Previously on Introduction to Linked Data…

- In the last lecture, you have learned how to distinguish between classes, properties and individuals in data modelling and to design domain vocabularies in RDFS by relating to terms from your vocabulary with terms from existing vocabularies.

- In the following lecture, you will learn:
  - How fundamental insights from philosophers and mathematicians form the basis for today‘s knowledge representation techniques
  - Details on how to formally define the syntax and semantics of RDF
  - How to use the formal semantics to derive new knowledge from existing knowledge

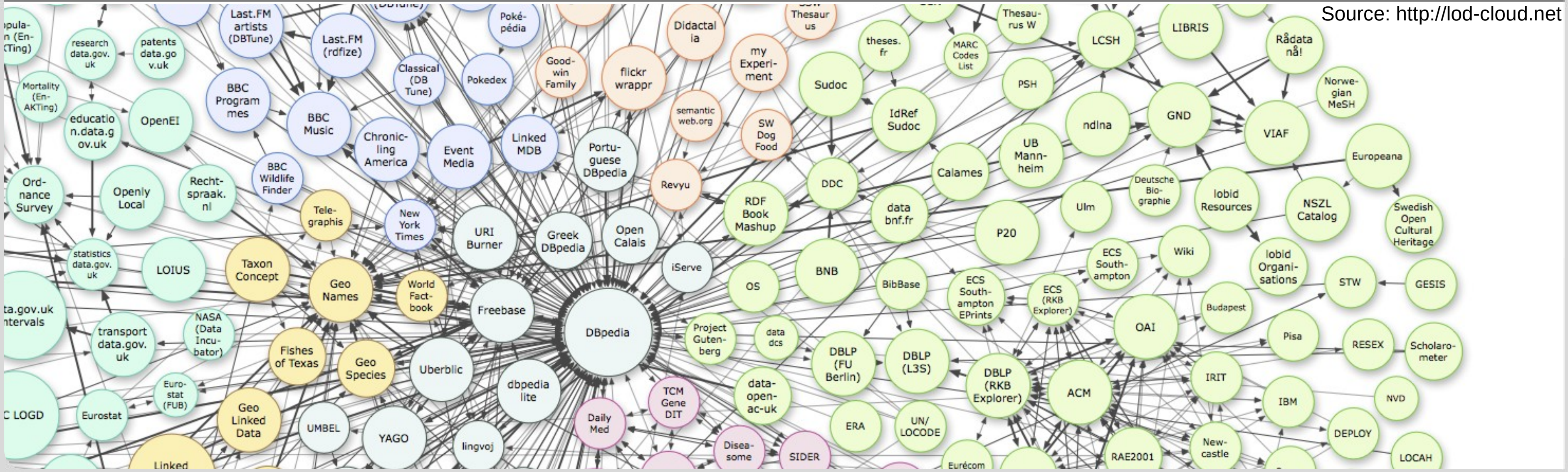
C09 Semantics of RDF and RDF Schema Vocabularies

How to derive everything that logically follows from descriptions?

Version 2022-06-27

Lecturer: Prof. Dr. Andreas Harth

CHAIR OF TECHNICAL INFORMATION SYSTEMS


Source: http://lod-cloud.net

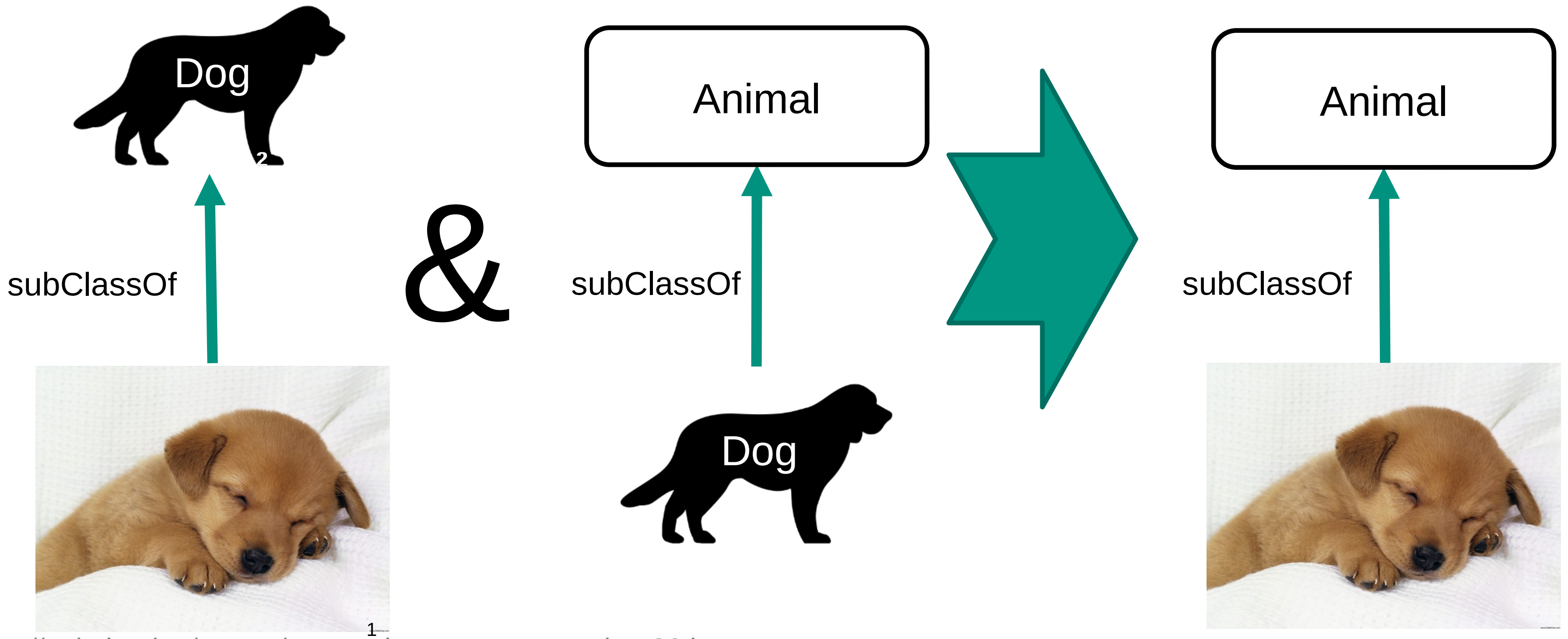
CC - Creative Commons Licensing

- This set of slides is part of the lecture „Semantic Web Technologies“ held at Karlsruhe Institute of Technology
- The content of the lecture was prepared by PD Dr. Andreas Harth based on his book „Introduction to Linked Data“
- The slides were prepared by Lars Heling, Andreas Harth, Tobias Käfer and Maribel Acosta

Agenda

Motivation: Inference

- We can use the RDF and RDFS vocabularies in RDF graphs
- From triples using the RDF and RDFS vocabularies, systems ("reasoners") should be able to infer new triples
- An example: (taxonomy)



[1] http://onin.london/assets/macexplorer.com-puppy-dog-26.jpg
[2] Created by Icon Island from Noun Project

Inference in Triples

- Given a graph G:

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix : <#> .

:GermanShepherd rdfs:subClassOf :Dog .
:Dog rdfs:subClassOf :Animal .
```

- We can infer the triple:

```
:GermanShepherd rdfs:subClassOf :Animal .
```

A Classical Example

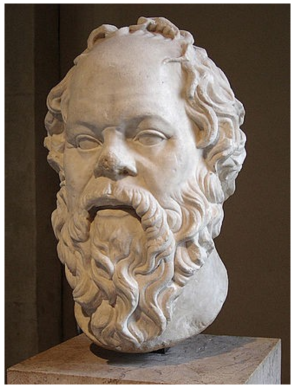


Photo from Wikipedia

- Premise: All men are mortal
- Premise: Socrates is a man
- Conclusion: Socrates is mortal

- In RDF using RDFS vocabulary:

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix : <#> .

:Man rdfs:subClassOf :Mortal .
:Socrates a :Man .

:Socrates a :Mortal .
```

Ways to Define A Logical Language

- Syntax: how sentences are constructed (here: RDF triples)
- Semantics: what the sentences mean

- Model-theoretic Definition
  - Construct „models“ from sentences in the language
  - Relate the models to each other
- Proof-theoretic Definition
  - Given sentences as premises, can we find out whether another sentence logically follows from the premises?
- Operational Definition
  - Given the premises, calculate all sentences that logically follow (the conclusions)

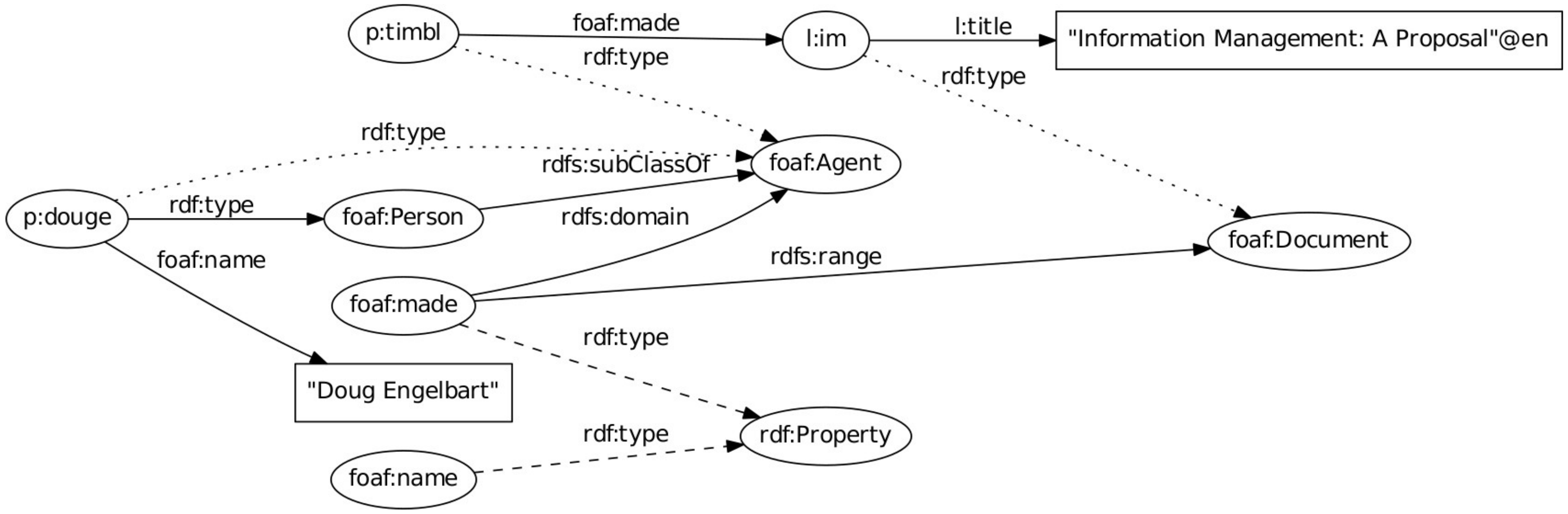
Why Formal Semantics?

- After introduction of RDF and RDFS, critcism of tool developers: different tools were incompatible (despite the existing specification)

- E.g.:
  - Same RDF document
  - Same entailment relation
  - Different results

- Thus, a model-theoretic semantics was defined for entailment:
  - provides a formal specification of when truth is preserved by transformations of RDF or operations which derive RDF triples from other RDF triples (logical consequence).

