Acknowledgment

These slides are based on the Latex version of slides by Sebastian Rudolph of TU Dresden
<table>
<thead>
<tr>
<th>Overview</th>
<th>What are Rules?</th>
<th>Kinds of Rules</th>
<th>Datalog</th>
<th>Semantics of Rules</th>
<th>Evaluating Datalog Programs</th>
</tr>
</thead>
</table>

**Overview**

1. Overview
2. What are Rules?
3. Kinds of Rules
4. Datalog
5. Semantics of Rules
6. Evaluating Datalog Programs
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**
  - 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**
<table>
<thead>
<tr>
<th>Overview</th>
<th>What are Rules?</th>
<th>Kinds of Rules</th>
<th>Datalog</th>
<th>Semantics of Rules</th>
<th>Evaluating Datalog Programs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Overview</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 What are Rules?</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 Kinds of Rules</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 Datalog</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5 Semantics of Rules</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6 Evaluating Datalog Programs</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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 \( \leadsto \) 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
What are Rules?

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

2. Procedural rules (e.g. production rules):
   - “If $X$ then $Y$ else $Z$”
   - executable commands $\sim$ **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**
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):
   - $\text{man}(X) \leftarrow \text{person}(X) \land \neg \text{woman}(X)$
   - 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:

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

\( H \) is 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)
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 \quad body

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

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

- Implications often written from right to left (\leftarrow \text{ 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:

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

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

- Rules
  - 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
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

What are Rules?

Kinds of Rules

Datalog

Semantics of Rules

Evaluating Datalog Programs

RDFS Rule-based Reasoning
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
  
  Man(\(x\)) \land Woman(\(x\)) \leftarrow
  
  “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
  - 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

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
  - \(\leftarrow Man(x) \land Woman(x)\)
  - “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
  - 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:

\[
\begin{align*}
\text{hasUncle}(x, y) & \leftarrow \text{hasBrother}(\text{mother}(x), y) \\
\text{hasFather}(x, \text{father}(x)) & \leftarrow \text{Person}(x)
\end{align*}
\]

→ new elements are dynamically generated
→ not considered here
→ see logic programming
Overview

1. Overview
2. What are Rules?
3. Kinds of Rules
4. Datalog
5. Semantics of Rules
6. Evaluating Datalog Programs
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
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 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

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

\( \sim \) 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
- interpretation of \( T \) has to be minimal
In principle, a Datalog rule

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

represents the FOL sentence

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

- $x_1, \ldots, x_n$ 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

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

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:

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

(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 (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:

\[
given: \quad E(a, b), E(b, c), E(c, d) \\
T(x, y) \leftarrow E(x, y) \quad (1) \\
T(x, y) \leftarrow E(x, z) \land T(z, y) \quad (2)
\]

(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
<table>
<thead>
<tr>
<th></th>
<th>Overview</th>
<th>What are Rules?</th>
<th>Kinds of Rules</th>
<th>Datalog</th>
<th>Semantics of Rules</th>
<th>Evaluating Datalog Programs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Overview</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>What are Rules?</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Kinds of Rules</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Datalog</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Semantics of Rules</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Evaluating Datalog Programs</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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:

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

1. Overview
2. What are Rules?
3. Kinds of Rules
4. Datalog
5. Semantics of Rules
6. Evaluating Datalog Programs
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 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>flat</th>
<th>down</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>g</td>
<td>l</td>
</tr>
<tr>
<td>e</td>
<td>f</td>
<td>f</td>
</tr>
<tr>
<td>a</td>
<td>m</td>
<td>m</td>
</tr>
<tr>
<td>f</td>
<td>n</td>
<td>f</td>
</tr>
<tr>
<td>m</td>
<td>o</td>
<td>g</td>
</tr>
<tr>
<td>n</td>
<td>m</td>
<td>b</td>
</tr>
<tr>
<td>g</td>
<td>p</td>
<td>h</td>
</tr>
<tr>
<td>n</td>
<td>m</td>
<td>c</td>
</tr>
<tr>
<td>i</td>
<td>i</td>
<td>i</td>
</tr>
<tr>
<td>o</td>
<td>d</td>
<td>d</td>
</tr>
<tr>
<td>j</td>
<td>k</td>
<td>k</td>
</tr>
</tbody>
</table>
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

