Semantic Technologies

Part 10: RDFS Rule-based Reasoning

Werner Nutt

Acknowledgment

These slides are based on the Latex version of slides
by Sebastian Rudolph of TU Dresden

Overview

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

Overview

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):
   - $F \to G$ equivalent to $\neg F \lor G$
   - logical extension of knowledge base $\leadsto$ **static**
   - open world
   - **declarative** (describing)
2. Procedural rules (e.g. production rules):
   - "*If X then Y else Z*"
   - executable commands $\leadsto$ **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**

What are Rules?

1. Logic rules (fragments of predicate logic):
   - $F \rightarrow G$ equivalent to $\neg F \vee G$
   - logical extension of knowledge base $\leadsto$ **static**
   - open world
   - **declarative** (describing)
2. Procedural rules (e.g. production rules):
   - "*If X then Y else Z*"
   - executable commands $\leadsto$ **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**

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:

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

$\leadsto$ 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)

Predicate Logic as a Rule Language

- Rules as implication formulae in predicate logic:

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

$\leadsto$ 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:

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

- 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

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

- Such a rewriting is also referred to as **Lloyd-Topor transformation**

Overview

Kinds of Rules

Names for “rules” in predicate logic:

- **Clause:** disjunction of atomic and negated atomic propositions
  - Woman$(x) \lor$ Man$(x) \;\leftarrow\;$ Person(x)
- **Horn clause:** clause with *at most* one non-negated atom
  - $\leftarrow\;$ Man$(x) \land$ Woman(x)

  $\leadsto$ “integrity constraints”
- **Definite clause:** Horn clause with *exactly one* non-negated atom
  - Father$(x) \;\leftarrow\;$ Man$(x) \land$ hasChild(x, y)
- **Fact:** clause containing just one non-negated atom
  - Woman(gisela)

Kinds of Rules

Names for "rules" in predicate logic:

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

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

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"
- **Definite clause:** Horn clause with *exactly one* non-negated atom
  - $\mathsf{Father}(x) \ \leftarrow\ \mathsf{Man}(x) \wedge \mathsf{hasChild}(x, y)$
- **Fact:** clause containing just one non-negated atom
  - Woman(gisela)

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"
- **Definite clause:** Horn clause with *exactly one* non-negated atom
  - $\mathsf{Father}(x) \ \leftarrow\ \mathsf{Man}(x) \wedge \mathsf{hasChild}(x, y)$
- **Fact:** clause containing just one non-negated atom
  - $\mathsf{Woman}(\mathsf{gisela})$

Kinds of Rules

Rules may also contain **function symbols**:

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

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

Overview

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

Horn rules without function symbols $\leadsto$ **Datalog rules**

- logical rule language, originally basis of *deductive databases*
- knowledge bases ("programs") consisting of Horn clauses without function symbols
- decidable
- efficient for big *data* sets, due to refined optimizations
- a lot of research done in the late 1980s and the 1990s

Datalog as Extension of the Relational Calculus

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

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

Datalog as Extension of the Relational Calculus

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

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

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

Datalog as Extension of the Relational Calculus

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

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

$\leadsto$ 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:

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

- not sufficient to uniquely determine a solution

$\leadsto$ interpretation of T has to be minimal

Model-theoretic Semantics of Datalog

In principle, a Datalog rule

$$\rho\colon R_1(\bar{u}_1) \leftarrow R_2(\bar{u}_2), \ldots, R_n(\bar{u}_n)$$

represents the FOL sentence

$$\forall x_1, \ldots, x_n \, (R_1(\bar{u}_1) \leftarrow R_2(\bar{u}_2) \wedge \ldots \wedge R_n(\bar{u}_n))$$

- $x_1, \ldots, x_n$ 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

  $$R_1(\nu(\bar{u}_1)) \leftarrow R_2(\nu(\bar{u}_2)), \ldots, R_n(\nu(\bar{u}_n))$$

  we find $R_1(\nu(\bar{u}_1))$ satisfied whenever $R_2(\nu(\bar{u}_2)), \ldots, R_n(\nu(\bar{u}_n))$ 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 ($\leadsto$ 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

