FOUNDATIONS OF SEMANTIC WEB TECHNOLOGIES

RDFS Rule-based Reasoning

Sebastian Rudolph
RDFS Rule-based Reasoning
RDFS Rule-based Reasoning
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 $\neg F \lor G$
   - 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 \( \neg F \lor G \)
   - 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)
What are Rules?

1. logic rules (fragments of predicate logic):
   - $F \rightarrow G$ 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):
   - $\text{man}(X) \leftarrow \text{person}(X) \text{ AND NOT woman}(X)$
   - 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:

\[ H \leftarrow A_1 \land A_2 \land \ldots \land A_n \]

head \hspace{1cm} body

- semantically equivalent to disjunction:

\[ H \lor \neg A_1 \lor \neg A_2 \lor \ldots \lor \neg A_n \]

- implications often written from right to left (← 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:

\[ H \leftarrow A_1 \land A_2 \land \ldots \land A_n \]

\[ \rightarrow \text{semantically equivalent to disjunction:} \]

\[ H \lor \neg A_1 \lor \neg A_2 \lor \ldots \lor \neg A_n \]

• 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)
Example:

\[ \text{hasUncle}(x, z) \leftarrow \text{hasParent}(x, y) \land \text{hasBrother}(y, z) \]

- 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

\[
H_1, H_2, \ldots, H_m \leftarrow A_1, A_2, \ldots, A_n
\]

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
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

\[ H_1 \lor H_2 \lor \ldots \lor H_m \leftarrow A_1 \land A_2 \land \ldots \land A_n \]

