



Semantic Web Grundlagen

RDFS Schlussfolgern & Datalog Regeln

Birte Glimm
Institut für Künstliche Intelligenz | 10. Nov 2011

Organisatorisches: Inhalt

Einleitung und XML	17. Okt	SPARQL Syntax & Intuition	12. Dez
Einführung in RDF	20. Okt	Übung 4	15. Dez
RDF Schema	24. Okt	SPARQL Semantik	19. Dez
fällt aus	27. Okt	SPARQL 1.1	22. Dez
Logik – Grundlagen	31. Okt	Übung 5	9. Jan
Übung 1	3. Nov	SPARQL Entailment	12. Jan
Semantik von RDF(S)	7. Nov	SPARQL Implementierung	16. Jan
RDF(S) & Datalog Regeln	10. Nov	Abfragen & RIF	19. Jan
OWL Syntax & Intuition	14. Nov	Übung 6	23. Jan
Übung 2	17. Nov	Ontology Editing	26. Jan
OWL & BLs	21. Nov	Ontology Engineering	30. Jan
OWL 2	24. Nov	Linked Data	2. Feb
Tableau	28. Nov	Übung 7	6. Feb
Übung 3	1. Dez	SemWeb Anwendungen	9. Feb
Blocking & Unravelling	5. Dez	Wiederholung	13. Feb
Hypertableau	8. Dez	Übung 8	16. Feb

Agenda

- ▶ **Regeln**
 - ▶ Llyod-Topor Transformation
- ▶ **Datalog**
 - ▶ Charakterisierungen der Semantik von Datalog Programmen
- ▶ **Evaluierung von Datalog Programmen**
 - ▶ Naive Evaluierung
 - ▶ Semi-naive Evaluierung
- ▶ **Regeln für RDFS mittels Triple Prädikat**
- ▶ **Regeln für RDFS durch direkte Übersetzung**

Bestandteile von Regeln

- ▶ **Grundelemente von Regeln sind Atome**
 - ▶ **Ground** Atome ohne freie Variablen
 - ▶ **Non-Ground** Atome mit freien Variablen

Was sind Regeln?

1. Logische Regeln (Fragmente von Prädikatenlogik):
 - ▶ $F \rightarrow G$ ist äquivalent zu $\neg F \vee G$
 - ▶ Logische Erweiterung der Wissensbasis \rightsquigarrow **statisch**
 - ▶ Open World
 - ▶ **Deklarativ** (beschreibend)
2. Prozedurale Regeln (z.B. Production Rules):
 - ▶ "If X then Y else Z"
 - ▶ Ausführbare Maschinen-Anweisungen \rightsquigarrow **dynamisch**
 - ▶ **Operational** (Bedeutung = Effekt bei Ausführung)
3. Logikprogrammierung et al. (z.B. PROLOG, F-Logik):
 - ▶ $\text{mann}(X) \leftarrow \text{person}(X) \text{ AND NOT } \text{frau}(X)$
 - ▶ Approximation logischer Semantik mit operationalen Aspekten, Built-ins möglich
 - ▶ häufig Closed World
 - ▶ **Semi-deklarativ**

Regelbeispiel

Beispiel:

$$\text{hatOnkel}(x,z) \leftarrow \text{hatElternteil}(x,y) \wedge \text{hatBruder}(y,z)$$

- ▶ Wir verwenden kurze Namen hier (hatOnkel) an Stelle von qualifizierten IRIs (<http://example.org/Beispiel#hatOnkel>) oder abgekürzten IRIs (ex:hatOnkel)
- ▶ Wir benutzen x, y, und z für Variablen

Prädikatenlogik als Regelsprache

- ▶ Regeln als Implikationsformeln der Prädikatenlogik:

$$\underbrace{H}_{\text{Kopf}} \leftarrow \underbrace{A_1 \wedge A_2 \wedge \dots \wedge A_n}_{\text{Rumpf}}$$

\rightsquigarrow Semantisch äquivalent zu Disjunktion:

$$H \vee \neg A_1 \vee \neg A_2 \vee \dots \vee \neg A_n$$

- ▶ Implikation von rechts nach links üblich (\leftarrow oder $:-$)
- ▶ Konstanten, Variablen und Funktionssymbole erlaubt
- ▶ Quantoren für Variablen werden oft weggelassen: freie Variablen als universell quantifiziert verstanden (d.h. Regel gilt für alle Belegungen)

Agenda

- ▶ Regeln
 - ▶ **Lloyd-Topor Transformation**
- ▶ Datalog
 - ▶ Charakterisierungen der Semantik von Datalog Programmen
- ▶ Evaluierung von Datalog Programmen
 - ▶ Naive Evaluierung
 - ▶ Semi-naive Evaluierung
- ▶ Regeln für RDFS mittels Triple Prädikat
- ▶ Regeln für RDFS durch direkte Übersetzung

