TECHNISCHE UNIVERSITÄT DRESDEN

FOUNDATIONS OF SEMANTIC WEB TECHNOLOGIES

RDFS Rule-based Reasoning

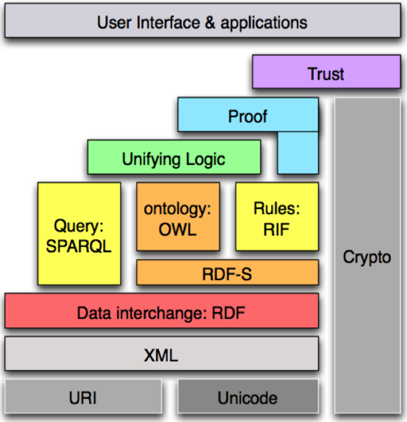
Sebastian Rudolph

Dresden, 16 April 2013

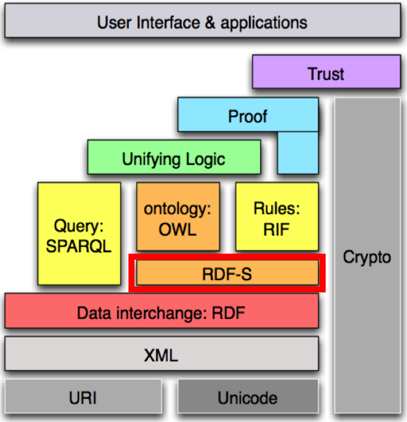
DRESDEN concept
Exzellenz aus Wissenschaft und Kultur

Content

RDF Schema


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

RDF Schema



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

Agenda

- Rules
  - Llyod-Topor Transformation
- Datalog
  - Characterizations of Datalog Program Semantics
- Evaluating Datalog Programs
  - Naïve Evaluation
  - Semi-naïve Evaluation
- Rules for RDFS via a Triple Predicate
- Rules for RDFS via Direct Translation

Agenda

- Rules
  - Llyod-Topor Transformation
- Datalog
  - Characterizations of Datalog Program Semantics
- Evaluating Datalog Programs
  - Naïve Evaluation
  - Semi-naïve Evaluation
- Rules for RDFS via a Triple Predicate
- Rules for RDFS via Direct Translation

Constituents of Rules

- basic elements of rules are atoms
  - ground atoms without free variables
  - non-ground atoms with free variables

What are Rules?

1. logic rules (fragments of predicate logic):
   - $F \rightarrow G$ equivalent to $\neg F \vee G$
   - logical extension of knowledge base $\rightsquigarrow$ static
   - open world
   - declarative (describing)

What are Rules?

1. logic rules (fragments of predicate logic):
   - $F \rightarrow G$ equivalent to $\neg F \vee G$
   - logical extension of knowledge base $\rightsquigarrow$ static
   - open world
   - declarative (describing)
2. procedural rules (e.g. production rules):
   - "If X then Y else Z"
   - executable commands $\rightsquigarrow$ dynamic
   - operational (meaning = effect caused when executed)

What are Rules?

1. logic rules (fragments of predicate logic):
   - $F \rightarrow G$ equivalent to $\neg F \vee G$
   - logical extension of knowledge base $\rightsquigarrow$ static
   - open world
   - declarative (describing)
2. procedural rules (e.g. production rules):
   - "If X then Y else Z"
   - executable commands $\rightsquigarrow$ dynamic
   - operational (meaning = effect caused when executed)
3. logic programming et al. (e.g. PROLOG, F-Logic):
   - `man(X) <- person(X) AND NOT woman(X)`
   - approximation of logical semantics with operational aspects, built-ins are possible
   - often closed-world
   - semi-declarative

Predicate Logic as a Rule Language

- rules as implication formulae in predicate logic:

$$\underbrace{H}_{\text{head}} \leftarrow \underbrace{A_1 \wedge A_2 \wedge \ldots \wedge A_n}_{\text{body}}$$

