CS 7810 - Knowledge Representation and Reasoning (for the Semantic Web)

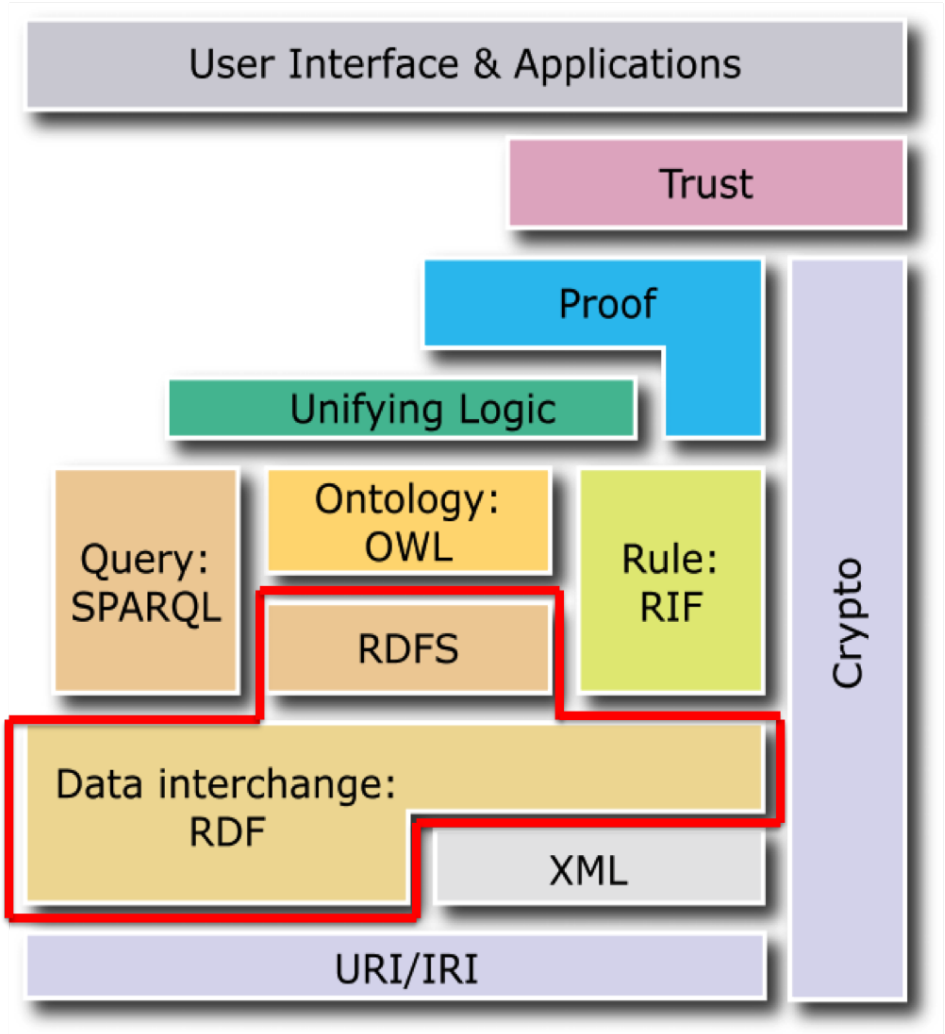
04 - RDF Semantics

Adila Krisnadhi

Data Semantics Lab
Wright State University, Dayton, OH

September 13, 2016

Outline


User Interface & Applications
Trust
Proof
Unifying Logic
Query: SPARQL
Ontology: OWL
Rule: RIF
RDFS
Crypto
Data interchange: RDF
XML
URI/IRI


W3C
RDF

Acknowledgements

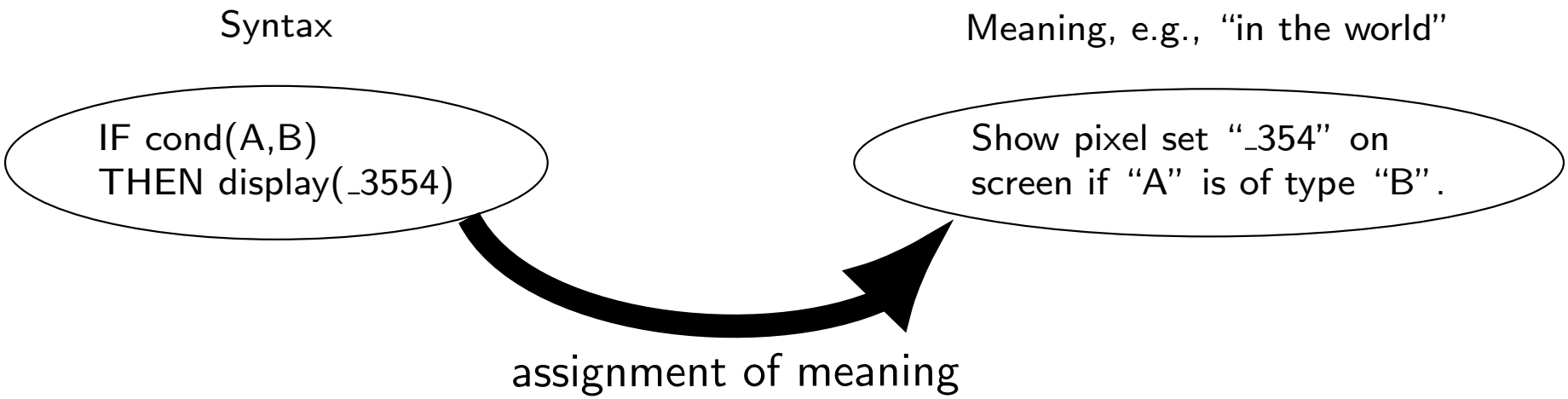
Most of the slides in this presentation are adapted from:

- Sebastian Rudolph, "Semantics of RDF(S)", slides for Foundations of Semantic Web Technologies course, Dresden, April 25, 2014.
- Pascal Hitzler, "Slides 4 01/12/2012", slides for Knowledge Representation for the Semantic Web course, Winter quarter 2012.