Lloyd-Topor Transformation

- ▶ Mehrere Atome im Kopf werden meist als Konjunktion verstanden

$$H_1, H_2, \dots, H_m \leftarrow A_1, A_2, \dots, A_n$$

äquivalent zu

$$H_1 \leftarrow A_1, A_2, \dots, A_n$$

$$H_2 \leftarrow A_1, A_2, \dots, A_n$$

...

$$H_m \leftarrow A_1, A_2, \dots, A_n$$

- ▶ Eine derartige Umformung bezeichnet man als **Lloyd-Topor Transformation**

Arten von Regeln

Bezeichnungen für "Regeln" der Prädikatenlogik:

- ▶ **Klausel:** Disjunktion von atomaren Aussagen oder negierten atomaren Aussagen
 - ▶ $\text{Frau}(x) \vee \text{Mann}(x) \leftarrow \text{Person}(x)$
- ▶ **Hornklausel:** Klausel mit *höchstens* einem nicht-negiertem Atom
 - ▶ $\leftarrow \text{Mann}(x) \wedge \text{Frau}(x)$
 - ↪ "Integritätsbedingungen"
- ▶ **Definite Klausel:** Klausel mit *genau einem* nicht negiertem Atom
 - ▶ $\text{Vater}(x) \leftarrow \text{Mann}(x) \wedge \text{hatKind}(x, y)$
- ▶ **Fakt:** Klausel aus einem einzigen nicht-negiertem Atom
 - ▶ $\text{Frau}(\text{gisela})$

Disjunktive Regeln

- ▶ Regel können auch Disjunktionen enthalten
- ↪ Mehrere Atome im Kopf werden als Alternativen verstanden:

$$H_1, H_2, \dots, H_m \leftarrow A_1, A_2, \dots, A_n$$

äquivalent zu

$$H_1 \vee H_2 \vee \dots \vee H_m \leftarrow A_1 \wedge A_2 \wedge \dots \wedge A_n$$

äquivalent zu

$$H_1 \vee H_2 \vee \dots \vee H_m \vee \neg A_1 \vee \neg A_2 \vee \dots \vee \neg A_n$$

- ↪ Hier nicht betrachtet

Arten von Regeln

Regeln können auch **Funktionssymbole** enthalten:

$$\begin{aligned} \text{OnkelVon}(x, y) &\leftarrow \text{hatBruder}(\text{mutter}(x), y) \\ \text{hatVater}(x, \text{vater}(x)) &\leftarrow \text{Person}(x) \end{aligned}$$

- ↪ Generiert dynamisch neue Elemente
- ↪ Hier nicht betrachtet
- ↪ Logische Programmierung

Agenda

- ▶ Regeln
 - ▶ Llyod-Topor Transformation
- ▶ Datalog
 - ▶ Charakterisierungen der Semantik von Datalog Programmen
- ▶ Evaluierung von Datalog Programmen
 - ▶ Naive Evaluierung
 - ▶ Semi-naive Evaluierung
- ▶ Regeln für RDFS mittels Triple Prädikat
- ▶ Regeln für RDFS durch direkte Übersetzung

Datalog

Horn-Regeln ohne Funktionssymbole \rightsquigarrow **Datalog-Regeln**

- ▶ Logische Regelsprache, ursprünglich Grundlage *deduktiver Datenbanken*
- ▶ Wissensbasen (“Programme”) aus Horn-Klauseln ohne Funktionssymbole
- ▶ Entscheidbar
- ▶ Effizient für große *Datenmengen*, Gesamtkomplexität EXPTIME
- ▶ Viel Forschung in den 80-er Jahren

Datalog als Erweiterung des Relationenkalküls

Datalog kann als Erweiterung des Relationenkalküls mit Rekursion gesehen werden

$$T(x, y) \leftarrow E(x, y)$$

$$T(x, y) \leftarrow E(x, z) \wedge T(z, y)$$

\rightsquigarrow Berechnet die transitive Hülle (T) der Relation E, z.B., wenn E die Kanten (edges) eines Graphen enthält

- ▶ Ein Set von (ground) Fakten nannte sich auch *Instanz*

Agenda

- ▶ Regeln
 - ▶ Llyod-Topor Transformation
- ▶ Datalog
 - ▶ Charakterisierungen der Semantik von Datalog Programmen
- ▶ Evaluierung von Datalog Programmen
 - ▶ Naive Evaluierung
 - ▶ Semi-naive Evaluierung
- ▶ Regeln für RDFS mittels Triple Prädikat
- ▶ Regeln für RDFS durch direkte Übersetzung

Semantik von Datalog

Drei verschiedene aber äquivalente Arten die Semantik zu definieren:

