Kein Vergleich von Literalen inkompatibler Typen (z.B. `xsd:string` und `xsd:integer`)

Arithmetische Operatoren: `+`, `-`, `*`, `/`

- Unterstützung für numerische Datentypen
- Verwendung zur Kombination von Werten in Filterbedingungen

Beispiel 3.16

```
FILTER( ?gewicht/(?groesse * ?groesse) >= 25 )
```

RDF-spezifische Filterfunktionen

`BOUND(A)`	`true` falls `A` eine gebundene Variable ist
`isURI(A)`	`true` falls `A` eine URI ist
`isBLANK(A)`	`true` falls `A` ein leerer Knoten ist
`isLITERAL(A)`	`true` falls `A` ein Literal ist
`sameTERM(A,B)`	`true` falls `A` und `B` dieselben Terme sind
`langMATCHES(A,B)`	`true` falls die Sprachangabe `A` auf das Muster `B` passt
`REGEX(A,B)`	`true` falls die Zeichenkette `A` auf den regulären Ausdruck `B` passt
`STR(A)`	lexikalische Darstellung (`xsd:string`) von URIs oder Literalen
`LANG(A)`	Sprachcode eines Literals (`rdf:langString`) (leerer String falls kein Sprachcode)
`DATATYPE(A)`	Datentyp-URI eines Literals (`xsd:string` bei untypisierten Literalen)

Beispiel 3.17

```
PREFIX ex: <http://example.org/>
SELECT ?buch ?text WHERE
       { ?buch ex:kritik ?text .
         FILTER ( langMATCHES( LANG(?text), "de") ) }
```

Filterfunktionen: Boolesche Operatoren

Verknüpfung von Bedingungen mit Booleschen Operatoren `&&`, `||`, `!`

Teilweise auch durch Graph-Muster ausdrückbar:

- Konjunktion entspricht Angaben mehrerer Filter
- Disjunktion entspricht Anwendung von Filtern in alternativen Mustern

Inhalt

Wozu Modifikatoren?

Bisher nur grundsätzliche Formatierungseinstellungen für Ergebnisse:

- Wie kann man definierte Teile der Ergebnismenge anfragen?
- Wie werden Ergebnisse geordnet?
- Können wiederholte Ergebniszeilen entfernt werden?

$\rightsquigarrow$ Modifikatoren der Lösungssequenz (solution sequence modifiers)

Ergebnisse sortieren

Sortierung von Ergebnissen mit Schlüsselwort ORDER BY

Beispiel 3.18

```
SELECT ?buch ?preis
  WHERE { ?buch ex:preis ?preis . }
  ORDER BY ?preis
```

- Sortierung wie bei Filter-Vergleichsoperatoren
- Sortierung von URIs alphabetisch als Zeichenketten
- Reihenfolge zwischen unterschiedlichen Arten von Elementen:
  Ungebundene Variable $<$ leere Knoten $<$ URIs $<$ RDF-Literale

Weitere mögliche Angaben:

- `ORDER BY DESC(?preis)`: absteigend
- `ORDER BY ASC(?preis)`: aufsteigend, Voreinstellung
- `ORDER BY DESC(?preis), ?titel`: hierarchische Ordnungskriterien

Einschränkung der Ergebnismenge

LIMIT maximale Anzahl von Ergebnissen (Tabellenzeilen)
⇝ Ende abschneiden

OFFSET Position des ersten gelieferten Ergebnisses
⇝ Anfang abschneiden

DISTINCT Entfernung von doppelten Tabellenzeilen

Beispiel 3.19

```
SELECT DISTINCT ?buch ?preis
WHERE { ?buch ex:preis ?preis . }
ORDER BY ?preis  LIMIT 5  OFFSET 25
```

⇝ Ausgabe: Ergebnisse 26 – 30, Duplikate entfernt

`LIMIT` und `OFFSET` meist nur mit `ORDER BY` sinnvoll.

Kombination von Modifikatoren

Reihenfolge bei Abarbeitung von Modifikatoren:

1. Sortierung gemäß `ORDER BY`
2. Entfernung der nicht ausgewählten Variablen
3. Entfernung doppelter Ergebnisse (`DISTINCT`)
4. Entfernung der ersten `OFFSET` Ergebnisse
5. Entfernung aller Ergebnisse bis auf `LIMIT`

Anmerkungen

- Sortierung auch nach nicht mit `SELECT` ausgewählten Variablen möglich
- `DISTINCT` bezieht sich nur auf ausgewählte Variablen

Vorgestellte SPARQL-Merkmale im Überblick

Grundstruktur
PREFIX
SELECT
WHERE

Graph-Muster
Basic Graph Patterns
{...}
OPTIONAL
UNION

Filter
BOUND
isURI
isBLANK
isLITERAL
sameTERM
langMATCHES
REGEX

Modifikatoren
ORDER BY
LIMIT
OFFSET
DISTINCT

Inhalt

Berechnete Variablen: BIND

- Variablen können aus anderen Variablen berechnet werden
- Arithmetische Operatoren wie bei Filtern
- Syntax: BIND (<ausdruck> AS ?var)

Beispiel 3.20

Graph:

```
ex:Buch1 ex:titel "SPARQL Tutorial" ;
         ex:preis  50 ;    # in Euro
         ex:rabatt 10 .    # in Prozent
```

Anfrage:

```
SELECT ?titel ?endpreis
WHERE  { ?buch ex:titel ?titel;
               ex:preis ?preis ;
               ex:rabatt ?rabatt .
          BIND (?preis * (1 - ?rabatt / 100) AS ?endpreis) }
```

Ergebnis:

titel	endpreis
"SPARQL Tutorial"	45

Inhalt

Aggregate

Aggregate erlauben

- Gruppierung von Lösungen
- Berechnung von Werten über die Gruppen

Beispiel 3.21

```
SELECT    ?vorlesung (COUNT(?student) AS ?c)
WHERE     { ?student ex:besucht ?vorlesung }
GROUP BY  ?vorlesung
HAVING    (?c > 5)
```

- `GROUP BY` gruppiert Lösungen
  - optional; default: alle Ergebnisse
  - Gruppierungs-Variable muss selektiert sein
  - hier: gleiche Vorlesung
- `COUNT` zählt Anzahl der Lösungen in einer Gruppe
  - hier: Anzahl Studenten mit gleicher Vorlesung
- `HAVING` beschränkt aggregierte Werte
  - optional
  - hier: mehr als 5

Aggregatfunktionen in SPARQL

`COUNT`	Zählen der Lösungen
`MIN`	Finden des Minimalwerts
`MAX`	Finden des Maximalwerts
`SUM`	Summieren der Werte
`AVG`	Bilden des Durchschnitts
`GROUP_CONCAT`	Stringkonkatenation Z.B.: `GROUP_CONCAT(?x ; separator=", ")`
`SAMPLE`	Wählen eines beliebigen Wertes

Übung: Aggregate

Übung 3.22

Gegeben sei der folgende Graph:

```
ex:Paul ex:note 2.0 .
ex:Paul ex:note 3.0 .
ex:Mary ex:note 2.0 .
ex:Peter ex:note 3.5 .
```

Welche Antwort erzeugt die folgende Anfrage?

```
SELECT ?student (AVG(?note) AS ?avg)
WHERE { ?student ex:note ?note }
GROUP BY ?student
HAVING (?avg > 2.0)
```

Inhalt

Property Paths

Regulärer Ausdrücke über Prädikaten:

- Verkettung von Pfaden: `?s ex:p1 / ex:p2 ?o`
- Alternative Pfade: `?s (ex:p1 | ex:p2) ?o`
- Pfade mit unbestimmter Länge: `?s ex:p+ ?o`, `?s ex:p* ?o`

Weitere Konstruktoren:

- Pfade bestimmter Länge: `?s ex:p{2, 4} ?o`
- Negation von Pfaden: `?s !ex:p ?o`
- Inverse Pfade: `?s ^ex:p ?o` (entspricht `?o ex:p ?s`)

Interne Verarbeitung von Property Paths

- wenn möglich: Übersetzung in bestehende SPARQL-Konstrukte (|, ^, /)
- eigene Verfahren für +, *, !

Beispiele: Property Paths

Beispiel 3.23 (Verkettung von Pfaden)

```
SELECT ?xName WHERE {
  ?x rdf:type foaf:Person .
  ?x foaf:name ?xName .
  ?x foaf:knows{2}/foaf:name "Bill Gates" .
}
```

Beispiel 3.24 (Transitiver Abschluss)

```
SELECT ?s WHERE {
  ?s  rdf:type  ?type .
  ?type rdfs:subClassOf* ex:Carnivore .
}
```

Inhalt

Negation mit MINUS

- Entfernt Variablenbindungen aus Ergebnis
- Syntax: <Muster1> MINUS { <Muster2> }
  1. Berechne Antwort für <Muster1>
  2. Entferne Variablenbindungen, die <Muster2> erfüllen

Beispiel 3.25

Graph:

```
ex:Peter rdf:type foaf:Person .
ex:Peter foaf:name "Peter" .
ex:Mary rdf:type foaf:Person .
```

Anfrage:

```
SELECT ?x
WHERE { ?x rdf:type foaf:Person .
        MINUS { ?x foaf:name ?name } }
```

Ergebnis:

x
ex:Mary

Übung 3.26

Laden Sie den folgenden Graphen in den Fuseki-Server:
`http://wwwlehre.dhbw-stuttgart.de/~hladik/SW/Ontologien/planets.ttl`

Formulieren Sie dann SPARQL-Anfragen, die Ihnen die folgenden Ergebnisse liefern:

1. Objekte, die um die Sonne oder um einen Satelliten der Sonne kreisen.
2. Objekte mit zwei oder mehr Satelliten (nehmen Sie an, dass unterschiedliche URIs hier unterschiedliche Objekte bezeichnen).
3. Objekte, die Satelliten eines anderen Himmelskörpers sind, selbst aber keine Satelliten haben.
4. Objekte mit einem Satelliten, für den ein englischsprachiger Name gegeben ist, die außerdem Satellit eines Objektes von über 3000 (km) Durchmesser sind.
5. Objekte mit einem Volumen von über 2×10^{10} (km^3) (Himmelskörper mit Radius können als kugelförmig angenommen werden, d.h. $V = \frac{4}{3}\pi r^3$) und, falls vorhanden, dem Objekt, dessen Satellit sie sind.

Zusammenfassung SPARQL

Muster	Einfach und komplex: Alternativen, Optionale Teile, Gruppen
Filter	Nur Resultate, die bestimmte Bedingungen erfüllen
Modifikatoren	Sortierung, Duplikate
Aggregate	Berechnungen über Ergebnisse
Property Paths	Komplexe Kombinationen von Properties
Negation	Ausschließen von Lösungen

Ausdrucksstärke von RDF und SPARQL

- RDF(S): sehr leichtgewichtig
  - keine Negation
  - keine Disjunktion
  - keine komplexen Property-Konstrukte
- SPARQL: zahlreiche ausdrucksstarke Anfrage-Konstrukte
  - Negation via `MINUS`
  - Disjunktion via `UNION`
  - Property Paths

Inhalt

Inhalt

Beschreibungslogiken: Ursprung

- Familie von WR-Sprachen (Logiken)
- Fragmente der Prädikatenlogik
  - weniger ausdrucksstark als PL
  - leichter verständlich
- entscheidbare Schlussfolgerungsprobleme
- Komplexität von P bis 2-NExpTime
  - auch ausdrucksstarke Logiken effizient „in der Praxis“
- Ursprung: KL-ONE (Brachman et al., 1985)
- Unterschiedliche Namen

  1986 KL-ONE-like languages
  1989 term subsumption languages
  1991 terminologial logics
  1992 description logics (DLs)

Ron Brachman
(*1949)

Inhalt

BL-Bausteine

Konzeptnamen `Mensch`, `Universität`, `Stadt`, usw.
- ⇝ einstellige Prädikate in PL
- ⇝ ähnlich Klassen in RDFS

Rollennamen `verheiratetMit`, `besucht`, `arbeitetMit`, usw.
- ⇝ zweistellige Prädikate in PL
- ⇝ ähnlich Properties in RDF

Individuennamen `John`, `DHBW`, `Stuttgart`, usw.
- ⇝ Konstanten in PL
- ⇝ ähnlich Knoten in RDF

Die Menge aller Konzept-, Rollen- und Individuennamen einer Ontologie heißt Signatur oder Vokabular.

