



TECHNISCHE  
UNIVERSITÄT  
DRESDEN

# FOUNDATIONS OF SEMANTIC WEB TECHNOLOGIES

## **RDFS Rule-based Reasoning**

**Sebastian Rudolph**

Dresden, 16 April 2013



DRESDEN  
SCHOOL  
OF INFORMATICS  
LEADERSHIP  
AND PARTNER

# Content

Overview & XML	9 APR DS2	Hypertableau II	7 JUN DS5
Introduction into RDF	9 APR DS3	Tutorial 5	11 JUN DS2
RDFS – Syntax & Intuition	12 APR DS5	SPARQL Syntax & Intuition	11 JUN DS3
RDFS – Semantics	16 APR DS2	SPARQL – Semantics	14 JUN DS2
RDFS Rule-based Reasoning	16 APR DS3	SPARQL 1.1	14 JUN DS3
Tutorial 1	19 APR DS5	SPARQL Entailment	14 JUN DS5
OWL – Syntax & Intuition	26 APR DS5	Tutorial 6	18 JUN DS2
Tutorial 2	3 MAY DS5	SPARQL Implementation	18 JUN DS3
OWL & Description Logics	7 MAY DS2	Ontology Editing	2 JUL DS2
OWL 2	7 MAY DS3	Ontology Engineering	2 JUL DS3
Tutorial 3	10 MAY DS5	Tutorial 7	9 JUL DS2
Tableau I	14 MAY DS2	Linked Data	9 JUL DS3
Tableau II	14 MAY DS3	Applications	12 JUL DS5
Tutorial 4	17 MAY DS5	Test Exam	16 JUL DS2
Tableau – Optimizations	7 JUN DS2	Test Exam Evaluation	16 JUL DS3
Hypertableau I	7 JUN DS3	Q&A Session	19 JUL DS5

# Agenda

- Rules
  - Lloyd-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
  - Lloyd-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):
  - $\text{man}(X) \leftarrow \text{person}(X) \text{ AND NOT } \text{woman}(X)$
  - 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:

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

# Predicate Logic as a Rule Language

- rules as implication formulae in predicate logic:

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

$\rightsquigarrow$  semantically equivalent to disjunction:

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

- 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:

$$\text{hasUncle}(x, z) \leftarrow \text{hasParent}(x, y) \wedge \text{hasBrother}(y, z)$$

- 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

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

equivalent to

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

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

...

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

- 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:

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

equivalent to

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

equivalent to

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

- ↪ (not considered here)

# Kinds of Rules

names for “rules” in predicate logic:

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

# 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”



# 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
  - $\leftarrow \text{Man}(x) \wedge \text{Woman}(x)$
  - ↔ “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**:

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

- ↪ new elements are dynamically generated
- ↪ not considered here
- ↪ see logic programming

# Agenda

- Rules
  - Lloyd-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

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
  - Lloyd-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
- ↪ interpretation of  $T$  has to be minimal

# Model-theoretic Semantics of Datalog

in principle, a Datalog rule

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

represents the FOL sentence

$$\forall x_1, \dots, x_n. (R_1(x_1) \leftarrow R_2(x_2) \wedge \dots \wedge R_n(x_n))$$

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

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

**we find  $R_1(\nu(u_1))$  satisfied whenever  $R_2(\nu(u_2)), \dots, R_n(\nu(u_n))$  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:

*given* :  $E(a, b), E(b, c), E(c, d)$

$$T(x, y) \leftarrow E(x, y) \tag{1}$$

$$T(x, y) \leftarrow E(x, z) \wedge T(z, y) \tag{2}$$

- (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  $A_1 \leftarrow A_2, \dots, A_n$  of a rule in  $P$ , such that the node is labeled with  $A_1$  and its children with  $A_2, \dots, A_n$

# Proof-theoretic Semantics of Datalog

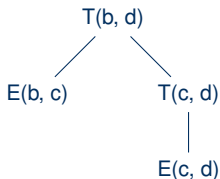
based on proofs for facts:

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

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

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

- (a)  $E(c, d)$  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  $A \leftarrow A_1, \dots, A_n$  is the instance of a rule from  $P$ , such that  $A_1, \dots, A_n \in I$
- 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”  
↔ 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. → IRIS)

## Extensions of Datalog:

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

# Agenda

- Rules
  - Lloyd-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:

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

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

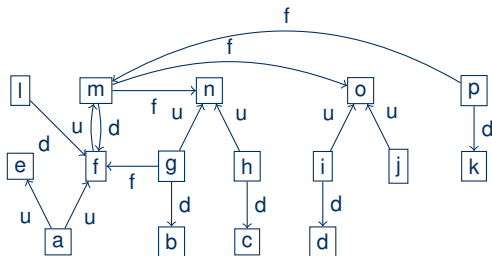
given data:

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

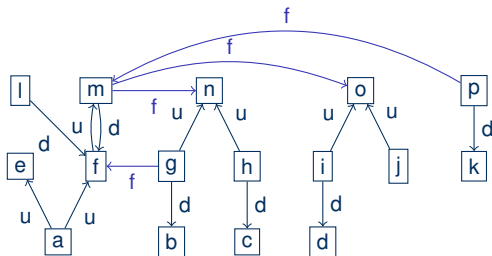
## Reverse-Same-Generation – Visualization



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

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

## Reverse-Same-Generation – Visualization

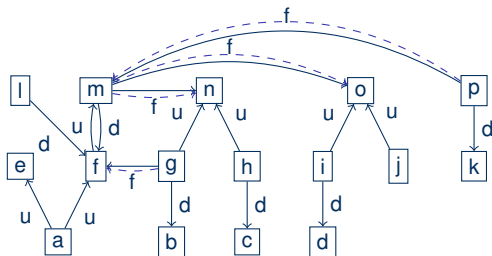


$$rsg(x, y) \leftarrow flat(x, y)$$

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



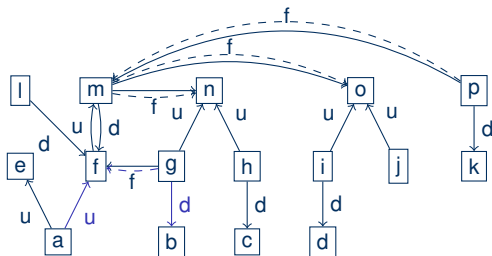
## Reverse-Same-Generation – Visualization



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

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

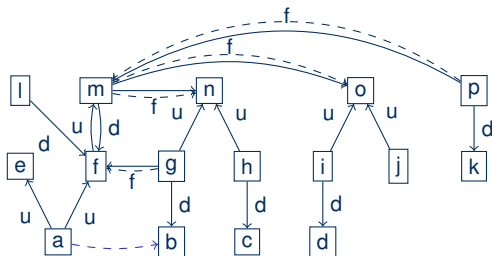
# Reverse-Same-Generation – Visualization



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

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

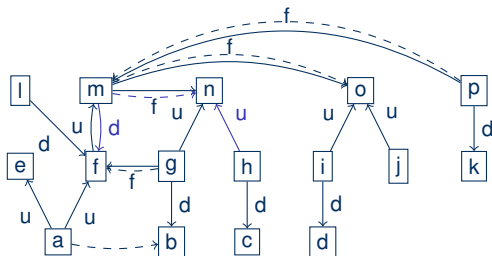
# Reverse-Same-Generation – Visualization



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

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

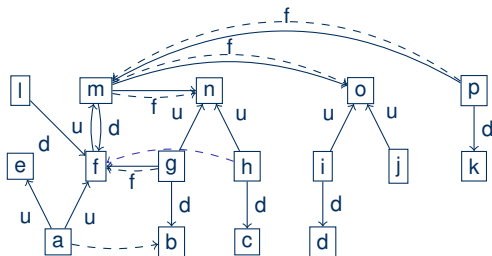
# Reverse-Same-Generation – Visualization



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

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

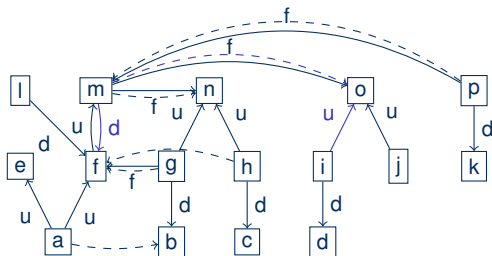
# Reverse-Same-Generation – Visualization



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

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

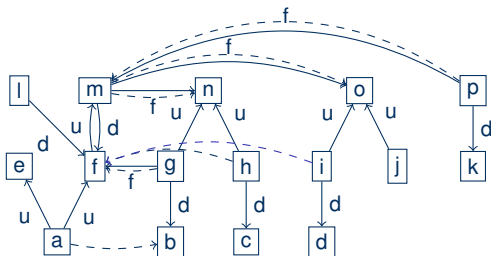
# Reverse-Same-Generation – Visualization