- ▶ Modell-Theoretisch
- ▶ Beweis-Theoretisch
- ▶ Fixpunkt-Semantik

Modell-Theoretische Semantik von Datalog

Sieht die Regeln als Sätze:

$$\forall x, y. (T(x, y) \leftarrow E(x, y))$$

$$\forall x, y. (T(x, y) \leftarrow E(x, z) \wedge T(z, y))$$

- ▶ Nicht genug für eine eindeutige Bestimmung des Ergebnisses
- ↔ Interpretation von T muss minimal sein

Modell-Theoretische Semantik von Datalog

Prinzipiell repräsentiert eine Datalog Regel

$$\rho: R_1(u_1) \leftarrow R_2(u_2), \dots, R_n(u_n)$$

den logischen Satz

$$\forall x_1, \dots, x_n. (R_1(u_1) \leftarrow R_2(u_2) \wedge \dots \wedge R_n(u_n))$$

- ▶ x_1, \dots, x_n sind die Variablen der Regel und \leftarrow ist die logische Implikation
- ▶ Eine Instanz I erfüllt die Regel ρ , geschrieben $I \models \rho$, genau dann wenn jede Instanziierung

$$R_1(\nu(u_1)) \leftarrow R_2(\nu(u_2)), \dots, R_n(\nu(u_n))$$

in der $R_2(\nu(u_2)), \dots, R_n(\nu(u_n))$ in I wahr sind, dann ist auch $R_1(\nu(u_1))$ wahr

Modell-Theoretische Semantik von Datalog

- ▶ Eine Instanz I ist ein Modell eines Datalog Programms P , wenn I jede Regel in P gesehen als logischer Satz wahr macht
- ▶ Die Semantik von P für die Eingabe I ist das *minimale* Modell das I enthält (wenn es existiert)
- ▶ Frage: Existiert so ein Modell immer?
- ▶ Wie können wir so ein Modell konstruieren?

Beweis-Theoretische Semantik von Datalog

Basiert auf Beweisen für Fakten:

Gegeben : $E(a, b), E(b, c), E(c, d)$

$$T(x, y) \leftarrow E(x, y) \quad (1)$$

$$T(x, y) \leftarrow E(x, z) \wedge T(z, y) \quad (2)$$

- (a) $E(c, d)$ ist ein gegebener Fakt
- (b) $T(c, d)$ folgt aus (1) und (a)
- (c) $E(b, c)$ ist ein gegebener Fakt
- (d) $T(b, d)$ folgt aus (c), (b) und (2)
- (e) ...

Beweis-Theoretische Semantik von Datalog

Ein Fakt ist beweisbar, wenn es einen Beweis gibt, belegt durch einen Beweisbaum:

Definition

Ein *Beweisbaum* (proof tree) für einen Fakt A für eine Instanz I und ein Datalog Programm P ist ein markierter Baum (labeled tree) in dem

1. Jeder Knoten mit einem Fakt markiert ist
2. Jedes Blatt mit einem Fakt aus I markiert ist
3. Die Wurzel ist mit A markiert
4. Für jedes innere Blatt existiert eine Instanziierung $A_1 \leftarrow A_2, \dots, A_n$ einer Regel in P , so dass der Knoten mit A_1 markiert ist und seine Kinder entsprechend mit A_2, \dots, A_n

Beweis-Theoretische Semantik von Datalog

- Programme können als "Fabriken" verstanden werden, die alle beweisbaren Fakten produzieren (**bottom-up** von bekannten Fakten werden mittels der Regeln neue Fakten abgeleitet)
- Alternativ: **top-down** Evaluierung; ausgehend von einem zu beweisenden Fakt werden Lemmata gesucht, die für den Beweis gebraucht werden (\rightsquigarrow Resolution)

Beweis-Theoretische Semantik von Datalog

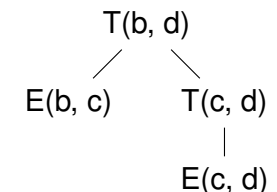
Basiert auf Beweisen für Fakten:

Gegeben : $E(a, b), E(b, c), E(c, d)$

$$T(x, y) \leftarrow E(x, y) \quad (1)$$

$$T(x, y) \leftarrow E(x, z) \wedge T(z, y) \quad (2)$$

- (a) $E(c, d)$ ist ein gegebener Fakt
- (b) $T(c, d)$ folgt aus (1) und (a)
- (c) $E(b, c)$ ist ein gegebener Fakt
- (d) $T(b, d)$ folgt aus (c), (b) und (2)
- (e) ...