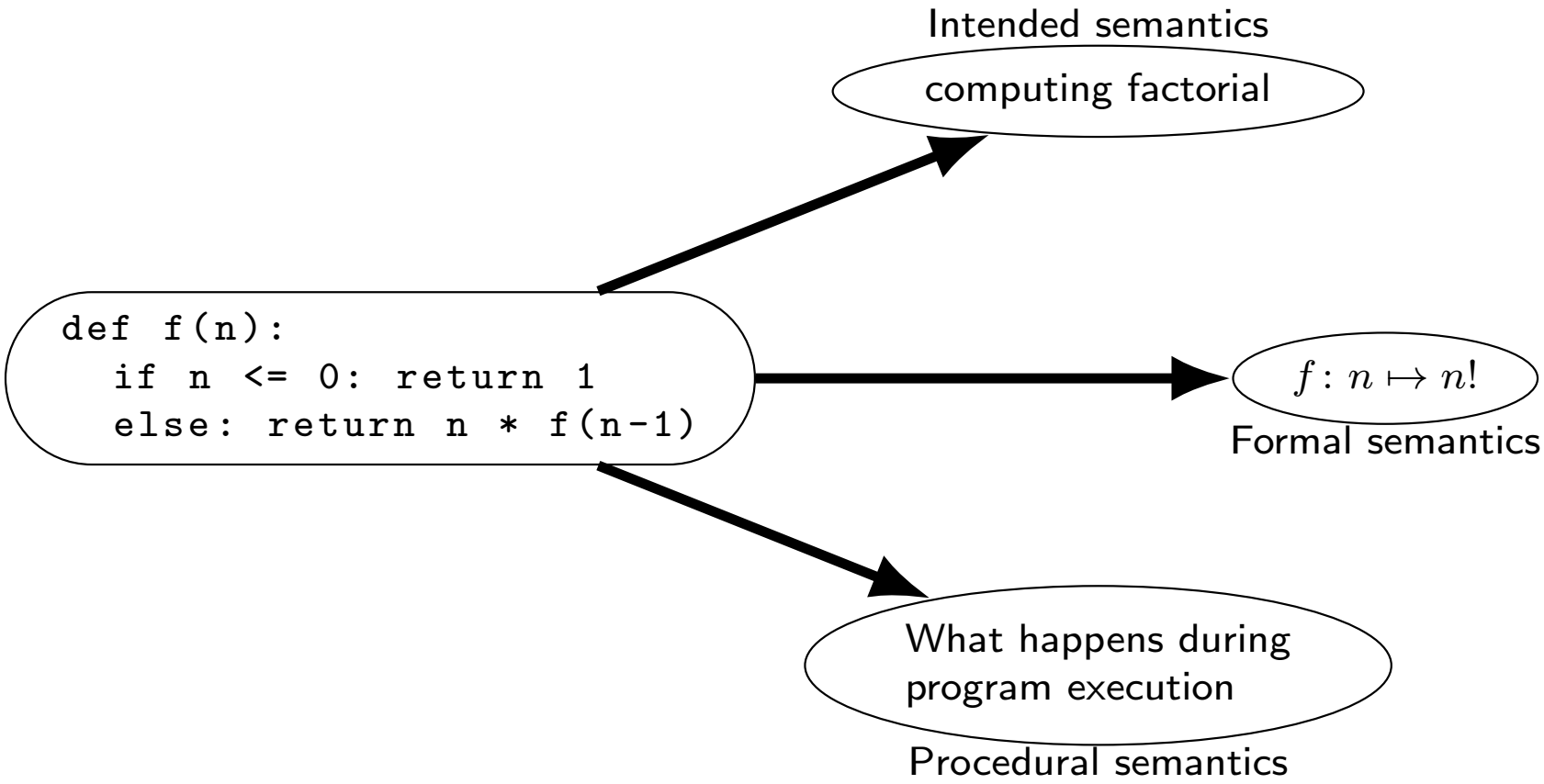
Outline

Syntax and Semantics

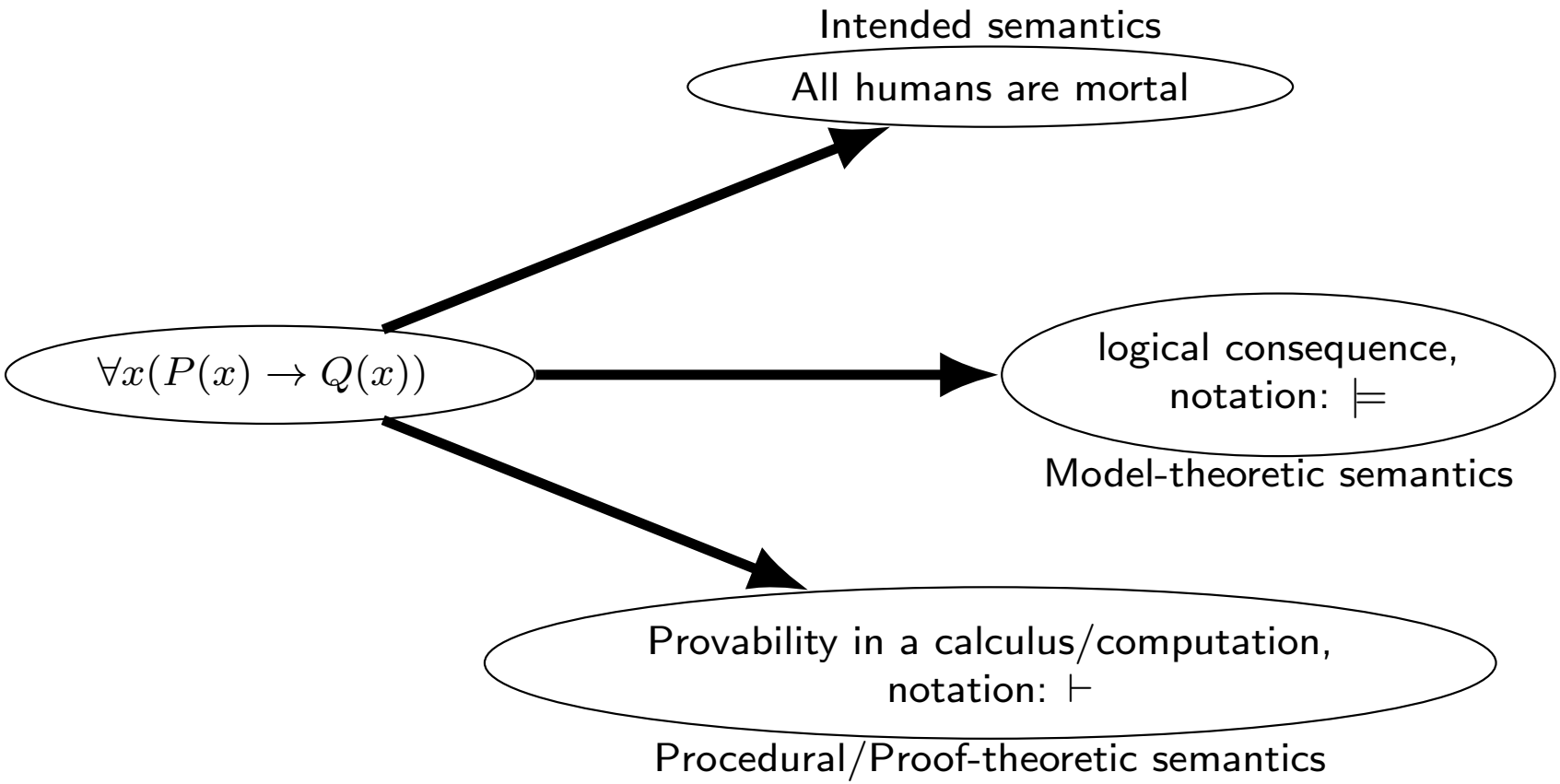
- Syntax: character strings without meaning.
- Semantics: meaning of character strings, e.g., in the world.
  - To connect syntax to semantics, we usually define a function/mapping that assigns a meaning to every syntax element.



Semantics in Programming Languages



Semantics of Logic: Example



Why Formal Semantics

DaSe Lab

- Criticism of tool developers: incompatibility of different tools (despite existing specification).
- Same RDF document, same SPARQL query, but different answers.
- Needs shareable, declarative, computable semantics.
- Ontology languages provide semantics formally based on a logical consequence relation.
- We capture the meaning of information, not by specifying the meaning (which is actually impossible), but by specifying how information interacts with other information.
  - Meaning is described indirectly through its effects.

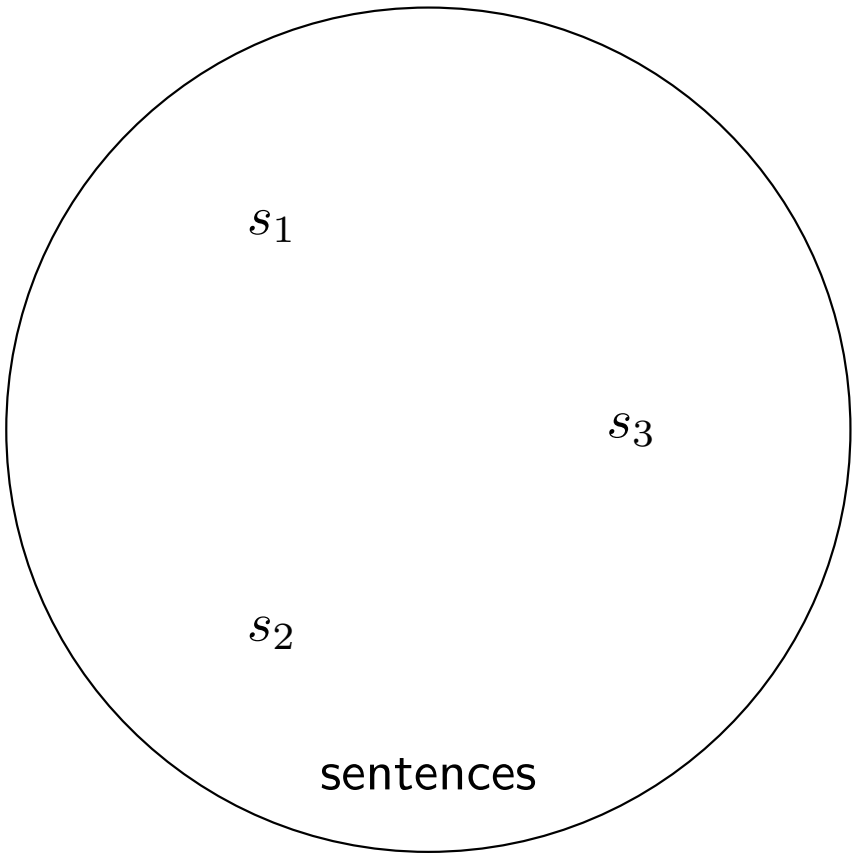
Model-theoretic Semantics

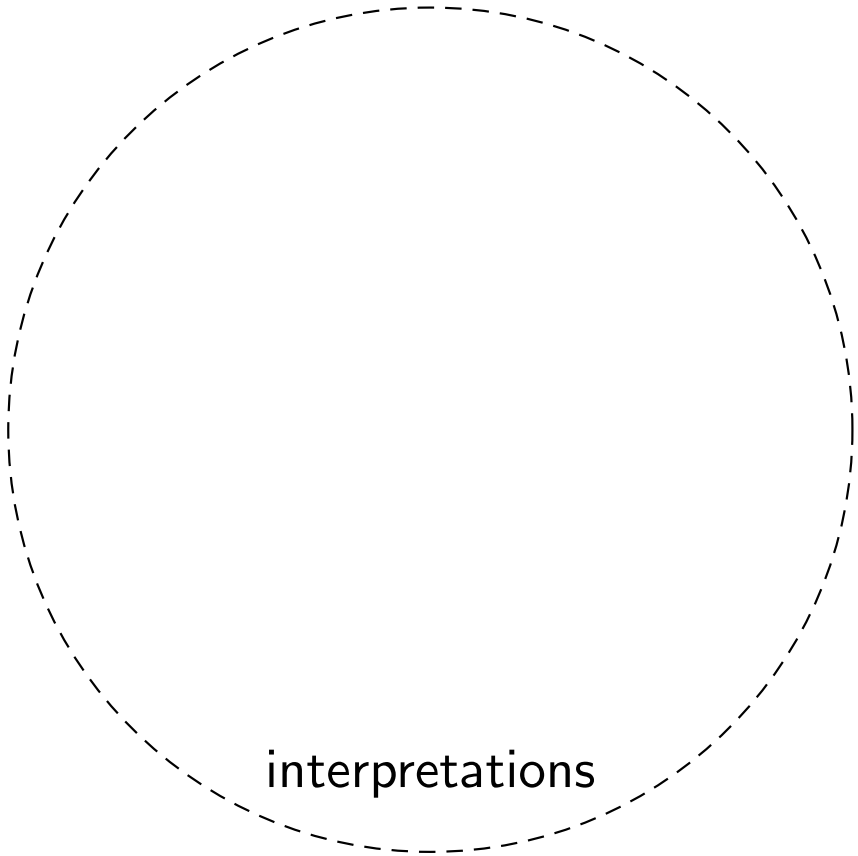
DaSe Lab

- Given a language/syntax (whose elements are sentences), we define a set of *interpretations*:
  - Each interpretation are made such that for each sentence, it is always possible to determine if the sentence is true or false.
  - If a sentence α is true in an interpretation $\mathcal{I}$, we say that $\mathcal{I}$ is a *model* of α, written $\mathcal{I} \models \alpha$.
  - An interpretation $\mathcal{I}$ is a model of a set K of sentences, written $\mathcal{I} \models K$, if $\mathcal{I} \models \alpha$ for every $\alpha \in K$.
- A sentence β is a *logical consequence* of α, written $\alpha \models \beta$, if for every interpretation $\mathcal{I}$ with $\mathcal{I} \models \alpha$, we also have that $\mathcal{I} \models \beta$.
- If K is a set of sentences, we write $K \models \beta$ if for each $\mathcal{I}$ such that $\mathcal{I} \models K$, we have that $\mathcal{I} \models \beta$.
- If J is another set of sentences, we write $K \models J$ if for $K \models \beta$ for each $\beta \in J$.

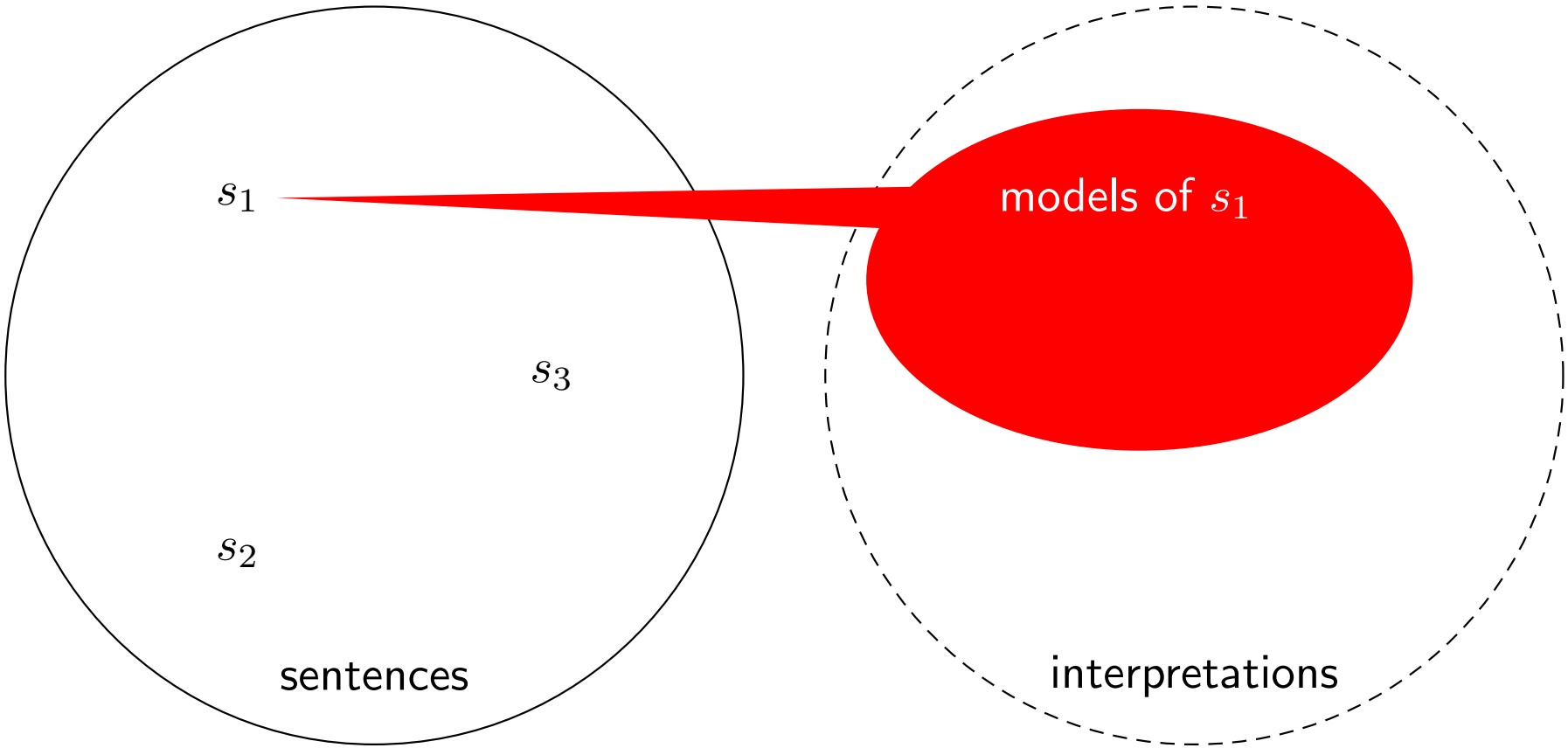
Note that the notation $\models$ is overloaded.