Predicate Logic as a Rule Language

- rules as implication formulae in predicate logic:

$$\underbrace{H}_{\text{head}} \leftarrow \underbrace{A_1 \wedge A_2 \wedge \ldots \wedge A_n}_{\text{body}}$$

$\rightsquigarrow$ semantically equivalent to disjunction:

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

- implications often written from right to left($\leftarrow$ or $:-$)
- constants, variables and function symbols allowed
- quantifiers for variables are often omitted:
  free variables are often understood as universally quantified
  (i.e. rule is valid for all variable assignments)

Rules – Example

Example:

$$\texttt{hasUncle}(x,z) \leftarrow \texttt{hasParent}(x,y) \land \texttt{hasBrother}(y,z)$$

- we use short names (hasUncle) instead of IRIs like http://example.org/Example#hasUncle
- we use x,y,z for variables

Agenda

- Rules
  - Llyod-Topor Transformation
- Datalog
  - Characterizations of Datalog Program Semantics
- Evaluating Datalog Programs
  - Naïve Evaluation
  - Semi-naïve Evaluation
- Rules for RDFS via a Triple Predicate
- Rules for RDFS via Direct Translation

Lloyd-Topor Transformation

- multiple heads in atoms are usually understood as conjunction

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

equivalent to

$$\begin{aligned} H_1 &\leftarrow A_1, A_2, \ldots, A_n \\ H_2 &\leftarrow A_1, A_2, \ldots, A_n \\ &\ldots \\ H_m &\leftarrow A_1, A_2, \ldots, A_n \end{aligned}$$

- such a rewriting is also referred to as Lloyd-Topor transformation

Disjunctive Rules

- some rule formalisms allow for disjunction

⇝ several atoms in the head are conceived as alternatives:

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

equivalent to

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

equivalent to

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

⇝ (not considered here)

Kinds of Rules

names for "rules" in predicate logic:

- clause: disjunction of atomic and negated atomic propositions
  - $\mathrm{Woman}(x) \vee \mathrm{Man}(x) \leftarrow \mathrm{Person}(x)$

Kinds of Rules

names for “rules” in predicate logic:

- clause: disjunction of atomic and negated atomic propositions
  - $\mathsf{Woman}(x) \vee \mathsf{Man}(x) \leftarrow \mathsf{Person}(x)$
- Horn clause: clause with at most one non-negated atom
  - $\leftarrow \mathsf{Man}(x) \wedge \mathsf{Woman}(x)$

  $\leadsto$ “integrity constraints”

Kinds of Rules

names for "rules" in predicate logic:

- clause: disjunction of atomic and negated atomic propositions
  - $\text{Woman}(x) \vee \text{Man}(x) \leftarrow \text{Person}(x)$
- Horn clause: clause with at most one non-negated atom
  - $\leftarrow \text{Man}(x) \wedge \text{Woman}(x)$

  $\rightsquigarrow$ "integrity constraints"
- definite clause: Horn clause with exactly one non-negated atom
  - $\text{Father}(x) \leftarrow \text{Man}(x) \wedge \text{hasChild}(x, y)$

Kinds of Rules

names for "rules" in predicate logic:

- clause: disjunction of atomic and negated atomic propositions
  - $\text{Woman}(x) \vee \text{Man}(x) \ \leftarrow\ \text{Person}(x)$
- Horn clause: clause with at most one non-negated atom
  - $\leftarrow\ \text{Man}(x) \wedge \text{Woman}(x)$

  $\rightsquigarrow$ "integrity constraints"
- definite clause: Horn clause with exactly one non-negated atom
  - $\text{Father}(x) \ \leftarrow\ \text{Man}(x) \wedge \text{hasChild}(x, y)$
- fact: clause containing just one non-negated atom
  - $\text{Woman}(\text{gisela})$

Kinds of Rules

Rules may also contain function symbols:

$$\text{hasUncle}(x,y) \leftarrow \text{hasBrother}(\text{mother}(x),y)$$
$$\text{hasFather}(x,\text{father}(x)) \leftarrow \text{Person}(x)$$