Fixpunkt Semantik

Definiert die Semantik eines Datalog Programms als die Lösung einer Fixpunkt Gleichung

- ▶ Prozedurale Definition (Iterationen bis zum Fixpunkt)
- ▶ Gegeben eine Instanz I und ein Datalog Programm P , ein Fakt A ist eine *direkte Konsequenz* für wenn P und I , wenn
 1. A in I ist oder
 2. $A \leftarrow A_1, \dots, A_n$ eine instantiierte Regel in P ist, so dass $A_1, \dots, A_n \in I$
- ▶ Wir können dann einen “direkte Konsequenz”-Operator definieren, welcher von einer Instanz ausgehend alle direkten Konsequenzen berechnet
- ▶ Ähnlich wie die bottom-up beweis-theoretische Semantik, aber generiert erst kürzere Beweise, bevor längere generiert werden

Extensionale und Intensionale Prädikate

- ▶ Aus Sicht von Datenbanken (anders in der logischen Programmierung) unterscheidet man Fakten und Regeln
- ▶ Bei den Regeln unterscheiden wir **extensionale** von **intensionalen** Prädikaten
- ▶ *Extensionale* Prädikate (extensional database – edb) sind diejenigen, die nicht im Kopf von Regeln vorkommen (im Beispiel Relation E)
- ▶ *Intensionale* Prädikate (intensional database – idb) sind diejenigen, die in mindestens einem Kopf einer Regel vorkommen (im Beispiel Relation T)
- ▶ Die Semantik eines Datalog Programms kann dann verstanden werden als Mapping der gegebenen Instanzen über die edb Prädikate zu den Instanzen der idb Prädikate

Semantik von Regeln

- ▶ Mit anderen prädikatenlogischen Ansätzen kompatibel (z.B. Beschreibungslogik!)
- ▶ Konjunktionen in Regelköpfen, Disjunktionen in Regelrumpfen: nicht nötig (*Übung*)
- ▶ Andere (nicht-monotone) Interpretationen auch möglich
 - ▶ well-founded semantics
 - ▶ stable model semantics
 - ▶ answer set semantics
- ▶ Für Horn Regeln unterscheiden sich diese Interpretationen nicht (außer Negation von Atomen ist erlaubt)
- ▶ Production rules/prozedurale Regeln betrachten die Konsequenz einer Regel als eine Aktion “if-then do”
 \rightsquigarrow Hier nicht betrachtet

Datalog in der Praxis

Datalog in der Praxis:

- ▶ Verschiedene Implementierungen verfügbar
- ▶ Anpassungen für das Semantic Web: XSD-Typen, URIs (z.B. \rightarrow IRIS)

Erweiterungen von Datalog:

- ▶ *Disjunktives Datalog* erlaubt Disjunktionen in Köpfen
- ▶ Nichtmonotone Negation (keine prädikatenlogische Semantik)

Agenda

- ▶ Regeln
 - ▶ Llyod-Topor Transformation
- ▶ Datalog
 - ▶ Charakterisierungen der Semantik von Datalog Programmen
- ▶ Evaluierung von Datalog Programmen
 - ▶ Naive Evaluierung
 - ▶ Semi-naive Evaluierung
- ▶ Regeln für RDFS mittels Triple Prädikat
- ▶ Regeln für RDFS durch direkte Übersetzung

Evaluierung von Datalog Programmen

- ▶ Top-down oder bottom-up Evaluierung
- ▶ Direkte Evaluierung versus Kompilierung in ein effizienteres Programm
- ▶ Einfluss auf die logische Programmierung
- ▶ Hier:
 1. Naive bottom-up Evaluierung
 2. Semi-naive bottom-up Evaluierung

Reverse-Same-Generation

Gegebenes Datalog Programm:

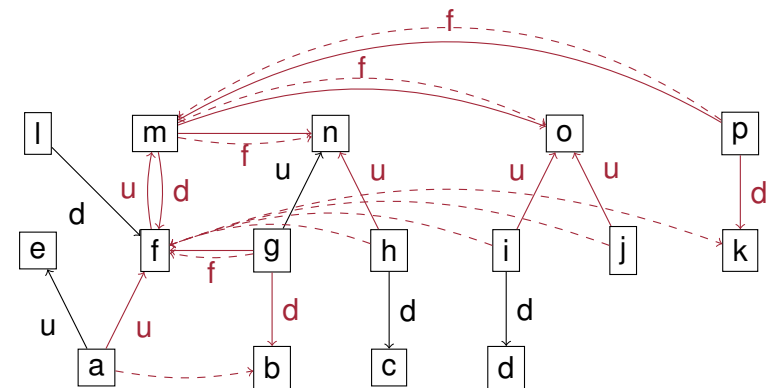
$$\text{rsg}(x, y) \leftarrow \text{flat}(x, y)$$

$$\text{rsg}(x, y) \leftarrow \text{up}(x, x_1), \text{rsg}(y_1, x_1), \text{down}(y_1, y)$$