equivalent to

\[ H_1 \lor H_2 \lor \ldots \lor H_m \lor \neg A_1 \lor \neg A_2 \lor \ldots \lor \neg A_n \]

~~~ (not considered here)
Kinds of Rules

names for “rules” in predicate logic:

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

- **Horn clause**: clause with at most one non-negated atom
  - $\leftarrow \text{Man}(x) \land \text{Woman}(x)$
  $\Rightarrow$ “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
  - $\text{Woman}(\text{gisela})$
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
  - \( \leftarrow \text{Man}(x) \land \text{Woman}(x) \)
  - \( \leadsto “\text{integrity constraints}” \)
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
  - \( \leftarrow \text{Man}(x) \land \text{Woman}(x) \)
  - “integrity constraints”

- **definite clause**: Horn clause with exactly one non-negated atom
  - \( \text{Father}(x) \leftarrow \text{Man}(x) \land \text{hasChild}(x, y) \)
Kinds of Rules

names for “rules” in predicate logic:

- **Clause**: disjunction of atomic and negated atomic propositions
  
  - Woman(x) ∨ Man(x) ← Person(x)

- **Horn clause**: clause with at most one non-negated atom
  
  - ← Man(x) ∧ Woman(x)
  
  ⇝ “integrity constraints”

- **Definite clause**: Horn clause with exactly one non-negated atom
  
  - Father(x) ← Man(x) ∧ hasChild(x, y)

- **Fact**: clause containing just one non-negated atom
  
  - Woman(gisela)
Kinds of Rules

Rules may also contain function symbols:

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

⇝ new elements are dynamically generated
⇝ not considered here
⇝ 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
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 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

\[
T(x, y) \leftarrow E(x, y)
\]
\[
T(x, y) \leftarrow E(x, z) \land T(z, y)
\]

\(\rightsquigarrow\) 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:

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

- not sufficient to uniquely determine a solution
  \[ \Rightarrow \] interpretation of \( T \) has to be minimal
Model-theoretic Semantics of Datalog

in principle, a Datalog rule

\[ \rho: R_1(u_1) \leftarrow R_2(u_2), \ldots, R_n(u_n) \]

represents the FOL sentence

\[ \forall x_1, \ldots, x_n. (R_1(u_1) \leftarrow R_2(u_2) \land \ldots \land R_n(u_n)) \]

- \( x_1, \ldots, x_n \) 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

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

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:

\[ \begin{align*}
given & : \ E(a, b), E(b, c), E(c, d) \\
T(x, y) & \leftarrow E(x, y) \quad (1) \\
T(x, y) & \leftarrow E(x, z) \land T(z, y) \quad (2)
\end{align*} \]

(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) \ldots
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 (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:

\[
\begin{align*}
given & : \quad E(a, b), E(b, c), E(c, d) \\
T(x, y) & \leftarrow E(x, y) \quad (1) \\
T(x, y) & \leftarrow E(x, z) \land T(z, y) \quad (2)
\end{align*}
\]

(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”
  ↝ not considered here
Extentional 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
  - 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 program:

\[
\begin{align*}
\text{rsg}(x, y) & \leftarrow \text{flat}(x, y) \\
\text{rsg}(x, y) & \leftarrow \text{up}(x, x_1), \text{rsg}(y_1, x_1), \text{down}(y_1, y)
\end{align*}
\]

given data:

<table>
<thead>
<tr>
<th>up</th>
<th></th>
<th>down</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>e</td>
<td>l</td>
<td>f</td>
</tr>
<tr>
<td>a</td>
<td>f</td>
<td>m</td>
<td>f</td>
</tr>
<tr>
<td>f</td>
<td>m</td>
<td>m</td>
<td>o</td>
</tr>
<tr>
<td>g</td>
<td>n</td>
<td>p</td>
<td>m</td>
</tr>
<tr>