$\leadsto$ new elements are dynamically generated
$\leadsto$ not considered here
$\leadsto$ see logic programming

Agenda

- Rules
  - Llyod-Topor Transformation
- Datalog
  - Characterizations of Datalog Program Semantics
- Evaluating Datalog Programs
  - Naïve Evaluation
  - Semi-naïve Evaluation
- Rules for RDFS via a Triple Predicate
- Rules for RDFS via Direct Translation

TECHNISCHE UNIVERSITÄT DRESDEN

Datalog

Horn rules without function symbols $\rightsquigarrow$ Datalog rules

- logical rule language, originally basis of deductive databases
- knowledge bases ("programs") consisting of Horn clauses without function symbols
- decidable
- efficient for big datasets, combined complexity ExpTime
- a lot of research done in the 1980s

Datalog as Extension of the Relation Calculus

Datalog can be conceived as Extension of the relation calculus by recursion

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

$\rightsquigarrow$ computes the transitive closure (T) of the binary relation E, (e.g. if E contains the edges of a graph)

- a set of (ground) facts is also called an instance

Agenda

- Rules
  - Llyod-Topor Transformation
- Datalog
  - Characterizations of Datalog Program Semantics
- Evaluating Datalog Programs
  - Naïve Evaluation
  - Semi-naïve Evaluation
- Rules for RDFS via a Triple Predicate
- Rules for RDFS via Direct Translation

Semantics of Datalog

three different but equivalent ways to define the semantics:

- model-theoretically
- proof-theoretically
- via fixpoints

Model-theoretic Semantics of Datalog

rules are seen as logical sentences:

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

- not sufficient to uniquely determine a solution

$\rightsquigarrow$ interpretation of T has to be minimal

Model-theoretic Semantics of Datalog

in principle, a Datalog rule

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

represents the FOL sentence

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

- $x_1, \ldots, x_n$ are the rule's variables and $\leftarrow$ is logical implication
- an instance I satisfies ρ, written $I \models \rho$, if and only if for every instantiation

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

we find $R_1(\nu(u_1))$ satisfied whenever $R_2(\nu(u_2)), \ldots, R_n(\nu(u_n))$ are satisfied

Model-theoretic Semantics of Datalog

- an instance I is a model of a Datalog program P, if I satisfies every rule in P (seen as a FOL formula)
- the semantics of P for the input I is the minimal model that contains I (if it exists)
- Question: does such a model always exist?
- If so, how can we construct it?

Proof-theoretic Semantics of Datalog

based on proofs for facts:

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

(a) E(c, d) is a given fact
(b) T(c, d) follows from (1) and (a)
(c) E(b, c) is a given fact
(d) T(b, d) follows from (c), (b) and (2)
(e) . . .

Proof-theoretic Semantics of Datalog

- programs can be seen as "factories" that produce all provable facts (deriving new facts from known ones in a bottom-up way by applying rules)
- alternative: top-down evaluation; starting from a to-be-proven fact, one looks for lemmata needed for the proof ($\rightsquigarrow$ Resolution)

Proof-theoretic Semantics of Datalog

a fact is provable, if it has a proof, represented by a proof-tree:

Definition

A proof tree for a fact A for an instance I and a Datalog program P is a labeled tree in which

1. every node is labeled with a fact
2. every leaf is labeled with a fact from I
3. the root is labeled with A
4. for each internal leaf there exists an instantiation $A_1 \leftarrow A_2, \ldots, A_n$ of a rule in P, such that the node is labeled with A_1 and its children with $A_2, \ldots, A_n$

Proof-theoretic Semantics of Datalog

based on proofs for facts:

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

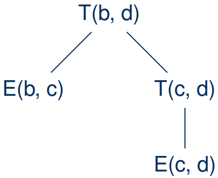
(a) E(c, d) is a given fact

(b) T(c, d) follows from (1) and (a)

(c) E(b, c) is a given fact

(d) T(b, d) follows from (c), (b) and (2)

(e) . . .