Think-Pair-Share



- The triples in the dashed lines are entailed via RDF entailment and the triples in the dotted lines via RDFS entailment.
- Explain why the dashed (dotted) triples are RDF- (RDFS-) entailed.
- What other triples are entailed?

Agenda

1. Defining a Logical Language
2. **Interpretations and Satisfiability**
3. Simple Entailment and *D*-Entailment
4. RDF Interpretations
5. RDFS Interpretations
6. Entailment Patterns for RDF and RDFS

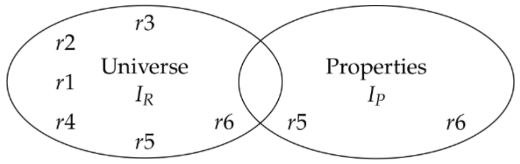
Syntax vs. Semantics

- Logicians write down a theory about the world in a logical language
- The structure (syntax) of the language is defined
  - By the symbols one can use
  - By the way the symbols can be combined to form sentences
- The meaning (semantics) of the language can be defined in different ways

Language (Syntax)

```
@prefix : <#> .
:tbl :born :london .
```

Model (Semantics)



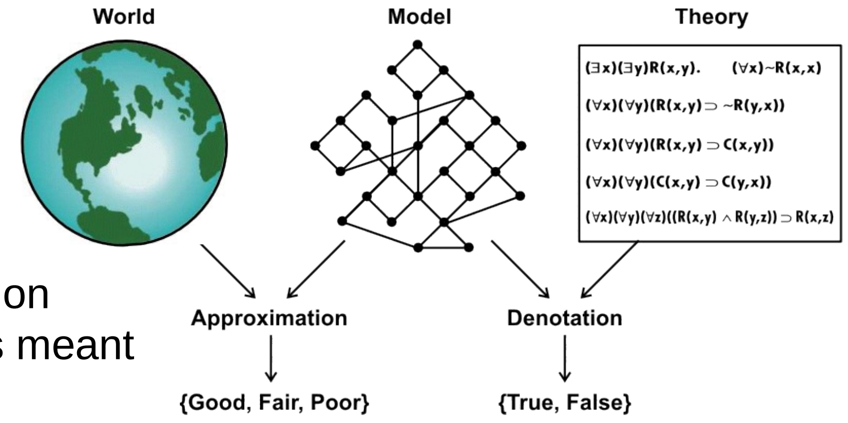
Reality (World)

Theories of Truth

- Correspondence theory of truth
  - True sentences are in agreement with the current state of affairs
- Coherence theory of truth
  - True sentences do not contradict existing knowledge
- Consensus theory of truth (also known as pragmatic theory of truth)
  - True sentences are sentences that people agree about
- We have learned about various aspects of these views
- For a comprehensive theory of truth, we can make use of different aspects of the different theories of truth

Names, Truth, Logic

- What does a name refer to?
  - Frege (1848 – 1925)
    - Name (Wort) – Sinn – Bedeutung
  - Russell (1872 – 1970)
    - Name = abbreviation for a description that excludes all but the one who is meant
  - Berners-Lee (1955 –)
    - Ask the one who coined the name (URI)
  - …
- How does truth of sentences and theories come about?
  - Correspondence theory of truth (e.g., Tarski (1901–1983))
    - Satisfiability ▭ RDF Model Theoretic Semantics
  - Consensus theory of truth (e.g., Habermas (1929 –))
    - Common acceptance
  - …



http://www.jfsowa.com/pubs/worlds.pdf

Ingredients of the RDF Model Theory

- Hayes built the RDF Model Theory on Tarski's ideas
  - First: define the syntax of the RDF language (URIs, literals, blank nodes, triples, graphs)
  - Second: use a mathematical structure called "interpretation" to decide whether a given graph is true
- An interpretation is (Hayes):
  "A minimal formal description of those aspects of a world which is just sufficient to establish the truth or falsity of any expression of a logic."
- An interpretation I defines how URIs, literals and blank nodes are being interpreted in world (the universe or domain).

Interpretations

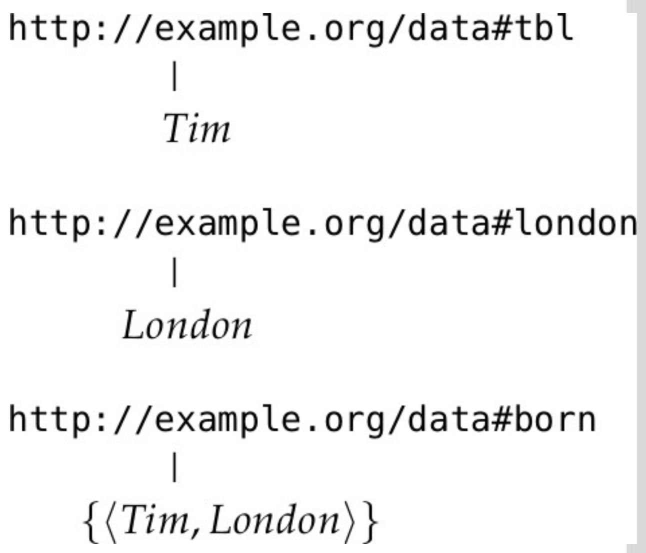
An interpretation assigns **meaning** (semantics) **to the terms** (i.e., URIs, literals, blank nodes) of a vocabulary and **to sentences** (i.e., RDF triples and graphs) involving the terms.

Interpretations

① Interpretation structure: set of resources, set of property resources, extension of property resources

- E.g., a set of resources *{ Tim, London }* and a set of properties *{ born }, with their extension*.

② Interpretation mapping: from syntactic expressions (URIs, literals) to the interpretation structure.


http://example.org/data#tbl
Tim
http://example.org/data#london
London
http://example.org/data#born
{⟨Tim, London⟩}

③ And then: conditions to decide whether the interpretation makes the graph true.

E.g., the graph at http://example.org/data:

```
@prefix : <#> .
:tbl :born :london
```

Truth and Models via Semantic Conditions

③ For an interpretation I to make a graph G true, written as I(G) = true:

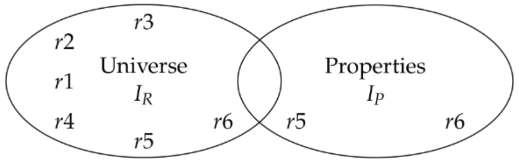
- All symbols (URIs, literals, blank nodes) in a graph are mapped to elements in the universe
- All triples in the graph are mapped to pairs of resources that the property of the triple applies to (the extension of the property)

- If I(G) = true, then we say that I *satisfies* G.
- If I(G) = true, then we also say that I *is a model of* G
- Hence the name "model theory"

Language (Syntax)

```
@prefix : <#> .
:tbl :born :london .
```

Model (Semantics)



Satisfiability and Models

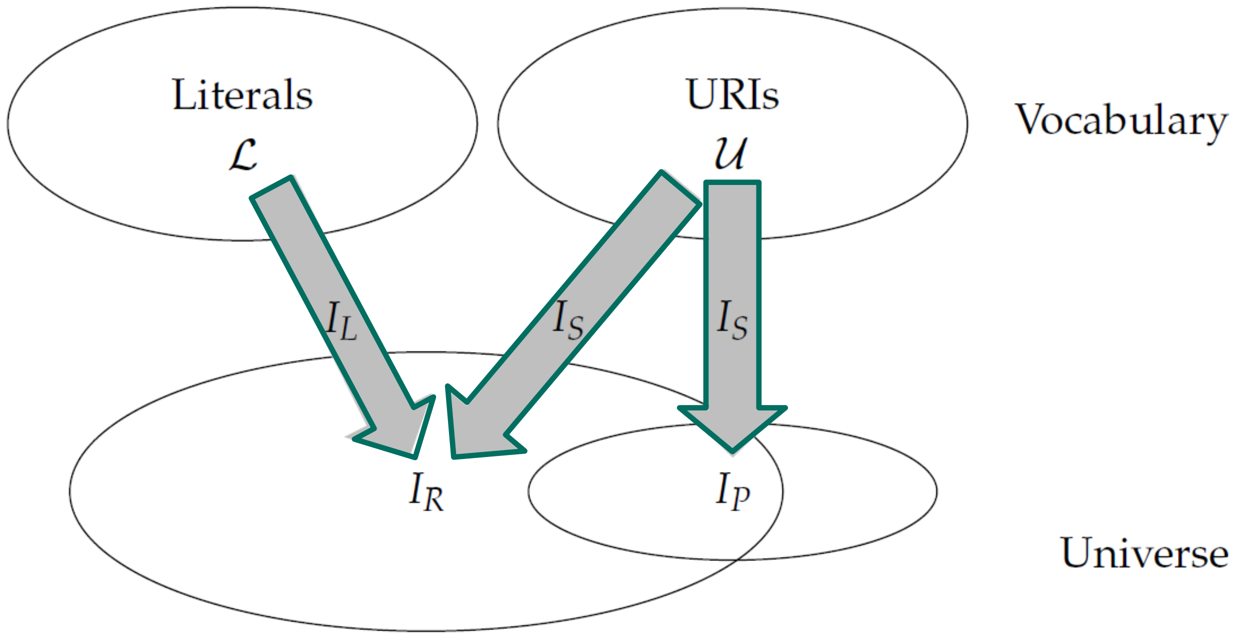
- If there exists an interpretation I that satisfies G (makes G true, written as I(G) = true), then G is *satisfiable*
- We then also write (I satisfies G)
- If there is no interpretation that satisfies G, G is *unsatisfiable*

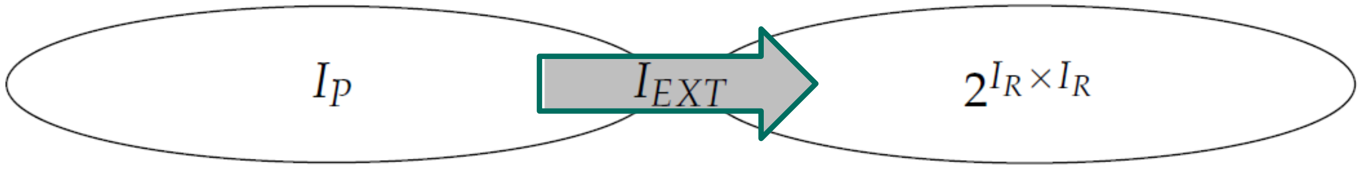
Simple Interpretation

Definition 7 (Interpretation). *Let $\mathcal{U}$ be the set of URIs and $\mathcal{L}$ be the set of RDF literals. A simple interpretation I consists of:*