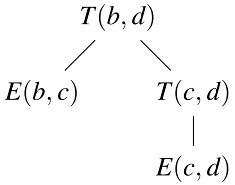
Based on proofs for facts:

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

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

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

(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) …



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 an 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

Overview

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"
  $\leadsto$ 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)

Overview

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:

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

Given data:

up

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

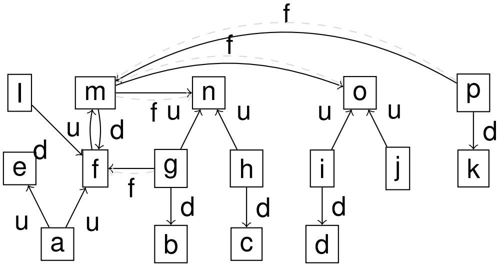
flat

g	f
m	n
m	o
p	m

down

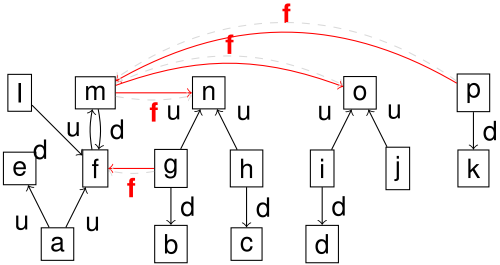
l	f
m	f
g	b
h	c
i	d
p	k

Reverse-Same-Generation – Visualization



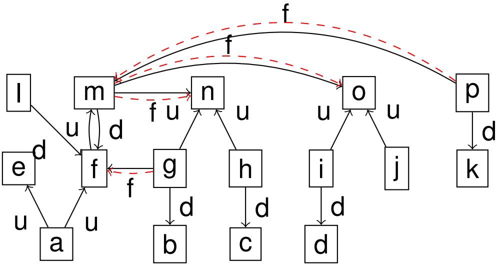
$$rsg(x,y) \leftarrow flat(x,y)$$
$$rsg(x,y) \leftarrow up(x,x_1), rsg(y_1,x_1), down(y_1,y)$$

Reverse-Same-Generation – Visualization



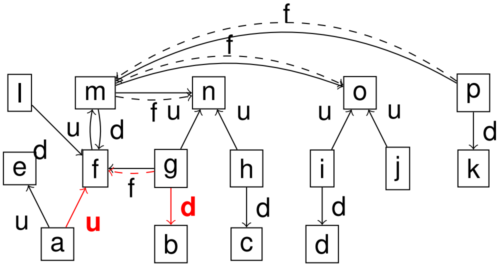
$$rsg(x,y) \leftarrow flat(x,y)$$
$$rsg(x,y) \leftarrow up(x,x_1), rsg(y_1,x_1), down(y_1,y)$$

Reverse-Same-Generation – Visualization



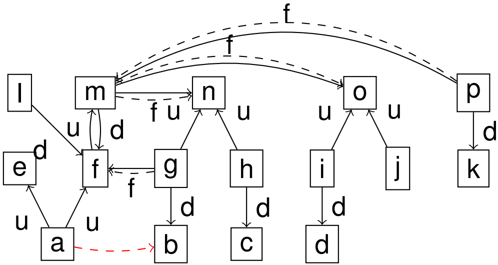
$$rsg(x,y) \leftarrow flat(x,y)$$
$$rsg(x,y) \leftarrow up(x,x_1), rsg(y_1,x_1), down(y_1,y)$$

Reverse-Same-Generation – Visualization



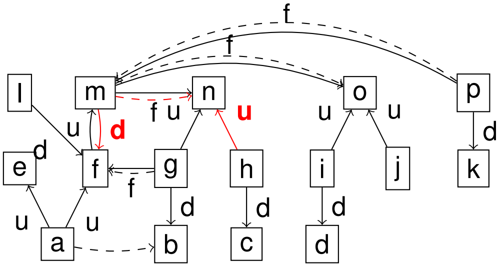
$$rsg(x, y) \leftarrow flat(x, y)$$
$$rsg(x, y) \leftarrow up(x, x_1), rsg(y_1, x_1), down(y_1, y)$$