<td>h</td>
<td>n</td>
<td>h</td>
<td>c</td>
</tr>
<tr>
<td>i</td>
<td>o</td>
<td>i</td>
<td>d</td>
</tr>
<tr>
<td>j</td>
<td>o</td>
<td>p</td>
<td>k</td>
</tr>
</tbody>
</table>
Reverse-Same-Generation – Visualization

\[
\text{rs}(x, y) \leftarrow \text{flat}(x, y) \\
\text{rs}(x, y) \leftarrow \text{up}(x, x_1), \text{rs}(y_1, x_1), \text{down}(y_1, y)
\]
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

\[
\begin{align*}
\text{rsg}(x, y) & \leftarrow \text{flat}(x, y) \\
\text{rsg}(x, y) & \leftarrow \text{up}(x, x_1), \text{rsg}(y_1, x_1), \text{down}(y_1, y)
\end{align*}
\]
Reverse-Same-Generation – Visualization

\[ \text{rsg}(x, y) \leftarrow \text{flat}(x, y) \]
\[ \text{rsg}(x, y) \leftarrow \text{up}(x, x_1), \text{rsg}(y_1, x_1), \text{down}(y_1, y) \]
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

\[
\begin{align*}
\text{rsg}(x, y) & \leftarrow \text{flat}(x, y) \\
\text{rsg}(x, y) & \leftarrow \text{up}(x, x_1), \text{rsg}(y_1, x_1), \text{down}(y_1, y)
\end{align*}
\]
$rsg(x, y) \leftarrow flat(x, y)$

$rsg(x, y) \leftarrow up(x, x_1), rsg(y_1, x_1), down(y_1, y)$
Reverse-Same-Generation – Visualization

\[
\begin{align*}
\text{rsg}(x, y) & \leftarrow \text{flat}(x, y) \\
\text{rsg}(x, y) & \leftarrow \text{up}(x, x_1), \text{rsg}(y_1, x_1), \text{down}(y_1, y)
\end{align*}
\]
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
  - 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**

\[
\begin{align*}
\text{rs g} & := \emptyset \\
\text{repeat} & \\
\quad \text{rs g} & := \text{rs g} \cup \text{fl at} \cup \pi_{16}(\sigma_{2=4}(\sigma_{3=5}(up \times \text{rs g} \times down))) \\
\text{until} & \text{ fixpoint reached}
\end{align*}
\]

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

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
  – 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 $RSG'$

\[
\begin{align*}
\Delta_{rsg}^1(x, y) &:= \text{flat}(x, y) \\
\Delta_{rsg}^{i+1}(x, y) &:= \text{up}(x, x_1), \Delta_{rsg}^i(y_1, x_1), \text{down}(y_1, y)
\end{align*}
\]

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

\[
\text{rsg}^{i+1} - \text{rsg}^i \subseteq \Delta_{rsg}^{i+1} \subseteq \text{rsg}^{i+1}
\]

- $RSG(I)(\text{rsg}) = \bigcup_{1 \leq i} (\Delta_{rsg}^i)$
- less redundancy
An Improvement

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

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

$\leadsto \text{rsg}(g, f) \in \text{rsg}^1$, because $\text{flat}(g, f)$,

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

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

Algorithm 3 \textit{RSG}''

\[
\begin{align*}
\Delta_{\text{rsg}}^1 (x, y) &:= \text{flat}(x, y) \\
\text{rsg}^1 &:= \Delta_{\text{rsg}}^1 \\
\text{tmp}_{\text{rsg}}^{i+1} (x, y) &:= \text{up}(x, x_1), \Delta_{\text{rsg}}^i (y_1, x_1), \text{down}(y_1, y) \\
\Delta_{\text{rsg}}^{i+1} (x, y) &:= \text{tmp}_{\text{rsg}}^{i+1} - \text{rsg}^i \\
\text{rsg}^{i+1} &:= \text{rsg}^i \cup \Delta_{\text{rsg}}^{i+1}
\end{align*}
\]
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)

\[
\begin{align*}
\text{a rdfs:domain x . u a y .} \\
\text{u rdf:type x .} \\
\end{align*}
\]

\[
\text{rdf:type}(u, x) \leftarrow \text{rdfs:domain}(a, x) \land a(u, y)
\]

- solution: use a triple predicate
Datalog Rules for RDFS (no Datatypes & Literals)

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

\[
\begin{align*}
\text{a rdfs:domain x . u a y .} \\
\text{u rdf:type x .} \quad \text{rdfs2}
\end{align*}
\]

\[
\text{rdf:type}(u, x) \leftarrow \text{rdfs:domain}(a, x) \land a(u, y)
\]

- 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)

\[
\begin{align*}
\text{a rdfs:domain } & \ x \ . \ u \ a \ y \ . \\
\de \ u \ rdf:type \ & \ x \ . \\