Die Beschreibungslogik $\mathcal{ALC}$

- „Attributive Language with Complement“
- Boole'sche Operatoren $\neg, \sqcap, \sqcup$
- Quantoren $\forall, \exists$

Beispiel 4.1 ($\mathcal{ALC}$-Konzept)

$\texttt{Student} \sqcap \forall \texttt{besucht}.\texttt{BachelorVorlesung}$
Gruppe der Studenten, die nur Bachelor-Vorlesungen besuchen

Definition 4.2 ($\mathcal{ALC}$-Konzept)

Die Menge der $\mathcal{ALC}$-Konzepte ist wie folgt induktiv definiert:

- jeder Konzeptname ist ein Konzept;
- $\top$ und $\bot$ sind Konzepte;
- Für Konzepte C und D sind auch $(\neg C)$, $(C \sqcap D)$, und $(C \sqcup D)$ Konzepte;
- Für einen Rollennamen r und ein Konzept C sind auch $(\exists r.C)$ und $(\forall r.C)$ Konzepte.

Vorrang wie in der Prädikatenlogik: $\forall, \exists \gg \neg \gg \sqcap \gg \sqcup$

Inhalt

Definition 4.3 (Interpretation für $\mathcal{ALC}$-Konzepte)

Sei C ein $\mathcal{ALC}$-Konzept mit Konzeptnamen N und und Rollennamen R.
Eine Interpretation $\mathcal{I}$ ist ein Paar $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ mit

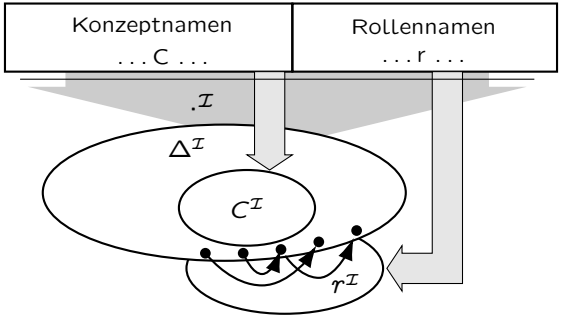
- $\Delta^{\mathcal{I}}$ ist eine nichtleere Menge;
- die Funktion $\cdot^{\mathcal{I}}$ weist
  - jedem Konzeptnamen $C \in N$ eine einstellige Relation $C^{\mathcal{I}}$ über $\Delta^{\mathcal{I}}$ und
  - jedem Rollennamen $r \in R$ eine zweistellige Relation $r^{\mathcal{I}}$ über $\Delta^{\mathcal{I}}$ zu.

Beispiel 4.4 ($\mathcal{ALC}$-Interpretation)

$$\texttt{Student} \sqcap \forall \texttt{besucht}.\texttt{BachelorVorlesung}$$

$$\begin{aligned}
\mathcal{I} &= (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}}) \\
\Delta^{\mathcal{I}} &= \{\texttt{Peter}, \texttt{Paul}, \texttt{Maria}, \texttt{Logik}, \texttt{SemanticWeb}\}, \\
\texttt{Student}^{\mathcal{I}} &= \{\texttt{Peter}, \texttt{Maria}\}, \\
\texttt{Bachelorvorlesung}^{\mathcal{I}} &= \{\texttt{Logik}\}, \\
\texttt{besucht}^{\mathcal{I}} &= \{(\texttt{Peter}, \texttt{Logik}), (\texttt{Peter}, \texttt{SemanticWeb}), (\texttt{Maria}, \texttt{Logik})\}
\end{aligned}$$

Schematische Darstellung einer Interpretation



Definition 4.5 (Interpretation komplexer Konzepte)

Die Interpretation komplexer Konzepte ist induktiv definiert:

Name	Syntax	Semantik
Top	$\top$	$\Delta^{\mathcal{I}}$
Bottom	$\bot$	$\emptyset$
Negation	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
Konjunktion	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
Disjunktion	$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
Allquantor	$\forall r.C$	$\{x \in \Delta^{\mathcal{I}} \mid \text{aus } (x, y) \in r^{\mathcal{I}} \text{ folgt } y \in C^{\mathcal{I}}\}$
Existenzquantor	$\exists r.C$	$\{x \in \Delta^{\mathcal{I}} \mid \text{es gibt ein } y \in \Delta^{\mathcal{I}}, \text{ so dass } (x, y) \in r^{\mathcal{I}} \text{ und } y \in C^{\mathcal{I}} \text{ gilt}\}$

Übung: $\mathcal{ALC}$-Konzepte

Übung 4.6

Verwenden Sie

- die Konzeptnamen `Mensch`, `Weiblich`, `Architekt`, `Ingenieur` und
- den Rollennamen `kind`,

um Konzepte für die folgenden Gruppen zu formulieren:

1. Menschen, die nicht weiblich sind
2. Menschen, die nur weibliche Kinder haben
3. Menschen, die ein Kind haben, das Architekt oder Ingenieur ist
4. Großeltern

Inhalt

Grundlegende BL-Schlussfolgerungsprobleme

Konsistenz Ist ein Konzept C widersprüchlich?

$\rightsquigarrow$ Ist C erfüllbar?

Subsumtion Ist jede Instanz von C auch Instanz von D?

$\rightsquigarrow$ Ist $C \sqcap \neg D$ erfüllbar?

- liefert Konzepthierarchie

Erfüllbarkeitstest kann beide Schlussfolgerungsprobleme entscheiden.

Tableau-Algorithmus für $\mathcal{ALC}$-Konzepte

- arbeitet auf $\mathcal{ALC}$-Konzepten in Negations-Normalform (NNF)
  - Negation nur direkt vor Konzeptnamen

$$\begin{array}{rcl} \neg(C \sqcap D) & \rightsquigarrow & \neg C \sqcup \neg D \\ \neg\forall r.C & \rightsquigarrow & \exists r.\neg C \\ \neg\top & \rightsquigarrow & \bot \\ \neg\neg C & \rightsquigarrow & C \end{array} \qquad \begin{array}{rcl} \neg(C \sqcup D) & \rightsquigarrow & \neg C \sqcap \neg D \\ \neg\exists r.C & \rightsquigarrow & \forall r.\neg C \\ \neg\bot & \rightsquigarrow & \top \end{array}$$

  - Notation: $\mathrm{nnf}(C)$
  - Abkürzung für $\mathrm{nnf}(\neg C)$: $\dot{\neg} C$
- erzeugt Tableau für Konzept C
  - Baum, der Modell für C repräsentiert (wenn er Clash-frei und vollständig ist)
  - Knoten beschriftet mit Mengen von Subkonzepten von C
  - Kanten beschriftet mit Rollennamen
  - Wurzel beschriftet mit C
  - Weitere Knoten von $\exists$-Regel erzeugt
  - Keine Tabellenstruktur für Disjunktionen, sondern depth-first-Expansion mit Backtracking

$\mathcal{ALC}$-Tableau: Formale Definition

Definition 4.7 ($\mathcal{ALC}$-Tableau)

Sei

- C ein $\mathcal{ALC}$-Konzept in NNF,
- $\mathrm{SC}(C)$ die Menge aller Subkonzepte von C und
- $\mathrm{Rol}(C)$ die Menge aller in C vorkommenden Rollennamen.

Ein Tableau für C ist ein Baum $G = (V, E, L, L_E)$ mit

- einer Knotenmenge V,
- einer Kantenmenge $E \subseteq V \times V$ und
- einer Knoten-Beschriftungsfunktion $L : V \to 2^{\mathrm{SC}(C)}$ und
- einer Kanten-Beschriftungsfunktion $L_E : E \to \mathrm{Rol}(C)$.

$\mathcal{ALC}$-Tableau-Algorithmus

- 4 Regeln (wegen NNF)
  - brechen komplexe Konzepte in einfachere auf
  - Anwendungsreihenfolge don't-care-nichtdeterministisch
  - $\sqcup$-Regel don't-know-nichtdeterministisch
  - Wenn keine Regel anwendbar ist, heißt das Tableau vollständig.
- Clash: Knotenlabel enthält C und $\neg C$ für ein Konzept C (oder $\bot$)
  - Wenn kein Knoten einen Clash enthält, heißt das Tableau Clash-frei.

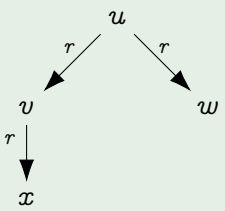
Definition 4.8 (r-Nachfolger)

Wenn $(v, v') \in E$ und $L_E(v, v') = r$ gelten, heißt v' r-Nachfolger von v.

- $\sqcap$ Wenn es ein $v \in V$ gibt mit $C \sqcap D \in L(v)$ und $\{C, D\} \not\subseteq L(v)$,
  dann $L(v) := L(v) \cup \{C, D\}$.
- $\sqcup$ Wenn es ein $v \in V$ gibt mit $C \sqcup D \in L(v)$ und $\{C, D\} \cap L(v) = \emptyset$,
  dann wähle $X \in \{C, D\}$ und setze $L(v) := L(v) \cup \{X\}$.
- $\exists$ Wenn es ein $v \in V$ gibt mit $\exists r.C \in L(v)$
  und kein r-Nachfolger v' von v existiert mit $C \in L(v')$,
  dann $V := V \cup \{v'\}$, $E := E \cup \{(v, v')\}$,
  $L(v') := \{C\}$ und $L_E(v, v') := r$ für einen neuen Knoten v'.
- $\forall$ Wenn es ein $v \in V$ gibt mit $\forall r.C \in L(v)$
  und ein r-Nachfolger v' von v existiert mit $C \notin L(v')$,
  dann $L(v') := L(v') \cup \{C\}$.

Beispiel: $\mathcal{ALC}$-Tableau-Algorithmus

Beispiel 4.9 (Tableau für $\exists r.(A \sqcup \exists r.B) \sqcap \exists r.\neg A \sqcap \forall r.(\neg A \sqcap \forall r.(\neg B \sqcup A)))$



$$
\begin{aligned}
L(u) &= \{\exists r.(A \sqcup \exists r.B) \sqcap \exists r.\neg A \sqcap \forall r.(\neg A \sqcap \forall r.(\neg B \sqcup A)),\\
&\quad\ \exists r.(A \sqcup \exists r.B), \exists r.\neg A, \forall r.(\neg A \sqcap \forall r.(\neg B \sqcup A))\}\\
L(v) &= \{A \sqcup \exists r.B, \neg A \sqcap \forall r.(\neg B \sqcup A), \neg A, \forall r.(\neg B \sqcup A), \cancel{A}, \exists r.B\}\\
L(w) &= \{\neg A, \neg A \sqcap \forall r.(\neg B \sqcup A), \forall r.(\neg B \sqcup A)\}\\
L(x) &= \{B, \neg B \sqcup A, \cancel{\neg B}, A\}
\end{aligned}
$$

$\mathcal{ALC}$-Tableau und Modell

- Tableaus und Modelle hängen zusammen
  - Tableau enthält die „entscheidenden“ Teile eines Modells
  - Tableau (clash-frei und vollständig) $\rightsquigarrow$ Modell
  - kein clash-freies und vollständiges Tableau $\rightsquigarrow$ kein Modell
- Aber: Tableaus und Modelle sind nicht identisch

Tableau	Modell
immer baumförmig	beliebige Struktur möglich
immer endlich	kann unendlich sein
muss nicht festlegen, ob x zu C gehört	$C^{\mathcal{I}}$ ist fix
kann Clashes enthalten	per Definition clash-frei
kann unvollständig sein	vom Element x erfüllte Konzepte stehen fest

Konstruktion eines Modells aus einem Tableau ist nicht trivial!