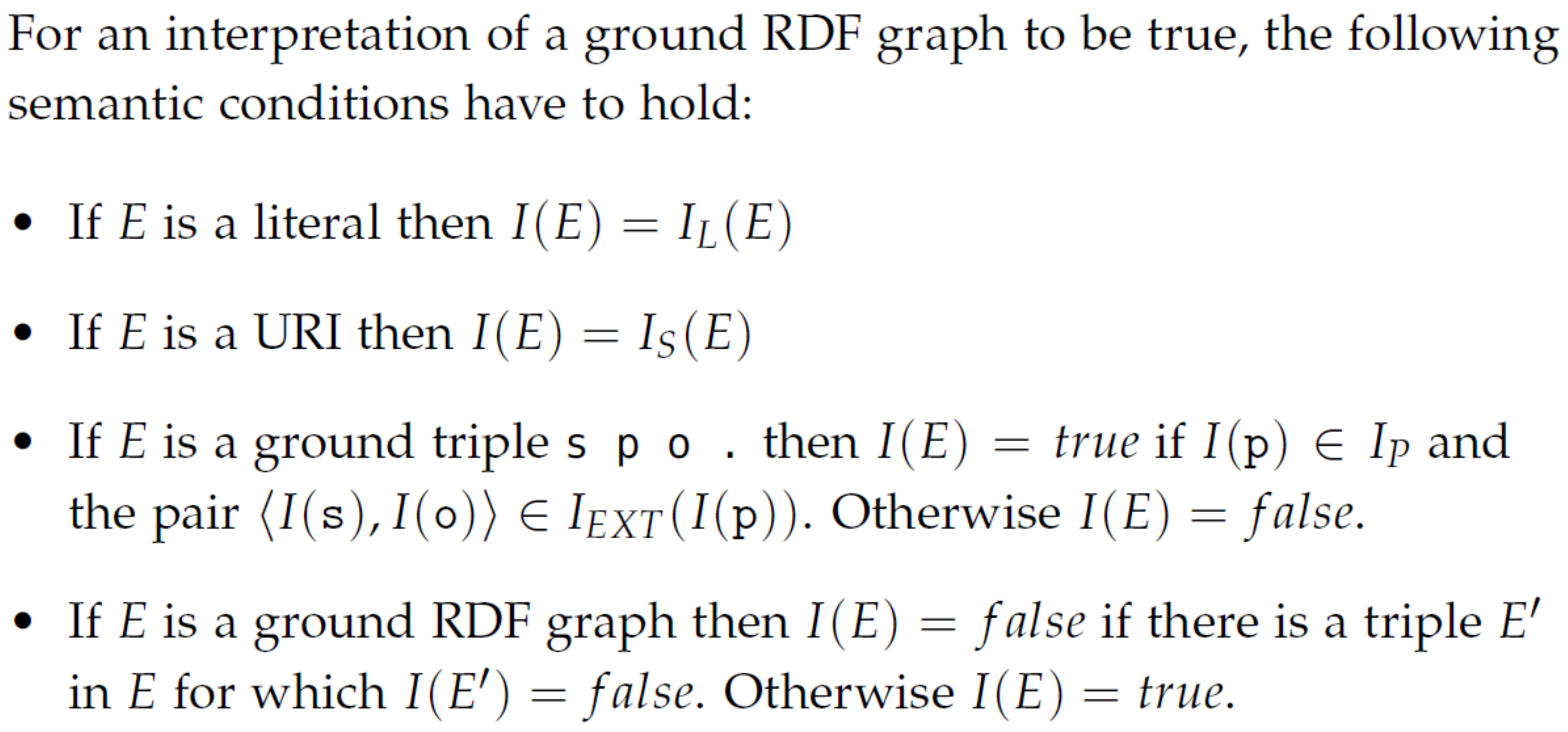
1. *A non-empty set I_R of resources called I's domain or universe.*
2. *A set I_P called the set of properties of I.*
3. *A function $I_{EXT} : I_P \mapsto 2^{I_R \times I_R}$ called the extension of a property, to assign a set of pairs $\langle s, o \rangle$ with $s, o \in I_R$ to a property resource.*
4. *A function $I_S : \mathcal{U} \mapsto I_R \cup I_P$ that maps URIs to resources in the universe.*
5. *A partial function $I_L : \mathcal{L} \mapsto I_R$ that maps RDF literals to resources in the universe.*

Simple Interpretations Graphically





Truth and Falsehood



For an interpretation of a ground RDF graph to be true, the following semantic conditions have to hold:

- If E is a literal then $I(E) = I_L(E)$
- If E is a URI then $I(E) = I_S(E)$
- If E is a ground triple `s p o .` then $I(E) = \mathit{true}$ if $I(\mathtt{p}) \in I_P$ and the pair $\langle I(\mathtt{s}), I(\mathtt{o}) \rangle \in I_{EXT}(I(\mathtt{p}))$. Otherwise $I(E) = \mathit{false}$.
- If E is a ground RDF graph then $I(E) = \mathit{false}$ if there is a triple E' in E for which $I(E') = \mathit{false}$. Otherwise $I(E) = \mathit{true}$.

Ground triple: triple without blank nodes
Ground graph: graph with only ground triples

Think-Pair-Share: Simple Interpretation

Given the following RDF graph *G*:

```
@prefix : <#> .
:Germany :capital :Berlin ;
         :name "Germany" .
```

Demonstrate whether the following simple interpretation ▭
$I = \langle I_R, I_P, I_{EXT}, I_S, I_L \rangle$
simply satisfies *G* or not

$I_R = \{r1, r2, r3, r4\}$

$I_P = \{p1, p2\}$

$I_{EXT}(p1) = \{\langle r1, r2 \rangle\}$

$I_{EXT}(p2) = \{\langle r1, r3 \rangle\}$

$I_S(\texttt{:Germany}) = r1$

$I_S(\texttt{:Berlin}) = r2$

$I_S(\texttt{:capital}) = p1$

$I_S(\texttt{:name}) = p2$

$I_L(\texttt{"Germany"}) = r3$

$I_L(\texttt{"Berlin"}) = r4$

Think-Pair-Share: Simple Interpretation

(a) Solution (part 1):

For each triple in G, we check if the semantic conditions hold.

```
:Germany :capital :Berlin .
```

1) We have that:

- I(`:Germany`) = I_S(`:Germany`)
- I(`:Germany`) = r1
- I(`:capital`) = I_S(`:capital`)
- I(`:capital`) = p1
- I(`:Berlin`) = I_S(`:Berlin`)
- I(`:Berlin`) = r2

2) Now, we must check that I(`:capital`) belongs to I_P and that <I(`:Germany`), I(`:Berlin`)> belongs to I_{EXT}(I(`:capital`)).

I(`:capital`) is p1, and by definition of I we have that p1 belongs to I_P. This condition holds.

<I(`:Germany`), I(`:Berlin`)> is <r1, r2>. By definition of I, we have that <r1, r2> ndition holds.

For this ground triple E, we have that I(E)=true.

Think-Pair-Share: Simple Interpretation

(a) Solution (part 2):

For each triple in G, we check if the semantic conditions hold.

`:Germany :name "Germany".`

1) We have that:

- I(`:Germany`) = I_S(`:Germany`)
- I(`:Germany`) = r1
- I(`:name`) = I_S(`:name`)
- I(`:name`) = p2
- I(`"Germany"`) = I_L(`"Germany"`)
- I(`"Germany"`) = r3

2) Now, we must check that I(`:name`) belongs to I_P and that <I(`:Germany`), I(`"Germany"`)> belongs to I_{EXT}(I(`:name`)).

I(`:name`) is p2, and by definition of I we have that p2 belongs to I_P. This condition holds.

<I(`:Germany`), I(`"Germany"`)> is <r1, r3>. By definition of I, we have that <r1, r3> belongs to I_{EXT}(p2). This condition holds.

For this ground triple E, we have that I(E)=true.

Think-Pair-Share: Simple Interpretation

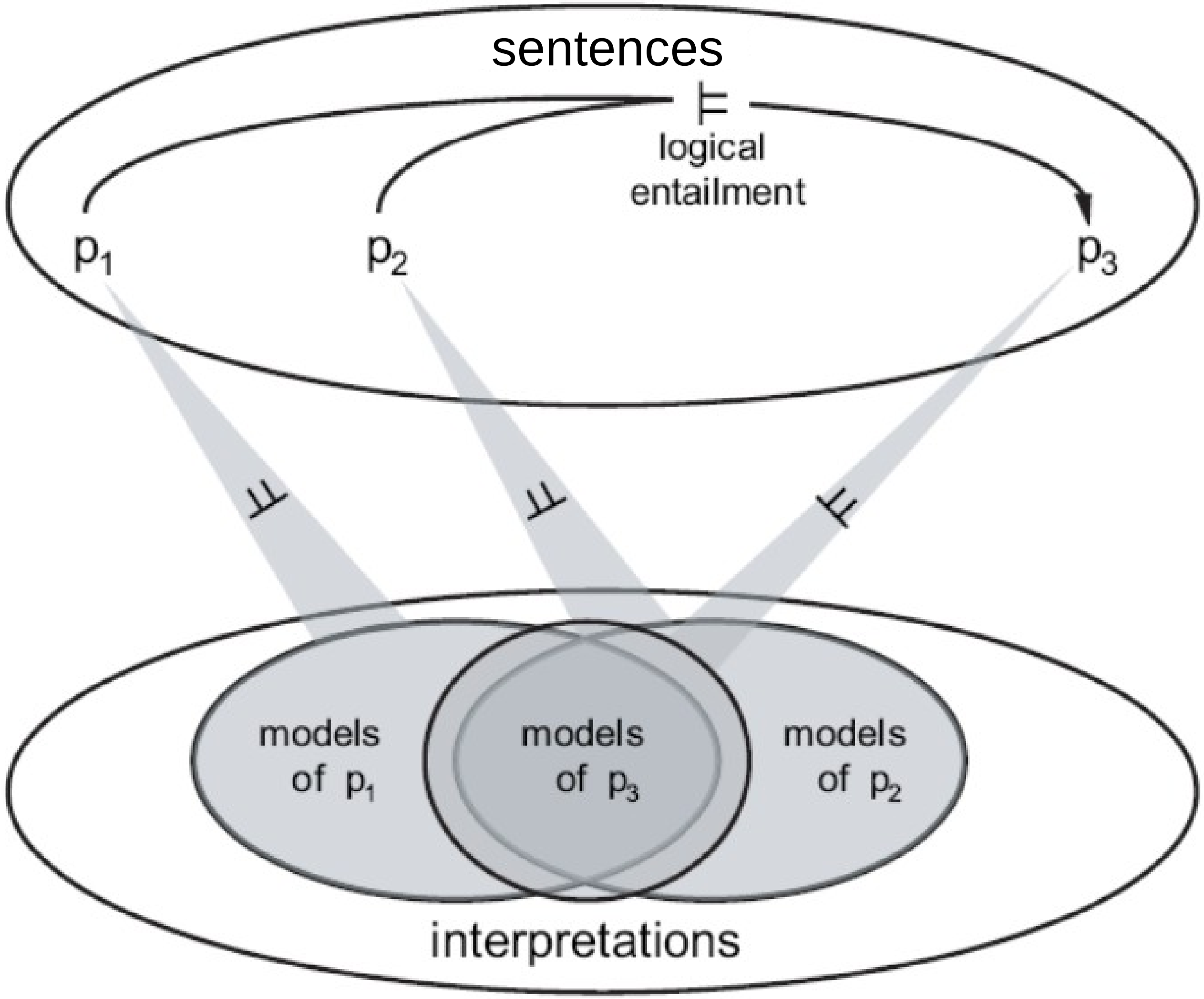
(a) Solution (part 3):

For all triples E’ in G we obtained that I(E’) = true. Therefore we conclude that **I(G) = true**, in other words: **I satisfies G**

Agenda

1. Defining a Logical Language
2. Interpretations and Satisfiability
3. **Simple Entailment and *D*-Entailment**
4. RDF Interpretations
5. RDFS Interpretations
6. Entailment Patterns for RDF and RDFS

