ulm university universität ulm

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 Implemetierung	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):
   - `mann(X) <- person(X) AND NOT frau(X)`
   - Approximation logischer Semantik mit operationalen Aspekten, Built-ins möglich
   - häufig Closed World
   - Semi-deklarativ

Prädikatenlogik als Regelsprache

- Regeln als Implikationsformeln der Prädikatenlogik:

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

  $\rightsquigarrow$ Semantisch äquivalent zu Disjunktion:

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

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

Regelbeispiel

Beispiel:

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

- 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

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

Lloyd-Topor Transformation

- Mehrere Atome im Kopf werden meist als Konjunktion verstanden

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

äquivalent zu

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

- Eine derartige Umformung bezeichnet man als Lloyd-Topor Transformation

Disjuktive Regeln

- Regel können auch Disjunktionen enthalten

⇝ Mehrere Atome im Kopf werden als Alternativen verstanden:

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

äquivalent zu

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

äquivalent zu

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

⇝ Hier nicht betrachtet

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
  - $\leftarrow \text{Mann}(x) \wedge \text{Frau}(x)$

  $\rightsquigarrow$ “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
  - $\text{Frau}(\text{gisela})$

Arten von Regeln

Regeln können auch Funktionssymbole enthalten:

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

- $\rightsquigarrow$ Generiert dynamisch neue Elemente
- $\rightsquigarrow$ Hier nicht betrachtet
- $\rightsquigarrow$ 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 *Daten*mengen, 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 nennte 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

$\rightsquigarrow$ Interpretation von T muss minimal sein

Modell-Theoretische Semantik von Datalog

Prinzipiell repräsentiert eine Datalog Regel

$$\rho\colon R_1(u_1) \leftarrow R_2(u_2), \ldots, R_n(u_n)$$

den logischen Satz

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

- $x_1, \ldots, x_n$ 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

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

  in der $R_2(\nu(u_2)), \ldots, R_n(\nu(u_n))$ 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:

$$\begin{aligned} &\textit{Gegeben}: \; E(a,b), E(b,c), E(c,d) \\ &T(x,y) \leftarrow E(x,y) && (1) \\ &T(x,y) \leftarrow E(x,z) \wedge T(z,y) && (2) \end{aligned}$$

(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

- Programme können alls “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

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 $A_1 \leftarrow A_2, \ldots, A_n$ einer Regel in P, so dass der Knoten mit A_1 markiert ist und seine Kinder entsprechend mit $A_2, \ldots, A_n$

Beweis-Theoretische Semantik von Datalog

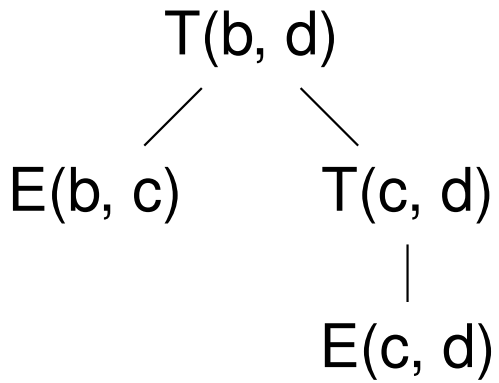
Basiert auf Beweisen für Fakten:

$$\textit{Gegeben}: \; E(a,b), E(b,c), E(c,d)$$

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

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

(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. $A \leftarrow A_1, \ldots, A_n$ eine instantiierte Regel in P ist, so dass $A_1, \ldots, A_n \in I$
- 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

Semantik von Regeln

- Mit anderen prädikatenlogischen Ansätzen kompatibel (z.B. Beschreibungslogik!)
- Konjunktionen in Regelköpfen, Disjunktionen in Regelrümpfen: 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

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 dijenigen, die nicht im Kopf von Regeln vorkommen (im Beispiel Relation E)
- *Intensionale* Prädikate (intensional database – idb) sind dijenigen, 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