Fixpoint Semantics

defines the semantics of a Datalog program as the solution of a fixpoint equation

- procedural definition (iteration until fixpoint reached)
- given an instance I and a Datalog program P, we call a fact A a direct consequence for P and I, if
  1. A is contained in I or
  2. $A \leftarrow A_1, \ldots, A_n$ is the instance of a rule from P, such that $A_1, \ldots, A_n \in I$
- then we can define a "direct consequence"-operator that computes, starting from an instance, all direct consequences
- similar to the bottom-up proof-theoretic semantics, but shorter proofs are always generated earlier than longer ones

Semantics of Rules

- compatible with other approaches that are based on FOL (e.g. description logics)
- conjunctions in rule heads and disjunction in bodies unnecessary
- other (non-monotonic) semantics definitions possible
  - well-founded semantics
  - stable model semantics
  - answer set semantics
- for Horn rules, these definitions do not differ
- production rules/procedural rules conceive the consequence of a rule as an action "If-then do"
  $\rightsquigarrow$ not considered here

Extensional and Intensional Predicates

- from the database perspective (and opposed to logic programming) one distinguishes facts and rules
- within rules, we distinguish extensional and intensional predicates
- extensional predicates (also: extensional database – edb) are those not occurring in rule heads (in our example: relation E)
- intensional predicates (also: intensional database – idb) are those occurring in at least one head of a rule (in our example: relation T)
- semantics of a datalog program can be understood as a mapping of given instances over edb predicates to instances of idb predicates

Datalog in Practice

Datalog in Practice:

- several implementations available
- some adaptations for Semantic Web: XSD types, URIs (e.g. $\rightarrow$ IRIS)

Extensions of Datalog:

- disjunctive Datalog allows for disjunctions in rule heads
- non-monotonic negation (no FOL semantics)

Agenda

- Rules
  - Llyod-Topor Transformation
- Datalog
  - Characterizations of Datalog Program Semantics
- Evaluating Datalog Programs
  - Naïve Evaluation
  - Semi-naïve Evaluation
- Rules for RDFS via a Triple Predicate
- Rules for RDFS via Direct Translation

Evaluating Datalog Programs

- top-down or bottom-up evaluation
- direct evaluation versus compilation into an efficient program
- here:
  1. Naïve bottom-up Evaluierung
  2. Semi-naïve bottom-up Evaluierung

Reverse-Same-Generation

given Datalog programm:

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

given data:

up

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

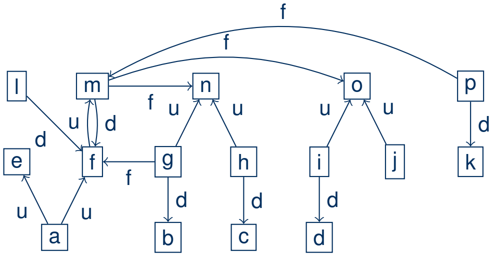
flat

g	f
m	n
m	o
p	m

down

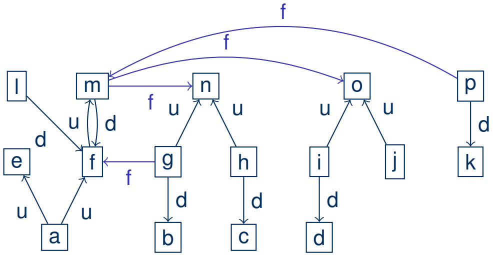
l	f
m	f
g	b
h	c
i	d
p	k

Reverse-Same-Generation – Visualization



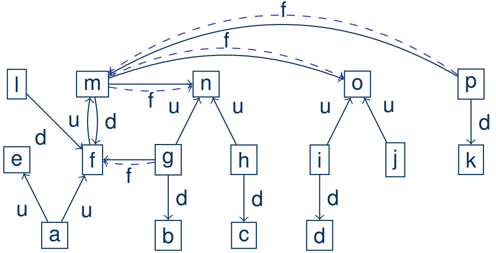
$$rsg(x,y) \leftarrow flat(x,y)$$
$$rsg(x,y) \leftarrow up(x,x_1), rsg(y_1,x_1), down(y_1,y)$$