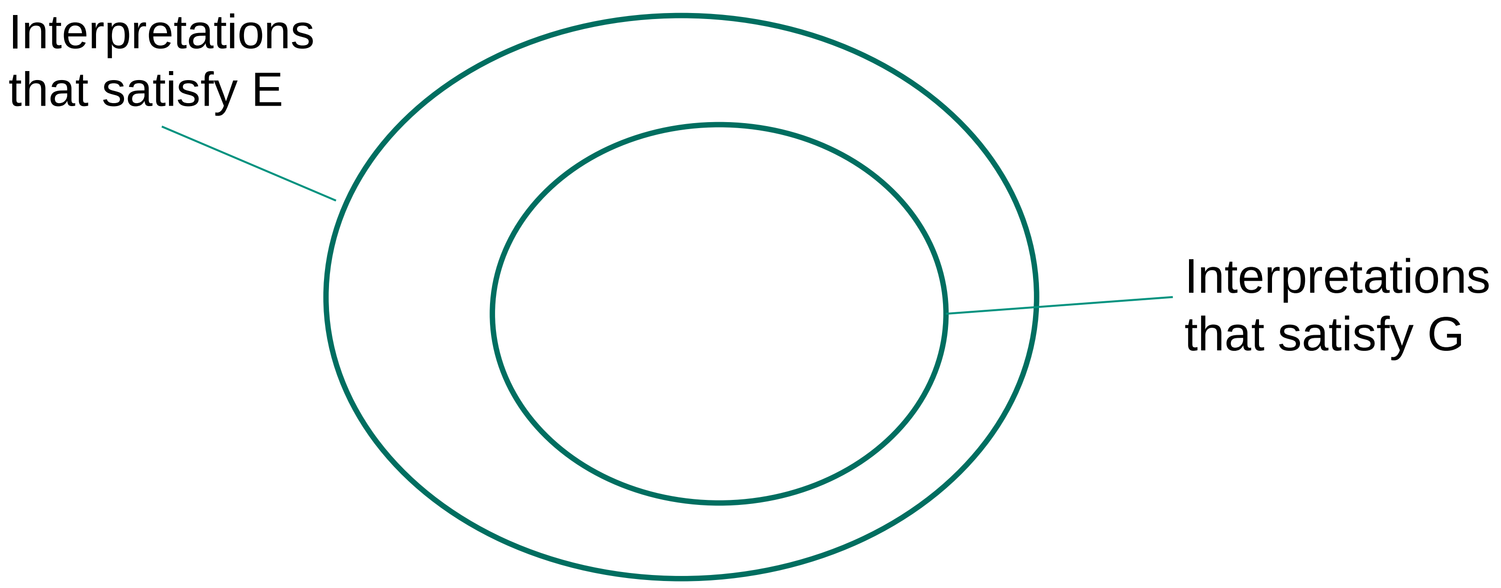
From Interpretations to Entailment



Pascal Hitzler, Markus Krötzsch, Sebastian Rudolph. Foundations of Semantic Web Technologies. Chapman & Hall/CRC, 2009.

Entailment Relation

- Entailment relation between two graphs G and E
- G |= E (G entails E)
- G entails E when every interpretation that satisfies G also satisfies E


Interpretations that satisfy E
Interpretations that satisfy G

Entailment and Logical Consequence

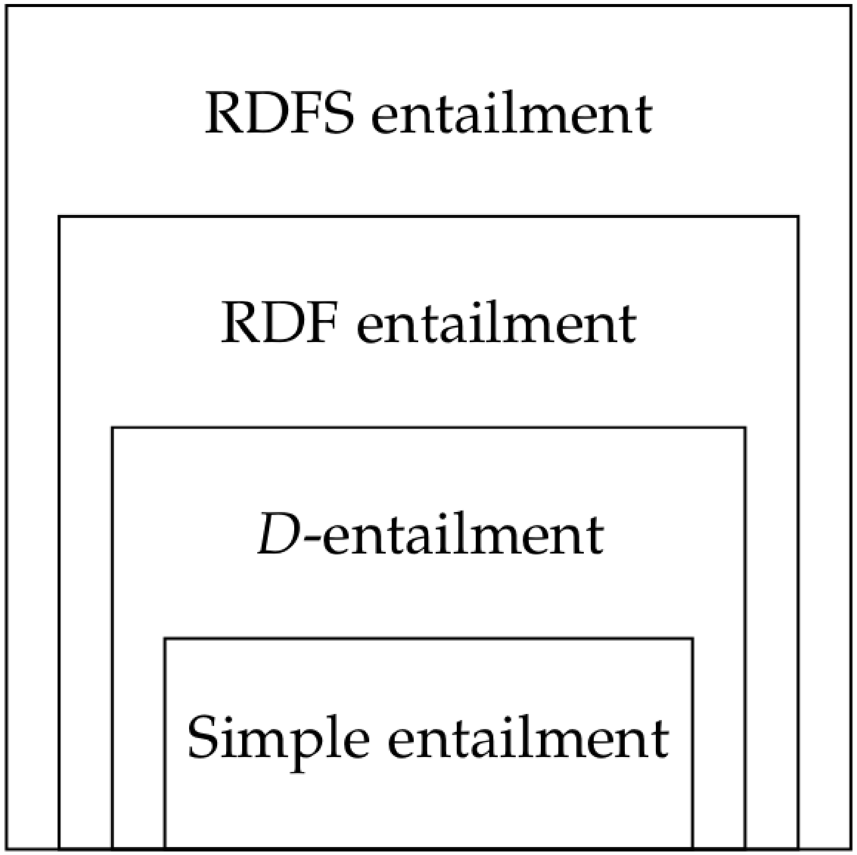
- Logical consequence states that true sentences follow from true premises.
- Applied on the level of syntax between two graphs.
- Given two graphs G and E, we write G |= E to mean that G entails E.
- Intuitively speaking, G |= E if the information in E is a subset of the information in G.
- Consider the simple case; does the following entailment hold?

```
<#s> <#p> <#o> . <#s> <#q> "23" . ⊨ <#s> <#p> <#o> . ?    (1)
```

Does the Entailment Relation Hold?

```
<#s> <#p> <#o> . <#s> <#q> "23" . ⊨ <#s> <#p> <#o> . ?      (1)

<#s> <#p> <#o> . ⊨ _:bn1 <#p> _:bn2 . ?                      (2)

_:bn1 <#p> _:bn2 . ⊨ <#s> <#p> <#o> . ?                      (3)

_:bn1 <#p> _:bn2 . ⊨ _:a <#p> _:b . ?                        (4)

_:a <#p> _:b . ⊨ _:bn1 <#p> _:bn2 . ?                        (5)

<#s> <#q> 23 . ⊨ <#s> <#q> 23.0 . ?                          (6)

<#s> <#q> 23.0 . ⊨ <#s> <#q> 23 . ?                          (7)

<#s> <#q> "23"^^xsd:float . ⊨ <#s> <#q> 23.0 . ?             (8)

<#s> <#q> "untrue"^^xsd:boolean . ⊨ <#s> <#q> false . ?      (9)

<#s> <#q> 23.0 . ⊨ <#s> <#q> 23 , "23"^^xsd:int . ?          (10)
```

Layered Entailment

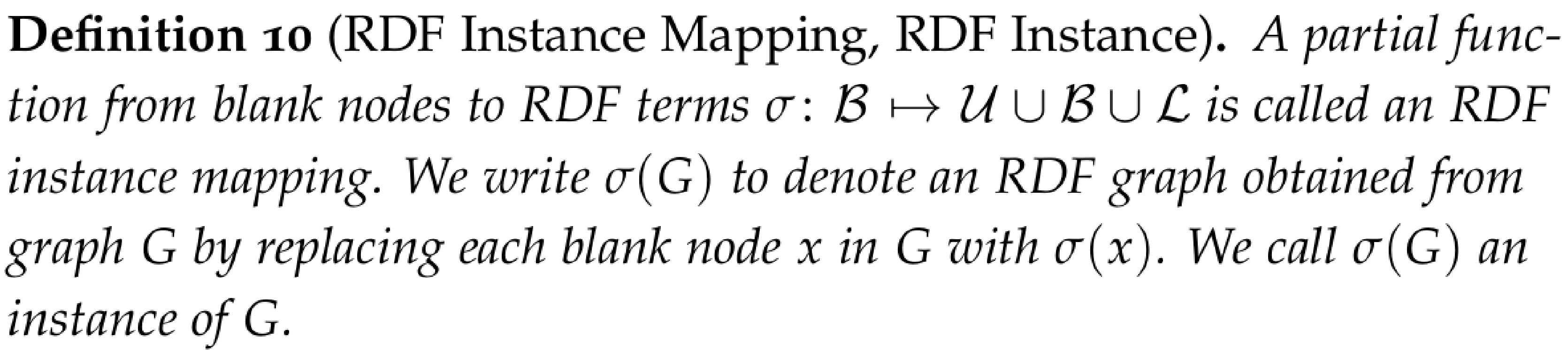


More expressive interpretations support more logical conclusions (entailments).

From Defining Entailment to Deciding Entailment

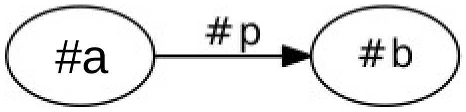
- Definitions for entailment are not constructive: there is no algorithm to generate interpretations which are models for a given RDF graph
- Interpretations are not minimal: interpretations may contain arbitrary additional statements; RDF and RDFS interpretations are infinite
- We need an algorithm to decide on entailment of RDF graphs on a computer
- For simple entailment: the definitions of the interpolation lemma coincide with SPARQL BGP matching (for which there are algorithms)

RDF Instance Mapping

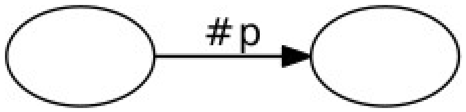


Definition 10 (RDF Instance Mapping, RDF Instance). *A partial function from blank nodes to RDF terms $\sigma : \mathcal{B} \mapsto \mathcal{U} \cup \mathcal{B} \cup \mathcal{L}$ is called an RDF instance mapping. We write $\sigma(G)$ to denote an RDF graph obtained from graph G by replacing each blank node x in G with $\sigma(x)$. We call $\sigma(G)$ an instance of G.*

- Graph G0: `<#a> <#p> <#b> .`

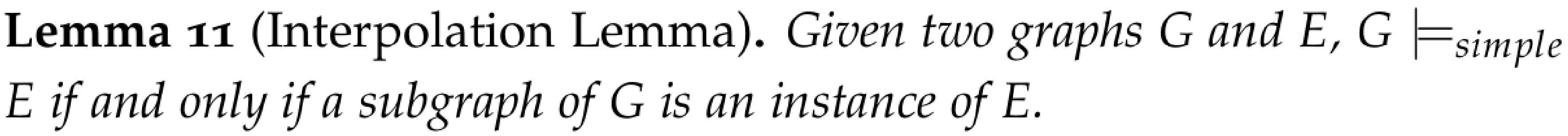


- Graph G1: `_:bn1 <#p> _:bn2 .`



- G0 is an instance of G1, assuming the RDF instance mapping :
  - (`_:bn1`) = <#a>
  - (`_:bn2`) = <#b>

The Interpolation Lemma



Lemma 11 (Interpolation Lemma). *Given two graphs G and E, G* $\models_{simple}$ *E if and only if a subgraph of G is an instance of E.*

- Short version of the implication of the interpolation lemma:
  1. Blank nodes in RDF graphs can be exchanged freely, if no accidental co-references are introduced
  2. URIs and literals in G can be substituted with blank nodes in E