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

a	e
a	f
f	m
g	n
h	n
i	o
j	o

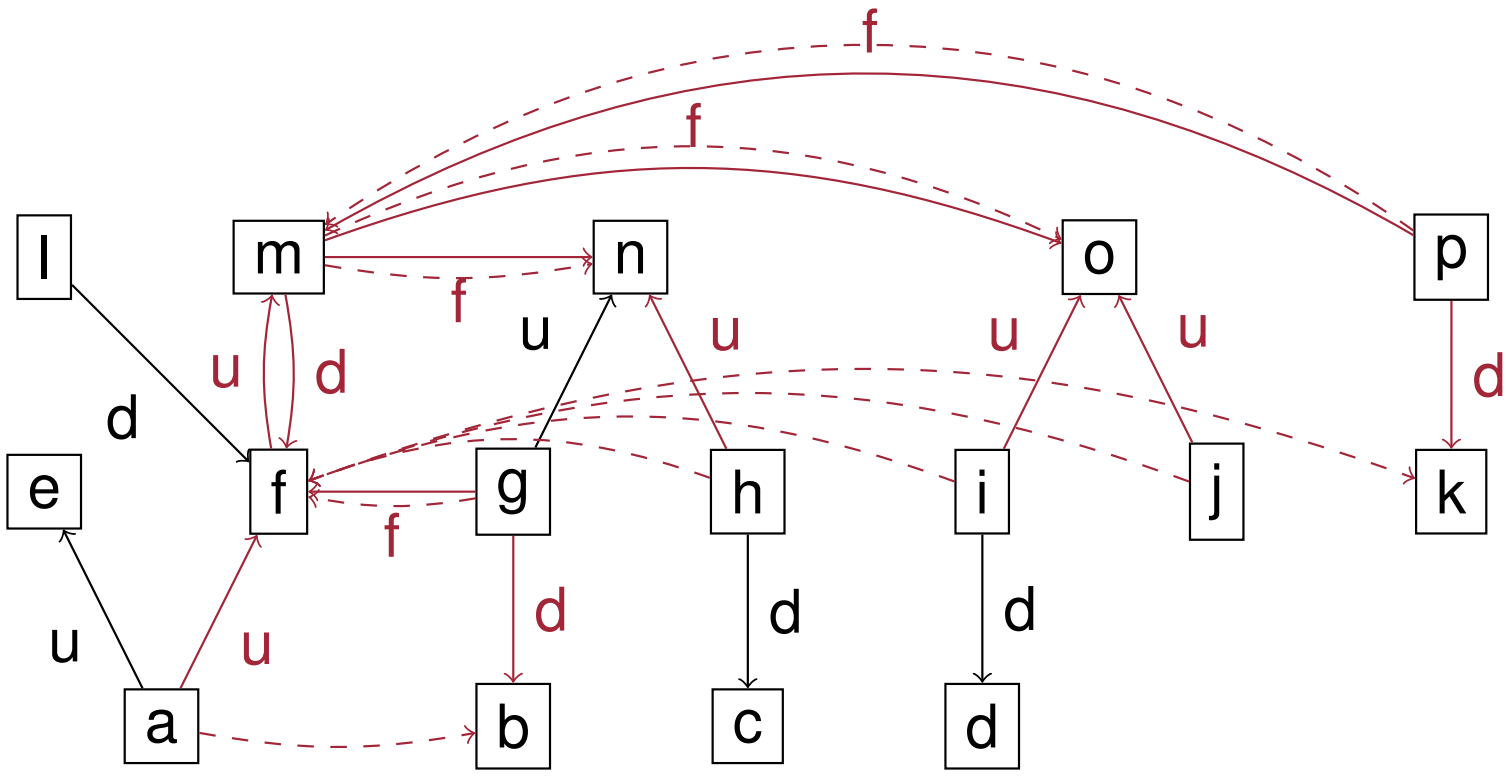
flat

g	f
m	n
m	o
p	m

down

l	f
m	f
g	b
h	c
i	d
p	k

Reverse-Same-Generation – Datenvisualisierung



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

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

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)

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'

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

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

$$rsg^{i+1} - rsg^i \subseteq \delta^{i+1}_{rsg} \subseteq rsg^{i+1}$$

- RSG(I)(rsg) = $\cup_{1 \leq i}(\delta^i_{rsg})$
- Weniger Redundanz

Eine Verbesserung

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

Z.B.: $(g,f) \in \delta^2_{rsg}, (g,f) \notin rsg^2 - rsg^1$
$\rightsquigarrow rsg(g,f) \in rsg^1$, weil $flat(g,f)$,
$\rightsquigarrow rsg(g,f) \in \delta^2_{rsg}$, weil $up(g,n), rsg(m,f), down(m,f)$

- Idee: Nutzung von $rsg^i - rsg^{i-1}$ an Stelle von Δ^i_{rsg} in der zweiten "Regel" von RSG′

Algorithm 3 RSG''

$\Delta^1_{rsg}(x,y) := flat(x,y)$
$rsg^1 := \Delta^1_{rsg}$
$tmp^{i+1}_{rsg}(x,y) := up(x,x_1), \Delta^i_{rsg}(y_1,x_1), down(y_1,y)$
$\Delta^{i+1}_{rsg}(x,y) := tmp^{i+1}_{rsg} - rsg^i$
$rsg^{i+1} := rsg^i \cup \Delta^{i+1}_{rsg}$

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)

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

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

- 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)

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

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

- 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

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

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

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)

$\rightsquigarrow$ 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)

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

$\rightsquigarrow$ *type*(a, rdf:Property) $\leftarrow$ *rel*(u, a, y)

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

$\rightsquigarrow$ *type*(u, x) $\leftarrow$ *domain*(a, x) $\wedge$ *rel*(u, a, y)

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

$\rightsquigarrow$ *type*(v, x) $\leftarrow$ *range*(a, x) $\wedge$ *rel*(u, a, v)

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

$\rightsquigarrow$ *type*(u, rdfs:Resource) $\leftarrow$ *rel*(u, a, x)

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

RDF-Folgerungsregeln (ohne Datentypen & Literale)

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

$\rightsquigarrow$ *type*(v, rdfs:Resource) $\leftarrow$ *rel*(u, a, v)

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

$\rightsquigarrow$ *subPropertyOf*(u, x) $\leftarrow$ *subPropertyOf*(u, v) $\wedge$ *subPropertyOf*(v, x)

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

$\rightsquigarrow$ *subPropertyOf*(u, u) $\leftarrow$ *type*(u, rdf:Property)

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

$\rightsquigarrow$ *rel*(u, b, y) $\leftarrow$ *subPropertyOf*(a, b) $\wedge$ *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)

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

$\rightsquigarrow$ *type*(u, rdfs:Resource) $\leftarrow$ *type*(u, rdfs:Class)

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

$\rightsquigarrow$ *type*(v, x) $\leftarrow$ *subClassOf*(u, x) $\wedge$ *type*(v, x)

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

$\rightsquigarrow$ *subClassOf*(u, u) $\leftarrow$ *type*(u, rdfs:Class)

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

$\rightsquigarrow$ *subClassOf*(u, x) $\leftarrow$ *subClassOf*(u, v) $\wedge$ *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)

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

$\rightsquigarrow$ *subPropertyOf*(u, rdfs:member) $\leftarrow$ *type*(u, rdfs:ContainerMembershipProperty)

a, b IRIs
u, v IRI oder blank node
x, y IRI, blank node oder Literal
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