\end{align*}
\]

\[
\text{Triple}(u, rdf:type, x) \leftarrow \text{Triple}(a, rdfs:domain, x) \land \text{Triple}(u, a, y)
\]

- 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

\[
\begin{align*}
\text{a rdfs:domain} & \text{x . u a y .} \\
\text{u rdf:type} & \text{x .} \\
\end{align*}
\]

\[
\text{type}(u, x) \leftarrow \text{domain}(a, x) \land \text{rel}(u, a, y)
\]
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

\[
\begin{align*}
\text{type}(\text{rdf:} & \text{type}, \text{rdf:} \text{Property}) \\
\text{type}(\text{rdf:} & \text{subject}, \text{rdf:} \text{Property}) \\
\text{type}(\text{rdf:} & \text{predicate}, \text{rdf:} \text{Property}) \\
\text{type}(\text{rdf:} & \text{object}, \text{rdf:} \text{Property}) \\
\text{type}(\text{rdf:} & \text{first}, \text{rdf:} \text{Property}) \\
\text{type}(\text{rdf:} & \text{rest}, \text{rdf:} \text{Property}) \\
\text{type}(\text{rdf:} & \text{value}, \text{rdf:} \text{Property}) \\
\text{type}(\text{rdf:}_1, \text{rdf:} \text{Property}) \\
\text{type}(\text{rdf:}_2, \text{rdf:} \text{Property}) \\
\text{type}(\ldots, \text{rdf:} \text{Property}) \\
\text{type}(\text{rdf:} \text{nil}, \text{rdf:} \text{List}) \\
\ldots \text{(plus RDFS axiomatic triples)}
\end{align*}
\]
Axiomatic Triples as Facts

\[
\begin{align*}
type(rdf:type, rdf:Property) \\
type(rdf:subject, rdf:Property) \\
type(rdf:predicate, rdf:Property) \\
type(rdf:object, rdf:Property) \\
type(rdf:first, rdf:Property) \\
type(rdf:rest, rdf:Property) \\
type(rdf:value, rdf:Property) \\
type(rdf:_1, rdf:Property) \\
type(rdf:_2, rdf:Property) \\
type(..., rdf:Property) \\
type(rdf:nil, rdf:List) \\
\ldots \text{(plus RDFS axiomatic triples)}
\end{align*}
\]

\[
\Rightarrow \text{only needed for those } rdf:i \text{ that occur in the graphs } G_1 \text{ and } G_2, \text{ if } G_1 \models ? G_2 \text{ is to be decided}
\]
RDF Entailment Rules (no Datatypes & Literals)

\[
\begin{align*}
\text{rdf1} & : a \text{ rdf:type } \text{ rdf:Property} \\
\text{rdfs1} & : \text{domain } x \\
\text{rdfs2} & : \text{range } x \\
\text{rdfs4} & : \text{type } \text{ rdf:Resource} \\
\end{align*}
\]

\[
\begin{align*}
\text{u a y} & \quad \text{rdf1} \\
\text{rdfs1} & : \text{domain} \ a \ y \\
\text{rdfs2} & : \text{range} \ a \ y \\
\text{u a x} & : \text{rdfs4} \\
\end{align*}
\]

\[
\begin{align*}
\text{a, b} & \quad \text{IRIs} \\
\text{x, y} & \quad \text{IRI, blank node or literal} \\
\text{u, v} & \quad \text{IRI or blank node} \\
\end{align*}
\]

TU Dresden Foundations of Semantic Web Technologies
RDF Entailment Rules (no Datatypes & Literals)

\[
\begin{align*}
\text{rdf1} & \quad \frac{u \ a \ v}{a \ \text{rdf:type} \ \text{rdf:Property}} \\
& \quad \leadsto \ \text{type}(a, \text{rdf:Property}) \leftarrow \ \text{rel}(u, a, y)
\end{align*}
\]

\[
\begin{align*}
\text{rdfs2} & \quad \frac{a \ \text{rdfs:domain} \ x \ . \ u \ a \ y \ .}{u \ \text{rdf:type} \ x .} \\
& \quad \leadsto \ \text{type}(u, x) \leftarrow \ \text{domain}(a, x) \land \ \text{rel}(u, a, y)
\end{align*}
\]

\(a, b\) IRIs \quad \(x, y\) IRI, blank node or literal

\(u, v\) IRI or blank node \quad \text{literal} \quad \_n\) blank nodes
RDF Entailment Rules (no Datatypes & Literals)

\[
\begin{align*}
\text{rdf1} & : \\
\frac{u \ a \ y}{a \ \text{rdf}:\text{type} \ \text{rdf}:\text{Property}} \\
\iff \text{type}(a, \ \text{rdf}:\text{Property}) \gets \text{rel}(u, a, y)
\end{align*}
\]

\[
\begin{align*}
\text{rdfs2} & : \\
\frac{a \ \text{rdfs}:\text{domain} \ x \ . \ u \ a \ y}{u \ \text{rdf}:\text{type} \ x} \\
\iff \text{type}(u, x) \gets \text{domain}(a, x) \land \text{rel}(u, a, y)
\end{align*}
\]

\[
\begin{align*}
\text{rdfs3} & : \\