Model-theoretic Semantics

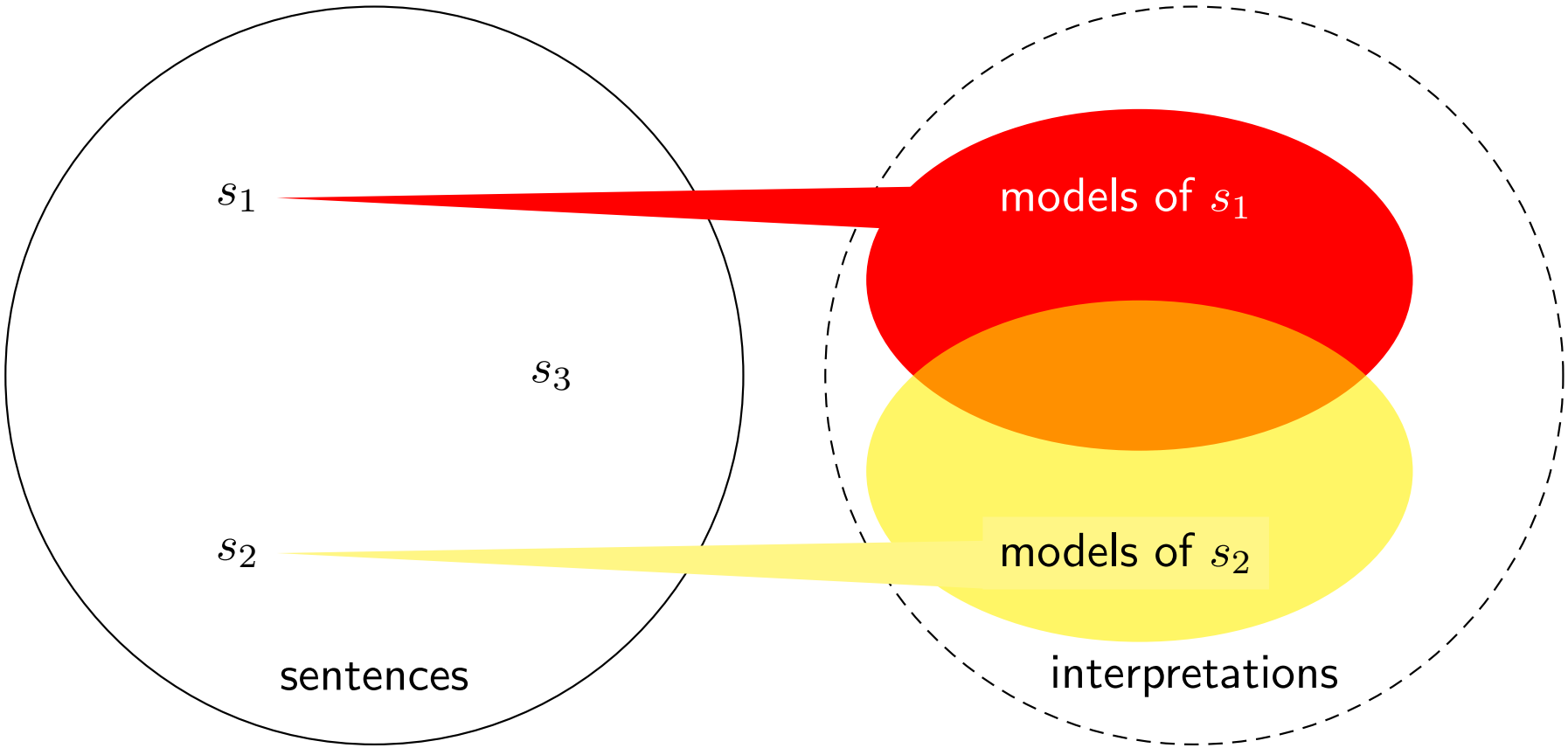




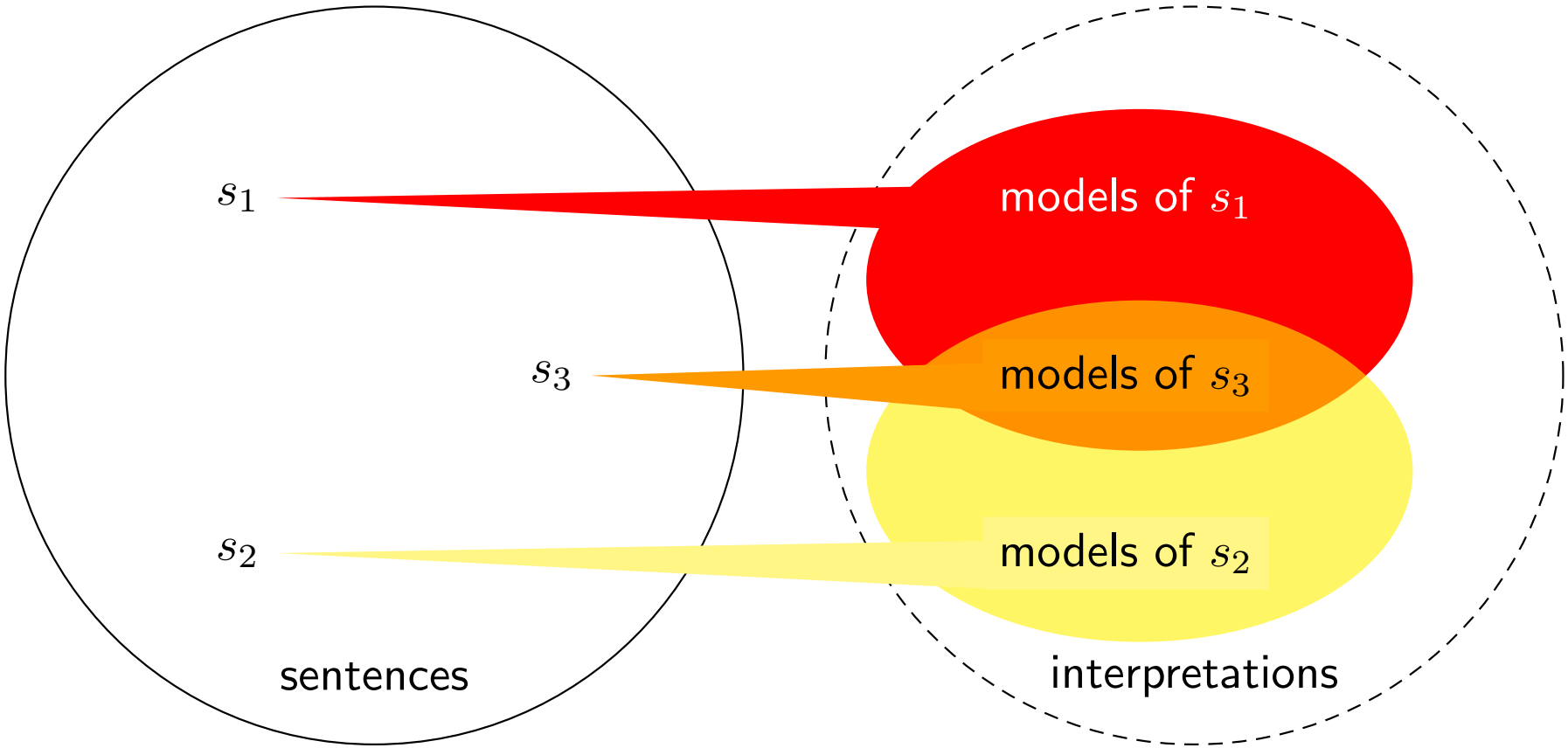
Model-theoretic Semantics



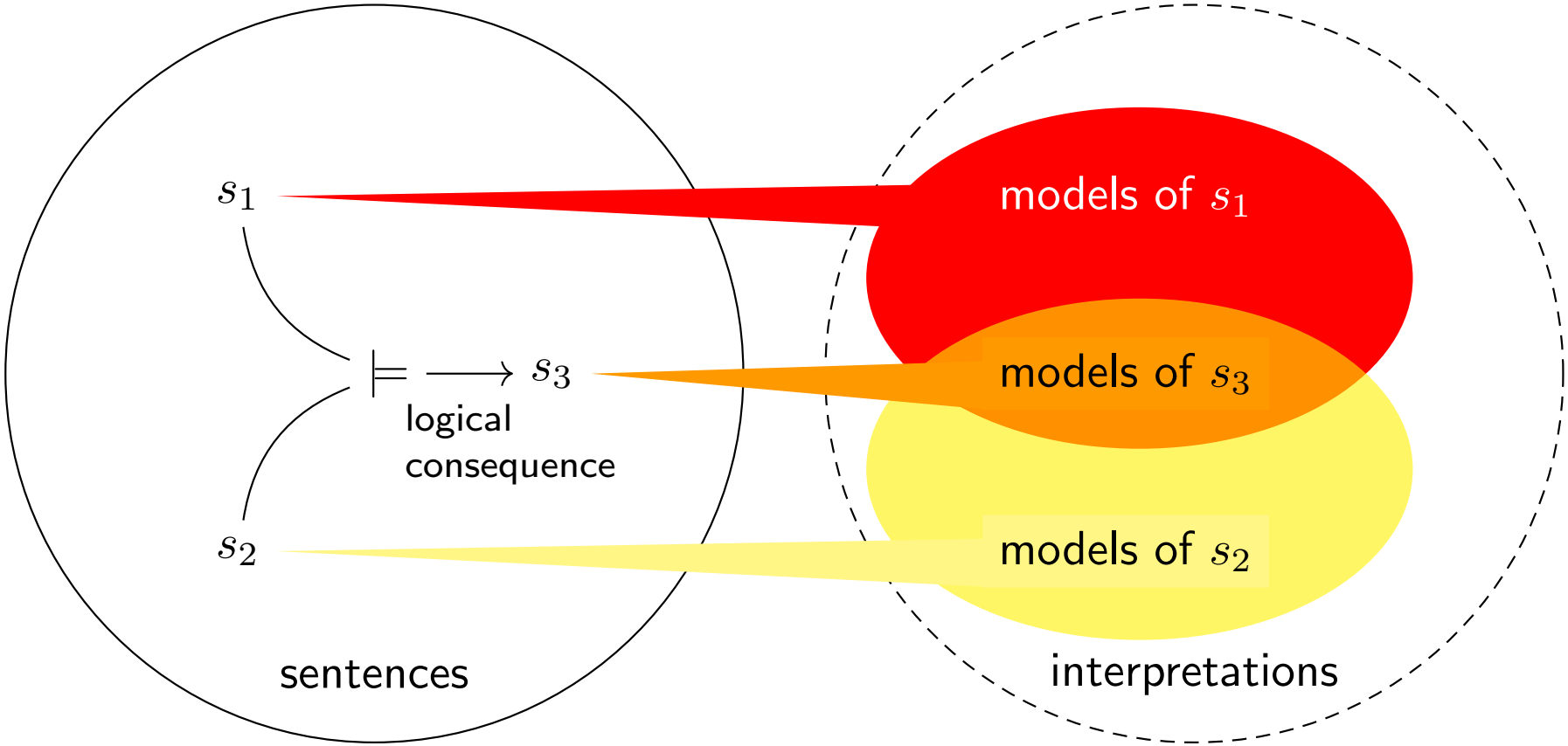
Model-theoretic Semantics



Model-theoretic Semantics



Model-theoretic Semantics



Example for Model-theoretic Semantics

- Language L:
  variables $\ldots, w, x, y, z, \ldots$
  symbol $\boxdot$
  allowed sentences: $a \boxdot b$ for any variables a, b

Example for Model-theoretic Semantics

- Language L:
  variables $\ldots, w, x, y, z, \ldots$
  symbol $\boxdot$
  allowed sentences: $a \boxdot b$ for any variables a, b
- What are the logical consequences for the set $\{x \boxdot y, y \boxdot z\}$?
  - We need to define what a model is for our language.