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

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

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

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

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

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

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

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

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

Algorithm 1 RSG

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

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

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** \( RSG' \)

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

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

\[
rsg^{i+1} - rs\cdot g^i \subseteq \Delta_{rsg}^{i+1} \subseteq rs\cdot g^{i+1}
\]

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

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

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

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

~ $rsg(g,f) \in \Delta^2_{rsg}$, because $up(g,n), rsg(m,n), down(m,f)$

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

Algorithm 3 $RSG''$

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

$rsg^1 := \Delta^1_{rsg}$

$\Delta^i_{rsg}(x, y) := up(x, x_1), \Delta^i_{rsg}(y_1, x_1), down(y_1, y)$

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

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

\[
\frac{a \text{ rdfs:domain } x . \ u \ a \ y .}{u \ \text{rdf:type } x .} \quad \text{rdfs2}
\]
Datalog Rules for RDFS (no Datatypes & Literals)

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

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

\[
\text{rdf:type}(u, x) \leftarrow \text{rdfs:domain}(a, x) \land a(u, y)
\]
Datalog Rules for RDFS (no Datatypes & Literals)

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

\[
\begin{align*}
\text{a rdfs:domain } & \ x \ \ . \ \ u \ a \ y \ . \\
\Rightarrow \ u \ \ rdf:\text{type} \ & \ x \ . \\
\text{rdf:}\text{type}(u,x) & \leftarrow \text{rdfs:domain}(a,x) \land a(u,y)
\end{align*}
\]
Datalog Rules for RDFS (no Datatypes & Literals)

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

\[ \text{a } \text{rdfs:domain } x \ . \ u \ a \ y \ . \ \text{rdfs2} \]
\[ u \ \text{rdf:type } x \ . \]
\[ \text{rdf:type}(u, x) \leftarrow \text{rdfs:domain}(a, x) \land a(u, y) \]

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)

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

\[\text{Triple}(u, \texttt{rdf:type}, x) \leftarrow \text{Triple}(a, \texttt{rdfs:domain}, x) \land \text{Triple}(u, a, y)\]
Datalog Rules for RDFS (no Datatypes & Literals)

\[
\frac{a \text{ rdfs:domain } x \quad u \ a \ y}{u \text{ rdf:type } x} \quad \text{rdfs2}
\]

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

- Usage of just one predicate reduces optimization potential
Datalog Rules for RDFS (no Datatypes & Literals)

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

\[\text{rdfs2}\]

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

- Usage of just one predicate reduces optimization potential
- All (newly derived) triples are potential candidates for any rule
Datalog Rules for RDFS (no Datatypes & Literals)

\[
\frac{a \text{ rdfs:domain } x \cdot u \ a \ y }{u \text{ rdf:type } x} \quad \text{rdfs2}
\]

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

- 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
Solution 2: introduce specific predicates

\[
\begin{array}{c}
\frac{\text{a rdfs:domain } x \ . \ u a y . \ u \ rdf:type \ x .}{\text{rdfs2}}
\end{array}
\]
Solution 2: introduce specific predicates

\[
\begin{align*}
\text{a rdfs:domain } x & . \quad u \; a \; y & . \\
\text{u rdf:type } x & . \\
\text{type}(u, x) & \leftarrow \text{domain}(a, x) \land \text{rel}(u, a, y)
\end{align*}
\]
Axiomatic Triples as Facts

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

\(\leadsto\) 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

\[
\text{type}(\text{rdf:type, rdf:Property}) \\
\text{type}(\text{rdf:subject, rdf:Property}) \\
\text{type}(\text{rdf:_predicate, rdf:Property}) \\
\text{type}(\text{rdf:object, rdf:Property}) \\
\text{type}(\text{rdf:subject, rdf:Property}) \\
\text{type}(\text{rdf:object, rdf:Property}) \\
\text{type}(\text{rdf:property, rdf:Property}) \\
\text{type}(\text{rdf:property, rdf:Property}) \\
\text{type}(\text{..., rdf:Property}) \\
\text{type}(\text{rdf:nil, rdf:List})
\]