Gegebene Daten:

up	<table style="border-collapse: collapse;"> <tr><td style="border-right: 1px solid black; padding: 2px 5px;">a</td><td style="padding: 2px 5px;">e</td></tr> <tr><td style="border-right: 1px solid black; padding: 2px 5px;">a</td><td style="padding: 2px 5px;">f</td></tr> <tr><td style="border-right: 1px solid black; padding: 2px 5px;">f</td><td style="padding: 2px 5px;">m</td></tr> <tr><td style="border-right: 1px solid black; padding: 2px 5px;">g</td><td style="padding: 2px 5px;">n</td></tr> <tr><td style="border-right: 1px solid black; padding: 2px 5px;">h</td><td style="padding: 2px 5px;">n</td></tr> <tr><td style="border-right: 1px solid black; padding: 2px 5px;">i</td><td style="padding: 2px 5px;">o</td></tr> <tr><td style="border-right: 1px solid black; padding: 2px 5px;">j</td><td style="padding: 2px 5px;">o</td></tr> </table>	a	e	a	f	f	m	g	n	h	n	i	o	j	o	<table style="border-collapse: collapse;"> <tr> <td style="border-right: 1px solid black; padding: 5px;">flat</td> <td style="padding: 5px;"> <table style="border-collapse: collapse;"> <tr><td style="border-right: 1px solid black; padding: 2px 5px;">g</td><td style="padding: 2px 5px;">f</td></tr> <tr><td style="border-right: 1px solid black; padding: 2px 5px;">m</td><td style="padding: 2px 5px;">n</td></tr> <tr><td style="border-right: 1px solid black; padding: 2px 5px;">m</td><td style="padding: 2px 5px;">o</td></tr> <tr><td style="border-right: 1px solid black; padding: 2px 5px;">p</td><td style="padding: 2px 5px;">m</td></tr> </table> </td> </tr> </table>	flat	<table style="border-collapse: collapse;"> <tr><td style="border-right: 1px solid black; padding: 2px 5px;">g</td><td style="padding: 2px 5px;">f</td></tr> <tr><td style="border-right: 1px solid black; padding: 2px 5px;">m</td><td style="padding: 2px 5px;">n</td></tr> <tr><td style="border-right: 1px solid black; padding: 2px 5px;">m</td><td style="padding: 2px 5px;">o</td></tr> <tr><td style="border-right: 1px solid black; padding: 2px 5px;">p</td><td style="padding: 2px 5px;">m</td></tr> </table>	g	f	m	n	m	o	p	m	<table style="border-collapse: collapse;"> <tr> <td style="border-right: 1px solid black; padding: 5px;">down</td> <td style="padding: 5px;"> <table style="border-collapse: collapse;"> <tr><td style="border-right: 1px solid black; padding: 2px 5px;">l</td><td style="padding: 2px 5px;">f</td></tr> <tr><td style="border-right: 1px solid black; padding: 2px 5px;">m</td><td style="padding: 2px 5px;">f</td></tr> <tr><td style="border-right: 1px solid black; padding: 2px 5px;">g</td><td style="padding: 2px 5px;">b</td></tr> <tr><td style="border-right: 1px solid black; padding: 2px 5px;">h</td><td style="padding: 2px 5px;">c</td></tr> <tr><td style="border-right: 1px solid black; padding: 2px 5px;">i</td><td style="padding: 2px 5px;">d</td></tr> <tr><td style="border-right: 1px solid black; padding: 2px 5px;">p</td><td style="padding: 2px 5px;">k</td></tr> </table> </td> </tr> </table>	down	<table style="border-collapse: collapse;"> <tr><td style="border-right: 1px solid black; padding: 2px 5px;">l</td><td style="padding: 2px 5px;">f</td></tr> <tr><td style="border-right: 1px solid black; padding: 2px 5px;">m</td><td style="padding: 2px 5px;">f</td></tr> <tr><td style="border-right: 1px solid black; padding: 2px 5px;">g</td><td style="padding: 2px 5px;">b</td></tr> <tr><td style="border-right: 1px solid black; padding: 2px 5px;">h</td><td style="padding: 2px 5px;">c</td></tr> <tr><td style="border-right: 1px solid black; padding: 2px 5px;">i</td><td style="padding: 2px 5px;">d</td></tr> <tr><td style="border-right: 1px solid black; padding: 2px 5px;">p</td><td style="padding: 2px 5px;">k</td></tr> </table>	l	f	m	f	g	b	h	c	i	d	p	k
a	e																																								
a	f																																								
f	m																																								
g	n																																								
h	n																																								
i	o																																								
j	o																																								
flat	<table style="border-collapse: collapse;"> <tr><td style="border-right: 1px solid black; padding: 2px 5px;">g</td><td style="padding: 2px 5px;">f</td></tr> <tr><td style="border-right: 1px solid black; padding: 2px 5px;">m</td><td style="padding: 2px 5px;">n</td></tr> <tr><td style="border-right: 1px solid black; padding: 2px 5px;">m</td><td style="padding: 2px 5px;">o</td></tr> <tr><td style="border-right: 1px solid black; padding: 2px 5px;">p</td><td style="padding: 2px 5px;">m</td></tr> </table>	g	f	m	n	m	o	p	m																																
g	f																																								
m	n																																								
m	o																																								
p	m																																								
down	<table style="border-collapse: collapse;"> <tr><td style="border-right: 1px solid black; padding: 2px 5px;">l</td><td style="padding: 2px 5px;">f</td></tr> <tr><td style="border-right: 1px solid black; padding: 2px 5px;">m</td><td style="padding: 2px 5px;">f</td></tr> <tr><td style="border-right: 1px solid black; padding: 2px 5px;">g</td><td style="padding: 2px 5px;">b</td></tr> <tr><td style="border-right: 1px solid black; padding: 2px 5px;">h</td><td style="padding: 2px 5px;">c</td></tr> <tr><td style="border-right: 1px solid black; padding: 2px 5px;">i</td><td style="padding: 2px 5px;">d</td></tr> <tr><td style="border-right: 1px solid black; padding: 2px 5px;">p</td><td style="padding: 2px 5px;">k</td></tr> </table>	l	f	m	f	g	b	h	c	i	d	p	k																												
l	f																																								
m	f																																								
g	b																																								
h	c																																								
i	d																																								
p	k																																								