- Example of 1:
  - Let G1 be:
    ```
    _:bn1 <#name> "Paul Otlet" .
    ```
  - Let G2 be:
    ```
    _:genid23 <#name> "Paul Otlet" .
    ```

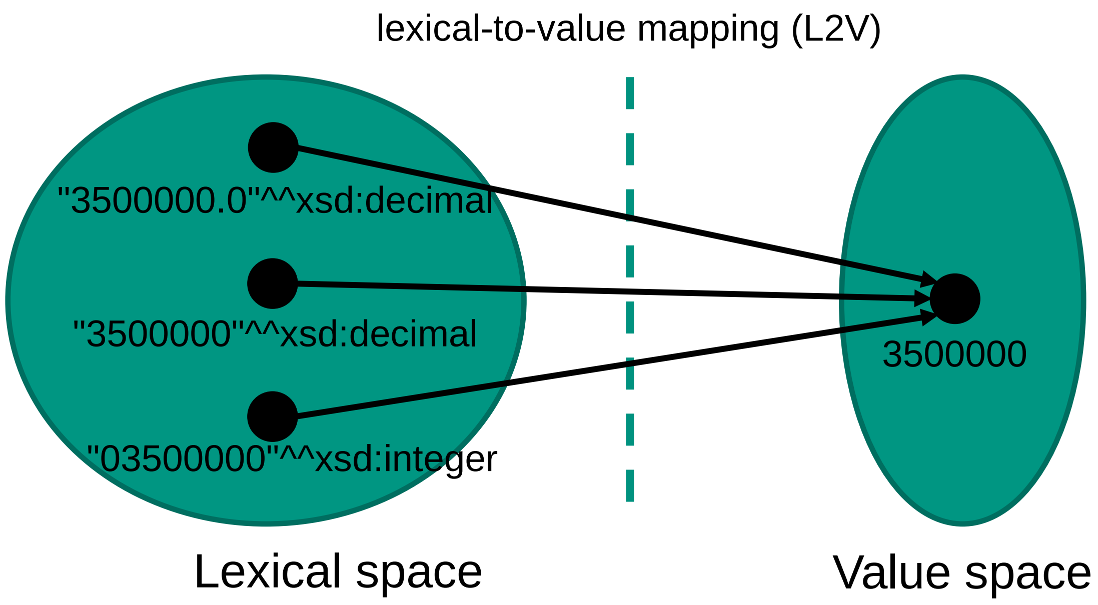
- Both graphs are equal under simple entailment: G1 $|=_{simple}$ G2 and G2 $|=_{simple}$ G1

XSD Datatypes

Definition 9 (Datatype). *A datatype d, identified via a URI, consists of a value space $V(d)$ (a non-empty set of values) and a lexical space $L(d)$ (a non-empty set of Unicode strings). The lexical-to-value mapping (or just lexical mapping) L2V for a datatype d is a mapping from the lexical space to the value space: $L(d) \mapsto V(d)$.*

Lexical Space vs. Value Space

- Lexical space: Consists of different representation of a value (as Unicode character strings)
- Value space: Consists of the actual values
- There is a mapping from the lexical space to the value space
- Example: `:Berlin :population "3500000"^^xsd:decimal .`



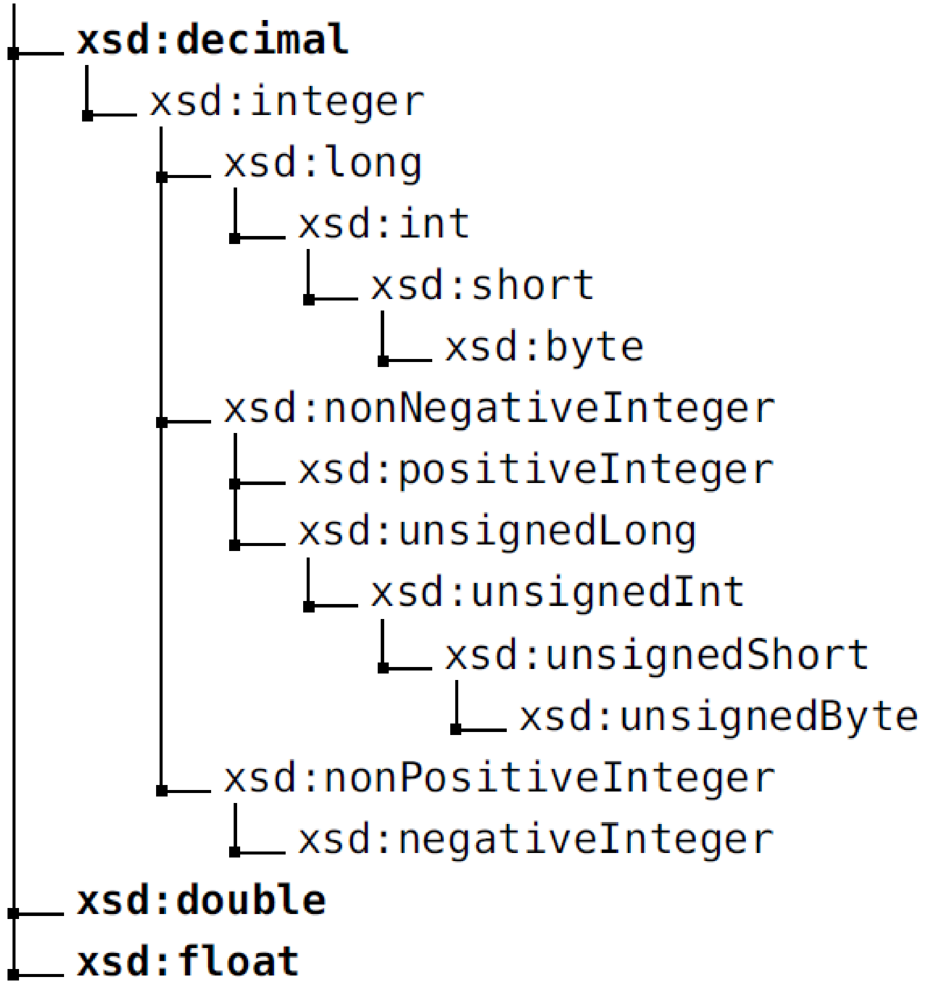
D-Interpretations

- Consider the correct handling of datatypes
- Untyped literals are considered to be of datatype `xsd:string`
- Literals with a language tag are of type `rdf:langString`
- For all other literals the L2V function is used to map literals to their corresponding value
- This function needs to be defined for all datatypes D that are supported
- Example for `xsd:boolean`:

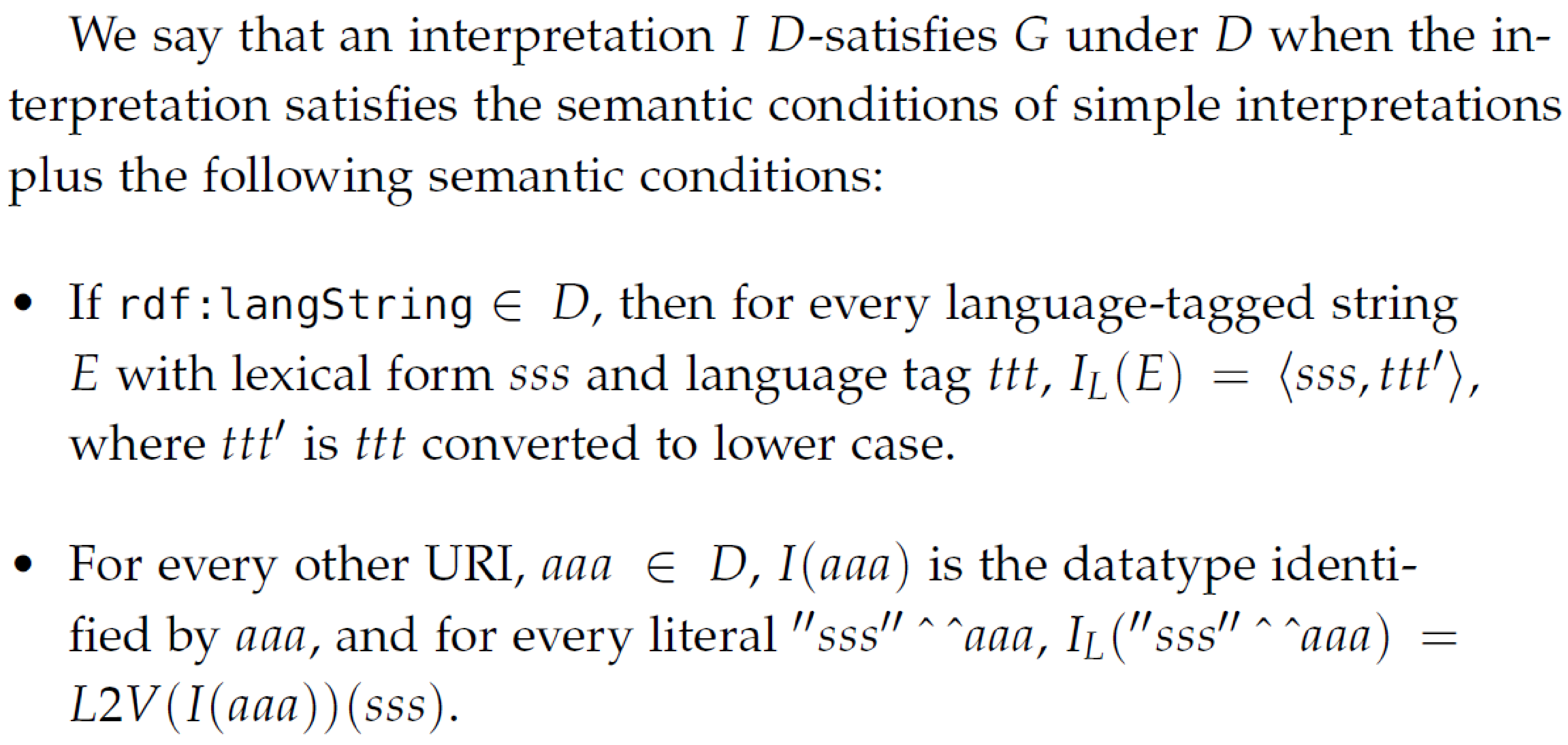
Literal	Value
< `"true"` , `xsd:boolean` >	`true`
< `"false"`, `xsd:boolean` >	`false`
< `"1"`, `xsd:boolean` >	`true`
< `"0"` , `xsd:boolean` >	`false`

Comparison of Datatypes

- Figure shows a subset of XSD datatypes
- Only datatypes from the same „base datatypes“ (bold) can be compared



Semantic Conditions for *D*-Interpretations ③



We say that an interpretation I D-satisfies G under D when the interpretation satisfies the semantic conditions of simple interpretations plus the following semantic conditions:

- If `rdf:langString` $\in D$, then for every language-tagged string E with lexical form sss and language tag ttt, $I_L(E) = \langle sss, ttt' \rangle$, where ttt' is ttt converted to lower case.

- For every other URI, $aaa \in D$, $I(aaa)$ is the datatype identified by aaa, and for every literal $''sss''\,\hat{}\,\hat{}aaa$, $I_L(''sss''\,\hat{}\,\hat{}aaa) = L2V(I(aaa))(sss)$.