Konstruktion Tableau $\rightsquigarrow$ Modell

Das vollständige und Clash-freie Tableau für C liefert ein Modell $\mathcal{I}$ für C:

Domäne	alle Knoten	$\Delta^{\mathcal{I}}$	$=$	$\{u, v, w, x\}$
Konzeptname	alle Knoten, die mit ihm beschriftet sind	$A^{\mathcal{I}}$	$=$	$\{x\}$
		$B^{\mathcal{I}}$	$=$	$\{x\}$
Rolle	alle Kanten, die mit ihr beschriftet sind	$r^{\mathcal{I}}$	$=$	$\{(u, v), (u, w), (v, x)\}$

Aus der Semantik von $\mathcal{ALC}$ folgt per Induktion:

- Jeder Knoten erfüllt jedes Konzept in seinem Label.
- Der Wurzelknoten erfüllt C.
- C ist erfüllbar.

Übung: $\mathcal{ALC}$-Tableaualgorithmus

Übung 4.10

Testen Sie die Konsistenz des Konzepts K mit Hilfe des $\mathcal{ALC}$-Tableau-Algorithmus:

$$K = \exists r.C \sqcap \exists r.\neg C \sqcap \forall r.(\neg C \sqcup D) \sqcap \forall r.(\neg D \sqcup \exists s.C)$$

Satz 4.11 (Korrektheit des $\mathcal{ALC}$-Tableau-Algorithmus)

Wenn der Algorithmus für ein Konzept C ein Clash-freies und vollständiges Tableau erzeugt, ist C konsistent.

Beweis.

Aus dem Tableau kann ein Modell abgeleitet werden. □

Satz 4.12 (Vollständigkeit des $\mathcal{ALC}$-Tableau-Algorithmus)

Wenn das Konzept C konsistent ist, kann die $\sqcup$-Regel so angewendet werden, dass der Algorithmus ein Clash-freies und vollständiges Tableau erzeugt.

Beweis.

Das Modell kann genutzt werden, um eine erfolgreiche Folge von Regelanwendungen zu finden. □

Terminierung

Satz 4.13 (Terminierung des $\mathcal{ALC}$-Tableau-Algorithmus)

Der Algorithmus terminiert für jedes Eingabekonzept C.

Beweis.

- Quantorentiefe sinkt mit jeder Ebene des Baums.
- Die $\exists$-Regel ist in jedem Knoten für jedes Konzept höchstens einmal anwendbar.
- Ein endlich verzweigter Baum, in dem nur endlich lange Pfade existieren, ist endlich. (Lemma von König)
- Jeder Knoten ist nur mit Subkonzepten von C beschriftet.
- C hat endlich viele Subkonzepte.
- Für jede Disjunktion wird höchstens einmal Backtracking durchgeführt. □

Satz 4.14 (Modelleigenschaft von $\mathcal{ALC}$)

Für $\mathcal{ALC}$-Konzepte gilt die Endliches-Baum-Modell-Eigenschaft:
Jedes konsistente Konzept C hat ein endliches Baum-Modell.

Zusammenfassung: $\mathcal{ALC}$

- ausdrucksstärker als Aussagenlogik
- ausdrucksschwächer als Prädikatenlogik
  - $\forall$ und $\exists$ nur eingeschränkt
  - nur ein- und zweistellige Prädikate
  - keine Funktionen
  - keine Variablen
- Tableau-Algorithmus
  - erzeugt endliches Baum-Modell
  - korrekt, vollständig, terminierend
- Schlussfolgerungsprobleme entscheidbar
  - Konsistenz
  - Subsumtion

Inhalt

DL-Wissensbanken

Bisher: Nur einzelne Konzepte

- Schlussfolgerungsverfahren
  - Konzept-Konsistenz
  - Konzept-Subsumtion
- Nicht möglich:
  - Begriffsdefinitionen („Ein `Bachelorstudent` ist . . . “)
  - Aussagen über Konzepte („Kein `Kind` isst einen `Kürbis`.“)
  - Aussagen über Individuen („`Tim` ist ein `Kind`.“)
  - Aussagen über Rollen („`teil` ist transitiv.“) („`kind` impliziert `nachkomme`“)

Jetzt: DL-Wissensbank $\mathcal{K}$

TBox $\mathcal{T}$	Informationen über Konzepte und ihre taxonomischen Abhängigkeiten
ABox $\mathcal{A}$	Informationen über Individuen, ihre Konzepte und Rollenverbindungen
RBox $\mathcal{R}$	Informationen über Rollen und ihre Abhängigkeiten

Inhalt

Definition 4.15 (GCI, TBox)

Seien C und D Konzepte.

- Ein allgemeines Konzept-Inklusions-Axiom (general concept inclusion axiom, GCI) hat die Form
$$C \sqsubseteq D$$
- Ein Konzept-Äquivalenz-Axiom $C \equiv D$ ist eine Abkürzung für $C \sqsubseteq D$ und $D \sqsubseteq C$.
- Eine TBox (terminologische Box) ist eine endliche Menge von Konzept-Axiomen.

TBox $\mathcal{T}$

Eine Beispiel-TBox

$\texttt{Tier}$	$\sqsubseteq$	$\texttt{Lebendig} \sqcup \texttt{Tot}$	„Jedes Tier ist lebendig oder tot.“
$\texttt{Gesund}$	$\sqsubseteq$	$\texttt{Lebendig}$	„Nur wer lebendig ist, kann gesund sein.“
$\texttt{Lebendig}$	$\equiv$	$\neg\, \texttt{Tot}$	„Lebendig sind genau diejenigen, die nicht tot sind.“
$\texttt{Katze}$	$\sqsubseteq$	$\texttt{Tier}$	„Jede Katze ist ein Tier“
$\texttt{Weiblich} \sqcap \exists\, \texttt{kind}.\top$	$\sqsubseteq$	$\texttt{Mutter}$	„Wer weiblich ist und ein Kind hat, ist eine Mutter.“

Modelle für TBoxen

Definition 4.16 (Modell für TBox)

- Eine Interpretation $\mathcal{I}$ ist ein Modell für ein GCI $C \sqsubseteq D$ ($\mathcal{I} \models C \sqsubseteq D$), wenn $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ gilt.
- $\mathcal{I}$ ist ein Modell für eine TBox $\mathcal{T}$ ($\mathcal{I} \models \mathcal{T}$), wenn sie Modell für jedes Axiom in $\mathcal{T}$ ist.
- $\mathcal{I}$ ist ein Modell für ein Konzept C bzgl. einer TBox $\mathcal{T}$, wenn $\mathcal{I} \models \mathcal{T}$ gilt und $C^{\mathcal{I}}$ nichtleer ist.
- Konsistenz von (Konzepten bzgl.) TBoxen ist wie üblich über die Existenz eines Modells definiert.

Übung: Formalisierung mit TBox

Übung 4.17

Formalisieren Sie die folgenden Aussagen als GCIs in $\mathcal{ALC}$:

1. Jeder Junge und jedes Mädchen ist ein Kind.
2. Jedes Kind bekommt ein Auto, eine Puppe oder eine Rute.
3. Kein Junge bekommt eine Puppe.
4. Kein braves Kind bekommt eine Rute.
5. Kein Kind bekommt ein Auto.

Verwenden Sie hierzu

- den Rollennamen `bekommt` und
- die Konzeptnamen `Junge`, `Maedchen`, `Kind`, `Auto`, `Puppe`, `Rute` und `Brav`.

Schlussfolgerungsprobleme für TBoxen

TBox-Konsistenz	Gibt es ein Modell für $\mathcal{T}$?
Konzept-Konsistenz bzgl. TBox	Gibt es ein Modell für C bzgl. $\mathcal{T}$?
Konzept-Subsumtion bzgl. TBox	Gilt $C \sqsubseteq D$ in jedem Modell von $\mathcal{T}$?

Zusammenhänge:

- TBox-Konsistenz $\leq$ Konzept-Konsistenz bzgl. TBox:
  $\mathcal{T}$ ist konsistent gdw. $\top$ bzgl. $\mathcal{T}$ konsistent ist.
- Konzept-Subsumtion bzgl. TBox $\leq$ Konzept-Konsistenz bzgl. TBox:
  $C \sqsubseteq D$ bzgl. $\mathcal{T}$ gdw. $C \sqcap \neg D$ bzgl. $\mathcal{T}$ inkonsistent ist.
- Nur Algorithmus für Konzept-Konsistenz bzgl. TBox wird benötigt

Schlussfolgerungsverfahren: Erweiterung des Tableau-Algorithmus für Konzept-Konsistenz

- zusätzliche Regel
- veränderte Anwendbarkeitsbedingungen

Tableau-Algorithmus für Konzept-Konsistenz bzgl. TBoxen

Neue Regel für GCIs:

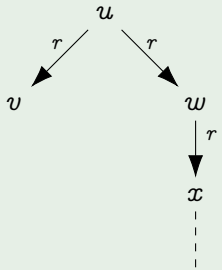
GCI Wenn $\mathcal{T}$ den GCI $C \sqsubseteq D$ enthält
und es ein $v \in V$ gibt mit $\text{nnf}(\neg C \sqcup D) \notin L(v)$,
dann $L(v) := L(v) \cup \{\text{nnf}(\neg C \sqcup D)\}$.

- Jeder Knoten wird mit $\text{nnf}(\neg C \sqcup D)$ beschriftet.
- Erinnerung: Äquivalenz-Axiome werden durch GCIs ersetzt.
- Frage: Warum kann man es sich nicht einfacher machen:

  GCI Wenn $C \in L(v)$ gilt, dann $L(v) := L(v) \cup \{D\}$?

Beispiel: Konzept-Konsistenz bzgl. TBox

Beispiel 4.18 (Tableau für $A \sqcap \exists r.B$ bzgl. $\{A \sqsubseteq \exists r.A\}$)



$K = A \sqcap \exists r.B \qquad \mathcal{T} = \{A \sqsubseteq \exists r.A\}$

$L(u) = \{A \sqcap \exists r.B, A, \exists r.B, \neg A \sqcup \exists r.A, \exists r.A\}$

$L(v) = \{B, \neg A \sqcup \exists r.A, \neg A\}$

$L(w) = \{A, \neg A \sqcup \exists r.A, \exists r.A\}$

$L(x) = \{A, \neg A \sqcup \exists r.A, \exists r.A\}$

$\vdots$

- Algorithmus terminiert nicht mehr!
  - Erzeugtes Baummodell ist unendlich
- Aber: Alle Nachfolger von w haben dasselbe Label
  - Wenn w clash-frei ist, sind es die Nachfolger auch
  - Um Konsistenz zu testen, müssen nicht alle Nachfolger erzeugt werden

Definition 4.19 (Blockierung)

Ein Knoten y in einem Tableau ist blockiert, wenn

- ein Knoten x existiert mit $L(y) \subseteq L(x)$,
- x nicht blockiert ist, und
- x vor y erzeugt wurde.

Ein Knoten y ist blockiert, wenn

- Ein älterer Knoten x existiert,
- dessen Label alle Konzepte für y schon enthält.

Modifizierte ∃-Regel für TBoxen

∃ Wenn es einen nicht blockierten Knoten $v \in V$ gibt mit $\exists r.C \in L(v)$
und kein r-Nachfolger v' von v existiert mit $C \in L(v)$,
dann $V := V \cup \{v'\}$, $E := E \cup \{(v, v')\}$,
$L(v') := \{C\}$ und $L_E(v, v') := r$ für einen neuen Knoten v'.

Satz 4.20

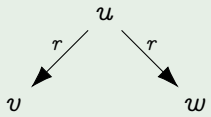
Der Tableau-Algorithmus für $\mathcal{ALC}(\mathcal{Q})$-Konzept-Konsistenz bzgl. TBoxen terminiert, wenn ∃-Regel (und $\geqslant$-Regel) nur auf nicht blockierte Knoten angewendet werden.