Reverse-Same-Generation – Datenvisualisierung



$$\text{rsg}(x, y) \leftarrow \text{flat}(x, y)$$

$$\text{rsg}(x, y) \leftarrow \text{up}(x, x_1), \text{rsg}(y_1, x_1), \text{down}(y_1, y)$$

Agenda

- ▶ Regeln
 - ▶ Llyod-Topor Transformation
- ▶ Datalog
 - ▶ Charakterisierungen der Semantik von Datalog Programmen
- ▶ Evaluierung von Datalog Programmen
 - ▶ Naive Evaluierung
 - ▶ Semi-naive Evaluierung
- ▶ Regeln für RDFS mittels Triple Prädikat
- ▶ Regeln für RDFS durch direkte Übersetzung

Naiver Algorithmus zur Evaluierung von Datalog Programmen

- ▶ Redundante Berechnungen (alle Elemente des vorigen Levels werden berücksichtigt)
- ▶ Auf jedem Level werden alle Elemente des vorigen Levels erneut berechnet
- ▶ Monotonie (rsg wird immer mehr erweitert)

Naiver Algorithmus zur Berechnung von rsg

$$rsg(x, y) \leftarrow flat(x, y)$$

$$rsg(x, y) \leftarrow up(x, x_1), rsg(y_1, x_1), down(y_1, y)$$

Algorithm 1 RSG

$rsg := \emptyset$

repeat

$rsg := rsg \cup flat \cup \pi_{16}(\sigma_{2=4}(\sigma_{3=5}(up \times rsg \times down)))$

until Fixpunkt erreicht

$$rsg^{i+1} := rsg^i \cup flat \cup \pi_{16}(\sigma_{2=4}(\sigma_{3=5}(up \times rsg \times down)))$$

Level 0: \emptyset

Level 1: $\{(g, f), (m, n), (m, o), (p, m)\}$

Level 2: $\{\text{Level 1}\} \cup \{(a, b), (h, f), (i, f), (j, f), (f, k)\}$

Level 3: $\{\text{Level 2}\} \cup \{(a, c), (a, d)\}$

Level 4: $\{\text{Level 3}\}$

Agenda

- ▶ Regeln
 - ▶ Llyod-Topor Transformation
- ▶ Datalog
 - ▶ Charakterisierungen der Semantik von Datalog Programmen
- ▶ Evaluierung von Datalog Programmen
 - ▶ Naive Evaluierung
 - ▶ Semi-naive Evaluierung
- ▶ Regeln für RDFS mittels Triple Prädikat
- ▶ Regeln für RDFS durch direkte Übersetzung

Semi-Naiver Algorithmus zur Berechnung von rsg

Fokus auf den im vorigen Level neu berechneten Fakten

Algorithm 2 RSG'

$$\Delta_{rsg}^1(x, y) := flat(x, y)$$

$$\Delta_{rsg}^{i+1}(x, y) := up(x, x_1), \Delta_{rsg}^i(y_1, x_1), down(y_1, y)$$