Reverse-Same-Generation – Visualization



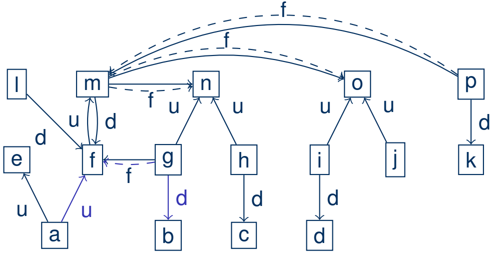
$$rsg(x, y) \leftarrow flat(x, y)$$
$$rsg(x, y) \leftarrow up(x, x_1), rsg(y_1, x_1), down(y_1, y)$$

Reverse-Same-Generation – Visualization



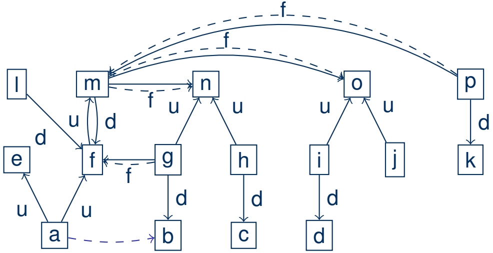
$$rsg(x, y) \leftarrow flat(x, y)$$
$$rsg(x, y) \leftarrow up(x, x_1), rsg(y_1, x_1), down(y_1, y)$$

Reverse-Same-Generation – Visualization



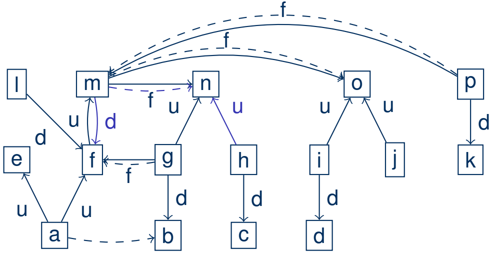
$$rsg(x,y) \leftarrow \mathit{flat}(x,y)$$
$$rsg(x,y) \leftarrow up(x,x_1), rsg(y_1,x_1), down(y_1,y)$$

Reverse-Same-Generation – Visualization



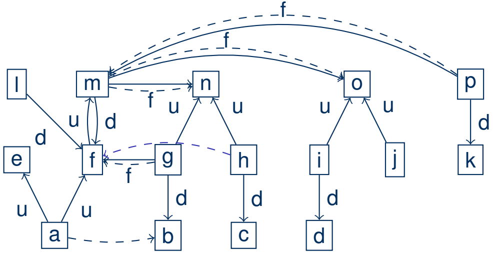
$$rsg(x, y) \leftarrow flat(x, y)$$
$$rsg(x, y) \leftarrow up(x, x_1), rsg(y_1, x_1), down(y_1, y)$$

Reverse-Same-Generation – Visualization



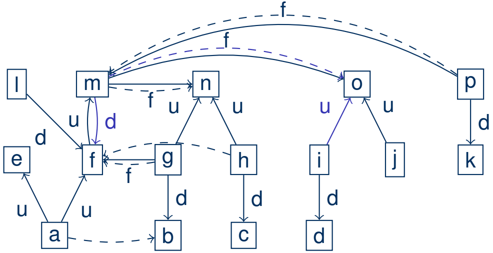
$$rsg(x, y) \leftarrow flat(x, y)$$
$$rsg(x, y) \leftarrow up(x, x_1), rsg(y_1, x_1), down(y_1, y)$$