Beweis.

- Die Menge der möglichen Knotenlabel ist endlich (Subkonzepte der Eingabe).
- Ein Pfad ohne blockierte Knoten kann nur endliche Länge haben. □

Beispiel 4.18 repariert

Beispiel 4.21



$$K = A \sqcap \exists r.B \qquad \mathcal{T} = \{A \sqsubseteq \exists r.A\}$$

$$L(u) = \{A \sqcap \exists r.B, A, \exists r.B, \neg A \sqcup \exists r.A, \exists r.A\}$$

$$L(v) = \{B, \neg A \sqcup \exists r.A, \neg A\}$$

$$L(w) = \{A, \neg A \sqcup \exists r.A, \exists r.A\}$$

- w ist durch u blockiert
- x und seine Nachfolger werden nicht erzeugt

Übung: Tableau-Algorithmus für $\mathcal{ALC}$ mit TBoxen

Übung 4.22

Testen Sie die Konsistenz des Konzepts

$$\texttt{Mensch}$$

bzgl. der TBox

$$\mathcal{T} = \{\texttt{Mensch} \sqsubseteq \exists \texttt{vater}.\texttt{Mensch} \sqcap \exists \texttt{mutter}.\texttt{Mensch}\}$$

Zusatzübung: Konzept-Konsistenz bzgl. TBox

Übung 4.23

Gegeben sei die TBox $\mathcal{T}$ aus Übung 4.17. Zeigen Sie dass kein Junge brav ist, indem Sie mit Hilfe des Tableau-Algorithmus nachweisen, dass das Konzept

$$\texttt{Junge} \sqcap \texttt{Brav}$$

inkonsistent bzgl. $\mathcal{T}$ ist.

Korrektheit der Blockierung

Schwierigkeit:

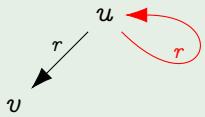
- Für blockierte Knoten werden keine Nachfolger erzeugt.
- Knoten ist mit $\exists r.C$ beschriftet, aber es gibt keinen entsprechenden Nachfolger
- Tableau liefert nicht direkt ein Modell

Lösung: Blockierender Knoten x ersetzt blockierten Knoten y

- y ist nicht Teil des Modells
- Kante zu y wird auf x „umgebogen“
- x erfüllt alle Anforderungen an y:
  - x enthält alle Konzepte im Label von y
  - x ist selbst nicht blockiert

Modell für Beispiel 4.18

Beispiel 4.24



$$K = A \sqcap \exists r.B \qquad \mathcal{T} = \{A \sqsubseteq \exists r.A\}$$

$$L(u) = \{A \sqcap \exists r.B, A, \exists r.B, \neg A \sqcup \exists r.A, \exists r.A\}$$

$$L(v) = \{B, \neg A \sqcup \exists r.A, \neg A\}$$

Problem:

- u blockiert w
- $\exists$-Regel wird nicht auf w angewendet
- $\exists r.A$ in $L(w)$ wird nicht erfüllt

Lösung:

- entferne w
- leite zu w führende Kante auf u um
- alle Konzepte im Label werden erfüllt

Zusammenfassung: TBoxen

- Terminologisches Wissen
- GCI $C \sqsubseteq D$: „Jede Entität, das C erfüllt, erfüllt auch D“.
- Terminierender Tableau-Algorithmus erfordert Blockierung
  - Tableau liefert nicht direkt ein Modell
  - Kanten müssen umgebogen werden
- Schlussfolgerungsprobleme
  - TBox-Konsistenz
  - Konzept-Konsistenz bzgl. TBox
  - Konzept-Subsumtion bzgl. TBox

Inhalt

Definition 4.25 (Fakt, ABox)

Sei C ein Konzept, r ein Rollenname und a, b Individuennamen.

- Ein Konzept-Fakt hat die Form $a : C$.
- Ein Rollen-Fakt hat die Form $(a, b) : r$.
- Eine ABox (Assertions-Box) ist eine endliche Menge von (Konzept- oder Rollen-) Fakten.

ABox $\mathcal{A}$

Eine Beispiel-Wissensbank

TBox $\mathcal{T}$

Gesund	$\sqsubseteq$	$\neg$ Tot	„Gesunde sind nicht tot.“
Katze	$\sqsubseteq$	Tot $\sqcup$ Lebendig	„Jede Katze ist tot oder lebendig.“
GlücklicherKatzenbesitzer	$\sqsubseteq$	$\exists$hat.Katze $\sqcap$ $\forall$hat.Gesund	„Ein glücklicher Katzenbesitzer hat eine Katze und alles, was er hat, ist gesund.“

ABox $\mathcal{A}$

(Schrödinger, Kitty)	:	hat	„Schrödinger hat Kitty.“
Schrödinger	:	GlücklicherKatzenbesitzer	„Schrödinger ist ein glücklicher Katzenbesitzer.“

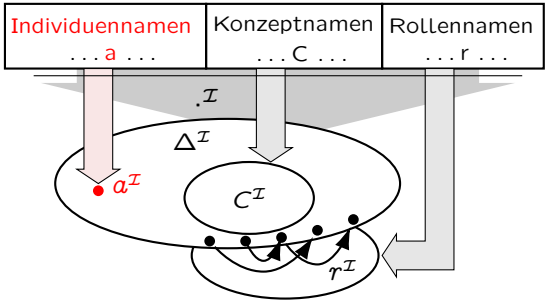
Definition 4.26 (Interpretation einer $\mathcal{ALC}$-Wissensbank)

Sei $\mathcal{K}$ eine $\mathcal{ALC}$-Wissensbank mit

- Konzeptnamen N,
- Rollennamen R und
- Individuennamen I.

Eine Interpretation $\mathcal{I}$ ist ein Paar $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ mit

- $\Delta^{\mathcal{I}}$ ist eine nichtleere Menge;
- die Funktion $\cdot^{\mathcal{I}}$ weist
  - jedem Konzeptnamen $C \in N$ eine einstellige Relation $C^{\mathcal{I}}$ über $\Delta^{\mathcal{I}}$,
  - jedem Rollennamen $r \in R$ eine zweistellige Relation $r^{\mathcal{I}}$ über $\Delta^{\mathcal{I}}$ und
  - jedem Individuennamen $a \in I$ ein Element $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$ zu.



Definition 4.27 (Modell für ABox-Fakten)

Eine Interpretation $\mathcal{I}$ ist ein Modell für einen ABox-Fakt A unter den folgenden Bedingungen:

Syntax	Semantik
$a : C$	$a^{\mathcal{I}} \in C^{\mathcal{I}}$
$(a, b) : r$	$(a^{\mathcal{I}}, b^{\mathcal{I}}) \in r^{\mathcal{I}}$

$\mathcal{I}$ ist ein Modell für eine ABox $\mathcal{A}$, wenn sie Modell für jeden Fakt in $\mathcal{A}$ ist.
$\mathcal{I}$ ist ein Modell für eine ABox $\mathcal{A}$ bzgl. einer TBox $\mathcal{T}$, wenn sie Modell für $\mathcal{A}$ und $\mathcal{T}$ ist.

Schlussfolgerungsprobleme für ABoxen

Definition 4.28 (ABox-Schlussfolgerungsprobleme)

- Eine ABox $\mathcal{A}$ ist konsistent (bzgl. einer TBox $\mathcal{T}$), wenn es eine Interpretation $\mathcal{I}$ gibt mit $\mathcal{I} \models \mathcal{A}$ (und $\mathcal{I} \models \mathcal{T}$).
- Ein Individuum a ist Instanz eines Konzepts C bzgl. einer Wissensbank $(\mathcal{T}, \mathcal{A})$, wenn jedes Modell von $(\mathcal{T}, \mathcal{A})$ auch Modell von $a : C$ ist.
- Die Extension eines Konzepts C bzgl. einer Wissensbank $(\mathcal{T}, \mathcal{A})$ ist die Menge aller Individuen, die Instanzen von C bzgl. $(\mathcal{T}, \mathcal{A})$ sind.

Zusammenhänge:

- Extension $\leq$ Instanz
  - endlich viele ABox-Individuen müssen getestet werden
- Instanz $\leq$ ABox-Konsistenz
  - a ist Instanz von C bzgl. $(\mathcal{T}, \mathcal{A})$ gdw. $\mathcal{A} \cup \{a : \neg C\}$ bzgl. $\mathcal{T}$ inkonsistent ist.
- $\rightsquigarrow$ nur Algorithmus für Konsistenz wird benötigt

Zusammenhang mit TBox-Schlussfolgerungsverfahren:

- Konzept-Konsistenz (bzgl. TBox) $\leq$ ABox-Konsistenz (bzgl. TBox).
  - C ist erfüllbar (bzgl. $\mathcal{T}$) gdw. $\{a : C\}$ konsistent (bzgl. $\mathcal{T}$) ist.

Schwierigkeit des Tableau-Algorithmus für ABox-Konsistenz

- ABox-Individuen können beliebig miteinander verbunden sein
  - Beispiel: $\{(a, b) : r, (b, c) : r, (a, c) : r\}$
- Keine Baummodell-Eigenschaft!
- Aber: Jedes ABox-Individuum ist Wurzel eines Baums $\rightsquigarrow$ „Wald-Modell-Eigenschaft“
- Individuen können r-Nachfolger anderer Individuen sein
  - relevant für Quantoren und Zahlenrestriktionen
- Wechselwirkung mit TBoxen
  - ABox-Individuen können nicht blockiert werden
  - denn: Es gibt keine älteren Knoten
- Wechselwirkung mit Zahlenrestriktionen
  - Knoten für Individuen können verschmolzen werden
  - denn: Verschiedene Namen für ein Individuum möglich (keine „Unique Name Assumption“)
  - wie bei URIs: `dbo:Stuttgart` und `ex:Stuttgart`

Tableau-Algorithmus für ABoxen

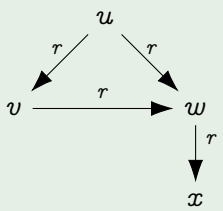
Vorverarbeitung: Transformiere Konzepte in Konzept-Fakten in NNF

Algorithmus:

- erzeuge für jede Individuennamen a einenen Knoten; beschrifte diesen mit a
- Für $a : C \in \mathcal{A}$: beschrifte Knoten für a mit C
- Für $(a, b) : r \in \mathcal{A}$: erzeuge r-Kante vom Knoten für a zum Knoten für b
- Wende Tableau-Regeln an

Beispiel 4.29

$$\mathcal{T} = \{\exists r.\top \sqsubseteq \forall r.\exists r.\top\} \qquad \mathcal{A} = \{(a, b) : r, (b, c) : r, (a, c) : r\}$$



$$\begin{aligned} L(u) &= \{a, \forall r.\bot \sqcup \forall r.\exists r.\top, \forall r.\exists r.\top\} \\ L(v) &= \{b, \forall r.\bot \sqcup \forall r.\exists r.\top, \exists r.\top, \forall r.\exists r.\top\} \\ L(w) &= \{c, \forall r.\bot \sqcup \forall r.\exists r.\top, \exists r.\top, \forall r.\exists r.\top\} \\ L(x) &= \{\forall r.\bot \sqcup \forall r.\exists r.\top, \exists r.\top, \forall r.\exists r.\top\} \end{aligned}$$

x ist durch v blockiert $\rightsquigarrow$ Algorithmus terminiert

Übung: Tableau für ABox-Konsistenz

Übung 4.30

Testen Sie mit Hilfe des gezeigten Tableau-Algorithmus, ob die ABox

$$\mathcal{A} = \{a : \exists r.C, \quad b : D, \quad (a, b) : r\}$$

bzgl. der TBox

$$\mathcal{T} = \{\exists r.C \sqsubseteq \forall r.C, \quad D \sqsubseteq \exists r.C\}$$

konsistent ist.