- ▶ Nicht rekursiv
- ▶ Kein Datalog Programm (infinite Menge an Regeln)
- ▶ Für jede Eingabe I und δ_{rsg}^i die neu berechneten Instanzen in Level i,

$$rsg^{i+1} - rsg^i \subseteq \delta_{rsg}^{i+1} \subseteq rsg^{i+1}$$

- ▶ $RSG(I)(rsg) = \bigcup_{1 \leq i} (\delta_{rsg}^i)$
- ▶ Weniger Redundanz

Eine Verbesserung

Aber: $\delta_{rsg}^{i+1} \neq rsg^{i+1} - rsg^i$

Z.B.: $(g, f) \in \delta_{rsg}^2, (g, f) \notin rsg^2 - rsg^1$

$\rightsquigarrow rsg(g, f) \in rsg^1$, weil $flat(g, f)$,

$\rightsquigarrow rsg(g, f) \in \delta_{rsg}^2$, weil $up(g, n), rsg(m, f), down(m, f)$

- ▶ Idee: Nutzung von $rsg^i - rsg^{i-1}$ an Stelle von Δ_{rsg}^i in der zweiten "Regel" von RSG'

Algorithm 3 RSG''

$$\Delta_{rsg}^1(x, y) := flat(x, y)$$

$$rsg^1 := \Delta_{rsg}^1$$

$$tmp_{rsg}^{i+1}(x, y) := up(x, x_1), \Delta_{rsg}^i(y_1, x_1), down(y_1, y)$$

$$\Delta_{rsg}^{i+1}(x, y) := tmp_{rsg}^{i+1} - rsg^i$$

$$rsg^{i+1} := rsg^i \cup \Delta_{rsg}^{i+1}$$

Agenda

- ▶ Regeln
 - ▶ Llyod-Topor Transformation
- ▶ Datalog
 - ▶ Charakterisierungen der Semantik von Datalog Programmen
- ▶ Evaluierung von Datalog Programmen
 - ▶ Naive Evaluierung
 - ▶ Semi-naive Evaluierung
- ▶ Regeln für RDFS mittels Triple Prädikat
- ▶ Regeln für RDFS durch direkte Übersetzung

Datalog Regeln für RDFS (ohne Datentypen & Literale)

Problem: Keine strikte Trennung von Daten und Schema (Prädikaten)

$$\frac{a \text{ rdfs:domain } x \ . \ u \ a \ y \ .}{u \text{ rdf:type } x \ .} \text{ rdfs2}$$

$$\text{rdf:type}(u, x) \leftarrow \text{rdfs:domain}(a, x) \wedge a(u, y)$$

- ▶ Lösung: Verwendung eines Triple Prädikats

Agenda

- ▶ Regeln
 - ▶ Llyod-Topor Transformation
- ▶ Datalog
 - ▶ Charakterisierungen der Semantik von Datalog Programmen
- ▶ Evaluierung von Datalog Programmen
 - ▶ Naive Evaluierung
 - ▶ Semi-naive Evaluierung
- ▶ Regeln für RDFS mittels Triple Prädikat
- ▶ Regeln für RDFS durch direkte Übersetzung

Datalog Regeln für RDFS (ohne Datentypen & Literale)

$$\frac{a \text{ rdfs:domain } x \ . \ u \ a \ y \ .}{u \text{ rdf:type } x \ .} \text{ rdfs2}$$

$$\text{Triple}(u, \text{rdf:type}, x) \leftarrow \text{Triple}(a, \text{rdfs:domain}, x) \wedge \text{Triple}(u, a, y)$$

- ▶ Nutzung eines Prädikats erlaubt weniger Optimierungen
- ▶ Alle (neuen) Triple kommen potenziell für jede Regel in Frage
- ▶ Regeln ändern sich wann immer sich die Daten ändern, da keine Trennung von Schema und Daten

Datalog Regeln für RDFS (ohne Datentypen & Literale)

- ▶ Lösung 2: Einführung spezieller Prädikate

$$\frac{a \text{ rdfs:domain } x \ . \ u \ a \ y \ .}{u \text{ rdf:type } x \ .} \text{ rdfs2}$$

$$\text{type}(u, x) \leftarrow \text{domain}(a, x) \wedge \text{rel}(u, a, y)$$

Axiomatische Triple als Fakten

```

type(rdf:type, rdf:Property)
type(rdf:subject, rdf:Property)
type(rdf:predicate, rdf:Property)
type(rdf:object, rdf:Property)
type(rdf:first, rdf:Property)
type(rdf:rest, rdf:Property)
type(rdf:value, rdf:Property)
type(rdf:_1, rdf:Property)
type(rdf:_2, rdf:Property)
type(..., rdf:Property)
type(rdf:nil, rdf:Property)
... (plus RDFS axiomatische Triple)

```

↔ Nur benötigt für `rdf:_i` die in den Graphen G_1 und G_2 vorkommen, bei der Entscheidung $G_1 \models G_2$