... (plus RDFS axiomatic triples)

\[\leadsto\text{only needed for those \text{ rdf:}i \text{ that occur in the graphs } G_1 \text{ and } G_2, if } G_1 \models ? G_2 \text{ is to be decided}\]
Axiomatic Triples as Facts

\[\text{type}(	ext{rdf:type}, \text{rdf:Property})\]
\[\text{type}(	ext{rdf:subject}, \text{rdf:Property})\]
\[\text{type}(	ext{rdf:predicate}, \text{rdf:Property})\]
\[\text{type}(	ext{rdf:object}, \text{rdf:Property})\]
\[\text{type}(	ext{rdf:subject}, \text{rdf:Property})\]
\[\text{type}(	ext{rdf:subject}, \text{rdf:Property})\]
\[\text{type}(	ext{rdf:property}, \text{rdf:Property})\]
\[\text{type}(	ext{rdf:property}, \text{rdf:Property})\]
\[\text{type}(	ext{rdf:property}, \text{rdf:Property})\]
\[\text{type}(	ext{rdf:property}, \text{rdf:Property})\]

only needed for those \text{rdf:property} 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)

\[
\begin{align*}
\text{rdf1:} & \quad u \ a \ y \\
\text{a rdf:type rdf:Property} & \\
\hline
\text{rdfs2:} & \quad a \ rdfs:domain \ x \ . \ u \ a \ y \\
\text{u rdf:type} & \quad x \\
\hline
\text{rdfs3:} & \quad a \ rdfs:range \ x \ . \ u \ a \ v \\
\text{v rdf:type} & \quad x \\
\hline
\text{rdfs4a:} & \quad u \ a \ x \\
\text{u rdf:type rdfs:Resource} & \\
\hline
\end{align*}
\]

\begin{itemize}
  \item \text{a, b IRI}s
  \item \text{x, y IRI, blank node or literal}
  \item \text{u, v IRI or blank node}
  \item \text{l literal}
  \item \text{n blank nodes}
\end{itemize}
RDF Entailment Rules (no Datatypes & Literals)

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

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

\[
\begin{align*}
\text{rdfs3: } & \\ a & \text{rdfs:range } x . & u & \ a & \ v . \\
\hline
\text{v rdf:type } x .
\end{align*}
\]

\[
\begin{align*}
\text{rdfs4a: } & \\ u & \ a & \ x . \\
\hline
\text{u rdf:type rdfs:Resource} .
\end{align*}
\]

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

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

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

\text{rdfs3:} & \quad \frac{a \, \text{rdfs:range} \, x \, . \, u \, a \, v \, .}{v \, \text{rdf:type} \, x \, .} \\
\text{range}(a, x) \land \text{rel}(u, a, v) & \quad \leadsto \text{type}(v, x) \leftarrow \text{range}(a, x) \land \text{rel}(u, a, v) \\

\text{rdfs4a:} & \quad \frac{u \, a \, x \, .}{u \, \text{rdf:type} \, \text{rdfs:Resource} \, .} \\
\text{Blank Nodes} & \quad \leadsto \text{type}(u, \text{rdfs:Resource}) \leftarrow \text{Blank Nodes} \\

\end{align*}
\]

a, b IRIs \\
u, v IRI or blank node \\
x, y IRI, blank node or literal \\
l literal \\
\_n blank nodes
RDF Entailment Rules (no Datatypes & Literals)

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

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

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

\[
\begin{align*}
\text{u a x} & \quad \text{rdfs4a} \\
u \text{ rdf:type rdfs:Resource} & .
\end{align*}
\]

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

\[
\begin{align*}
1 & \text{ blank node} \\
\end{align*}
\]

In blank nodes
RDF Entailment Rules (no Datatypes & Literals)

\[
\begin{align*}
\text{rdf1} & : \quad \frac{a \text{ type} \text{ rdf:Property}}{\text{type}(a, \text{ rdf:Property}) \leftarrow \text{rel}(u, a, y)} \\
\text{rdfs2} & : \quad \frac{a \text{ rdfs:domain} x . ~ u \ a \ y .}{\text{type}(u, x) \leftarrow \text{domain}(a, x) \land \text{rel}(u, a, y)} \\
\text{rdfs3} & : \quad \frac{a \text{ rdfs:range} x . ~ u \ a \ v .}{\text{type}(v, x) \leftarrow \text{range}(a, x) \land \text{rel}(u, a, v)} \\
\text{rdfs4a} & : \quad \frac{u \ a \ x .}{\text{type}(u, \text{ rdfs:Resource}) \leftarrow \text{rel}(u, a, x)}
\end{align*}
\]