Reverse-Same-Generation – Visualization



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

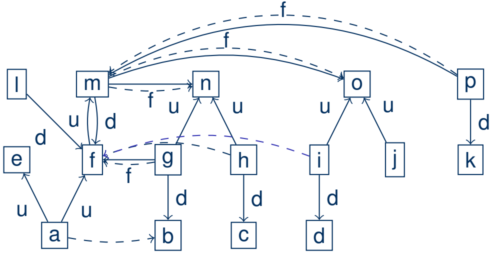
Reverse-Same-Generation – Visualization



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

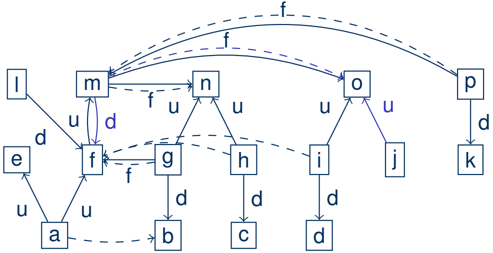
$rsg(x, y) \leftarrow up(x, x_1), rsg(y_1, x_1), down(y_1, y)$

Reverse-Same-Generation – Visualization



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

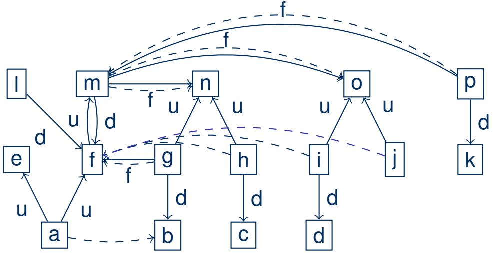
Reverse-Same-Generation – Visualization



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

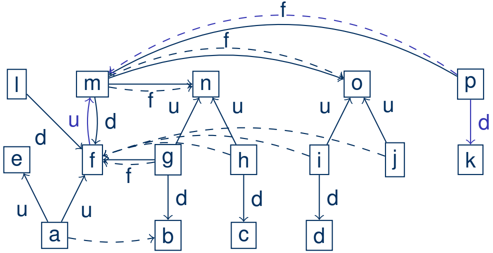
$rsg(x,y) \leftarrow up(x,x_1), rsg(y_1,x_1), down(y_1,y)$

Reverse-Same-Generation – Visualization



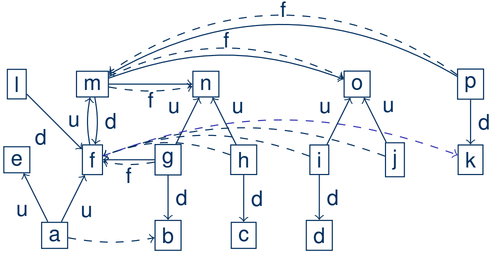
$$rsg(x, y) \leftarrow flat(x, y)$$
$$rsg(x, y) \leftarrow up(x, x_1), rsg(y_1, x_1), down(y_1, y)$$

Reverse-Same-Generation – Visualization



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

Reverse-Same-Generation – Visualization



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

Agenda

- Rules
  - Llyod-Topor Transformation
- Datalog
  - Characterizations of Datalog Program Semantics
- Evaluating Datalog Programs
  - Naïve Evaluation
  - Semi-naïve Evaluation
- Rules for RDFS via a Triple Predicate
- Rules for RDFS via Direct Translation

Naïve Algorithm for Computing 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 fixpoint reached

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

Level 0: $\emptyset$
Level 1: $\{(g,f), (m,n), (m,o), (p,m)\}$
Level 2: {Level 1} $\cup \{(a,b), (h,f), (i,f), (j,f), (f,k)\}$
Level 3: {Level 2} $\cup \{(a,c), (a,d)\}$
Level 4: {Level 3}

Naïve Algorithm for Evaluating Datalog Programs

- redundant computations (all elements of the preceding level are taken into account)
- on each level, all elements of the preceding level are re-computed
- monotone (rsg is extended more and more)

Agenda

- Rules
  - Llyod-Topor Transformation
- Datalog
  - Characterizations of Datalog Program Semantics
- Evaluating Datalog Programs
  - Naïve Evaluation
  - Semi-naïve Evaluation
- Rules for RDFS via a Triple Predicate
- Rules for RDFS via Direct Translation