Hausaufgabe: Formalisierung und Schlussfolgerung

Übung 4.31

1. Formalisieren Sie die folgenden Aussagen als $\mathcal{ALC}$-Wissensbank (TBox und ABox):
   1. Anna hat nur erfolgreiche Freunde.
   2. Bert ist ein Student.
   3. Ein Student, der nicht lernt, besteht nicht.
   4. Ein Student ist nur (!) dann erfolgreich, wenn er besteht.
   5. Bert lernt nicht.

   Verwenden Sie hierzu
   - die Konzeptnamen `Erfolgreich`, `Student`, `Lernt` und `Besteht`;
   - den Rollennamen `freund`;
   - die Individuennamen `Anna` und `Bert`.
2. Verwenden Sie den Tableaualgorithmus, um die Wissensbank auf Konsistenz zu testen.
3. Zeigen Sie, dass Anna nicht Bert als Freund hat, indem Sie nachweisen, dass der zusätzliche Rollen-Fakt $(\texttt{Anna}, \texttt{Bert}) : \texttt{freund}$ zu Inkonsistenz führt.

Zusammenfassung: ABoxen

- Faktenwissen: Wissen über Individuen

  Konzept-Fakt $a : C$ a erfüllt C

  Rollen-Fakt $(a, b) : r$ a und b stehen in Beziehung r
- Tableau-Algorithmus erzeugt Wald-Modell
  - endliche Menge von Bäumen
  - Baumwurzeln: Individuen
  - beliebige Beziehungen zwischen Individuen möglich
- Schlussfolgerungsprobleme
  - ABox-Konsistenz
  - Instanz
  - Konzept-Extension

Inhalt

Definition 4.32 (Rollenaxiom, RBox)

Seien r und s Rollen.

- Ein Rollen-Inklusions-Axiom (RIA) hat die Form $r \sqsubseteq s$.
- Ein Transitivitäts-Axiom hat die Form $\mathrm{Trans}(r)$.
- Eine RBox ist eine Menge von Rollenaxiomen.

RBox $\mathcal{R}$

Eine Beispiel-Wissensbank

TBox $\mathcal{T}$		
	$\texttt{Mensch} \sqsubseteq \exists\, \texttt{mutter.Mensch}$	„Jeder Mensch hat eine Mutter.“
ABox $\mathcal{A}$		
$\texttt{(Bert,Anna)} : \texttt{mutter}$		„Bert hat Anna als Mutter.“
RBox $\mathcal{R}$		
	$\texttt{mutter} \sqsubseteq \texttt{vorfahr}$	„Die Mutter ist ein Vorfahr.“
$\texttt{Trans(vorfahr)}$		„Ein Vorfahr eines Vorfahren ist auch ein Vorfahr.“

Definition 4.33 (Modell für Rollen-Axiome)

- Eine Interpretation $\mathcal{I}$ ist ein Modell für ein Rollen-Axiom unter diesen Bedingungen:

Syntax	Semantik
$r \sqsubseteq s$	$r^{\mathcal{I}} \subseteq s^{\mathcal{I}}$
Trans(r)	$(x, y) \in r^{\mathcal{I}} \wedge (y, z) \in r^{\mathcal{I}} \models (x, z) \in r^{\mathcal{I}}$

- $\mathcal{I}$ ist ein Modell für eine RBox $\mathcal{R}$, wenn sie Modell für jedes Axiom in $\mathcal{R}$ ist.
- Konzept- und ABox-Konsistenz bzgl. einer RBox ist entsprechend definiert.

Anpassungen des Tableau-Algorithmus

- für Rollen-Inklusion: Erweiterung der Definition von r-Nachfolger

Definition 4.34 (r-Nachfolger, erweitert für RBoxen)

Für eine ABox $\mathcal{A}$ bezeichne $\sqsubseteq^*$ den reflexiv-transitiven Abschluss der Rollen-Inklusions-Axiome in $\mathcal{A}$.
Ein Knoten v' heißt r-Nachfolger von einem Knoten v, wenn v' ein s-Nachfolger von v ist und $s \sqsubseteq^* r$ gilt.

Ergebnis: $\forall$-Regel für $\forall r.C$ wirkt sich auch auf s-Nachfolger aus

- für Transitivität: Zusätzliche Regel

$\forall^+$ Wenn es ein $v \in V$ gibt mit $\forall r.C \in L(v)$ und $\mathrm{Trans}(r) \in \mathcal{R}$
und ein r-Nachfolger v' von v existiert mit $\forall r.C \notin L(v')$,
dann $L(v') := L(v') \cup \{\forall r.C\}$

Ergebnis:

- r-Nachfolger werden mit C und $\forall r.C$ beschriftet
- alle indirekten r-Nachfolger werden ebenfalls mit C beschriftet

Beispiel: Tableau-Algorithmus für RBoxen

Beispiel 4.35

$$\mathcal{T} = \{\texttt{Mensch} \sqsubseteq \forall \texttt{vorfahr}.\texttt{Mensch}\} \quad \mathcal{A} = \{\texttt{j}:\texttt{Mensch} \sqcap \exists \texttt{mutter}.\top\}$$

$$\mathcal{R} = \{\texttt{mutter} \sqsubseteq \texttt{vorfahr}, \mathsf{Trans}(\texttt{vorfahr})\}$$

u — mutter → x

$$\begin{aligned} L(u) &= \{j, \texttt{Mensch} \sqcap \exists \texttt{mutter}.\top, \neg\texttt{Mensch} \sqcup \forall \texttt{vorfahr}.\texttt{Mensch}, \\ &\quad\ \texttt{Mensch}, \exists \texttt{mutter}.\top, \forall \texttt{vorfahr}.\texttt{Mensch}\} \\ L(x) &= \{\texttt{Mensch}, \forall \texttt{vorfahr}.\texttt{Mensch}, \neg\texttt{Mensch} \sqcup \forall \texttt{vorfahr}.\texttt{Mensch}\} \end{aligned}$$

Ausdrucksstärke von RBoxen

Transitivität und Rolleninklusion erlauben Internalisierung von TBoxen

- TBox kann in Teil des Eingabekonzepts (bzw. der ABox) umformuliert werden
- TBox ist dann überflüssig
- Verfahren: Erzeuge RBox $\mathcal{R}$
  - neue Rolle s ist Ober-Rolle aller anderen Rollen
    $r_1 \sqsubseteq s, r_2 \sqsubseteq s, \ldots$
  - s ist transitiv
    $\mathrm{Trans}(s)$
  - erzeuge Konzept G für alle TBox-GCIs
    $\mathcal{T} = \{D \sqsubseteq E, F \sqsubseteq H\} \quad \rightsquigarrow \quad G = \mathrm{nnf}((\neg D \sqcup E) \sqcap (\neg F \sqcup H))$
- C erfüllbar bzgl. $\mathcal{T}$ gdw. $C \sqcap G \sqcap \forall s.G$ erfüllbar bzgl. $\mathcal{R}$
- $\mathcal{A}$ erfüllbar bzgl. $\mathcal{T}$ gdw. $\mathcal{A} \cup \{a_1 : G \sqcap \forall s.G, a_2 : G \sqcap \forall s.G, \ldots\}$ erfüllbar bzgl. $\mathcal{R}$

Zusammenfassung: RBoxen

- Wissen über Eigenschaften von Rollen und deren Beziehungen zueinander
  - Rolleninklusion
  - Transitivität
- Anpassungen des Tableau-Algorithmus
  - Erweiterung der Definition von r-Nachfolger ($\rightsquigarrow$ Rolleninklusion)
  - $\forall^+$-Regel propagiert $\forall$-Restriktionen an Nachfolger ($\rightsquigarrow$ Transitivität)
- Ermöglicht Internalisierung von TBoxen

Analogien zwischen RDFS und $\mathcal{ALC}$

RDFS	$\mathcal{ALC}$		
`s p o .`	(s, o)	:	p
`s rdf:type o .`	s	:	O
`s rdfs:subClassOf o .`	S	$\sqsubseteq$	O
`s rdfs:subPropertyOf o .`	s	$\sqsubseteq$	o
`s rdfs:domain o .`	$\exists s.\top$	$\sqsubseteq$	O
`s rdfs:range o .`	$\top$	$\sqsubseteq$	$\forall s.O$
`s p [ rdf:type c ] .`	s	:	$\exists p.C$

Einschränkungen von DLs:

- strikte Trennung zwischen Konzepten, Rollen und Individuen

Einschränkungen von RDFS:

- keine komplexen Klassen-Ausdrücke
- keine Boole'schen Operatoren
- keine Werte-/Zahlen-Restriktionen

Inhalt

Zahlenrestriktionen

- ermöglichen Aussagen über die Anzahl der Rollennachfolger
- Mensch $\sqcap \geqslant 5\,\text{freund}.\top$ „Menschen, die mindestens 5 Freunde haben“
- Computer $\sqcap \leqslant 2\,\text{teil}.\text{Prozessor}$: „Computer, die höchstens 2 Prozessoren haben“
- nicht modellierbar als $\exists r.C \sqcap \exists r.C$!

Definition 4.36 (Qualifizierende Zahlenrestriktionen)

Sei n eine natürliche Zahl, r ein Rollenname und C ein Konzept.
Dann sind auch $\geqslant n\, r.C$ und $\leqslant n\, r.C$ Konzepte.
Die Erweiterung von $\mathcal{ALC}$ um qualifizierende Zahlenrestriktionen heißt $\mathcal{ALCQ}$.

Definition 4.37 (Semantik qualifizierender Zahlenrestriktionen)

Die Interpretation von Zahlenrestriktionen ist wie folgt definiert:

Syntax	Semantik
$\geqslant n\, r.C$	$\{x \in \Delta^{\mathcal{I}} \mid \#\{y \mid (x,y) \in r^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\} \geq n\}$
$\leqslant n\, r.C$	$\{x \in \Delta^{\mathcal{I}} \mid \#\{y \mid (x,y) \in r^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\} \leq n\}$

Für eine Menge M bezeichnet $\#M$ die Mächtigkeit von M.

Tableau-Regeln für Zahlenrestriktionen

Bisher
- Keine Notwendigkeit, auszudrücken, dass Knoten für verschiedende Entitäten stehen
- Keine Notwendigkeit, existierende Knoten zu löschen

Jetzt
- $\geqslant n\, r.C$ erfordert verschiedene Entitäten als Nachfolger
  $\rightsquigarrow$ Neue Knotenbeschriftungen vom Typ $\neq x$ für Knoten x
- $\leqslant n\, r.C$ erfordert Reduktion von Nachfolgern
  $\rightsquigarrow$ $\leqslant$-Regel verschmilzt Nachfolger

$\geqslant$ Wenn es ein $v \in V$ gibt mit $\geqslant n\, r.C \in L(v)$
und es keine n r-Nachfolger $v_1, \ldots, v_n$ von v gibt mit
$\{C\} \cup \{\neq v_j \mid j \neq i\} \subseteq L(v_i)$ für alle $1 \leq i, j \leq n$
dann $V := V \cup \{v_1, \ldots, v_n\}$, $E := E \cup \{(v, v_1), \ldots, (v, v_n)\}$,
$L(v_i) := \{C\} \cup \{\neq v_j \mid j \neq i\}$ und $L_E(v, v_i) = r$ für alle $1 \leq i, j \leq n$

$\leqslant$ Wenn es ein $v \in V$ gibt mit $\leqslant n\, r.C \in L(v)$
und es mehr als n r-Nachfolger $v_1, \ldots, v_n$ von v gibt mit $C \in L(v_i)$
und es zwei Knoten $\{v_i, v_j\} \subseteq \{v_1, \ldots, v_n\}$ gibt mit $\neq v_j \notin L(v_i)$,
dann setze $L(i) := L(i) \cup L(j)$ und $V := V \setminus \{j\}$
und mache alle Nachfolger von j zu Nachfolgern von i

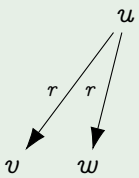
Clash-Bedingung für Zahlenrestriktionen

- $\leqslant$-Regel respektiert Ungleichheits-Assertionen $\neq x$
- nicht anwendbar, wenn es keine verschmelzbaren Knoten gibt
- Tableau liefert dann kein Modell
  $\rightsquigarrow$ zusätzliche Clash-Bedingung