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

\[ \frac{u \ a \ v \ . \ v \ \text{rdf:type} \ \text{rdfs:Resource} \ . \ rdfs4b}{u \ \text{rdfs:subPropertyOf} \ v \ . \ v \ \text{rdfs:subPropertyOf} \ x \ . \ u \ \text{rdfs:subPropertyOf} \ x \ . \ rdfs5} \]

\[ \frac{u \ \text{rdfs:subPropertyOf} \ v \ . \ v \ \text{rdfs:subPropertyOf} \ x \ . \ u \ \text{rdfs:subPropertyOf} \ u \ . \ rdfs6}{a \ \text{rdfs:subPropertyOf} \ b \ . \ u \ a \ y \ . \ rdfs7} \]

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

\[ u \ a \ v . \quad \text{rdfs}4b \]
\[ v \ \text{rdf:type} \ rdfs:Resource . \quad \text{rdfs}4b \]
\[ \sim \text{type}(v, rdfs:Resource) \leftarrow \text{rel}(u, a, v) \]

\[ u \ \text{rdfs:subPropertyOf} \ v . \quad \text{v \ rdfs:subPropertyOf} \ x . \quad \text{rdfs}5 \]
\[ u \ \text{rdfs:subPropertyOf} \ x . \quad \text{rdfs}5 \]

\[ u \ \text{rdf:type} \ rdfs:Property . \quad \text{rdfs}6 \]
\[ u \ \text{rdfs:subPropertyOf} \ u . \quad \text{rdfs}6 \]

\[ a \ \text{rdfs:subPropertyOf} \ b . \quad u \ a \ y . \quad \text{rdfs}7 \]
\[ u \ b \ y . \quad \text{rdfs}7 \]

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)

\[
\begin{align*}
&\text{u a v .} \quad \text{rdfs4b} \\
&\text{v rdfs:type rdfs:Resource .} \\
&\text{rdfs:type(v, rdfs:Resource) ← rel(u, a, v)} \\
\end{align*}
\]

\[
\begin{align*}
&\text{u rdfs:subPropertyOf v .} \quad \text{v rdfs:subPropertyOf x .} \quad \text{rdfs5} \\
&\text{u rdfs:subPropertyOf x .} \\
&\text{rdfs:subPropertyOf(u, x) ← subPropertyOf(u, v) \land subPropertyOf(v, x)} \\
\end{align*}
\]

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

\[
\begin{align*}
&\text{a rdfs:subPropertyOf b .} \quad \text{u a y .} \quad \text{rdfs7} \\
&\text{u b y .} \\
\end{align*}
\]

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

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

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

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

\[
\frac{a \ rdfs:\text{subPropertyOf} \ b \quad u \ a \ y}{u \ b \ y} \quad \text{rdfs7}
\]

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

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

\[ \text{u rdfs\text{:subPropertyOf v .} v rdfs\text{:subPropertyOf x .} \}
\[ \text{u rdfs\text{:subPropertyOf x} .} \]
\[ \leadsto \text{subPropertyOf}(u, x) \leftarrow \text{subPropertyOf}(u, v) \land \text{subPropertyOf}(v, x) \]

\[ \text{u rdf\text{:type rdfs\text{:Property} .} \]
\[ \text{u rdfs\text{:subPropertyOf u} .} \]
\[ \leadsto \text{subPropertyOf}(u, u) \leftarrow \text{type}(u, \text{rdfs\text{:Property})} \]

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

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)

u rdfs:type rdfs:Class . rdfs8
u rdfs:subClassOf rdfs:Resource .

u rdfs:subClassOf x . v rdf:type u . rdfs9
   v rdf:type x .

u rdfs:type rdfs:Class . rdfs10
u rdfs:subClassOf u .