RDF-Folgerungsregeln (ohne Datentypen & Literale)

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

$\rightsquigarrow \text{type}(a, \text{rdf:Property}) \leftarrow \text{rel}(u, a, y)$

$$\frac{a \ \text{rdfs:domain} \ x \ . \ u \ a \ y \ .}{u \ \text{rdf:type} \ x \ .} \text{ rdfs2}$$

$\rightsquigarrow \text{type}(u, x) \leftarrow \text{domain}(a, x) \wedge \text{rel}(u, a, y)$

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

$\rightsquigarrow \text{type}(v, x) \leftarrow \text{range}(a, x) \wedge \text{rel}(u, a, v)$

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

$\rightsquigarrow \text{type}(u, \text{rdfs:Resource}) \leftarrow \text{rel}(u, a, x)$

a, b IRIs x, y IRI, blank node oder Literal
u, v IRI oder blank node l Literal _:n blank nodes

RDF-Folgerungsregeln (ohne Datentypen & Literale)

$$\frac{u \ a \ v \ .}{v \ \text{rdf:type} \ \text{rdfs:Resource} \ .} \text{ rdfs4b}$$

$\rightsquigarrow \text{type}(v, \text{rdfs:Resource}) \leftarrow \text{rel}(u, a, v)$

$$\frac{u \ \text{rdfs:subPropertyOf} \ v \ . \ v \ \text{rdfs:subPropertyOf} \ x \ .}{u \ \text{rdfs:subPropertyOf} \ x \ .} \text{ rdfs5}$$

$\rightsquigarrow \text{subPropertyOf}(u, x) \leftarrow \text{subPropertyOf}(u, v) \wedge \text{subPropertyOf}(v, x)$

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

$\rightsquigarrow \text{subPropertyOf}(u, u) \leftarrow \text{type}(u, \text{rdf:Property})$

$$\frac{a \ \text{rdfs:subPropertyOf} \ b \ . \ u \ a \ y \ .}{u \ b \ y \ .} \text{ rdfs7}$$

$\rightsquigarrow \text{rel}(u, b, y) \leftarrow \text{subPropertyOf}(a, b) \wedge \text{rel}(u, a, y)$

a, b IRIs x, y IRI, blank node oder Literal
u, v IRI oder blank node l Literal _:n blank nodes

RDF-Folgerungsregeln (ohne Datentypen & Literale)

$$\frac{u \ \text{rdf:type} \ \text{rdfs:Class} \ .}{u \ \text{rdf:type} \ \text{rdfs:Resource} \ .} \text{ rdfs8}$$

$\rightsquigarrow \text{type}(u, \text{rdfs:Resource}) \leftarrow \text{type}(u, \text{rdfs:Class})$

$$\frac{u \ \text{rdfs:subClassOf} \ x \ . \ v \ \text{rdf:type} \ u \ .}{v \ \text{rdf:type} \ x \ .} \text{ rdfs9}$$

$\rightsquigarrow \text{type}(v, x) \leftarrow \text{subClassOf}(u, x) \wedge \text{type}(v, u)$

$$\frac{u \ \text{rdf:type} \ \text{rdfs:Class} \ .}{u \ \text{rdfs:subClassOf} \ u \ .} \text{ rdfs10}$$

$\rightsquigarrow \text{subClassOf}(u, u) \leftarrow \text{type}(u, \text{rdfs:Class})$

$$\frac{u \ \text{rdfs:subClassOf} \ v \ . \ v \ \text{rdfs:subClassOf} \ x \ .}{u \ \text{rdfs:subClassOf} \ x \ .} \text{ rdfs11}$$

$\rightsquigarrow \text{subClassOf}(u, x) \leftarrow \text{subClassOf}(u, v) \wedge \text{subClassOf}(v, x)$

a, b IRIs x, y IRI, blank node oder Literal
u, v IRI oder blank node l Literal _:n blank nodes

RDF-Folgerungsregeln (ohne Datentypen & Literale)

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

$\rightsquigarrow \text{subPropertyOf}(u, \text{rdfs:member}) \leftarrow \text{type}(u, \text{rdfs:ContainerMembershipProperty})$

a, b IRIs x, y IRI, blank node oder Literal
u, v IRI oder blank node l Literal _:n blank nodes

Agenda

- ▶ Regeln
 - ▶ Llyod-Topor Transformation
- ▶ Datalog
 - ▶ Charakterisierungen der Semantik von Datalog Programmen
- ▶ Evaluierung von Datalog Programmen
 - ▶ Naive Evaluierung
 - ▶ Semi-naive Evaluierung
- ▶ Regeln für RDFS mittels Triple Prädikat
- ▶ Regeln für RDFS durch direkte Übersetzung