Ein $\mathcal{ALCQ}$-Tableau enthält einen Clash, wenn

- ein Knoten v existiert mit $\leqslant n\, r.C \in L(v)$ und
- $n+1$ r-Nachfolger $v_1, \ldots, v_{n+1}$ von v existieren mit
  - $C \in L(v_i)$ für alle $1 \leq i \leq n+1$ und
  - $\neq v_j \in L(v_i)$ für alle $1 \leq i, j \leq n+1$

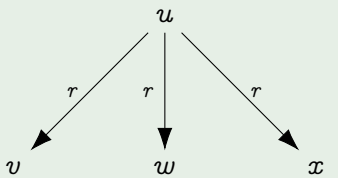
Beispiel 4.38 (Tableau für $\exists r.C \sqcap \exists r.\neg C \sqcap \geqslant 2\, r.D \sqcap \leqslant 2\, r.\top$)



$$
\begin{aligned}
L(u) &= \{\exists r.C \sqcap \exists r.\neg C \sqcap \geqslant 2\, r.D \sqcap \leqslant 2\, r.\top, \\
&\quad\ \exists r.C, \exists r.\neg C, \geqslant 2\, r.D, \leqslant 2\, r.\top\} \\
L(v) &= \{C, D, \neq w\} \\
L(w) &= \{\neg C, D, \neq v\}
\end{aligned}
$$

Unkorrektheit des bisherigen Algorithums

Beispiel 4.39 (Tableau für $\leqslant 1\, r.C \sqcap \leqslant 1\, r.\neg C \sqcap \geqslant 3\, r.D$)



$$
\begin{aligned}
L(u) &= \{\leqslant 1\, r.C \sqcap \leqslant 1\, r.\neg C \sqcap \geqslant 3\, r.D, \\
&\quad\ \leqslant 1\, r.C, \leqslant 1\, r.\neg C, \geqslant 3\, r.D\} \\
L(v) &= \{D, \neq w, \neq x, C \sqcup \neg C, C\} \\
L(w) &= \{D, \neq v, \neq x, C \sqcup \neg C, C\} \\
L(x) &= \{D, \neq v, \neq w, C \sqcup \neg C, \neg C\}
\end{aligned}
$$

- kein Nachfolgeknoten enthält C oder $\neg C$
- aber: jedes Domänenelement erfüllt $C \sqcup \neg C$
- Konzept ist inkonsistent, erzeugt aber keinen Clash!

choose Wenn es ein $v \in V$ gibt mit $\leqslant n\, r.C \in L(v)$
und es einen r-Nachfolger v' von v gibt mit $C \sqcup \dot{\neg} C \notin L(v')$
dann $L(v') := L(v') \cup \{C \sqcup \dot{\neg} C\}$

Übung: $\mathcal{ALCQ}$-Tableau-Algorithmus

Übung 4.40

Wenden Sie den Tableau-Algorithmus für $\mathcal{ALCQ}$ auf das folgende Konzept an:

$$K = \exists r.\neg C \sqcap \geqslant 3\, r.C \sqcap \geqslant 2\, r.D \sqcap \leqslant 3\, r.D \sqcap \leqslant 4\, r.\top$$

Anmerkungen:

- $\top$ wird per Definition von jedem Domänenelement erfüllt ($\top^{\mathcal{I}} = \Delta^{\mathcal{I}}$)
- Jeder Knoten ist implizit mit $\top$ beschriftet
- Choose-Regel muss für Konzepte $\leqslant n\, r.\top$ nicht angewendet werden, da es nur $\top \sqcup \bot$ hinzufügen würde

$\mathcal{ALCQ}$-Tableau-Algorithmus mit choose-Regel

Entscheidungsalgorithmus:

korrekt
- wegen choose-Regel kann aus einem clash-freien und vollständigen Tableau ein Modell abgeleitet werden

vollständig
- wenn es ein Modell gibt, kann ein clash-freies und vollständiges Tableau erzeugt werden

terminierend
- nicht trivial wegen $\leqslant$-Regel
- $\neq x$ verhindert „Jojo-Effekt“ von $\leqslant$- und $\geqslant$-Regel

Ineffizienz durch nichtdeterministische Regeln für $\leqslant$-Restriktionen

$\leqslant$ Wie viele Möglichkeiten gibt es, 10 existierende r-Nachfolger auf 4 zu reduzieren?
$\rightsquigarrow$ exponentielle Komplexität

choose Wie viele Möglichkeiten gibt es, Disjunktionen in 10 Nachfolgern zu verarbeiten?
$\rightsquigarrow$ exponentielle Komplexität

Konsequenz: $\leqslant$-Restriktionen sollten mit Bedacht eingesetzt werden

Transformationen

Negations-Normalform

- $\neg(\leqslant n\, r.C) \quad \rightsquigarrow \quad \geqslant (n+1)\, r.C$
- $\neg(\geqslant n\, r.C) \quad \rightsquigarrow \quad \leqslant (n-1)\, r.C$
- „Anna hat nicht 3 oder mehr Katzen.“ $\rightsquigarrow$ „Anna hat 2 oder weniger Katzen.“

Zusammenhang mit Existenz- und Werte-Restriktionen

- Zahlenrestriktionen verallgemeinern Existenz- und Werte-Restriktionen
- Existenz- und Werterestriktionen lassen sich durch Zahlenrestriktionen ausdrücken
- $\exists r.C \quad \rightsquigarrow \quad \geqslant 1\, r.C$
- $\forall r.C \quad \rightsquigarrow \quad \leqslant 0\, r.\neg C$
- „Bert hat nur Katzen.“ $\rightsquigarrow$ „Bert hat höchstens 0 Entitäten, die keine Katzen sind.“
- Effizienz: $\exists$ und $\forall$ verwenden, wenn möglich

RBoxen und Zahlenrestriktionen

Problem: „Maria hat maximal 3 Vorfahren.“

- $\mathcal{A} = \{\texttt{Maria:Mensch} \sqcap \leqslant 3\,\texttt{vorfahr}.\top\}$
- $\mathcal{T} = \{\texttt{Mensch} \sqsubseteq \exists\texttt{vorfahr}.\texttt{Mensch}\}$
- $\mathcal{R} = \{\text{Trans}(\texttt{vorfahr})\}$
- Wurzelknoten blockiert Nachfolger
- $\leqslant$-Regel muss Knoten aus unterschiedlichen Baumebenen verschmelzen

Lösung:

- Zahlenrestriktionen nur mit einfachen Rollen zulässig
- r ist komplex (d.h. nicht einfach)
  - wenn r transitiv ist oder
  - wenn r eine transitive Sub-Rolle hat.

Zusammenfassung: Zahlenrestriktionen

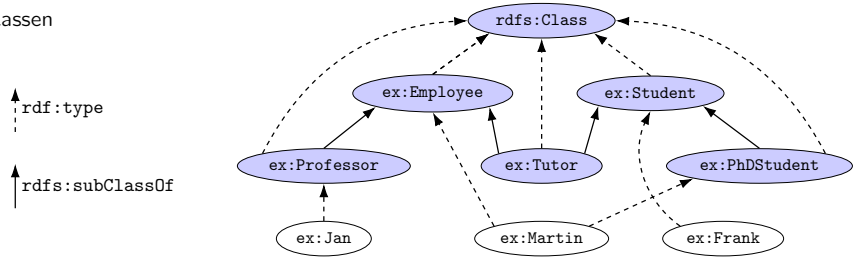
- $\mathcal{ALCQ}$ erlaubt $\geqslant n\, r.C$ und $\leqslant n\, r.C$
- schränkt Anzahl der r-Nachfolger ein
- Modifikationen des Tableau-Algorithmus
  - $\geqslant$-Regel erzeugt neue Nachfolger mit Ungleicheits-Assertionen $\neq x$
  - $\leqslant$-Regel verschmilzt Knoten
  - choose-Regel für $\leqslant n\, r.C$ erzwingt, dass alle r-Nachfolger mit C oder $\neg C$ beschriftet sind
  - Clash-Bedingung für Knoten, die mit $\leqslant n\, r.C$ beschriftet sind, aber mehr als n ungleiche r-Nachfolger haben, die mit C beschriftet sind
- $\leqslant$-Restriktionen können zu Ineffizienz führen

Inhalt

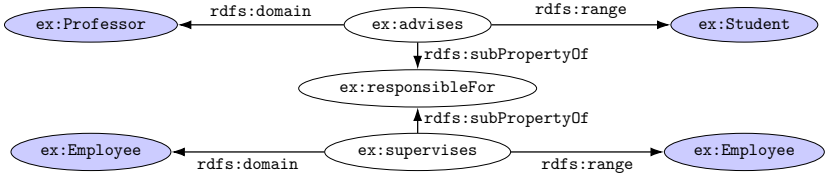
Inhalt

Einfache Ontologien in RDFS

Klassen



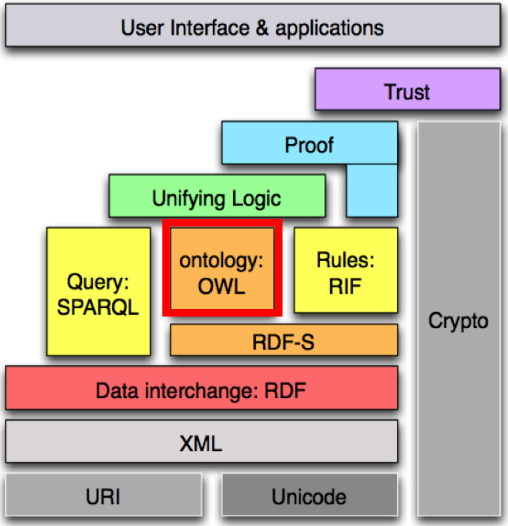
Properties



RDFS als Ontologiesprache

- Geeignet für einfache Ontologien
  - Automatisches Schlussfolgern relativ effizient
  - Leicht zu verstehen
  - Keine inkonsistenten Ontologien
- Aber: für komplexere Modellierungen ungeeignet
- Rückgriff auf mächtigere Sprachen wie OWL

OWL im Schichtenmodell



- „Web Ontology Language“
- W3C Recommendations
  - OWL 1 2004
  - OWL 2 2012
- basiert auf Beschreibungslogiken
  - DL-Semantik
  - DL-Tools verwendbar
  - unterstützt nicht alle RDF-Graphen
    - keine Klassen als Instanzen anderer Klassen
    - keine Klassen als Subjekte / Objekte von Properties

OWL-Ontologie in TTL-Syntax

Beispiel 5.1

```
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix ex: <http://example.org/> .

ex:Beispielontologie rdf:type owl:Ontology .

ex:Person rdf:type owl:Class .

ex:Charlie rdf:type owl:NamedIndividual ,
                    ex:Person ;
         ex:owns ex:Snoopy ;
         ex:age "6"^^xsd:integer .
```

OWL-Ontologie im Manchester-Format

Beispiel 5.2

```
Prefix: owl: <http://www.w3.org/2002/07/owl#>
Prefix: ex: <http://www.example.org/>

Ontology: ex:Beispielontologie

ObjectProperty: ex:owns
DataProperty: ex:age
Class: ex:Person

Individual: ex:Charlie
  Types:
    ex:Person
  Facts:
    ex:owns  ex:Snoopy
    ex:age  6
```

Inhalt

Klassen, Properties und Individuen

- Klassen (`owl:Class`)
  - DL-Konzepte
  - vergleichbar mit Klassen in RDFS
  - aber: OWL-Klasse kann nicht Instanz einer anderen Klasse sein
- Objekt-, Daten- und Annotations-Properties (`owl:Object/Data/AnnotationProperty`)
  - DL-Rollen
  - Vergleichbar mit Properties in RDFS
  - aber: Unterscheidung, ob Objekt URI oder Literal ist
  - aber: Property kann nicht Instanz einer Klasse sein
  - aber: Properties sind disjunkt von Klassen und Individuen
  - Annotation Property: Nur zur Annotation der Entitäten, keine Semantik
- Individuen (`owl:NamedIndividual`)
  - DL-Individuen
  - vergleichbar mit URIs in RDFS
  - aber: OWL-Individuum kann keine Klasse sein
  - keine eigene Klasse in RDFS, da jeder Knoten als Individuum betrachtet werden kann

Beispiel 5.3 (Definition von OWL-Klassen)

DL:	Konzeptname
Semantik:	Teilmenge der Domäne
Manchester:	`Prefix: : <http://example.org/>` `Class: Professor`
TTL:	`@prefix : <http//example.org/> .` `:Professor a owl:Class .`

`owl:Thing` enthält alle Entitäten und ist Oberklasse aller Klassen (DL: $\top$)

`owl:Nothing` enthält keine Entitäten und ist Unterklasse aller Klassen (DL: $\bot$)

Leeres Präfix „`:`“ kann in Manchester OWL weggelassen werden, in TTL nicht!