Reverse-Same-Generation – Visualization



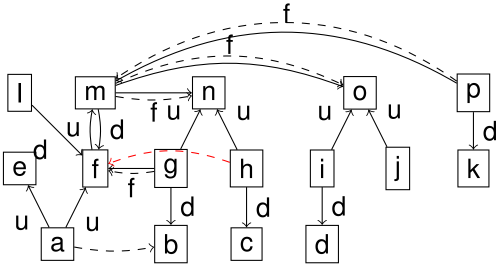
$$rsg(x,y) \leftarrow \mathit{flat}(x,y)$$
$$rsg(x,y) \leftarrow up(x,x_1), rsg(y_1,x_1), down(y_1,y)$$

Reverse-Same-Generation – Visualization



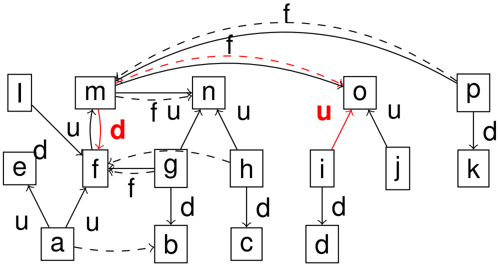
$$rsg(x, y) \leftarrow flat(x, y)$$
$$rsg(x, y) \leftarrow up(x, x_1), rsg(y_1, x_1), down(y_1, y)$$

Reverse-Same-Generation – Visualization



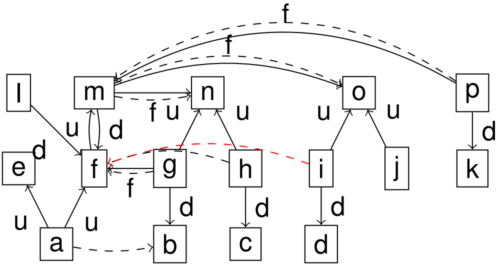
$$rsg(x,y) \leftarrow flat(x,y)$$
$$rsg(x,y) \leftarrow up(x,x_1), rsg(y_1,x_1), down(y_1,y)$$

Reverse-Same-Generation – Visualization



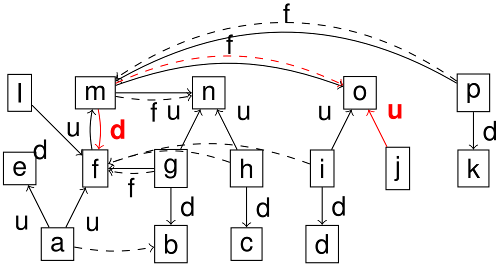
$$rsg(x, y) \leftarrow flat(x, y)$$
$$rsg(x, y) \leftarrow up(x, x_1), rsg(y_1, x_1), down(y_1, y)$$

Reverse-Same-Generation – Visualization



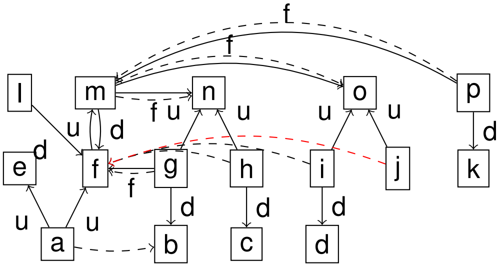
$$rsg(x,y) \leftarrow flat(x,y)$$
$$rsg(x,y) \leftarrow up(x,x_1), rsg(y_1,x_1), down(y_1,y)$$