Unsatisfiability in *D*-Interpretations

- To check whether datatypes are handled correctly in a given graph, *D*-interpretations can be used
- Concept:
  - Let us assume there is a literal with a recognised datatype

    `:Berlin :population "3500000"^^xsd:boolean .`

  - However, there is no value assignment for the lexical-to-value (L2V) mapping
    - There is **no** mapping from the lexical space `"3500000"` to the value space of `xsd:boolean`

The literal is considered to be ill-typed

- A triple containing an ill-type considered to be false

⇓

Graphs containing a false triple are considered to be unsatisfiable

⇒ *D*-unsatisfiable

Entailment with Datatypes

- The way how interpretations and satisfiability are defined has implications on how datatypes are treated
- Short version: RDF triples containing literals with datatypes that are in the hierarchy are equal.

- Let G1 be:

```
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
<#s> <#p> "23"^^xsd:byte .
```

- Let G2 be:

```
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
<#s> <#p> "23"^^xsd:integer .
```

- Assume $D = \{$ `xsd:byte`, `xsd:integer` $\}$
- Both graphs are equal under *D*-entailment: G1 $|=_D$ G2 and G2 $|=_D$ G1

Does the Entailment Relation Hold?

<#s> <#p> <#o> . <#s> <#q> "23" . $\models_{\text{simple}}$ <#s> <#p> <#o> . ? (1)

<#s> <#p> <#o> . $\models_{\text{simple}}$ _:bn1 <#p> _:bn2 . ? (2)

_:bn1 <#p> _:bn2 . $\models_{\text{simple}}$ <#s> <#p> <#o> . ? (3)

_:bn1 <#p> _:bn2 . $\models_{\text{simple}}$ _:a <#p> _:b . ? (4)

_:a <#p> _:b . $\models_{\text{simple}}$ _:bn1 <#p> _:bn2 . ? (5)

Assume *D* = { `xsd:integer`, `xsd:int`, `xsd:decimal`, `xsd:boolean`, `xsd:float` }

<#s> <#q> 23 . $\models_{D}$ <#s> <#q> 23.0 . ? (6)

<#s> <#q> 23.0 . $\models_{D}$ <#s> <#q> 23 . ? (7)

<#s> <#q> "23"^^xsd:float . $\models_{D}$ <#s> <#q> 23.0 . ? (8)

<#s> <#q> "untrue"^^xsd:boolean . $\models_{D}$ <#s> <#q> false . ? (9)

<#s> <#q> 23.0 . $\models_{D}$ <#s> <#q> 23 , "23"^^xsd:int . ? (10)

Agenda

1. Defining a Logical Language
2. Interpretations and Satisfiability
3. Simple Entailment and *D*-Entailment
4. **RDF Interpretations**
5. RDFS Interpretations
6. Entailment Patterns for RDF and RDFS

RDF Vocabulary

- RDF interpretations consider the RDF vocabulary terms[1].
- RDF vocabulary terms are:

Properties:	Classes:	Datatypes:	Instances:
`rdf:type`	`rdf:Property`	`rdf:HTML`	`rdf:nil`
`rdf:subject`	`rdf:List`	`rdf:XMLLiteral`	
`rdf:predicate`	`rdf:Bag`	`rdf:langString`	
`rdf:object`	`rdf:Alt`	`rdf:PlainLiteral`	
`rdf:first`	`rdf:Seq`		
`rdf:rest`	`rdf:Statement`		
`rdf:value`			

$\left.\begin{array}{l}\texttt{rdf:\_1}\\ \texttt{rdf:\_2}\\ \dots\end{array}\right\}$ Number of elements in a container may be infinite

[1]http://www.w3.org/1999/02/22-rdf-syntax-ns#

RDF Axiomatic Triples

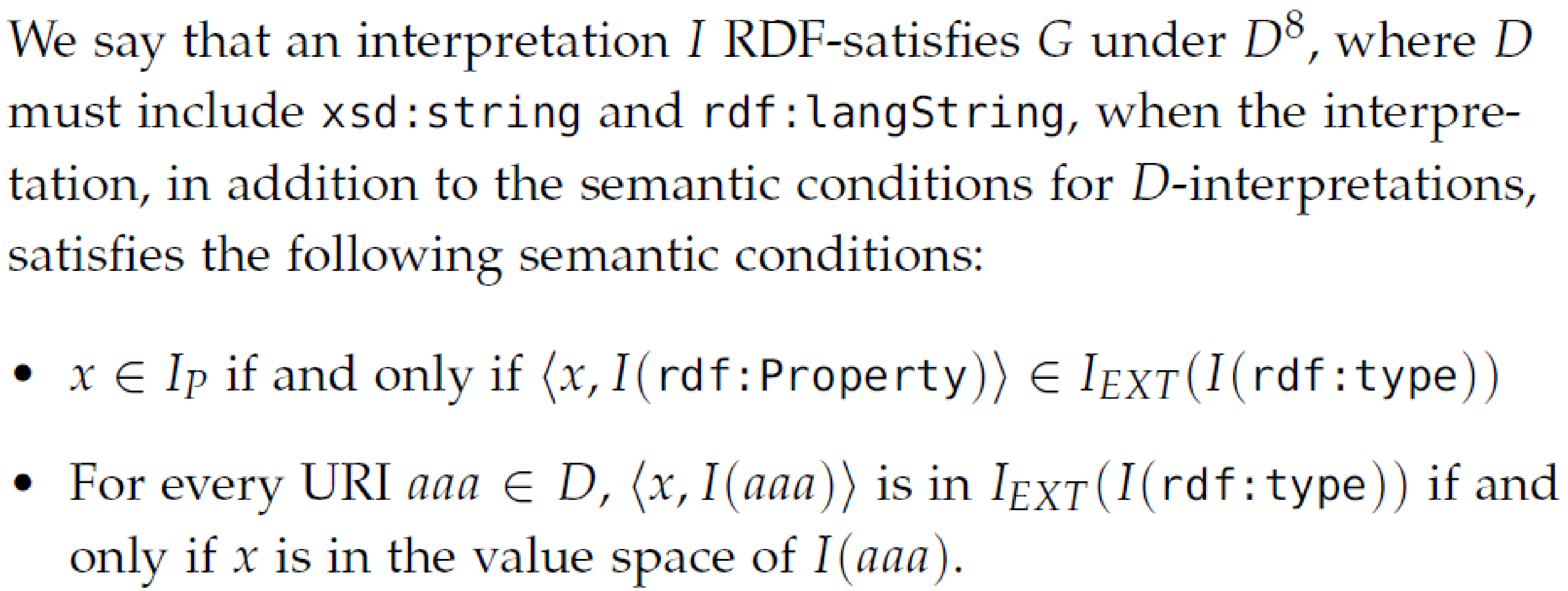
- What is an axiom?
  - A self-evident or universally recognised truth [1]
  - An established rule, principle, or law [1]
- The following triples have to be true in any RDF interpretation, by definition:

```
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 .
...
```

Since the elements of a container may be infinite, the application of the axiomatic triples results in an infinite interpretation

[1] http://www.thefreedictionary.com/axiom

Semantic Conditions for RDF Interpretations



We say that an interpretation I RDF-satisfies G under D[8], where D must include `xsd:string` and `rdf:langString`, when the interpretation, in addition to the semantic conditions for D-interpretations, satisfies the following semantic conditions:

- $x \in I_P$ if and only if $\langle x, I(\texttt{rdf:Property})\rangle \in I_{EXT}(I(\texttt{rdf:type}))$
- For every URI $aaa \in D$, $\langle x, I(aaa)\rangle$ is in $I_{EXT}(I(\texttt{rdf:type}))$ if and only if x is in the value space of $I(aaa)$.

The first condition states that predicates of triples are of type `rdf:Property`.

The second condition states that literals are an instance of their datatype.

Agenda

1. Defining a Logical Language
2. Interpretations and Satisfiability
3. Simple Entailment and *D*-Entailment
4. RDF Interpretations
5. **RDFS Interpretations**
6. Entailment Patterns for RDF and RDFS

RDFS Vocabulary

- RDFS interpretations consider the correct handling of RDFS[1] terms
- Similar to RDF there is also a set of axiomatic triples that are valid by definition
- RDFS terms are:

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

[1] http://www.w3.org/TR/rdf-schema/

RDFS Axiomatic Triples

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

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 a rdfs:ContainerMembershipProperty ; rdfs:domain rdfs:Resource ; rdfs:range rdfs:Resource .
rdf:_2 a rdfs:ContainerMembershipProperty ; rdfs:domain rdfs:Resource ; rdfs:range rdfs:Resource .
...
```

Semantic Conditions for RDFS - Shortcuts

- We introduce the following shortcuts:
  - $I_{CEXT}(y)$ is defined to be $\{x : \langle x, y\rangle \in I_{EXT}(I(\texttt{rdf:type}))\}$
  - I_C is defined to be $I_{CEXT}(I(\texttt{rdfs:Class}))$

I_{CEXT}(y) is the set of instances of y.

Semantic Conditions for RDFS Interpretations (1)

- Conditions related to literals/datatypes
  - $I_{CEXT}(I(\texttt{rdf:langString})) \in \{I(E) : E \text{ is a language-tagged string }\}$
  - For every URI $aaa \in D$ other than `rdf:langString`, $I_{CEXT}(I(aaa))$ is the value space of $I(aaa)$
  - For every URI $aaa \in D$, $I(aaa) \in I_{CEXT}(I(\texttt{rdfs:Datatype}))$

- Every member of I_R is of `rdf:type rdfs:Resource`
  - $I_{CEXT}(I(\texttt{rdfs:Resource})) = I_R$

- Conditions related to `rdfs:domain`/`rdfs:range`
  - If $\langle x, y\rangle \in I_{EXT}(I(\texttt{rdfs:domain}))$ and $\langle u, v\rangle \in I_{EXT}(x)$ then $u \in I_{CEXT}(y)$
  - If $\langle x, y\rangle \in I_{EXT}(I(\texttt{rdfs:range}))$ and $\langle u, v\rangle \in I_{EXT}(x)$ then $v \in I_{CEXT}(y)$

Semantic Conditions for RDFS Interpretations (2)

- Conditions related to `rdfs:subPropertyOf`
  - $I_{EXT}(I(\texttt{rdfs:subPropertyOf}))$ is transitive and reflexive on I_P
  - If $\langle x, y \rangle \in I_{EXT}(I(\texttt{rdfs:subPropertyOf}))$ then x and $y \in I_P$ and $I_{EXT}(x) \subset I_{EXT}(y)$

- Conditions related to `rdfs:subClassOf`
  - If $x \in I_C$ then $\langle x, I(\texttt{rdfs:Resource}) \rangle \in I_{EXT}(I(\texttt{rdfs:subClassOf}))$
  - $I_{EXT}(I(\texttt{rdfs:subClassOf}))$ is transitive and reflexive on I_C
  - If $\langle x, y \rangle \in I_{EXT}(I(\texttt{rdfs:subClassOf}))$ then x and $y \in I_C$ and $I_{CEXT}(x) \subset I_{CEXT}(y)$

Semantic Conditions for RDFS Interpretations (3)

- Condition for the `rdfs:member` property
  - If $x \in I_{CEXT}(I(\texttt{rdfs:ContainerMembershipProperty}))$ then $\langle x, I(\texttt{rdfs:member})\rangle \in I_{EXT}(I(\texttt{rdfs:subPropertyOf}))$

- Condition ensuring every literal is of type `rdfs:Literal`
  - If $x \in I_{CEXT}(I(\texttt{rdfs:Datatype}))$ then $\langle x, I(\texttt{rdfs:Literal})\rangle \in I_{EXT}(I(\texttt{rdfs:subClassOf}))$

Think-Pair-Share: Satisfiability

- Consider graph G and interpretation I

Graph G