Beispiel 5.4 (Definition von Object Properties)

DL: Rolle

Semantik: zweistellige Relation über Domäne (Menge von Paaren)

Manchester:
```
ObjectProperty: mitglied
  Domain: Mensch
  Range: Organisation
```

TTL:
```
:mitglied a owl:ObjectProperty ;
  rdfs:domain :Mensch ;
  rdfs:range :Organisation .
```

- gleiche Semantik wie in RDFS
- deshalb keine neuen Properties `owl:domain` / `owl:range`

Beispiel 5.5 (Definition von Data Properties)

DL: Rolle mit konkreter Domäne (nicht behandelt)

Semantik: binäre Relation zwischen Domäne und konkreter Menge

Manchester:
```
DataProperty: vorname
  Domain: Mensch
  Range: xsd:string
```

TTL:
```
:vorname a owl:DatatypeProperty ;
  rdfs:domain :Mensch ;
  rdfs:range xsd:string .
```

Viele XML-Datentypen können als `range` verwendet werden.

Annotation Properties

- erlauben Meta-Modellierung: Beschreiben Modell selbst, nicht Domäne
- zulässige Subjekte: alle Arten von OWL-Entitäten
- zulässige Objekte: Literale, Individuen
- erlauben keine Schlussfolgerungen
  - werden von Inferenzmaschine ignoriert (wie Kommentare vom Compiler)
  - keine `domain`/`range`-Restriktionen
  - keine `subPropertyOf`-Axiome

Beispiel 5.6 (Definition von Annotation Properties)

DL:	(nicht vorhanden)
Semantik:	keine
Manchester:	`AnnotationProperty: creator`
TTL:	`:creator a owl:AnnotationProperty .`

Vordefinierte Annotation Properties:

- `owl:versionInfo`
- `rdfs:label`
- `rdfs:comment`
- `rdfs:seeAlso`
- `rdfs:isDefinedBy`

Beispiel 5.7 (Definition von Individuen und Fakten)

DL: ABox-Individuum / ABox-Fakt

Semantik: Element der Domäne / Relation über Domäne

Manchester:

```
Individual: John
  Types: Professor
  Facts: worksFor DHBW
Individual: Mary
  Types: Student
  Facts: age 25
```

TTL:

```
:John a owl:NamedIndividual , :Professor ;
  :worksFor :DHBW .
:Mary a owl:NamedIndividual , :Student ;
  :age "25".
```

Gleichheit und Verschiedenheit von Individuen

Beispiel 5.8 (SameAs und DifferentFrom)

DL: $\texttt{Bill} \doteq \texttt{William}$ / $\neg(\texttt{Bill} \doteq \texttt{Mary})$ (nicht behandelt)

Semantik: $\texttt{Bill}^{\mathcal{I}} = \texttt{William}^{\mathcal{I}}$ / $\texttt{Bill}^{\mathcal{I}} \neq \texttt{Mary}^{\mathcal{I}}$

Manchester:
```
Individual: Bill
  SameAs: William
  DifferentFrom: Mary
```

TTL:
```
:Bill owl:sameAs :William ;
  owl:differentFrom :Mary.
```

Erinnerung: Nicht möglich in RDF(S)

Beispiel 5.9 (Abkürzung mit DifferentIndividuals)

Manchester:
```
DifferentIndividuals: John, Paul, George, Ringo
```

TTL:
```
[] a owl:AllDifferent ;
  owl:members ( :John :Paul :George :Ringo ) .
```

Inhalt

Semantik von OWL

Die Semantik ist modelltheoretisch definiert, d.h. mittels Interpretationen

Definition 5.10 (Interpretation für OWL)

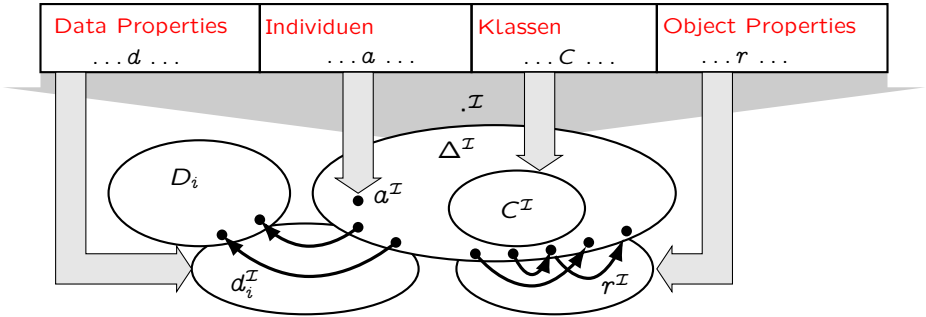
Sei A eine OWL-Ontologie mit

- einer Menge I von Individuen,
- einer Menge K von Klassen,
- einer Menge P von Object Properties und
- einer Menge $D = \{d_1, d_2, \ldots\}$ von Data Properties mit zugehörigen Datentypen $D_1, D_2, \ldots$

Eine Interpretation $\mathcal{I}$ für A besteht aus einer Domäne $\Delta^{\mathcal{I}}$ und einer Funktion $\cdot^{\mathcal{I}}$, die

- jedes Individuum $a \in I$ auf ein Domänenelement abbildet ($a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$),
- jede Klasse $C \in K$ auf eine Menge von Domänenelementen abbildet ($C^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$),
- jede Object Property $r \in P$ auf eine Menge von Paaren von Domänenelementen abbildet ($r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$),
- jede Data Property $d_i \in D$ auf eine Menge von Paaren aus Elementen der Domäne und Elementen des Datentyps D_i abbildet ($d_i^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times D_i$).

Schematische Darstellung einer Interpretation



Definition 5.11 (Modell)

Eine Interpretation $\mathcal{I}$ ist ein Modell für eine OWL-Ontologie, wenn es ein Modell für jedes Axiom der Ontologie ist.

Unterschiede der Semantiken von RDFS und OWL

	RDFS	OWL
Klasse	Domänenelement mit Klassen-Extension	Menge von Domänenelementen
Property	Domänenelement mit Property-Extension	Menge von Paaren von Domänenelementen
Literal	Domänenelement vom Typ `rdfs:Literal`	Element einer konkreten Domäne

Beispiel 5.12 (Zulässig in RDFS, unzulässig in OWL)

- ```
  ex:Tom rdf:type ex:Katze .
  ex:Katze rdf:type ex:Spezies .
  ```
- ```
  ex:Kind rdf:type rdfs:Class .
  ex:Kind rdf:type rdf:Property .
  ```
- ```
  ex:Mary ex:mag ex:mag .
  ```
- ```
  ex:Golf rdf:type rdfs:Class .
  ex:VolkswagenAG ex:baut ex:Golf .
  ```
- ```
  ex:Mensch rdfs:subClassOf "Lebewesen".
  ```
  ```

Inhalt

Einfache Klassenbeziehungen: rdfs:subClassOf

Beispiel 5.13 („Jeder Professor ist ein Mensch.“)

DL: $\texttt{Professor} \sqsubseteq \texttt{Mensch}$

Semantik: $\texttt{Professor}^{\mathcal{I}} \subseteq \texttt{Mensch}^{\mathcal{I}}$

Manchester:
```
Class: Professor
  SubClassOf: Mensch
```

TTL:
```
:Professor a owl:Class ;
  rdfs:subClassOf :Mensch .
```

- gleiche Semantik wie in RDFS
- daher keine neue Property owl:subClassOf
- aber: rdfs:Class $\neq$ owl:Class!

Einfache Klassenbeziehungen: owl:equivalentClass

Äquivalente Klassen haben dieselben Elemente

Beispiel 5.14 („Publikationen und Veröfentlichungen sind identisch.“)

DL: $\texttt{Veroeffentlichung} \equiv \texttt{Publikation}$

Semantik: $\texttt{Veroeffentlichung}^{\mathcal{I}} = \texttt{Publikation}^{\mathcal{I}}$

Manchester:
```
Class: Veroeffentlichung
  EquivalentTo: Publikation
```

TTL:
```
:Veroeffentlichtung a owl:Class ;
  owl:equivalentClass :Publikation
```

Inhalt

Durchschnitt und Vereinigung von Klassen

Beispiel 5.15

DL: $\texttt{Mensch} \sqcap \texttt{Weiblich} \quad / \quad \texttt{Junge} \sqcup \texttt{Maedchen}$

Semantik: $\texttt{Mensch}^{\mathcal{I}} \cap \texttt{Weiblich}^{\mathcal{I}} \quad / \quad \texttt{Junge}^{\mathcal{I}} \cup \texttt{Maedchen}^{\mathcal{I}}$

Manchester:
```
Class: Frau
  EquivalentTo: Mensch and Weiblich
Class: Kind
  EquivalentTo: Junge or Maedchen
```

TTL:
```
:Frau a owl:Class ;
  owl:equivalentClass [
    a owl:Class ;
    owl:intersectionOf ( :Mensch :Weiblich ) ] .
:Kind a owl:Class ;
  owl:equivalentClass [
    a owl:Class ;
    owl:unionOf ( :Junge :Maedchen ) ] .
```

Komplement von Klassen

Beispiel 5.16 („Kein Mensch ist ein Ding oder ein Tier.“)

DL: $\neg \texttt{Ding}$

Semantik: $\Delta^{\mathcal{I}} \setminus \texttt{Ding}^{\mathcal{I}}$

Manchester:
```
Class: Mensch
  DisjointWith: Ding, Tier
äquivalent:
  SubClassOf: not (Ding or Tier)
```

TTL
```
:Mensch a owl:Class ;
  rdfs:subClassOf [
    a owl:Class ;
    owl:complementOf [
      a owl:Class ;
      owl:unionOf ( :Ding :Tier ) ] ] .
```

Übung: OWL-Ontologien

Übung 5.17

Gegeben sei das Vokabular:

- die Klassen `Pizza`, `Lebensmittel`, `Wurst`, `Gemuese`, `Belag`
- die ObjectProperty `belegtMit`,
- das Individuum `Salami`.

Modellieren Sie die folgenden Aussagen als OWL-Ontologie in Manchester-Syntax.

1. Jede Pizza ist ein Lebensmittel.
2. Alles, das mit irgendetwas belegt ist, ist eine Pizza.
3. Alles, womit irgendetwas belegt ist, ist ein Belag.
4. Jede Wurst ist sowohl Lebensmittel als auch Belag.
5. Salami ist eine Wurst.
6. Keine Wurst ist ein Gemüse.

Inhalt

Existenz- und Werterestriktion

Beispiel 5.18

DL: $\forall \texttt{pruefer.Dozent} \quad / \quad \exists \texttt{pruefer.Professor}$

Semantik: $\{x \in \Delta^{\mathcal{I}} \mid \text{aus } (x, y) \in \texttt{pruefer}^{\mathcal{I}} \text{ folgt } y \in \texttt{Dozent}^{\mathcal{I}}\}$
$\{x \in \Delta^{\mathcal{I}} \mid \text{es gibt } y \in \Delta^{\mathcal{I}} \text{ mit } (x, y) \in \texttt{pruefer}^{\mathcal{I}} \text{ und } y \in \texttt{Professor}^{\mathcal{I}}\}$

Manchester:
```
Class: Pruefung
  SubClassOf: pruefer only Dozent
  SubClassOf: pruefer some Professor
```

TTL:
```
ex:Pruefung a owl:Class ;
  rdfs:subClassOf [   # Realisierung mit Bnode
    a owl:Restriction ;
    owl:onProperty ex:pruefer ;
    owl:allValuesFrom ex:Dozent ] ;
  rdfs:subClassOf [
    a owl:Restriction ;
    owl:onProperty ex:pruefer ;
    owl:someValuesFrom ex:Professor ] .
```

Beispiel 5.19 („Eine Prüfung hat 3–6 Dozenten als Prüfer.“)

DL: $\geqslant 3\,\mathsf{pruefer}.\mathsf{Dozent} \quad / \quad \leqslant 6\,\mathsf{pruefer}.\mathsf{Dozent}$

Semantik: $\{x \in \Delta^{\mathcal{I}} \mid 3 \leq \#\{y \mid (x,y) \in r^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\} \leq 6\}$

Manchester:
```
Class: Pruefung
  SubClassOf: pruefer min 3 Dozent
  SubClassOf: pruefer max 6 Dozent
```

TTL:
```
ex:Pruefung a owl:Class ;
  rdfs:subClassOf [   # Realisierung mit Bnode
    a owl:Restriction ;
    owl:onProperty ex:pruefer ;
    owl:minQualifiedCardinality 3^^xsd:nonNegativeInteger ;
    owl:onClass ex:Dozent ] ;
  rdfs:subClassOf [
    a owl:Restriction ;
    owl:onProperty ex:pruefer ;
    owl:maxQualifiedCardinality 6^^xsd:nonNegativeInteger ;
    owl:onClass ex:Dozent ] .
```

Vorrang in Manchester OWL

- Quantoren (einschließlich $\leqslant$ und $\geqslant$) binden stärker als Junktoren
- `kind some Mensch and Weiblich` bedeutet `(kind some Mensch) and Weiblich`

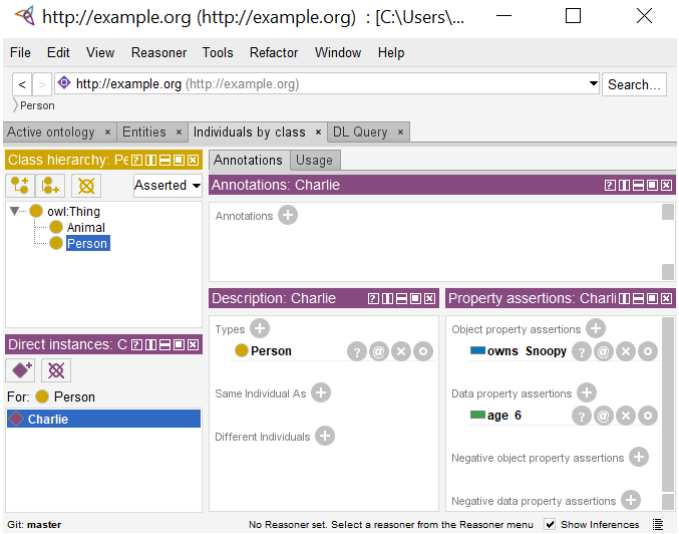
Beispiel 5.20

„Ein Mensch, der ein Haus mit Garten besitzt, und mit jemandem verheiratet ist, der mindestens 2 glückliche Kinder hat“