Example for Model-theoretic Semantics

- Language L:
  variables $\ldots, w, x, y, z, \ldots$
  symbol $\boxdot$
  allowed sentences: $a \boxdot b$ for any variables a, b
- What are the logical consequences for the set $\{x \boxdot y, y \boxdot z\}$?
  - We need to define what a model is for our language.
- An interpretation $\mathcal{I}$ for our language consists of:
  - a set Δ of cars
  - a function $\mathcal{I}(\cdot)$ such that:
    - $\mathcal{I}(x)$ is a car in Δ for every variable x;
    - $\mathcal{I}(x \boxdot y) = \mathbf{true}$ iff $\mathcal{I}(x)$ has more mileage than $\mathcal{I}(y)$

  Given a sentence s, when $\mathcal{I}(s) = \mathbf{true}$, we say that $\mathcal{I}$ is a model for s.

Example for Model-theoretic Semantics

DaSe Lab

- Language L:
  variables $\ldots, w, x, y, z, \ldots$
  symbol $\boxdot$
  allowed sentences: $a \boxdot b$ for any variables a, b
- What are the logical consequences for the set $\{x \boxdot y, y \boxdot z\}$?
  - We need to define what a model is for our language.
- An interpretation $\mathcal{I}$ for our language consists of:
  - a set Δ of cars
  - a function $\mathcal{I}(\cdot)$ such that:
    - $\mathcal{I}(x)$ is a car in Δ for every variable x;
    - $\mathcal{I}(x \boxdot y) = \mathbf{true}$ iff $\mathcal{I}(x)$ has more mileage than $\mathcal{I}(y)$

  Given a sentence s, when $\mathcal{I}(s) = \mathbf{true}$, we say that $\mathcal{I}$ is a model for s.
- Give an example of a model for $x \boxdot y$, and an example of interpretation that is not a model of $x \boxdot y$.
- Can you show that $\{x \boxdot y, y \boxdot z\} \models x \boxdot z$?

Computing Logical Consequences

- In principle, we wish to have an algorithm capable of computing logical consequences of a set of sentences.

Computing Logical Consequences

- In principle, we wish to have an algorithm capable of computing logical consequences of a set of sentences.
- Desiderata: termination, soundness, completeness.

Computing Logical Consequences

- In principle, we wish to have an algorithm capable of computing logical consequences of a set of sentences.
- Desiderata: termination, soundness, completeness.
- An algorithm terminates if it never runs indefinitely given any possible input.

Computing Logical Consequences

- In principle, we wish to have an algorithm capable of computing logical consequences of a set of sentences.
- Desiderata: termination, soundness, completeness.
- An algorithm terminates if it never runs indefinitely given any possible input.
- An algorithm is sound: if it never produces an incorrect solution.
  - If an unsound algorithm produces a solution, we cannot be sure that the solution is correct.
  - Soundness guarantees that the algorithm produces no false positives (i.e., 100% precision).

Computing Logical Consequences

- In principle, we wish to have an algorithm capable of computing logical consequences of a set of sentences.
- Desiderata: termination, soundness, completeness.
- An algorithm terminates if it never runs indefinitely given any possible input.
- An algorithm is sound: if it never produces an incorrect solution.
  - If an unsound algorithm produces a solution, we cannot be sure that the solution is correct.
  - Soundness guarantees that the algorithm produces no false positives (i.e., 100% precision).
- An algorithm is complete: it always finds a solution if one exists; otherwise it correctly reports that no solution is possible.
  - An incomplete algorithm may not return some of the correct solutions (though if it is sound, anything it returns is correct).
  - Completeness guarantees that the algorithm produces no false negatives (i.e., 100% recall).

- Problem: given a set K of sentences in the language L, find a set K' of sentences such that $K \models K'$.
  - The description of a problem always specifies, given an instance of a problem, what would be its correct output.
  - In our context, the model theory specifies, given a set K of sentences, the characterization of sentences that would be entailed by K: those sentences for which every model of K is also a model of the sentences.
- Suppose A is an algorithm that solves the previous problem:
  - A is sound if, given any input K, it outputs a set K' with $K \models K'$.
  - A is complete if, given any input K, every set K' of sentences that can be entailed by K can also be returned by A.

- Algorithm input: set K of sentences.
  1. Non-deterministically select any two sentences from K. If the first sentence is $a \boxdot b$ and the second is $b \boxdot c$, then add $a \boxdot c$ to K.
  2. Repeat step (1) until no selection of two sentences yields a change in K.
  3. Non-deterministically return a non-empty subset of K.

- Algorithm input: set K of sentences.
  1. Non-deterministically select any two sentences from K. If the first sentence is $a \boxdot b$ and the second is $b \boxdot c$, then add $a \boxdot c$ to K.
  2. Repeat step (1) until no selection of two sentences yields a change in K.
  3. Non-deterministically return a non-empty subset of K.
- Does the algorithm terminate?
- Is the algorithm sound? complete?
- The algorithm is non-deterministic. What does it mean?
- Computational complexity of this algorithm?

What do we gain?

- Note that $\alpha \models \beta$ if for all $\mathcal{I}$ with $\mathcal{I} \models \alpha$, $\mathcal{I} \models \beta$ also holds.
- Implementing model-theoretic semantics is not feasible:
  - Must deal with *all* models of a knowledge base.
  - There are a lot of cars in the world, hence we must check a lot of possibilities.
- Proof theory reduces model-theoretic semantics to symbol manipulation!
  - No need to refer to the models directly in the process.

- Step 1 in the previous algorithm can be written as:

  IF $a \boxdot b \in K$ and $b \boxdot c \in K$ THEN $K \leftarrow \{a \boxdot c\}$

- Such a rule is called deduction rule, and are schematically written as:

$$\frac{a \boxdot b \qquad b \boxdot c}{a \boxdot c}$$

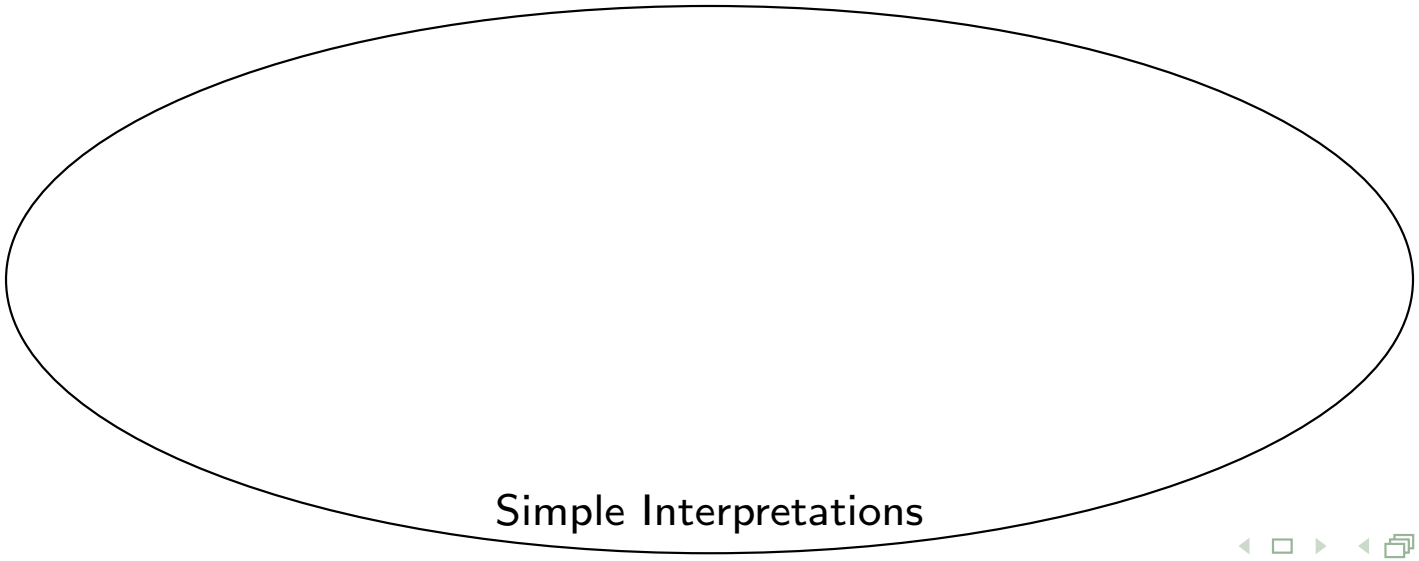
Model-theoretic Semantics for RDF(S)

DaSe Lab

- Language: RDF(S)
  Sentences are triples, possibly containing RDF(S) vocabulary terms. (Graphs are sets of triples).
- Interpretations are given as sets and functions from language vocabulary to these sets.
- Models are defined such that they capture the intended meaning of RDF(S) vocabulary terms.
- Three kinds of interpretations – the more we restrict the set of interpretations the stronger the consequence relation becomes:

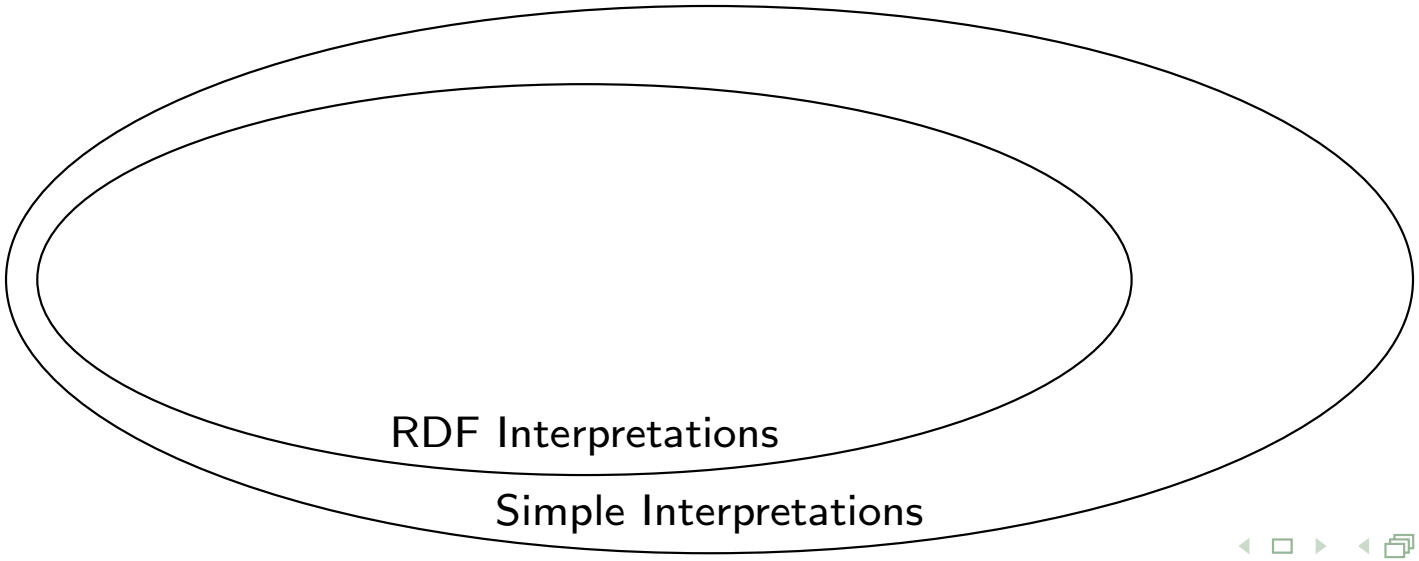
Model-theoretic Semantics for RDF(S)