Semi-Naïve Algorithm for Computing rsg

focus on facts that have been newly computed on the preceding level

Algorithm 2 RSG'

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

- not recursive
- no Datalog program (set of rules is infinite)
- for each input I and Δ^i_{rsg} (the newly computed instances on level i),

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

- RSG(I)(rsg) = $\cup_{1\leq i}(\Delta^i_{rsg})$
- less redundancy

TECHNISCHE UNIVERSITÄT DRESDEN

An Improvement

But: $\Delta^{i+1}_{rsg} \neq rsg^{i+1} - rsg^{i}$

e.g.: $(g,f) \in \Delta^{2}_{rsg}$, $(g,f) \notin rsg^{2} - rsg^{1}$
$\rightsquigarrow rsg(g,f) \in rsg^{1}$, because $flat(g,f)$,
$\rightsquigarrow rsg(g,f) \in \Delta^{2}_{rsg}$, because $up(g,n), rsg(m,f), down(m,f)$

- idea: use $rsg^{i} - rsg^{i-1}$ instead of Δ^{i}_{rsg} in the second “rule” of 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

- Rules
  - Llyod-Topor Transformation
- Datalog
  - Characterizations of Datalog Program Semantics
- Evaluating Datalog Programs
  - Naïve Evaluation
  - Semi-naïve Evaluation
- Rules for RDFS via a Triple Predicate
- Rules for RDFS via Direct Translation

Datalog Rules for RDFS (no Datatypes & Literals)

problem: no strict separation between data and schema (predicates)

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

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

- solution: use a triple predicate

Datalog Rules for RDFS (no Datatypes & Literals)

problem: no strict separation between data and schema (predicates)

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

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

- solution: use a triple predicate

Agenda

- Rules
  - Llyod-Topor Transformation
- Datalog
  - Characterizations of Datalog Program Semantics
- Evaluating Datalog Programs
  - Naïve Evaluation
  - Semi-naïve Evaluation
- Rules for RDFS via a Triple Predicate
- Rules for RDFS via Direct Translation

Datalog Rules for RDFS (no Datatypes & Literals)

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

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

- usage of just one predicate reduces optimization potential
- all (newly derived) triples are potential candidates for any rule
- rules change when the data changes, no separation between schema and data

Datalog Rules for RDFS (no Datatypes & Literals)

- solution 2: introduce specific predicates

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

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

TECHNISCHE UNIVERSITÄT DRESDEN

Axiomatic Triples as Facts

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:List`)
. . . (plus RDFS axiomatic triples)

Axiomatic Triples as Facts

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:List`)
. . . (plus RDFS axiomatic triples)

Axiomatic Triples as Facts

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:List`)
. . . (plus RDFS axiomatic triples)

$\rightsquigarrow$ only needed for those `rdf:_i` that occur in the graphs G_1 and G_2, if $G_1 \models^? G_2$ is to be decided

RDF Entailment Rules (no Datatypes & Literals)

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

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

$\texttt{a}$, $\texttt{b}$ IRIs $\texttt{x}$, $\texttt{y}$ IRI, blank node or literal
$\texttt{u}$, $\texttt{v}$ IRI or blank node $\texttt{l}$ literal $\texttt{\_:n}$ blank nodes

RDF Entailment Rules (no Datatypes & Literals)

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

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

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

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

$\texttt{a}$, $\texttt{b}$ IRIs	$\texttt{x}$, $\texttt{y}$ IRI, blank node or literal		
$\texttt{u}$, $\texttt{v}$ IRI or blank node	$\texttt{l}$ literal	$\texttt{\_:n}$ blank nodes	

RDF Entailment Rules (no Datatypes & Literals)

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

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

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

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

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

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

`a`, `b` IRIs	`x`, `y` IRI, blank node or literal	
`u`, `v` IRI or blank node	`l` literal	`_:n` blank nodes