```
Mensch and besitzt some (Haus and hat some Garten)
       and verheiratet some (kind min 2 Glücklich)
```

Der Ontologie-Editor Protégé

- Grafischer Editor für OWL 2
- Syntax: XML, TTL, Functional
- Schnittstelle für Reasoner
- `http://protege.stanford.edu`



Übung: Erweiterung der Pizza-Ontologie

Übung 5.21

Öffnen Sie die von Ihnen erstellte Pizza-Ontologie (Übung 5.17) in Protégé.
Gegeben seien die zusätzlichen Klassen `Margherita`, `QuattroStagioni`, `Tomate`, `Käse` und `Artischocke`.
Erweitern Sie die Ontologie um die folgenden Aussagen:

1. Jede Pizza Margherita ist mit Tomate und Käse belegt.
2. Keine Pizza Margherita ist mit Wurst belegt.
3. Jede Pizza hat höchstens 8 Beläge.
4. Jede Pizza Quattro Stagioni hat mindestens 4 Beläge, darunter `Artischocke`.
5. Artischocken und Tomaten sind Gemüse.

Property-Axiome in OWL

Eigenschaften von Properties: `p a owl:...Property`

TransitiveProperty $(x, y) \in p^{\mathcal{I}} \wedge (y, z) \in p^{\mathcal{I}} \rightarrow (x, z) \in p^{\mathcal{I}}$
Beispiel: `hatNachkomme`
DL: Transitivitäts-Axiom

ReflexiveProperty $(x, x) \in p^{\mathcal{I}}$
Beispiel: `kennt`

IrreflexiveProperty $(x, x) \notin p^{\mathcal{I}}$
Beispiel: `hatKind`

SymmetricProperty $(x, y) \in p^{\mathcal{I}} \rightarrow (y, x) \in p^{\mathcal{I}}$
Beispiel: `verheiratetMit`

AsymmtericProperty $(x, y) \in p^{\mathcal{I}} \rightarrow (y, x) \notin p^{\mathcal{I}}$
Beispiel: `hatKind`

FunctionalProperty $(x, y) \in p^{\mathcal{I}} \wedge (x, z) \in p^{\mathcal{I}} \rightarrow y = z$
Beispiel: `hatMutter`

InverseFunctionalProperty $(x, z) \in p^{\mathcal{I}} \wedge (y, z) \in p^{\mathcal{I}} \rightarrow x = y$
Beispiel: `istMutterVon`

Beziehungen zwischen Properties

Beziehungen zwischen zwei Properties: `p ... q`

`rdfs:subPropertyOf`	$(x, y) \in p^{\mathcal{I}} \rightarrow (x, y) \in q^{\mathcal{I}}$ Beispiel: `hatFreund rdfs:subPropertyOf kennt` DL: Rolleninklusions-Axiom
`owl:equivalentProperty`	$p^{\mathcal{I}} = q^{\mathcal{I}}$ Beispiel: `kennt owl:equivalentProperty hatBekannten`
`owl:propertyDisjointWith`	$(x, y) \in p^{\mathcal{I}} \rightarrow (x, y) \notin q^{\mathcal{I}}$ Beispiel: `hatVater owl:propertyDisjointWith hatSohn`
`owl:inverseOf`	$(x, y) \in p^{\mathcal{I}} \leftrightarrow (y, x) \in q^{\mathcal{I}}$ Beispiel: `hatVorfahr owl:inverseOf hatNachkomme`

Übung: Property-Axiome

Übung 5.22

1. Erweitern Sie die Pizza-Ontologie um die Property `hatZutat`, die eine Ober-Property von `hatBelag` ist.
2. Welche der genannten Property-Eigenschaften treffen auf die Pizza-Properties `hatBelag` und `hatZutat` zu?

Inhalt

Terminologische Schlussfolgerungen

Terminologisch: nur Klassen und Properties betreffend

- Unter-/Oberklassenbeziehung, Äquivalenz
  - Gilt $\texttt{ex:Dog}^{\mathcal{I}} \subseteq \texttt{ex:Mammal}^{\mathcal{I}}$?
  - DL: Konzeptsubsumtion (bzgl. TBox)
  - Protégé: DL Query → query for sub- / super- / equivalent classes
  - Ermöglicht Erstellen der Klassenhierarchie
- Klassenkonsistenz
  - Gilt $\texttt{ex:Mensch}^{\mathcal{I}} = \emptyset$?
  - DL: Konzeptkonsistenz (bzgl. TBox)
  - Protégé: Klassenname in rot (bei aktiviertem Reasoner)
  - inkonsistente Klasse ist äquivalent zu `owl:Nothing`
  - deutet auf Modellierungsfehler hin

Beispiel 5.23 (inkonsistente Klasse)

```
ex:Buch a owl:Class ;
  rdfs:subClassOf ex:Publikation ;
  owl:disjointWith ex:Publikation .
```

Assertionale Schlussfolgerungen

Assertional: auch Individuen betreffend

- Klassen-Instanz
  - Gehört `ex:Charlie` zur Klasse `ex:Mensch`?
  - DL: Konzept-Instanz
- Klassenextension
  - Finde alle Individuen, die (notwendig) in einer Klasse enthalten sind
  - Welche Individuen sind (sicher) Menschen?
  - DL: Konzept-Extension
  - Protégé: Individuals by class
- Property-Instanz
  - Werden zwei gegebene Individuen durch eine bestimmte Property verknüpft?
  - Ist John ein Freund von Mary?
  - Protégé: Individuals by class, Property assertions
- Property-Extension
  - Suche nach allen Individuenpaaren, die durch eine Property verknüpft sind
  - Welche Paare sind verheiratet?
  - Protégé: Object Properties → Object Property Usage

Inhalt

Überblick OWL-Sprachelemente: Klassen

Klassenkonstruktoren	
Klassennamen	`owl:Class`
Konjunktion	`owl:intersectionOf`
Disjunktion	`owl:unionOf`
Negation	`owl:complementOf`
Existentielle Restriktion	`owl:someValuesFrom`
Universelle Restriktion	`owl:allValuesFrom`
Größer-als	`owl:minQualifiedCardinality`
Kleiner-als	`owl:maxQualfifiedCardinality`

Klassenaxiome	
Inklusion	`rdfs:subClassOf`
Äquivalenz	`owl:equivalentClass`

Überblick OWL-Sprachelemente: Object Properties

Properties

Object Properties	`owl:ObjectProperty`

Property-Axiome

Inklusion	`rdfs:subPropertyOf`
Gleichheit	`owl:equivalentProperty`
Disjunktheit	`owl:propertyDisjointWith`
Inverse Property	`owl:inverseOf`
Transitivität	`owl:TransitiveProperty`
Symmetrie	`owl:(A)SymmetricProperty`
Reflexivität	`owl:(Ir)ReflexiveProperty`
Funktionalität	`owl:(Inverse)FunctionalProperty`

Überblick OWL-Sprachelemente: Individuen

Assertionen

Klassenzugehörigkeit	`rdf:type` (a)
Property-Beziehung	(RDF-Tripel)
Gleichheit	`owl:sameAs`
Ungleichheit	`owl:differentFrom`

Weiterführende Literatur

- https://www.w3.org/TR/owl-overview/
  Zentrale W3C Webseite für OWL
- https://www.w3.org/TR/owl-primer/
  Einführung mit Beispielen
- https://www.w3.org/TR/owl2-manchester-syntax/
  Manchester-Syntax
- https://www.w3.org/TR/owl-syntax/
  Funktionale Syntax
- https://www.w3.org/TR/owl-mapping-to-rdf/
  Übersetzung der funktionalen Syntax in TTL
- https://www.w3.org/TR/owl-direct-semantics/
  Direkte Semantik (mit funktionaler Syntax)

Zusammenfassung: Semantic Web

- Ontologien
  - formale und explizite Spezifikation
  - einer allgemein akzeptierten Konzeptualisierung
  - Wissensbank und Inferenzmaschine
- Linked Data und RDF(S)
  - leichgewichtige Ontologie-Sprache
  - Tripel, URIs, Literale, Bnodes
  - wesentliche Werkzeuge: domain, range, subclass, subproperty
  - SPARQL-Anfragen
- Beschreibungslogiken und OWL
  - ausdrucksstarke Ontologie-Sprache
  - Boolesche Operatoren, Zahlenrestriktionen
  - TBox, ABox, RBox
  - Tableau-Algorithmen
  - Protégé und Hermit