\frac{a \ \text{rdfs}:\text{range} \ x \ . \ u \ a \ v}{v \ \text{rdf}:\text{type} \ x} \\
\iff \text{type}(v, x) \gets \text{range}(a, x) \land \text{rel}(u, a, v)
\end{align*}
\]

\[
\begin{align*}
\text{rdfs4a} & : \\
\frac{u \ a \ x}{u \ \text{rdf}:\text{type} \ \text{rdfs}:\text{Resource}} \\
\iff \text{type}(u, \ \text{rdfs}:\text{Resource}) \gets \text{rel}(u, a, x)
\end{align*}
\]

\[
\begin{align*}
\text{a, b IRIs} & \quad x, y \text{ IRI, blank node or literal} \\
u, v \text{ IRI or blank node} & \quad \text{l IRI, literal} \quad \text{_:n blank nodes}
\end{align*}
\]
RDF Entailment Rules (no Datatypes & Literals)

\[ \text{RDF1: } \begin{align*} & u \ a \ y \\ & \text{a rdf:type rdf:Property} \\ & \leadsto \text{type(a, rdf:Property) } \leftarrow \text{rel(u, a, y)} \end{align*} \]

\[ \text{RDFS2: } \begin{align*} & a \ \text{rdfs:domain x} \ . \ u \ a \ y \\ & u \ \text{rdf:type x} \\ & \leadsto \text{type(u, x) } \leftarrow \text{domain(a, x) } \land \text{rel(u, a, y)} \end{align*} \]

\[ \text{RDFS3: } \begin{align*} & a \ \text{rdfs:range x} \ . \ u \ a \ v \\ & v \ \text{rdf:type x} \\ & \leadsto \text{type(v, x) } \leftarrow \text{range(a, x) } \land \text{rel(u, a, v)} \end{align*} \]

\[ \text{RDFS4a: } \begin{align*} & u \ a \ x \\ & u \ \text{rdf:type rdfs:Resource} \\ & \leadsto \text{type(u, rdfs:Resource) } \leftarrow \text{rel(u, a, x)} \end{align*} \]

\[ \begin{align*} & a, b \text{ IRIs} \\ & x, y \text{ IRI, blank node or literal} \\ & u, v \text{ IRI or blank node} \\ & l \text{ literal} \\ & \_n \text{ blank nodes} \end{align*} \]
RDF Entailment Rules (no Datatypes & Literals)

\[ \frac{u \ a \ v}{v \ \text{rdfs:type} \ \text{rdfs:Resource}} \ \ \ \text{rdfs4b} \]
\[ \text{\sim} \ \text{type}(v, \ \text{rdfs:Resource}) \leftarrow \text{rel}(u, \ a, \ v) \]

- \( a, b \) IRIs
- \( x, y \) IRI, blank node or literal
- \( u, v \) IRI or blank node
- \( \text{liter} \) literal
- \( n \) blank nodes
RDF Entailment Rules (no Datatypes & Literals)

\[ u \, a \, v. \]
\[ v \, rdf:type \, rdfs:Resource. \quad rdfs4b \]
\[ \leadsto \, type(v, \, rdfs:Resource) \leftarrow \, rel(u, \, a, \, v) \]

\[ u \, rdfs:subPropertyOf \, v. \quad v \, rdfs:subPropertyOf \, x. \quad rdfs5 \]
\[ \quad u \, rdfs:subPropertyOf \, x. \]
\[ \leadsto \, subPropertyOf(u, \, x) \leftarrow \, subPropertyOf(u, \, v) \land \, subPropertyOf(v, \, x) \]

\[ a, \, b \, \text{IRIs} \quad x, \, y \, \text{IRI, blank node or literal} \]
\[ u, \, v \, \text{IRI or blank node} \quad \_l \, \text{literal} \quad \_n \, \text{blank nodes} \]
RDF Entailment Rules (no Datatypes & Literals)

\[
\begin{align*}
\text{u a v . } \\
v \text{ rdf:type rdfs:Resource . } \\
\leadsto \text{type}(v, \text{rdfs:Resource}) \leftarrow \text{rel}(u, a, v)
\end{align*}
\]

\[
\begin{align*}
u \text{ rdfs:subPropertyOf v } . \\
v \text{ rdfs:subPropertyOf x } . \\
\text{rdfs5}
\end{align*}
\]

\[
\begin{align*}
\text{u rdfs:subPropertyOf x } . \\
\text{rdfs5}
\end{align*}
\]

\[
\begin{align*}
\text{u rdfs:subPropertyOf u } . \\
\text{rdfs6}
\end{align*}
\]

\[
\begin{align*}
\text{u rdf:type rdf:Property } . \\
\text{rdfs6}
\end{align*}
\]

\[
\begin{align*}
\text{a, b IRI, } x, y \text{ IRI, blank node or literal} \\
u, v \text{ IRI or blank node} \\
l \text{ literal} \\
\_n \text{ blank nodes}
\end{align*}
\]
RDF Entailment Rules (no Datatypes & Literals)

\[
\frac{\text{u a v} .}{\text{v rdf:type rdfs:Resource} .} \quad \text{rdfs4b}
\]
\[\sim \text{type}(v, \text{rdfs:Resource}) \leftarrow \text{rel}(u, a, v)\]

\[
\frac{u \text{rdfs:subPropertyOf} v . \quad v \text{rdfs:subPropertyOf} x .}{u \text{rdfs:subPropertyOf} x .} \quad \text{rdfs5}
\]
\[\sim \text{subPropertyOf}(u, x) \leftarrow \text{subPropertyOf}(u, v) \land \text{subPropertyOf}(v, x)\]