- Language: RDF(S)
  Sentences are triples, possibly containing RDF(S) vocabulary terms. (Graphs are sets of triples).
- Interpretations are given as sets and functions from language vocabulary to these sets.
- Models are defined such that they capture the intended meaning of RDF(S) vocabulary terms.
- Three kinds of interpretations – the more we restrict the set of interpretations the stronger the consequence relation becomes:



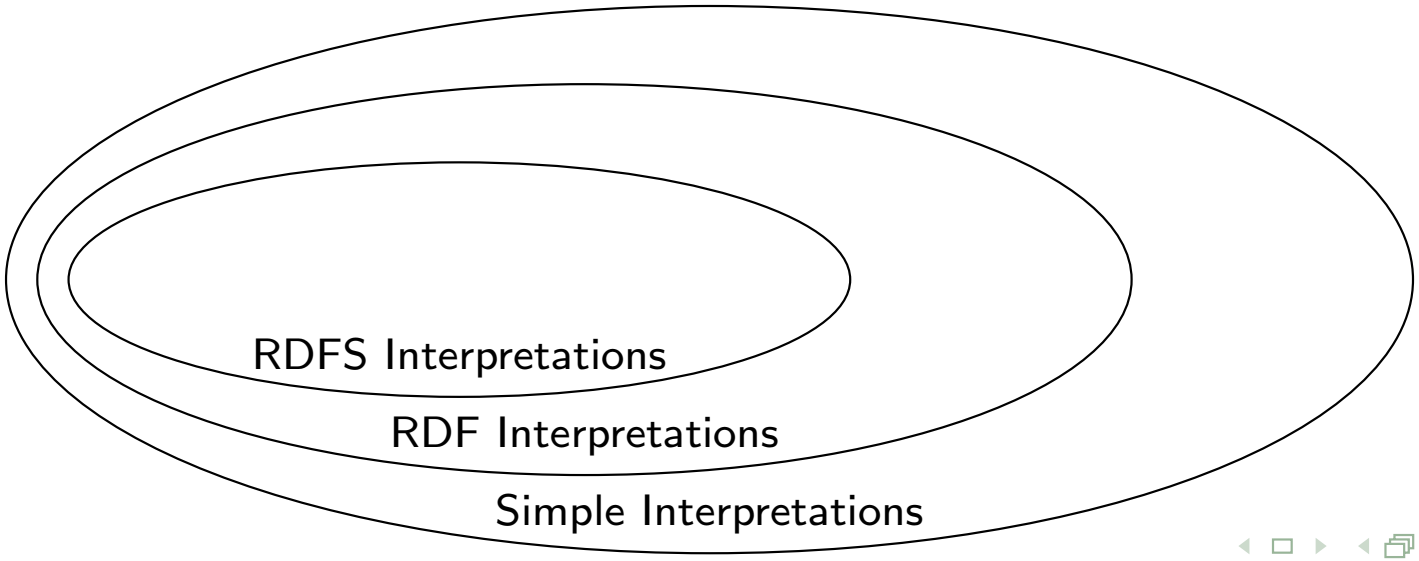
Model-theoretic Semantics for RDF(S)

- Language: RDF(S)
  Sentences are triples, possibly containing RDF(S) vocabulary terms. (Graphs are sets of triples).
- Interpretations are given as sets and functions from language vocabulary to these sets.
- Models are defined such that they capture the intended meaning of RDF(S) vocabulary terms.
- Three kinds of interpretations – the more we restrict the set of interpretations the stronger the consequence relation becomes:



Model-theoretic Semantics for RDF(S)

- Language: RDF(S)
  Sentences are triples, possibly containing RDF(S) vocabulary terms. (Graphs are sets of triples).
- Interpretations are given as sets and functions from language vocabulary to these sets.
- Models are defined such that they capture the intended meaning of RDF(S) vocabulary terms.
- Three kinds of interpretations – the more we restrict the set of interpretations the stronger the consequence relation becomes:



Outline

Simple Interpretation

We first define simple interpretations for ground graphs, i.e., graphs without any occurrence of blank nodes.

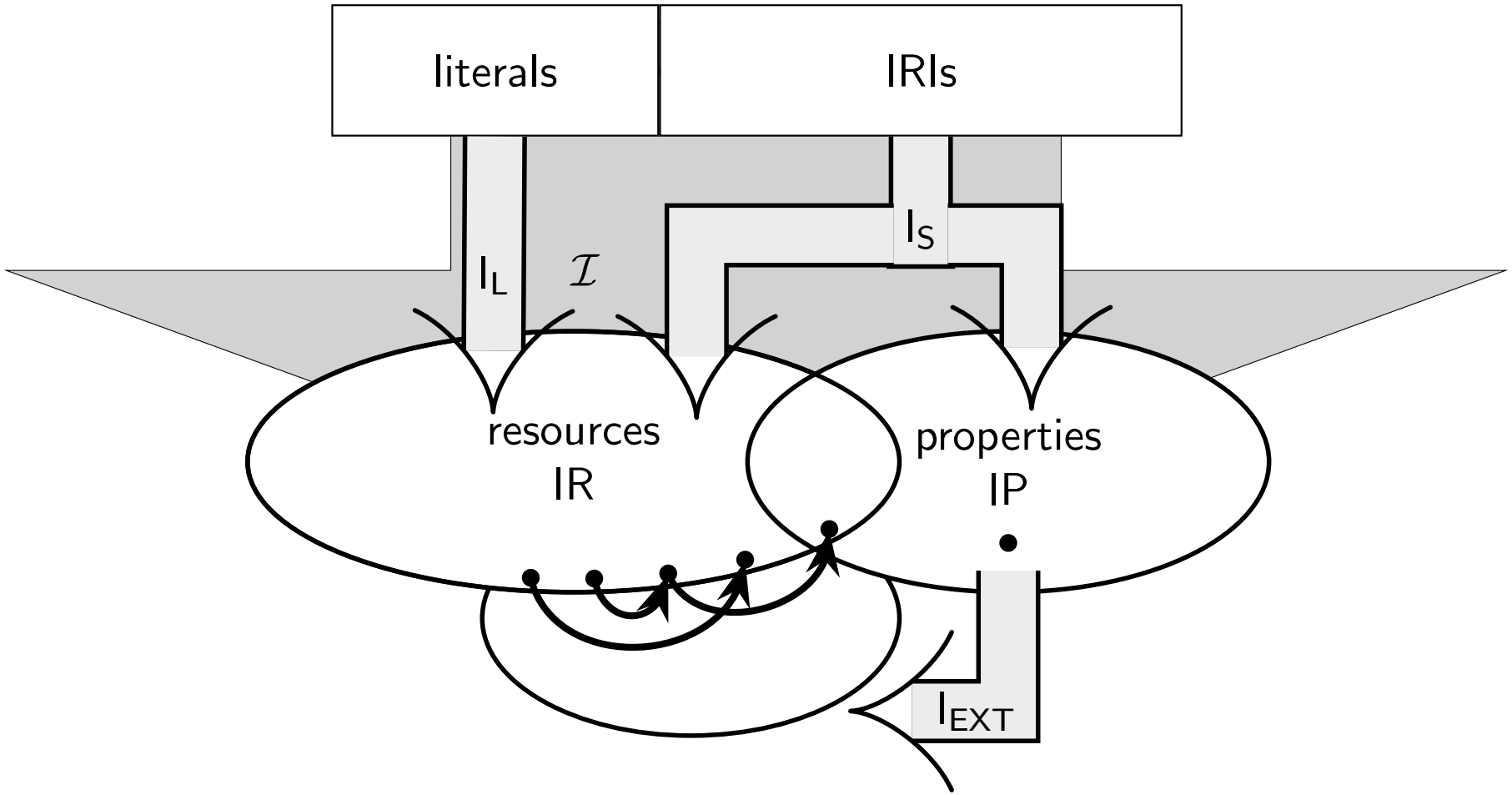
Definition (Simple Interpretation)

A simple interpretation $\mathcal{I}$ for an RDF graph consists of:

- IR, a non-empty set of resources, called the universe of $\mathcal{I}$
- IP, the set of properties of $\mathcal{I}$, possibly overlapping with IR.
- $\mathsf{I_S}$, a mapping from the set of IRIs to IR $\cup$ IP
- $\mathsf{I_{EXT}}$, a mapping from IP to the powerset of IR $\times$ IR, i.e., the set of pairs $\langle x, y \rangle$ with $x, y \in$ IR
- $\mathsf{I_L}$, a partial mapping from the set of literals to IR.

In addition, $\mathcal{I}(\cdot)$ is the corresponding mapping for $\mathcal{I}$ from names/terms, triples, and graphs to the elements of the universe IR and truth values where $\mathcal{I}(\cdot)$ satisfies the semantic conditions given next.

Schematically ...



Semantic Interpretations: Intuition

DaSe Lab

- When is a given interpretation a model of the graph?

Semantic Interpretations: Intuition

- When is a given interpretation a model of the graph?
- ... whenever it is a model for every triple in the graph.

Semantic Interpretations: Intuition

- When is a given interpretation a model of the graph?
- ... whenever it is a model for every triple in the graph.
- When is a given interpretation a model of a triple `s p o`?

Semantic Interpretations: Intuition

- When is a given interpretation a model of the graph?
- ... whenever it is a model for every triple in the graph.
- When is a given interpretation a model of a triple `s p o`?
- ... whenever s, p, and o all are appropriately interpreted as resources in IR (through $\mathsf{I_L}$ and $\mathsf{I_S}$), the predicate correspond to a property in IP, and additionally, $\langle \mathcal{I}(\mathsf{s}), \mathcal{I}(\mathsf{o}) \rangle \in \mathsf{I_{EXT}}(\mathcal{I}(\mathsf{p}))$ holds.