```
    :s :p :o .
```

- Does I simply satisfy G?
- Does I *D*-satisfy G?
- Does I RDF-satisfy G?
- Does I RDFS-satisfies G?

Interpretation I

$I_R = \{r1,r2,r3\}$

$I_P = \{p1,p2\}$

$I_{EXT}(p1) = \{\langle r1, r2\rangle\}$

$I_{EXT}(p2) = \{\langle r1, r3\rangle\}$

$I_S(\texttt{:s}) = r1$

$I_S(\texttt{:o}) = r2$

$I_S(\texttt{:p}) = p1$

$I_S(\texttt{:name}) = p2$

$I_L(\texttt{"Germany"}) = r3$

Think-Pair-Share: Unsatisfiability

- Assume the following prefixes:

```
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
```

- Are the following graphs simply satisfiable? *D*-satisfiable? RDF-satisfiable? RDFS-satisfiable?

Graph G1:

```
<#p> rdfs:range xsd:boolean .
<#s> <#p> “false"^^xsd:boolean .
```

Graph G2:

```
<#p> rdfs:range xsd:boolean .
<#s> <#p> "3.14"^^xsd:decimal .
```

No Formalisation

- The RDF (and RDFS) model theory does not provide detailed meaning of:
  - Containers: `rdf:Bag`, `rdf:Alt`, `rdf:Seq` (although discussions are currently taking place to provide a semantics for “arrays”, commonly used in JSON)
  - Collections (`rdf:List`)
  - Reification
  - `rdf:value`, `rdfs:label`, …

Agenda

From Defining Entailment to Deciding Entailment

- The model-theoretic definitions are elegant
- But do not give a decision procedure (an algorithm) to check entailment

- To apply entailment in practice, we need an algorithm

Generalised RDF Triple

- To apply RDF and RDFS entailment to a graph we need the concept of a generalised RDF triple

- RDF triple:

$$\langle s, p, o \rangle \in (U \cup B) \times U \times (U \cup B \cup L)$$

- Generalised triple:
  - Allows literals in subject position
  - Allows blank nodes and literals in predicate position

$$\Rightarrow \quad \langle s, p, o \rangle \in (U \cup B \cup L) \times (U \cup B \cup L) \times (U \cup B \cup L)$$

RDF Entailment Patterns

- The following entailment patterns can be used as an easy way to apply the RDF entailment rules to a graph
- Variables are denoted with a “?” (as in SPARQL)
- The patterns are applied by assigning values to the variables in the “If” statement and adding (inferring) the “Then” statement
- Patterns*:

	If …	Then …
rdfD1	`?x ?p "sss"^^ddd .`	`?x ?p _:n . _:n rdf:type ddd .`
rdfD2	`?x ?p ?y .`	`?p rdf:type rdf:Property .`

- Alternative pattern to rdfD1 (assuming generalised RDF)

	If …	Then …
GrdfD1	`?x ?p "sss"^^ddd .`	`"sss"^^ddd rdf:type ddd .`

- For the following examples we consider our graph: http://example.org/cities.ttl

* `"sss"` represents some Unicode string

RDF Entailment Patterns – GrdfD1

- Example:

If:

```
:Berlin :population "3500000"^^xsd:integer .
   ?x        ?p           "sss" ^^    ddd      .
```

```
?x ?p "sss"^^ddd .
```

Then:

```
"3500000"^^xsd:integer rdf:type xsd:integer .
```

RDF Entailment Patterns – rdfD2

- Example:

If:

```
:Pankow     :borough     :Berlin  .
  ?x           ?p          ?y      .
```

`?p rdf:type rdf:Property .`

Then:

```
:borough     rdf:type     rdf:Property .
```

RDFS Entailment Patterns

	If…	Then…
rdfs1	Any URI ddd in D	`ddd rdf:type rdfs:Datatype .`
rdfs2	`?p rdfs:domain ?x . ?y ?p ?z .`	`?y rdf:type ?x .`
rdfs3	`?p rdfs:range ?x . ?y ?p ?z .`	`?z rdf:type ?x .`
rdfs4a	`?x ?p ?z .`	`?x rdf:type rdfs:Resource .`
rdfs4b	`?y ?p ?z .`	`?z rdf:type rdfs:Resource .`

rdfs5	`?x rdfs:subPropertyOf ?y .`
	`?y rdfs:subPropertyOf ?z .`
		`?x rdfs:subPropertyOf ?z .`

rdfs6	`?x rdf:type rdf:Property .`	`?x rdfs:subPropertyOf ?x .`

rdfs7	`?p2 rdfs:subPropertyOf ?p1 .`
	`?x ?p2 ?y.`
		`?x ?p1 ?y .`

rdfs8	`?x rdf:type rdfs:Class .`	`?x rdfs:subClassOf`
		`rdfs:Resource .`

rdfs9	`?x rdfs:subClassOf ?y .`
	`?z rdf:type ?x .`
		`?z rdf:type ?y .`

rdfs10	`?x rdf:type rdfs:Class .`	`?x rdfs:subClassOf ?x .`

rdfs11	`?x rdfs:subClassOf ?y .`
	`?y rdfs:subClassOf ?z .`
		`?x rdfs:subClassOf ?z .`

rdfs12	`?x rdf:type`
	`rdfs:ContainerMembershipProperty.`
		`?x rdfs:subPropertyOf rdfs:member`
		`.`

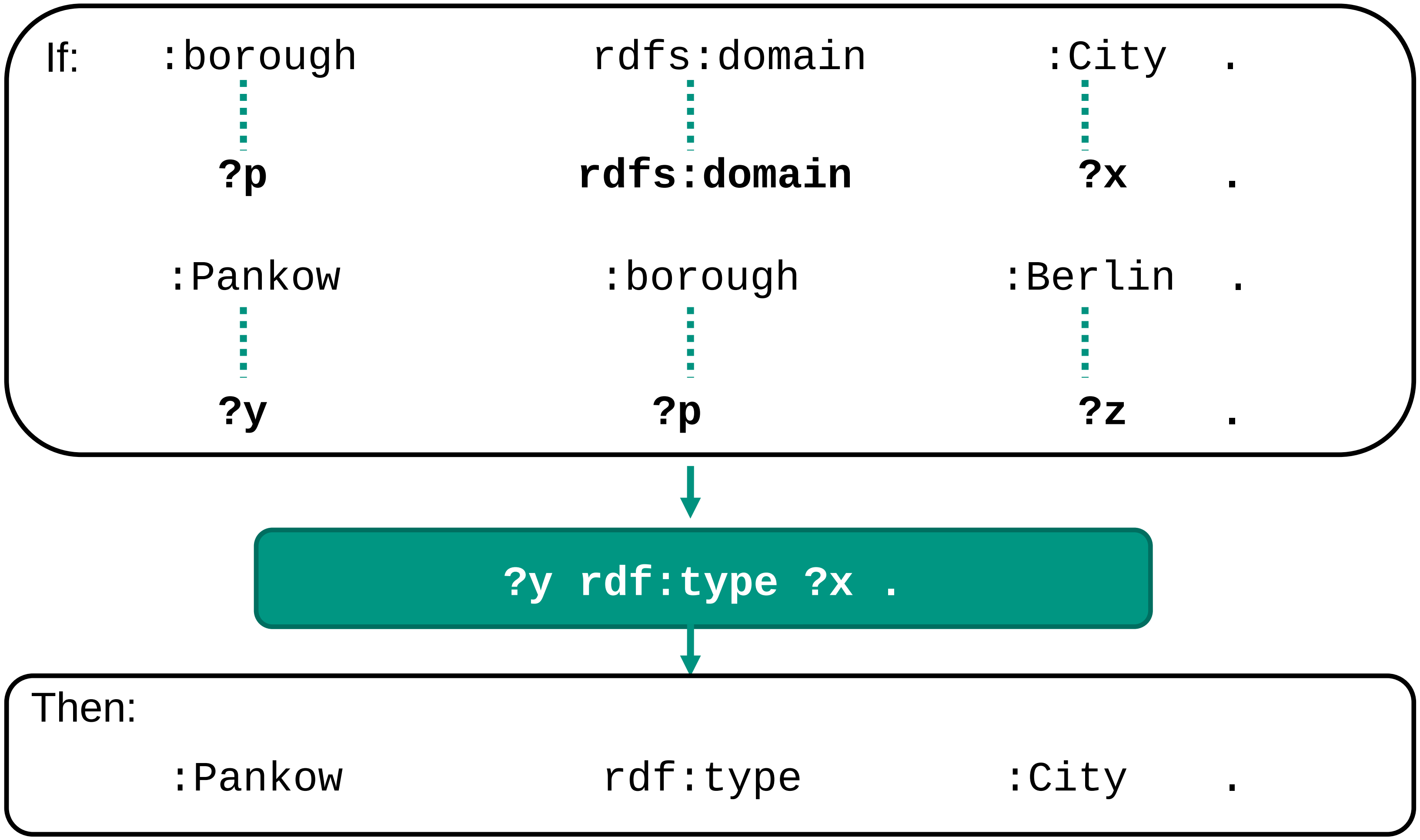
rdfs13	`?x rdf:type rdfs:Datatype .`	`?x rdfs:subClassOf rdfs:Literal .`

Relation Between RDFS Entailment Patterns and Semantic Conditions for RDFS Interpretations

- $I_{CEXT}(I(\texttt{rdfs:Resource})) = I_R$ **rdfs4a and rdfs4b**
- $I_{CEXT}(I(\texttt{rdf:langString})) \in \{I(E) : E \text{ is a language-tagged string }\}$
- for every other URI $aaa \in D$, $I_{CEXT}(I(aaa))$ is the value space of $I(aaa)$
- for every URI $aaa \in D$, $I(aaa) \in I_{CEXT}(I(\texttt{rdfs:Datatype}))$ **~rdfs1**
- If $\langle x, y\rangle \in I_{EXT}(I(\texttt{rdfs:domain}))$ and $\langle u, v\rangle \in I_{EXT}(x)$ then $u \in I_{CEXT}(y)$ **rdfs2**
- If $\langle x, y\rangle \in I_{EXT}(I(\texttt{rdfs:range}))$ and $\langle u, v\rangle \in I_{EXT}(x)$ then $v \in I_{CEXT}(y)$ **rdfs3**
- $I_{EXT}(I(\texttt{rdfs:subPropertyOf}))$ is transitive and reflexive on I_P **rdfs5, rdfs6**
- If $\langle x, y\rangle \in I_{EXT}(I(\texttt{rdfs:subPropertyOf}))$ then x and $y \in I_P$ and $I_{EXT}(x) \subset I_{EXT}(y)$ **rdfs7**
- If $x \in I_C$ then $\langle x, I(\texttt{rdfs:Resource})\rangle \in I_{EXT}(I(\texttt{rdfs:subClassOf}))$ **rdfs8**
- $I_{EXT}(I(\texttt{rdfs:subClassOf}))$ is transitive and reflexive on I_C **rdfs10, rdfs11**
- If $\langle x, y\rangle \in I_{EXT}(I(\texttt{rdfs:subClassOf}))$ then x and $y \in I_C$ and $I_{CEXT}(x) \subset I_{CEXT}(y)$ **~rdfs9**
- If $x \in I_{CEXT}(I(\texttt{rdfs:ContainerMembershipProperty}))$ then: $\langle x, I(rdfs : member)\rangle \in I_{EXT}(I(\texttt{rdfs:subPropertyOf}))$ **rdfs12**
- If $x \in I_{CEXT}(I(\texttt{rdfs:Datatype}))$ then $\langle x, I(\texttt{rdfs:Literal})\rangle \in I_{EXT}(I(\texttt{rdfs:subClassOf}))$ **rdfs13**

RDFS Entailment Patterns – rdfs2

- Example:



If:

```
:borough     rdfs:domain    :City    .
?p           rdfs:domain    ?x       .