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

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

# Reverse-Same-Generation – Visualization



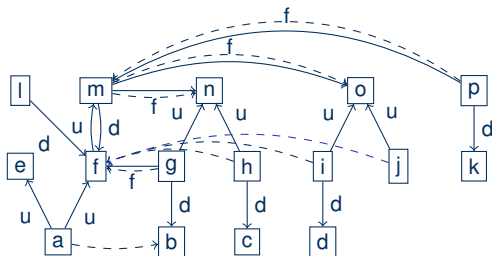
$$rsg(x, y) \leftarrow flat(x, y)$$

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





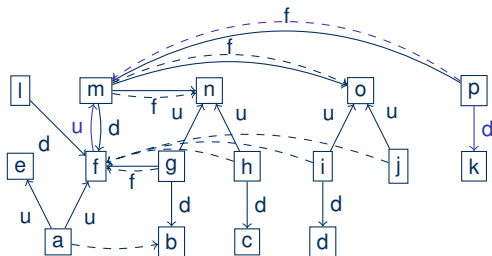
# Reverse-Same-Generation – Visualization



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

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

# Reverse-Same-Generation – Visualization



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

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



# Agenda

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

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

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

# 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
  - Lloyd-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 $RSG'$

---

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

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

---

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

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

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



## An Improvement

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

e.g.:  $(g, f) \in \Delta_{rsg}^2, (g, f) \notin rsg^2 - rsg^1$

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

$\rightsquigarrow rsg(g, f) \in \Delta_{rsg}^2$ , because *up*( $g, n$ ), *rsg*( $m, f$ ), *down*( $m, f$ )

- idea: use  $rsg^i - rsg^{i-1}$  instead of  $\Delta_{rsg}^i$  in the second “rule” of RSG’

---

### Algorithm 3 RSG’’

---

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

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

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

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

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


---

# Agenda

- Rules
  - Lloyd-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)

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

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

- solution: use a triple predicate

# Datalog Rules for RDFS (no Datatypes & Literals)

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

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

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

- solution: use a triple predicate

# Agenda

- Rules
  - Lloyd-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)

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

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

- 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

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

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

## 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)
```

↪ 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)

$$\frac{u \ a \ y}{a \ \text{rdf:type} \ \text{rdf:Property} \quad \text{rdf1}} \rightsquigarrow \text{type}(a, \text{rdf:Property}) \leftarrow \text{rel}(u, a, y)$$

$a, b$  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)

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

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

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

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

$a, b$  IRIs                       $x, y$  IRI, blank node or literal  
 $u, v$  IRI or blank node    $l$  literal    $u, v$  in blank nodes

# RDF Entailment Rules (no Datatypes & Literals)

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

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

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

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

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

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

$a, b$  IRIs                       $x, y$  IRI, blank node or literal  
 $u, v$  IRI or blank node    $l$  literal    $..$  in blank nodes

# RDF Entailment Rules (no Datatypes & Literals)

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

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

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

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

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

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

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

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

$a, b$  IRIs                       $x, y$  IRI, blank node or literal  
 $u, v$  IRI or blank node    $l$  literal    $..$  in blank nodes

# RDF Entailment Rules (no Datatypes & Literals)

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

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

$a, b$  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)

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

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

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

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

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)

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

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

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

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

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

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

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)

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

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

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

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

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

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

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

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

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

# RDF Entailment Rules (no Datatypes & Literals)

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

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

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)

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

↪ `subClassOf(u, rdfs:Resource) ← type(u, rdfs:Class)`

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

↪ `type(v, x) ← subClassOf(u, x) ∧ type(v, u)`

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)

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

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

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

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

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

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

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)

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

↪ **subClassOf**(u, rdfs:Resource) ← **type**(u, rdfs:Class)

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

↪ **type**(v, x) ← **subClassOf**(u, x) ∧ **type**(v, u)

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

↪ **subClassOf**(u, u) ← **type**(u, rdfs:Class)

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

↪ **subClassOf**(u, x) ← **subClassOf**(u, v) ∧ **subClassOf**(v, x)

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

# RDF Entailment Rules (no Datatypes & Literals)

u rdfs:type rdfs:ContainerMembershipProperty . rdfs12  
u rdfs:subPropertyOf rdfs:member .

↪ **subPropertyOf**(u, rdfs:member) ← **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
  - Lloyd-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