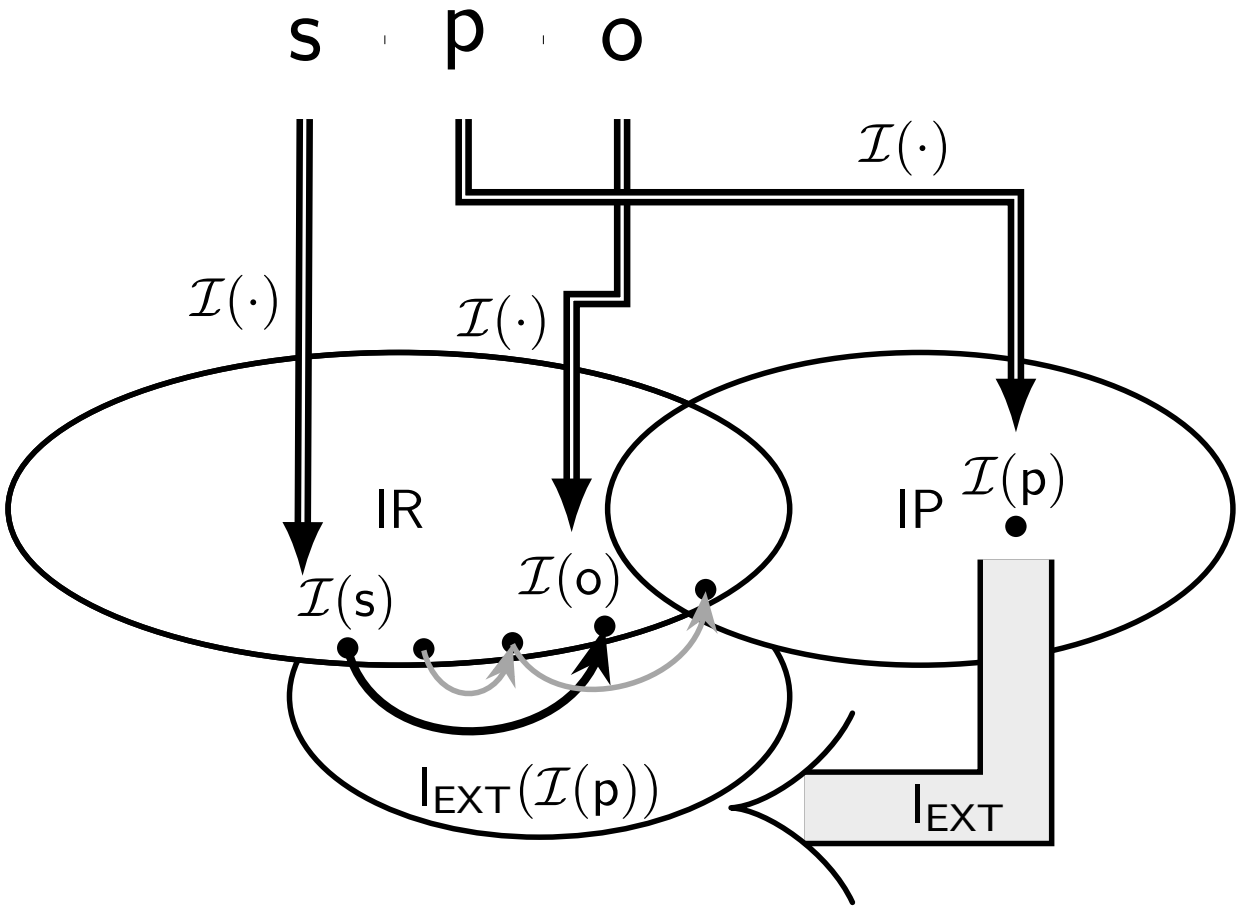
Definition (Semantic Conditions for Ground Graphs)

Let $\mathcal{I}$ be a simple interpretation of a ground graph G. Then,

1. if x is a literal in G, then $\mathsf{I_L}(x) = \mathsf{I_L}(x)$
2. if x is a IRI in G, then $\mathcal{I}(x) = \mathsf{I_S}(x)$
3. for each ground triple s p o in G,
   - $\mathcal{I}(\mathsf{s\ p\ o}) =$ **true** if $\mathcal{I}(\mathsf{p}) \in \mathsf{IP}$ and the pair $\langle \mathcal{I}(\mathsf{s}), \mathcal{I}(\mathsf{o}) \rangle \in \mathsf{I_{EXT}}(\mathsf{p})$
   - otherwise, $\mathcal{I}(\mathsf{s\ p\ o}) =$ **false**
4. $\mathcal{I}(G) =$ **false** if $\mathcal{I}(\mathsf{s\ p\ o}) =$ **false** for some triple s p o in G, otherwise $\mathcal{I}(G) =$ **true**.

- If $\mathsf{I_L}(x)$ is undefined for some literal x, then x has no semantic value, and any triple (and thus any graph) containing it will be false.
- The empty graph is always true under any simple interpretation (but not necessarily so in other notions of interpretation).

Schematically ...



What about blank nodes?

Intuition: extend $\mathcal{I}$ with a special mapping that maps blank nodes to resources.

Definition (Additional mapping for blank nodes)

Let $\mathcal{I}$ be a simple interpretation. For blank nodes, let A be a mapping that maps all blank nodes to elements of IR.
Let $\mathcal{I}_A$ be a mapping from all literals, IRIs, blank nodes to IR where

- $\mathcal{I}_A(x) = A(x)$ if x is a blank node
- $\mathcal{I}_A(x) = \mathcal{I}(x)$ if x is a IRI or literal
- $\mathcal{I}_A(\text{s p o}) =$ **true** if $\mathcal{I}(\text{p}) \in \mathcal{I}$ and $\langle \mathcal{I}_A(\text{s}), \mathcal{I}_A(\text{o}) \rangle \in$ IP, otherwise $\mathcal{I}_A(\text{s p o}) =$ **false**
- $\mathcal{I}_A(G) =$ **false** if $\mathcal{I}_A(\text{s p o}) =$ **false** for some triple s p o in G, otherwise $\mathcal{I}_A(G) =$ **true**.

What about blank nodes?

DaSe Lab

Definition (Semantic Condition for Blank Nodes)

If G is an RDF graph (possibly with blank nodes), then $\mathcal{I}(G) =$ **true** if $\mathcal{I}_A(G) =$ **true** for some mapping A from blank nodes to IR. Otherwise, $\mathcal{I}(G) =$ **false**.

- The semantic condition above modifies the 4th semantic condition for ground graphs.
- If two or more graphs share a blank node, their semantics can only be captured by considering the union of those graphs.

DaSe Lab

Definition

- A simple interpretation $\mathcal{I}$ is a simple model of an RDF graph G iff $\mathcal{I}(G) = \textbf{true}$.
- We also say that $\mathcal{I}$ simply satisfies G.
- G is simply satisfiable if it has a simple model.
- A graph G simply entails a graph G', written $G \models G'$, if every simple interpretation that simply satisfies G also simply satisfies G'.
- G and G' are equivalent if each entails the other.

Simple Interpretations: Example

Given a graph G:

```
@prefix ex: <http://example.org/> .
ex:Chutney ex:hasIngredient [ ex:ingredient  ex:greenMango ;
                              ex:amount      "1 lb" ] .
```

Simple interpretation $\mathcal{I}$:

$$\mathsf{IR} = \{c, g, h, z, l, m, j\} \qquad \mathsf{IP} = \{h, z, m\}$$

$$\mathsf{I_{EXT}} = \{h \mapsto \{\langle c, l\rangle\}, z \mapsto \{\langle l, g\rangle\}, m \mapsto \{\langle l, j\rangle\}\}$$

$$\mathsf{I_S} = \{\texttt{ex:Chutney} \mapsto c, \texttt{ex:hasIngredient} \mapsto h, \texttt{ex:greenMango} \mapsto g,$$
$$\texttt{ex:ingredient} \mapsto z, \texttt{ex:amount} \mapsto m\}$$

$$\mathsf{I_L} = \{\texttt{"1 lb"} \mapsto j\}$$

Is $\mathcal{I}$ a simple model for G?

Simple Interpretations: Example

Given a graph G:

```
@prefix ex: <http://example.org/> .
ex:Chutney ex:hasIngredient [ ex:ingredient  ex:greenMango ;
                              ex:amount      "1 lb" ] .
```

Simple interpretation $\mathcal{I}$:

$$\begin{aligned}
\mathsf{IR} &= \{c, g, h, z, l, m, j\} \qquad \mathsf{IP} = \{h, z, m\} \\
\mathsf{I_{EXT}} &= \{h \mapsto \{\langle c, l\rangle\}, z \mapsto \{\langle l, g\rangle\}, m \mapsto \{\langle l, j\rangle\}\} \\
\mathsf{I_S} &= \{\texttt{ex:Chutney} \mapsto c, \texttt{ex:hasIngredient} \mapsto h, \texttt{ex:greenMango} \mapsto g, \\
&\qquad \texttt{ex:ingredient} \mapsto z, \texttt{ex:amount} \mapsto m\} \\
\mathsf{I_L} &= \{\texttt{"1 lb"} \mapsto j\}
\end{aligned}$$

If we pick a mapping A with $A = \{\texttt{\_:id1} \mapsto l\}$, then:

$$\begin{aligned}
\langle \mathcal{I}_A(\texttt{ex:Chutney}), \mathcal{I}_A(\texttt{_:id1})\rangle &= \langle c, l\rangle \in \mathsf{I_{EXT}}(h) = \mathsf{I_{EXT}}(\mathcal{I}_A(\texttt{ex:hasIngredient})) \\
\langle \mathcal{I}_A(\texttt{_:id1}), \mathcal{I}_A(\texttt{ex:greenMango})\rangle &= \langle l, g\rangle \in \mathsf{I_{EXT}}(z) = \mathsf{I_{EXT}}(\mathcal{I}_A(\texttt{ex:ingredient})) \\
\langle \mathcal{I}_A(\texttt{_:id1}), \mathcal{I}_A(\texttt{"1 lb"})\rangle &= \langle l, j\rangle \in \mathsf{I_{EXT}}(m) = \mathsf{I_{EXT}}(\mathcal{I}_A(\texttt{ex:amount}))
\end{aligned}$$

Therefore, $\mathcal{I}$ is a model for G.

Simple Entailment

- Simple entailment definition describes the condition of when a graph simply entail another graph.
- But it does not say anything regarding how to actually compute the entailment.
- Solution: proof-theoretic semantics in the form of deduction rules (see notation next).

Simple Entailment Deduction Rule Notation

$$\frac{\text{Premise}}{\text{Antecedent}}\ \text{Label}$$

- Premise and Antecedent are sequences of triples (each triple ended with a fullstop) and Label is the name of the rule. Some deduction rules have additional preconditions described together with them.
- The rule is applied to a graph G if G contains triples matching Premise, and the additional preconditions, if any, are satisfied. (If Premise has two triples, then the rule is applied if G contains two triples (possibly the same) matching Premise).
- The result of rule application is the triples given by Antecedent and all of them are added to G.
- `u` and `v` refer to arbitrary IRIs or blank node IDs (any possible subject of a triple).
- `p` and `q` refer to arbitrary IRIs (any possible predicate of a triple).
- `x` and `y` refer to arbitrary IRIs, blank node IDs or literals (any possible object of a triple).
- Literals are denoted with `l` or sometimes, a string of the form `"lll"^^d` where `d` refers to a datatype IRI.

Simple Entailment Deduction Rules

Definition (se1)

$$\frac{\texttt{u p x .}}{\texttt{u p \_:n .}}\ \text{se1}$$

Precondition: blank node `_:n` was created for (IRI/literal/blanknode) `x` by earlier application of se1 or se2; if no such `_:n` exists, `_:n` must be a new blank node not yet occurring in the graph.

Definition (se2)

$$\frac{\texttt{u p x .}}{\texttt{\_:n p x .}}\ \text{se2}$$

Precondition: blank node `_:n` was created for (IRI/blank node) `u` by earlier application of se1 or se2; if no such `_:n` exists, `_:n` must be a new blank node not yet occurring in the graph.