\[
\frac{u \text{ rdf:type rdf:Property} .}{u \text{rdfs:subPropertyOf} u .} \quad \text{rdfs6}
\]
\[\sim \text{subPropertyOf}(u, u) \leftarrow \text{type}(u, \text{rdf:Property})\]

\[
\frac{a \text{rdfs:subPropertyOf} b . \quad u a y .}{u b y .} \quad \text{rdfs7}
\]
\[\sim \text{rel}(u, b, y) \leftarrow \text{subPropertyOf}(a, b) \land \text{rel}(u, a, y)\]

\[a, b \text{ IRIs} \quad x, y \text{ IRI, blank node or literal} \quad u, v \text{ IRI or blank node} \quad \text{literal} \quad \text{n blank nodes}\]

TU Dresden Foundations of Semantic Web Technologies
RDF Entailment Rules (no Datatypes & Literals)

\[
\text{u rdf:type rdfs:Class .} \\
\text{u rdf:subClassOf rdfs:Resource .} \\
\rightarrow \text{subClassOf(u, rdfs:Resource) } \leftarrow \text{type(u, rdfs:Class)}
\]

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)

\[
\begin{align*}
\text{rdfs} & 8 \\
\text{u rdf:type rdfs:Class} & \\ 
\text{u rdf:subClassOf rdfs:Resource} & \\ 
\leadsto \text{subClassOf}(u, \text{rdfs:Resource}) & \leftarrow \text{type}(u, \text{rdfs:Class})
\end{align*}
\]

\[
\begin{align*}
\text{rdfs} & 9 \\
\text{u rdfs:subClassOf x} & . \text{v rdf:type u} . \text{v rdf:type x} . \\
\leadsto \text{type}(v, x) & \leftarrow \text{subClassOf}(u, x) \land \text{type}(v, x)
\end{align*}
\]
RDF Entailment Rules (no Datatypes & Literals)

\[ \frac{u \text{ rdf:}\text{type rdfs:Class}}{u \text{ rdf:}\text{subClassOf rdfs:Resource}} \quad \text{rdfs8} \]
\[ \frac{\text{u rdf:}\text{subClassOf rdfs:Resource}}{\text{~}\Rightarrow \text{subClassOf(u, rdfs:Resource)} \iff \text{type(u, rdfs:Class)}} \]

\[ \frac{u \text{ rdfs:subClassOf x} \quad v \text{ rdf:}\text{type u}}{v \text{ rdf:}\text{type x}} \quad \text{rdfs9} \]
\[ \text{~}\Rightarrow \text{type(v, x) } \iff \text{subClassOf}(u, x) \land \text{type(v, x)}} \]

\[ \frac{u \text{ rdf:}\text{type rdfs:Class}}{u \text{ rdfs:subClassOf u}} \quad \text{rdfs10} \]
\[ \frac{\text{u rdfs:subClassOf u}}{\text{~}\Rightarrow \text{subClassOf(u, u)} \iff \text{type(u, rdfs:Class)}} \]

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)

\[
\begin{align*}
\text{rdfs8} & : \\
\text{u rdf:type rdfs:Class .} & \quad \text{u rdf:subClassOf rdfs:Resource .} \\
\to & \quad \text{subClassOf(u, rdfs:Resource) } \leftarrow \text{type(u, rdfs:Class)}
\end{align*}
\]

\[
\begin{align*}
\text{rdfs9} & : \\
\text{u rdfs:subClassOf x . v rdf:type u .} & \quad \text{v rdf:type x .} \\
\to & \quad \text{type(v, x) } \leftarrow \text{subClassOf(u, x) } \land \text{type(v, x)}
\end{align*}
\]

\[
\begin{align*}
\text{rdfs10} & : \\
\text{u rdf:type rdfs:Class .} & \quad \text{u rdfs:subClassOf u .} \\
\to & \quad \text{subClassOf(u, u) } \leftarrow \text{type(u, rdfs:Class)}
\end{align*}
\]

\[
\begin{align*}
\text{rdfs11} & : \\
\text{u rdfs:subClassOf v . v rdfs:subClassOf x .} & \quad \text{u rdfs:subClassOf x .} \\
\to & \quad \text{subClassOf(u, x) } \leftarrow \text{subClassOf(u, v) } \land \text{subClassOf(v, x)}
\end{align*}
\]

\[
\begin{align*}
\text{a, b IRIs} & \\
\text{x, y IRI, blank node or literal} & \\
\text{u, v IRI or blank node} & \quad \text{1 literal} & \quad \text{n blank nodes}
\end{align*}
\]

TU Dresden Foundations of Semantic Web Technologies
RDF Entailment Rules (no Datatypes & Literals)

\[
\operatorname{u} \text{ rdf:} \text{type} \text{ rdfs:} \text{ContainerMembershipProperty} . \quad \text{rdfs12}
\]

\[
\quad \text{u} \text{ rdfs:} \text{subPropertyOf} \text{ rdfs:} \text{member} .
\]

\[
\leadsto \text{subPropertyOf}(\text{u, rdfs:} \text{member}) \leftarrow \text{type}(\text{u, rdfs:} \text{ContainerMembershipProperty})
\]

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