Reverse-Same-Generation – Visualization



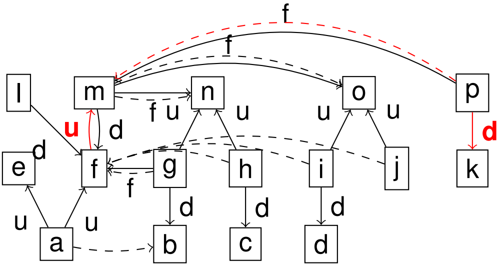
$$rsg(x, y) \leftarrow flat(x, y)$$
$$rsg(x, y) \leftarrow up(x, x_1), rsg(y_1, x_1), down(y_1, y)$$

Reverse-Same-Generation – Visualization



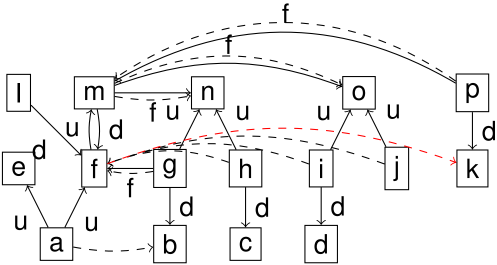
$$rsg(x,y) \leftarrow flat(x,y)$$
$$rsg(x,y) \leftarrow up(x,x_1), rsg(y_1,x_1), down(y_1,y)$$

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



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

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'

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

- 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) = $\cup_{1 \le i}(\Delta^i_{rsg})$
- 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 $\mathit{flat}(g,f)$,
$\rightsquigarrow rsg(g,f) \in \Delta_{rsg}^{2}$, because $up(g,n), rsg(m,n), down(m,f)$

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

Algorithm 3 RSG''

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

Agenda

- 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)

How can we translate an inference rule into a Datalog rule?

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

Datalog Rules for RDFS (no Datatypes & Literals)

How can we translate an inference rule into a Datalog rule?

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

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

Datalog Rules for RDFS (no Datatypes & Literals)

How can we translate an inference rule into a Datalog rule?

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

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

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

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)

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

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

Datalog Rules for RDFS (no Datatypes & Literals)

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

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

- Usage of just one predicate reduces optimization potential

Datalog Rules for RDFS (no Datatypes & Literals)

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

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

- Usage of just one predicate reduces optimization potential
- All (newly derived) triples are potential candidates for any rule

Datalog Rules for RDFS (no Datatypes & Literals)

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

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

- 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

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

Datalog Rules for RDFS (no Datatypes & Literals)

- Solution 2: introduce specific predicates

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

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

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)

$\leadsto$ 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

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)

$\leadsto$ 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

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)

$\leadsto$ 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)

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

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

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

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

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)

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

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

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

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

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

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)

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

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

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

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

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

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

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)

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

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

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

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

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

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

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

$\texttt{a}$, $\texttt{b}$ 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)

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

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

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

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

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

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

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

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

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

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

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

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

`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)

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

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

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

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

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

`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***(v, rdfs:Resource) $\longleftarrow$ ***rel***(u, a, v)

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

$\leadsto$ ***subPropertyOf***(u, x) $\longleftarrow$ ***subPropertyOf***(u, v) $\wedge$ ***subPropertyOf***(v, x)

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

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

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***$(\texttt{u}, \texttt{a}, \texttt{v})$

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

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

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

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

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

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***$(\texttt{u}, \texttt{a}, \texttt{v})$

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

$\leadsto$ ***subPropertyOf***$(\texttt{u}, \texttt{x}) \leftarrow$ ***subPropertyOf***$(\texttt{u}, \texttt{v}) \land$ ***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***$(\texttt{u}, \texttt{rdf:Property})$

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

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

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)

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

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

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

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

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

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

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

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

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***$(\texttt{u}, \texttt{rdfs:Class})$

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

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

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

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

`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***$(\texttt{u}, \texttt{rdfs:Class})$

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

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

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

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

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

`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***$(\texttt{u}, \texttt{rdfs:Class})$

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

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

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

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

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

$\leadsto$ ***subClassOf***$(\texttt{u}, \texttt{x}) \leftarrow$ ***subClassOf***$(\texttt{u}, \texttt{v}) \land$ ***subClassOf***$(\texttt{v}, \texttt{x})$

`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)

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

`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)

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

$\leadsto$ *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