Simple Entailment Deduction Rules

DaSe Lab

Theorem

A graph G_1 simply entails a graph G_2 if G_1 can be extended to a graph G_1' by applying the rules se1 and se2 such that $G_2 \subseteq G_1'$.

- Algorithm: apply se1 and se2 as needed until obtaining G_1' such that $G_2 \subseteq G_1'$
- The algorithm terminates? Why?
- The algorithm is sound? Why?
- The algorithm is complete? Why?

Simple Entailment: Example

The graph:

```
ex:fost  ex:publishedBy crc:uri  ;
         ex:title "Foundations of Semantic Web Technologies" .
crc:uri  ex:name "CRC Press" .
```

entails (how?)

```
ex:fost  ex:publishedBy [ ex:name "CRC Press" ;
                          ex:name [] ] .
```

Simple Entailment: Example

G_1:

```
ex:adila foaf:knows ex:pascal .
ex:adila foaf:name "Adila" .
foaf:knows rdfs:domain foaf:Person .
```

G_2:

```
_:adila foaf:knows ex:pascal .
_:adila foaf:name _:name .
```

$G_1 \models G_2$?

Simple Entailment: Example

G_1:

```
ex:adila foaf:knows ex:pascal .
ex:adila foaf:name "Adila" .
foaf:knows rdfs:domain foaf:Person .
```

G_2:

```
_:adila foaf:knows ex:pascal .
_:adila foaf:name _:name .
```

$G_1 \models G_2$.

G_3:

```
_:adila foaf:knows ex:pascal .
_:adila foaf:name _:adila .
```

$G_1 \models G_3$?

Simple Entailment: Example

G_1:

```
ex:adila foaf:knows ex:pascal .
ex:adila foaf:name "Adila" .
foaf:knows rdfs:domain foaf:Person .
```

G_2:

```
_:adila foaf:knows ex:pascal .
_:adila foaf:name _:name .
```

$G_1 \models G_2$.

G_3:

```
_:adila foaf:knows ex:pascal .
_:adila foaf:name _:adila .
```

$G_1 \not\models G_3$

G_4:

```
ex:adila rdf:type foaf:Person .
```

Simple Entailment: Example

G_1:

```
ex:adila foaf:knows ex:pascal .
ex:adila foaf:name "Adila" .
foaf:knows rdfs:domain foaf:Person .
```

G_2:

```
_:adila foaf:knows ex:pascal .
_:adila foaf:name _:name .
```

$G_1 \models G_2$.

G_3:

```
_:adila foaf:knows ex:pascal .
_:adila foaf:name _:adila .
```

$G_1 \not\models G_3$

G_4:

```
ex:adila rdf:type foaf:Person .
```

$G_1 \models G_4$?

Simple Entailment: Example

G_1:

```
ex:adila foaf:knows ex:pascal .
ex:adila foaf:name "Adila" .
foaf:knows rdfs:domain foaf:Person .
```

G_2:

```
_:adila foaf:knows ex:pascal .
_:adila foaf:name _:name .
```

$G_1 \models G_2$.

G_3:

```
_:adila foaf:knows ex:pascal .
_:adila foaf:name _:adila .
```

$G_1 \not\models G_3$

G_4:

```
ex:adila rdf:type foaf:Person .
```

$G_1 \not\models G_4$

Need ontology language to specify particular meaning of `rdf:type` and `rdfs:domain` to infer G_4 from G_1.

How about (typed) literals?

- Recall that every datatype would have a lexical-to-value mapping.
- Let d be a datatype. Then, we denote the lexical-to-value mapping for d with L2V_d, itself maps any literal string `sss` in the lexical space of d to its corresponding value in the value space of d.
- E.g., $\text{L2V}_{\text{integer}}(\texttt{"01"}) = 1$.
- If `sss` is not in the lexical space of d, then $\text{L2V}_d(\texttt{sss})$ is undefined.
- Literals of the form `"sss"` (i.e., without explicitly specified datatype) is syntactic sugar to the literals `"sss"^^xsd:string`.
- Languaged-tagged literals always (implicitly) have `rdf:langString` as their datatype.

Ill-typed Literals

- An ill-typed literal is a literal whose datatype IRI is recognized, but its value is undefined according to the lexical-value mapping of the datatype.
- E.g., $\text{L2V}_{\text{integer}}(\texttt{"adila"})$ is undefined.
- RDF allows any IRI to be used as the datatype IRI of a typed literal, *even when it is not recognized as a datatype*.
  - This should not be treated as errors, although applications may issue a warning.
  - Such literals should be treated like IRIs, i.e., denoting something in the universe IR.
  - RDF processors which do not recognize a datatype IRI will not be able to detect some entailments relevant to the unrecognized datatype IRI.
- We modify the notion of simple interpretation to accommodate datatypes next.

DaSe Lab

Definition

Let D be a set of IRIs identifying datatypes. A simple D-interpretation $\mathcal{I}$ is a simple interpretation that satisfies the following conditions:

- If `rdf:langString` $\in D$, then for every language-tagged string `"sss"@ttt`, $\mathrm{I_L}(\texttt{"sss"@ttt}) = \langle \texttt{sss}, \texttt{ttt}' \rangle$ where `ttt`' is `ttt` converted to lower case using US-ASCII rules.
- For every other datatype IRI `ddd` $\in D$, $\mathcal{I}(\texttt{ddd})$ is the datatype identified by `ddd`, and for every literal `"sss"^^ddd`, $\mathrm{I_L}(\texttt{"sss"\^\^ddd}) = \mathrm{L2V}_{\mathcal{I}(\texttt{ddd})}(\texttt{sss})$

- If the literal is ill-typed, $\mathrm{L2V}_{\mathcal{I}(\texttt{ddd})}$ is undefined on it. Thus, the literal cannot denote anything, and in this case, any triple containing it must be false.
- Hence a graph containing an ill-typed literal is always false.
- `rdf:langString` has no ill-typed literals.
- The only ill-typed literals of `xsd:string` are those containing Unicode point that does not match the Char production rule in XML specification (they cannot be written in any XML-compatible surface syntax).

Datatype Entailment

- A graph is (simply) D-satisfiable (or satisfiable recognizing D) if it is true in some D-interpretation. If no such D-interpretation exists, then the graph is D-unsatisfiable.
- A graph G (simply) D-entails a graph G' if every D-interpretation that satisfies G also D-satisfies G'.
- A triple

  `s p "xxx"^^ddd`

  entails the triple

  `s p "yyy"^^eee`

  whenever `"xxx"^^ddd` and `"yyy"^^eee` map to the same value (provided that `ddd` and `eee` are both recognized datatypes and have the *same primitive base* datatype).
- A D-unsatisfiable graph D-entails any graph. (Why?)

Datatype Entailment: Example

```
eg:Jane eg:age "15"^^xsd:nonNegativeInteger .
```

entails

```
eg:Jane eg:age "15"^^xsd:byte .
```

as well as

```
eg:Jane eg:age "15.0"^^xsd:decimal .
```

because both `xsd:byte`, `xsd:nonNegativeInteger`, and `xsd:decimal` have the same primitive base datatype (`xsd:decimal`) and the literals denote the same value.

Datatype Entailment: Example 2

DaSe Lab

```
eg:car eg:engineSizeInLitres "1.3"^^xsd:decimal .
```

does not entail

```
eg:car eg:engineSizeInLitres "1.3"^^xsd:float .
```

`xsd:decimal` has a different primitive base datatype to `xsd:float`.

Datatype Entailment: Example 3

DaSe Lab

```
eg:car eg:engineSizeInLitres "1.3"^^xsd:double .
```

does not entail

```
eg:car eg:engineSizeInLitres "1.3"^^xsd:float .
```

`xsd:double` has a different primitive base datatype to `xsd:float`.

Outline

RDF Interpretations

DaSe Lab

An RDF interpretation is a simple D-interpretation where additional semantic conditions are imposed on the IRIs of the RDF vocabulary (below).

```
rdf:type rdf:Property rdf:XMLLiteral rdf:nil
rdf:List rdf:Statement rdf:subject rdf:predicate
rdf:object rdf:first rdf:rest rdf:Seq rdf:Bag
rdf:Alt rdf:_1 rdf:_2 ...
```

Definition

An RDF interpretation (recognizing D) is a D-interpretation $\mathcal{I}$ where D includes `rdf:langString` and `xsd:string` and

- $x \in \mathsf{IP}$ if and only if $\langle x, \mathcal{I}(\texttt{rdf:Property})\rangle \in \mathsf{I_{EXT}}(\mathcal{I}(\texttt{rdf:type}))$;
- For every IRI `uuu` in D, $\langle x, \mathcal{I}(\texttt{uuu})\rangle \in \mathsf{I_{EXT}}(\mathcal{I}(\texttt{rdf:type})$ if and only if x is in the value space of $\mathcal{I}(\texttt{uuu})$; and
- $\mathcal{I}$ satisfies every triple in the following infinite set (called axiomatic triples):

```
rdf:type rdf:type rdf:Property .
rdf:subject rdf:type rdf:Property .
rdf:predicate rdf:type rdf:Property .
rdf:object rdf:type rdf:Property .
rdf:first rdf:type rdf:Property .
rdf:rest rdf:type rdf:Property .
rdf:value rdf:type rdf:Property .
rdf:nil rdf:type rdf:List .
rdf:_1 rdf:type rdf:Property .
rdf:_2 rdf:type rdf:Property .
...
```

RDF Entailment

- Graph G RDF-entails a graph G' recognizing D when every RDF interpretation recognizing D that satisfies G also satisfies G'.
  - D must contain `rdf:langString` and `xsd:string`.
- G is RDF unsatisfiable (recognizing D) when it has no satisfying RDF interpretation (recognizing D).

RDF Entailment Deduction Rules I

- RDF entailment deduction rules include rules se1 and se2 described earlier.
- Formally, we need to account for D-entailments.
  - Unfortunately, there is no single global syntactic criterion to detect all D-entailments.
  - RDF semantics only requires detection of entailments involving literals in `rdf:langString` and `xsd:string`.
- The remaining rules are as follows.

Definition (rdfD1)

$$\frac{\texttt{u p "lll"\^\^d .}}{\texttt{u p \_:n .} \quad \texttt{\_:n rdf:type d .}}\ \text{rdfD1}$$

Precondition: blank node `_:n` is not associated with other IRIs/blank nodes/literals; needs to be newly created.

RDF Entailment Deduction Rules II

Definition (rdfD2)

$$\frac{\texttt{u p y .}}{\texttt{p rdf:type rdf:Property .}}\ \text{rdfD2}$$

Definition (rdfax)

For every **axiomatic** triples `u p x .` in the definition of RDF interpretation:

$$\frac{}{\texttt{u p x .}}\ \text{rdfax}$$

RDF Entailment Algorithm via Deduction Rules

To check if a graph G_1 RDF-entails a graph G_2, we perform the following:

1. Add all RDF axiomatic triples to G_1 according to rdfax, except those containing the container membership property IRIs `rdf:_1`, `rdf:_2`, ...
2. For every container membership property IRI occurring in G, add the RDF axiomatic triples that contain that IRI.
3. Apply rdfD1 and rdfD2 exhaustively to G_1 until they generate no new triples. Let G_1' be the resulting graph.
4. If G_1' simply entails G_2, then G_1 RDF-entails G_2.

Theorem

The algorithm terminates and is sound.

RDF Entailment via Deduction Rules II

- The algorithm, however, is **not** complete. For example, the graph G_1 RDF-entails the graph G_2 below, but cannot be derived by the algorithm.