:Pankow      :borough       :Berlin  .
?y           ?p             ?z       .
```

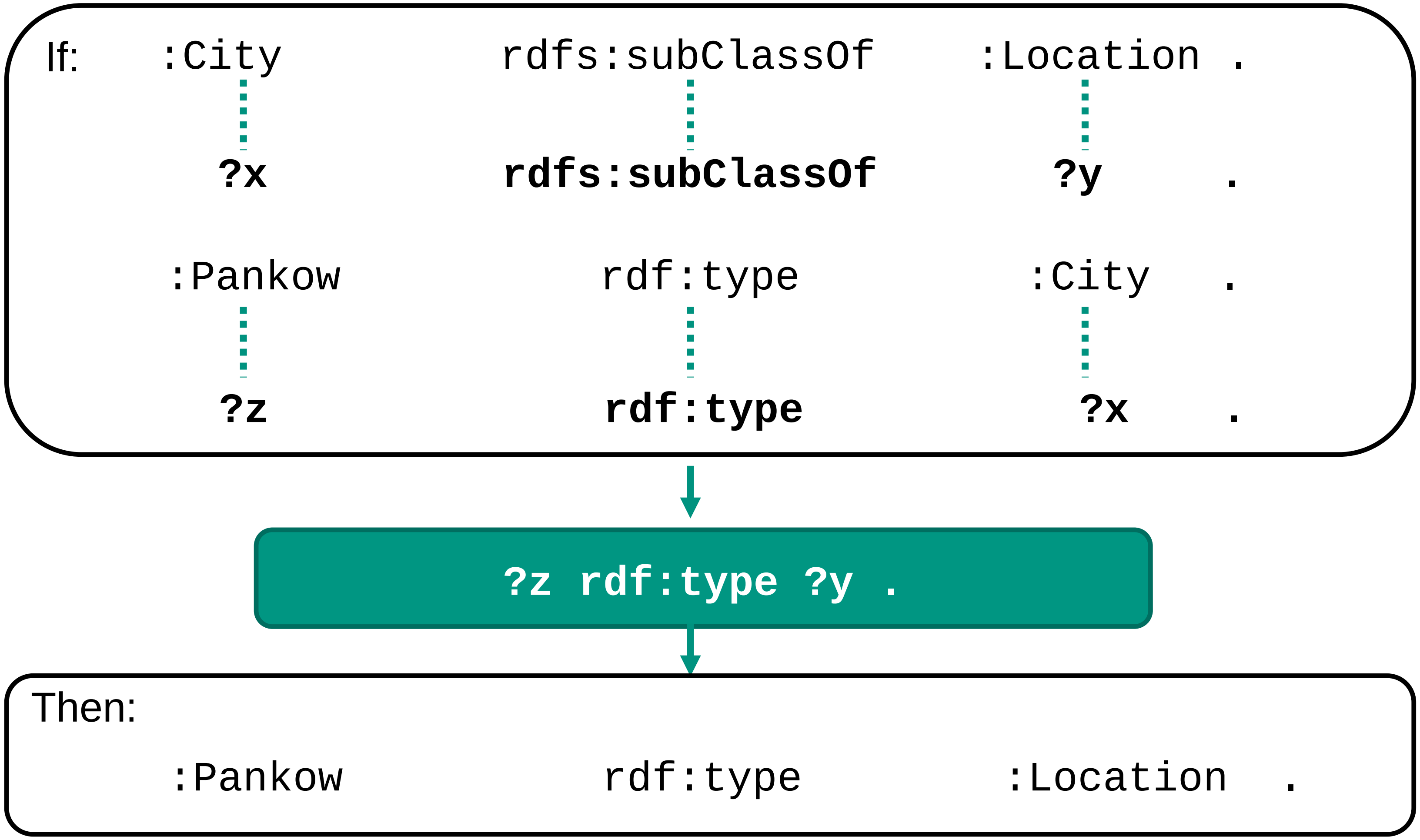
```
?y rdf:type ?x .
```

Then:

```
:Pankow      rdf:type       :City    .
```

RDFS Entailment Patterns – rdfs9

- Example:



If:
```
:City          rdfs:subClassOf   :Location .
?x             rdfs:subClassOf   ?y        .

:Pankow        rdf:type          :City     .
?z             rdf:type          ?x        .
```

```
?z rdf:type ?y .
```

Then:
```
:Pankow        rdf:type          :Location .
```

Think-Pair-Share: RDFS Entailment

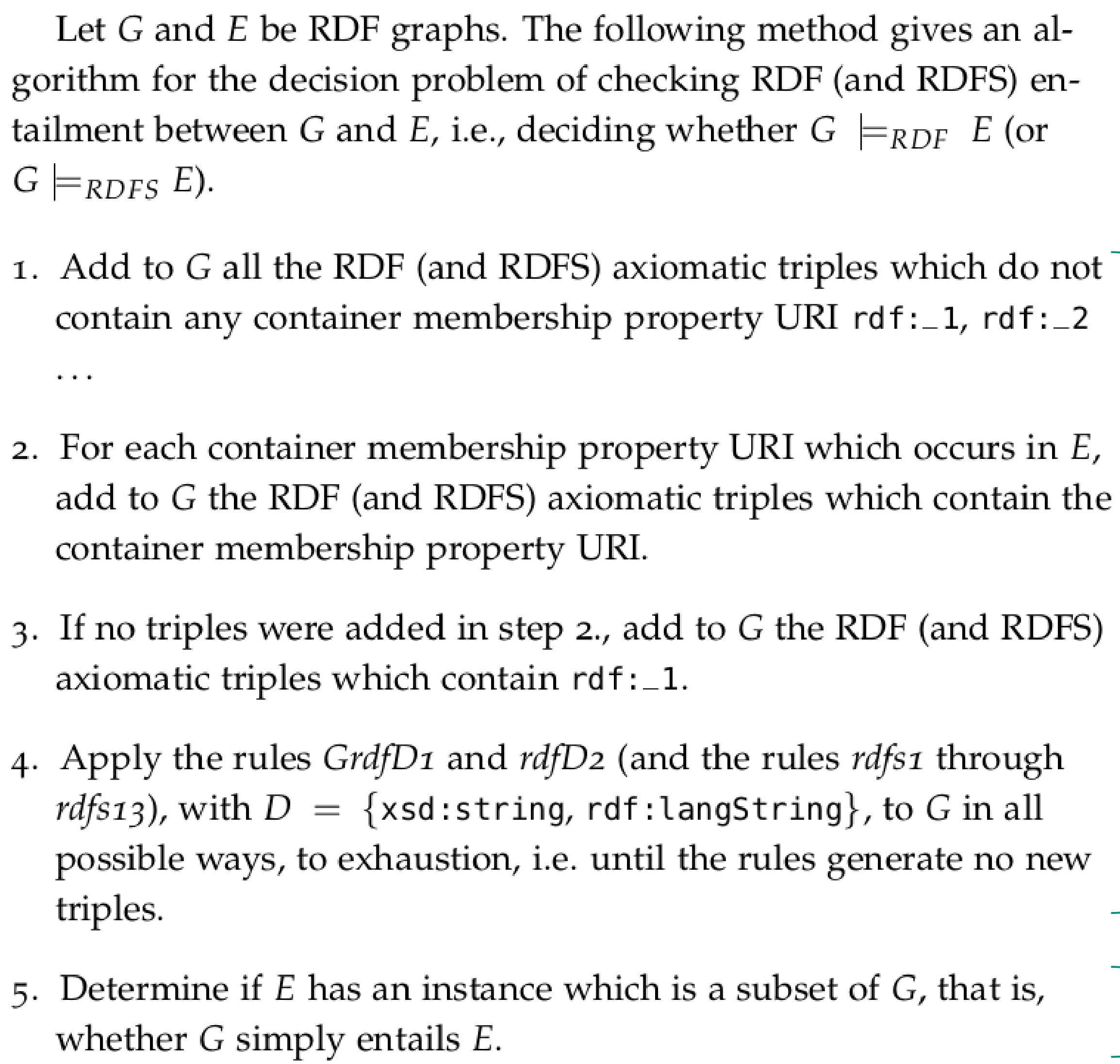
- Given the following RDF Graph G:

```
@prefix ex: <#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.

ex:speaksWith rdfs:domain ex:HomoSapiens .
ex:HomoSapiens rdfs:subClassOf ex:Primates .
ex:Alice ex:speaksWith ex:Bob .
```

For each of the following triples, answer whether the triple can be entailed under RDFS semantics from the graph G. For each triple that can be RDFS entailed, specify the RDFS axiomatic triples and RDFS entailment patterns that have to be applied to G in order to entail the triple.

a) `ex:Alice rdf:type ex:HomoSapiens .`
b) `ex:speaksWith rdfs:domain ex:Primates .`

Algorithm for Checking RDF (RDFS) Entailment *



Let G and E be RDF graphs. The following method gives an algorithm for the decision problem of checking RDF (and RDFS) entailment between G and E, i.e., deciding whether $G \models_{RDF} E$ (or $G \models_{RDFS} E$).

1. Add to G all the RDF (and RDFS) axiomatic triples which do not contain any container membership property URI `rdf:_1`, `rdf:_2` ...
2. For each container membership property URI which occurs in E, add to G the RDF (and RDFS) axiomatic triples which contain the container membership property URI.
3. If no triples were added in step 2., add to G the RDF (and RDFS) axiomatic triples which contain `rdf:_1`.
4. Apply the rules *GrdfD1* and *rdfD2* (and the rules *rdfs1* through *rdfs13*), with $D = \{$`xsd:string`, `rdf:langString`$\}$, to G in all possible ways, to exhaustion, i.e. until the rules generate no new triples.
5. Determine if E has an instance which is a subset of G, that is, whether G simply entails E.

* Missing: check lexical space of literals of datatype URIs in D

Compute the minimal model

Blank nodes and subgraph

Learning Goals

- G 9.1 Compare and contrast the correspondence theory of truth, the coherence theory of truth and the consensus theory of truth.
- G 9.2 Given an RDF graph, an interpretation and the semantic conditions for simple (*D*-, RDF, RDFS) interpretations, check whether the interpretation simply (*D*-, RDF, RDFS) satisfies the graph.
- G 9.3 Identify reasons for *D*- (RDF, RDFS) unsatisfiability of RDF graphs.
- G 9.4 Explain which triples follow from a given graph according to simple (*D*-, RDF, RDFS) entailment.
- G 9.5 Decide whether a given triple follows according to simple (*D*-, RDF, RDFS) entailment; specify which entailment patterns and axiomatic triples are involved.

Outlook – Chapter 10

- More ontology modelling constructs with the Web Ontology Language OWL