u rdfs:subClassOf v . v rdfs:subClassOf x . rdfs11
   u rdfs:subClassOf 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)

\[
\begin{align*}
\text{u rdf:type rdfs:Class .} & \quad \text{rdfs8} \\
\text{u rdf:subClassOf rdfs:Resource .} & \quad \text{rdfs8} \\
\rightarrow \text{subClassOf(u, rdfs:Resource) \leftarrow type(u, rdfs:Class)} & \\
\text{u rdfs:subClassOf x . v rdf:type u .} & \quad \text{rdfs9} \\
\text{v rdf:type x .} & \\
\text{u rdf:type rdfs:Class .} & \quad \text{rdfs10} \\
\text{u rdfs:subClassOf u .} & \\
\text{u rdfs:subClassOf v . v rdfs:subClassOf x .} & \quad \text{rdfs11} \\
\text{u rdfs:subClassOf x .} & \\
\text{a, b IRIs} & \quad x, y \text{ IRI, blank node or literal} \\
\text{u, v IRI or blank node} & \quad l \text{ literal} \quad n \text{ blank nodes}
\end{align*}
\]
RDF Entailment Rules (no Datatypes & Literals)

\[
\text{u rdf:type rdfs:Class } \quad \text{rdfs8}
\]
\[
\text{u rdf:subClassOf rdfs:Resource . } \quad \text{rdfs8}
\]
\[
\sim subClassOf(u, rdfs:Resource) \leftarrow \text{type}(u, rdfs:Class)
\]

\[
\text{u rdfs:subClassOf x . } \quad \text{v rdf:type u . } \quad \text{rdfs9}
\]
\[
\text{v rdf:type x . } \quad \text{rdfs9}
\]
\[
\sim \text{type(v, x)} \leftarrow \text{subClassOf}(u, x) \land \text{type}(v, x)
\]

\[
\text{u rdf:type rdfs:Class . } \quad \text{rdfs10}
\]
\[
\text{u rdf:subClassOf u . } \quad \text{rdfs10}
\]

\[
\text{u rdfs:subClassOf v . } \quad \text{v rdfs:subClassOf x . } \quad \text{rdfs11}
\]
\[
\text{u rdfs:subClassOf x . } \quad \text{rdfs11}
\]

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

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

\[\text{u rdfs:subClassOf x} . \text{ v rdf:type u} . \]
\[\text{ v rdf:type x} . \]
\[\sim \text{ type}(\text{v}, \text{x}) \leftarrow \text{subClassOf}(\text{u}, \text{x}) \land \text{type}(\text{v}, \text{x})\]

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

\[\text{u rdfs:subClassOf v} . \text{ v rdfs:subClassOf x} . \]
\[\text{ u rdfs:subClassOf x} . \]

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

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

& \text{u \ rdfs:subClassOf \ x} . \quad \text{v \ rdf:type \ u} . \quad \text{rdfs9} \\
& \quad \text{v \ rdf:type \ x} . \quad \text{rdfs9} \\
& \rightarrow \ type(v, x) \leftarrow subClassOf(u, x) \land type(v, x) \\

& \text{u \ rdf:type \ rdfs:Class} . \quad \text{rdfs10} \\
& \quad \text{u \ rdfs:subClassOf \ u} . \quad \text{rdfs10} \\
& \rightarrow \ subClassOf(u, u) \leftarrow type(u, \ rdfs:Class) \\

& \text{u \ rdfs:subClassOf \ v} . \quad \text{v \ rdfs:subClassOf \ x} . \quad \text{rdfs11} \\
& \quad \text{u \ rdfs:subClassOf \ x} . \quad \text{rdfs11} \\
& \rightarrow \ subClassOf(u, x) \leftarrow subClassOf(u, v) \land subClassOf(v, x) \\

\end{align*}
\]

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

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

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

\[
\begin{align*}
\text{u rdf:type rdfs:ContainerMembershipProperty} & \quad \rightarrow \quad \text{rdfs12} \\
\text{u rdfs:subPropertyOf rdfs:member} & \\
\rightarrow \quad \text{subPropertyOf(u, rdfs:member) \leftarrow type(u, rdfs:ContainerMembershipProperty)}
\end{align*}
\]

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