G_1:

```
ex:a ex:p "string"^^xsd:string .
ex:b ex:q "string"^^xsd:string .
```

G_2:

```
ex:a ex:p _:b .
ex:b ex:q _:b .
_:b rdf:type xsd:string .
```

Outline

RDFS Interpretation

An RDFS interpretation is an RDF interpretation where additional semantic conditions are applied to RDFS vocabulary below:

```
rdfs:domain rdfs:range rdfs:Resource rdfs:Literal rdfs:Datatype
rdfs:Class rdfs:subClassOf rdfs:subPropertyOf rdfs:member
rdfs:Container rdfs:ContainerMembershipProperty rdfs:comment
rdfs:seeAlso rdfs:isDefinedBy rdfs:label
```

Definition (RDFS Interpretation)

An RDFS interpretation (recognizing D) is an RDF interpretation (recognizing D) $\mathcal{I}$ satisfying the following conditions:

- $\mathcal{I}$ is extended with one additional mapping and two additional sets:
  - $\mathsf{I}_{\mathsf{CEXT}}(y)$ is defined as the set $\{x \mid \langle x, y\rangle \in \mathsf{I}_{\mathsf{EXT}}(\texttt{rdf:type})\}$
  - IC is defined to be $\mathsf{I}_{\mathsf{CEXT}}(\mathcal{I}(\texttt{rdfs:Class}))$
  - LV is defined to be $\mathsf{I}_{\mathsf{CEXT}}(\mathcal{I}(\texttt{rdfs:Literal}))$
- $\mathsf{I}_{\mathsf{CEXT}}(\mathcal{I}(\texttt{rdfs:Resource})) = \mathsf{IR}$
- $\mathsf{I}_{\mathsf{CEXT}}(\mathcal{I}(\texttt{rdf:langString})) = \{\mathcal{I}(x) \mid x$ is a language-tagged string$\}$
- for every other datatype $d \in D$, $\mathsf{I}_{\mathsf{CEXT}}(\mathcal{I}(d))$ is the value space of $\mathcal{I}(d)$
- for all datatypes $d \in D$, $\mathcal{I}(d) \in \mathsf{I}_{\mathsf{CEXT}}(\mathcal{I}(\texttt{rdfs:Datatype}))$

... continued to the next slide ...

Definition (RDFS Interpretation – contd.)

- If $\langle x, y\rangle \in \mathsf{I}_{\mathsf{EXT}}(\mathcal{I}(\texttt{rdfs:domain}))$ and $\langle u, v\rangle \in \mathsf{I}_{\mathsf{EXT}}(x)$, then $u \in \mathsf{I}_{\mathsf{CEXT}}(y)$
- If $\langle x, y\rangle \in \mathsf{I}_{\mathsf{EXT}}(\mathcal{I}(\texttt{rdfs:range}))$ and $\langle u, v\rangle \in \mathsf{I}_{\mathsf{EXT}}(x)$, then $v \in \mathsf{I}_{\mathsf{CEXT}}(y)$
- $\mathsf{I}_{\mathsf{EXT}}(\mathcal{I}(\texttt{rdfs:subPropertyOf}))$ is transitive and reflexive on IP
- If $\langle x, y\rangle \in \mathsf{I}_{\mathsf{EXT}}(\mathcal{I}(\texttt{rdfs:subPropertyOf}))$, then $x, y \in$ IP and $\mathsf{I}_{\mathsf{EXT}}(x) \subseteq \mathsf{I}_{\mathsf{EXT}}(y)$
- If $x \in$ IC, then $\langle x, \mathcal{I}(\texttt{rdfs:Resource})\rangle \in \mathsf{I}_{\mathsf{EXT}}(\mathcal{I}(\texttt{rdfs:subClassOf}))$
- $\mathsf{I}_{\mathsf{EXT}}(\mathcal{I}(\texttt{rdfs:subClassOf}))$ is transitive and reflexive on IC
- If $\langle x, y\rangle \in \mathsf{I}_{\mathsf{EXT}}(\mathcal{I}(\texttt{rdfs:subClassOf}))$, then $x, y \in$ IC and $\mathsf{I}_{\mathsf{CEXT}}(x) \subseteq \mathsf{I}_{\mathsf{CEXT}}(y))$
- If $x \in \mathsf{I}_{\mathsf{CEXT}}(\mathcal{I}(\texttt{rdfs:ContainerMembershipProperty}))$ then $\langle x, \mathcal{I}(\texttt{rdfs:member})\rangle \in \mathsf{I}_{\mathsf{EXT}}(\mathcal{I}(\texttt{rdfs:subPropertyOf}))$
- If $x \in \mathsf{I}_{\mathsf{CEXT}}(\mathcal{I}(\texttt{rdfs:Datatype}))$ then $\langle x, \mathcal{I}(\texttt{rdfs:Literal})\rangle \in \mathsf{I}_{\mathsf{EXT}}(\mathcal{I}(\texttt{rdfs:subClassOf}))$
- $\mathcal{I}$ satisfies all axiomatic triples given next.

RDFS Axiomatic Triples I

```
rdf:type rdfs:domain rdfs:Resource .
rdfs:domain rdfs:domain rdf:Property .
rdfs:range rdfs:domain rdf:Property .
rdfs:subPropertyOf rdfs:domain rdf:Property .
rdfs:subClassOf rdfs:domain rdfs:Class .
rdf:subject rdfs:domain rdf:Statement .
rdf:predicate rdfs:domain rdf:Statement .
rdf:object rdfs:domain rdf:Statement .
rdfs:member rdfs:domain rdfs:Resource .
rdf:first rdfs:domain rdf:List .
rdf:rest rdfs:domain rdf:List .
rdfs:seeAlso rdfs:domain rdfs:Resource .
rdfs:isDefinedBy rdfs:domain rdfs:Resource .
rdfs:comment rdfs:domain rdfs:Resource .
rdfs:label rdfs:domain rdfs:Resource .
rdf:value rdfs:domain rdfs:Resource .
```

RDFS Axiomatic Triples II

```
rdf:type rdfs:range rdfs:Class .
rdfs:domain rdfs:range rdfs:Class .
rdfs:range rdfs:range rdfs:Class .
rdfs:subPropertyOf rdfs:range rdf:Property .
rdfs:subClassOf rdfs:range rdfs:Class .
rdf:subject rdfs:range rdfs:Resource .
rdf:predicate rdfs:range rdfs:Resource .
rdf:object rdfs:range rdfs:Resource .
rdfs:member rdfs:range rdfs:Resource .
rdf:first rdfs:range rdfs:Resource .
rdf:rest rdfs:range rdf:List .
rdfs:seeAlso rdfs:range rdfs:Resource .
rdfs:isDefinedBy rdfs:range rdfs:Resource .
rdfs:comment rdfs:range rdfs:Literal .
rdfs:label rdfs:range rdfs:Literal .
rdf:value rdfs:range rdfs:Resource .
```

RDFS Axiomatic Triples III

```
rdf:Alt rdfs:subClassOf rdfs:Container .
rdf:Bag rdfs:subClassOf rdfs:Container .
rdf:Seq rdfs:subClassOf rdfs:Container .
rdfs:ContainerMembershipProperty rdfs:subClassOf rdf:Property .

rdfs:isDefinedBy rdfs:subPropertyOf rdfs:seeAlso .

rdfs:Datatype rdfs:subClassOf rdfs:Class .

rdf:_1 rdf:type rdfs:ContainerMembershipProperty .
rdf:_1 rdfs:domain rdfs:Resource .
rdf:_1 rdfs:range rdfs:Resource .
rdf:_2 rdf:type rdfs:ContainerMembershipProperty .
rdf:_2 rdfs:domain rdfs:Resource .
rdf:_2 rdfs:range rdfs:Resource .
...
```

RDFS Entailment

- An graph G RDFS-entails a graph G' recognizing a set D of datatypes if every RDFS interpretation recognizing D that satisfies G also satisfies G'.
- Every RDFS interpretation is RDF interpretation. So, if G RDFS-entails G', then G RDF-entails G', but not vice versa.

RDFS Entailment Deduction Rules I

- RDFS entailment rules include all RDF entailment rules: se1, se2, D-entailment principle, rdfD1, rdfD2, rdfax.
- The remaining rules are as follows.

Definition (rdfsax)

For every **axiomatic** triples `u p x .` in the definition RDFS interpretation:

$$\frac{}{\texttt{u p x .}}\ \text{rdfsax}$$

Definition (rdfs1)

For every recognized datatype IRI d:

$$\frac{}{\texttt{d rdf:type rdfs:Datatype.}}\ \text{rdfs1}$$

RDFS Entailment Deduction Rules II

DaSe Lab

Definition (rdfs2)

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

Definition (rdfs3)

$$\frac{\texttt{p rdfs:range x .} \qquad \texttt{u p y .}}{\texttt{y rdf:type x .}}\ \text{rdfs3}$$

RDFS Entailment Deduction Rules III

DaSe Lab

Definition (rdfs4a)

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

Definition (rdfs4b)

$$\frac{\texttt{u p x .}}{\texttt{x rdf:type rdfs:Resource.}}\ \text{rdfs4b}$$

Definition (rdfs5)

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

RDFS Entailment Deduction Rules IV

Definition (rdfs6)

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

Definition (rdfs7)

$$\frac{\texttt{u rdfs:subPropertyOf x .} \qquad \texttt{v u y .}}{\texttt{v x y .}}\ \text{rdfs7}$$

Definition (rdfs8)

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

RDFS Entailment Deduction Rules V

DaSe Lab

Definition (rdfs9)

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

Definition (rdfs10)

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

Definition (rdfs11)

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

RDFS Entailment Deduction Rules VI

DaSe Lab

Definition (rdfs12)

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

Definition (rdfs13)

$$\frac{\texttt{u rdf:type rdfs:Datatype.}}{\texttt{u rdfs:Literal.}}\ \text{rdfs6}$$

RDFS Entailment Theorem I

RDFS Entailment Algorithm via Deduction Rules

To check if a graph G_1 RDF-entails a graph G_2, we perform the following:

1. Add all RDF and RDFS axiomatic triples to G_1 according to rdfax and rdfsax, except those containing the container membership property IRIs `rdf:_1`, `rdf:_2`, ...
2. For every container membership property IRI occurring in G, add the RDF and RDFS axiomatic triples that contain that IRI.
3. Apply rdfD1, rdfD2, rdfs1, ..., rdfs13 exhaustively to G_1 until they generate no new triples. Let G_1' be the resulting graph.
4. If G_1' simply entails G_2, then G_1 RDF-entails G_2.

Theorem

The algorithm terminates and is sound.

RDFS Entailment Theorem II

- The algorithm, however, is **not** complete. For example, the graph G_1 RDFS-entails the graph G_2 below, but cannot be derived by the algorithm.

G_1:

```
ex:a rdfs:subPropertyOf _:b .
_:b rdfs:domain ex:c .
ex:d ex:a ex:e .
```

G_2:

```
ex:d rdf:type ex:c .
```

The Complete Algorithm?

- Both RDF and RDFS entailment algorithms shown previously can be made complete if we allow a generalized syntax, i.e., allow literals in the subject position and blank nodes in the predicate position.
- Simple, RDF, RDFS entailments are NP-complete.
- If we disallow blank nodes, the entailments are polynomial.

Outline

What RDFS Cannot Do

- Some sensible consequences are not supported by RDFS, e.g.,

```
ex:hasTitle rdfs:domain ex:Book .
ex:Book rdfs:subClassOf ex:Publication .
```

should logically imply (but does not RDFS-entail)

```
ex:hasTitle rdfs:domain ex:Publication .
```

- No possibility to express negation.