RDF Entailment Rules (no Datatypes & Literals)

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

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

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

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

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

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

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

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

`a`, `b` IRIs	`x`, `y` IRI, blank node or literal	
`u`, `v` IRI or blank node	`l` literal	`_:n` blank nodes

RDF Entailment Rules (no Datatypes & Literals)

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

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

`a`, `b` IRIs	`x`, `y` IRI, blank node or literal		
`u`, `v` IRI or blank node	`l`	literal	`_:n` blank nodes

RDF Entailment Rules (no Datatypes & Literals)

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

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

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

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

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

RDF Entailment Rules (no Datatypes & Literals)

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

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

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

$\rightsquigarrow$ 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)

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

RDF Entailment Rules (no Datatypes & Literals)

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

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

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

$\leadsto$ subPropertyOf($\texttt{u}$, $\texttt{x}$) $\leftarrow$ subPropertyOf($\texttt{u}$, $\texttt{v}$) $\wedge$ subPropertyOf($\texttt{v}$, $\texttt{x}$)

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

$\leadsto$ subPropertyOf($\texttt{u}$, $\texttt{u}$) $\leftarrow$ type($\texttt{u}$, $\texttt{rdf:Property}$)

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

$\leadsto$ rel($\texttt{u}$, $\texttt{b}$, $\texttt{y}$) $\leftarrow$ subPropertyOf($\texttt{a}$, $\texttt{b}$) $\wedge$ rel($\texttt{u}$, $\texttt{a}$, $\texttt{y}$)

$\texttt{a}$, $\texttt{b}$ IRIs $\quad$ $\texttt{x}$, $\texttt{y}$ IRI, blank node or literal
$\texttt{u}$, $\texttt{v}$ IRI or blank node $\quad$ $\texttt{l}$ literal $\quad$ $\texttt{\_:n}$ blank nodes

RDF Entailment Rules (no Datatypes & Literals)

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

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

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

RDF Entailment Rules (no Datatypes & Literals)

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

$\leadsto$ subClassOf($\texttt{u}$, $\texttt{rdfs:Resource}$) $\leftarrow$ type($\texttt{u}$, $\texttt{rdfs:Class}$)

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

$\leadsto$ type($\texttt{v}$, $\texttt{x}$) $\leftarrow$ subClassOf($\texttt{u}$, $\texttt{x}$) $\wedge$ type($\texttt{v}$, $\texttt{x}$)

`a`, `b` IRIs `x`, `y` IRI, blank node or literal
`u`, `v` IRI or blank node `l` literal `_:n` blank nodes

RDF Entailment Rules (no Datatypes & Literals)

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

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

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

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

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

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

`a`, `b` IRIs   `x`, `y` IRI, blank node or literal
`u`, `v` IRI or blank node   `l` literal   `_:n` blank nodes

TECHNISCHE UNIVERSITÄT DRESDEN

RDF Entailment Rules (no Datatypes & Literals)

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

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

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

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

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

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

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

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

$\texttt{a}$, $\texttt{b}$ IRIs　　$\texttt{x}$, $\texttt{y}$ IRI, blank node or literal
$\texttt{u}$, $\texttt{v}$ IRI or blank node　$\texttt{l}$ literal　$\texttt{\_:n}$ blank nodes

RDF Entailment Rules (no Datatypes & Literals)

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

$\leadsto$ subPropertyOf(`u`, `rdfs:member`) $\leftarrow$ type(`u`, `rdfs:ContainerMembershipProperty`)

`a`, `b` IRIs	`x`, `y` IRI, blank node or literal		
`u`, `v` IRI or blank node	`l` literal	`_:n` blank nodes	

Agenda

- Rules
  - Llyod-Topor Transformation
- Datalog
  - Characterizations of Datalog Program Semantics
- Evaluating Datalog Programs
  - Naïve Evaluation
  - Semi-naïve Evaluation
- Rules for RDFS via a Triple Predicate
- Rules for RDFS via Direct Translation