

Knowledge Representation for the Semantic Web

Winter Quarter 2012

Slides 4 – 01/12/2012

Pascal Hitzler

Kno.e.sis Center

Wright State University, Dayton, OH

<http://www.knoesis.org/pascal/>



Textbook (required)

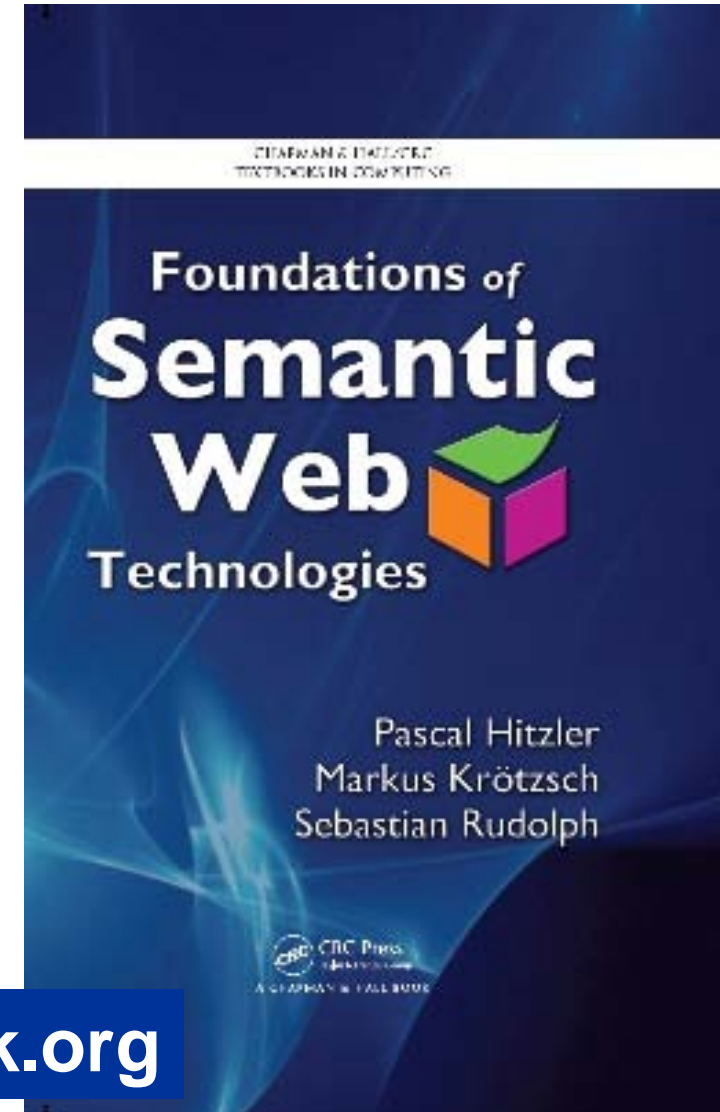
**Pascal Hitzler, Markus Krötzsch,
Sebastian Rudolph**

**Foundations of Semantic Web
Technologies**

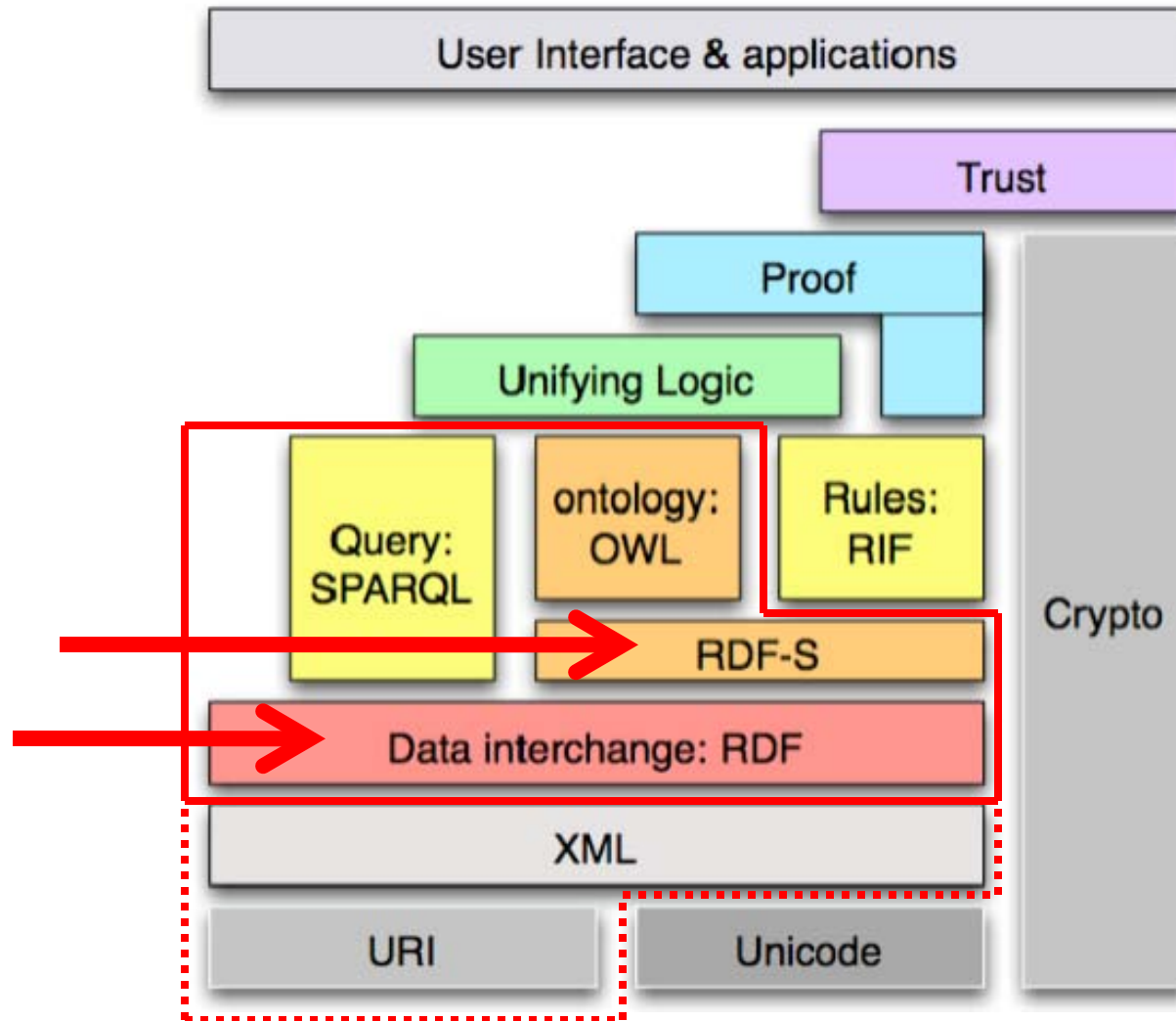
Chapman & Hall/CRC, 2010

**Choice Magazine Outstanding Academic
Title 2010 (one out of seven in Information
& Computer Science)**

<http://www.semantic-web-book.org>



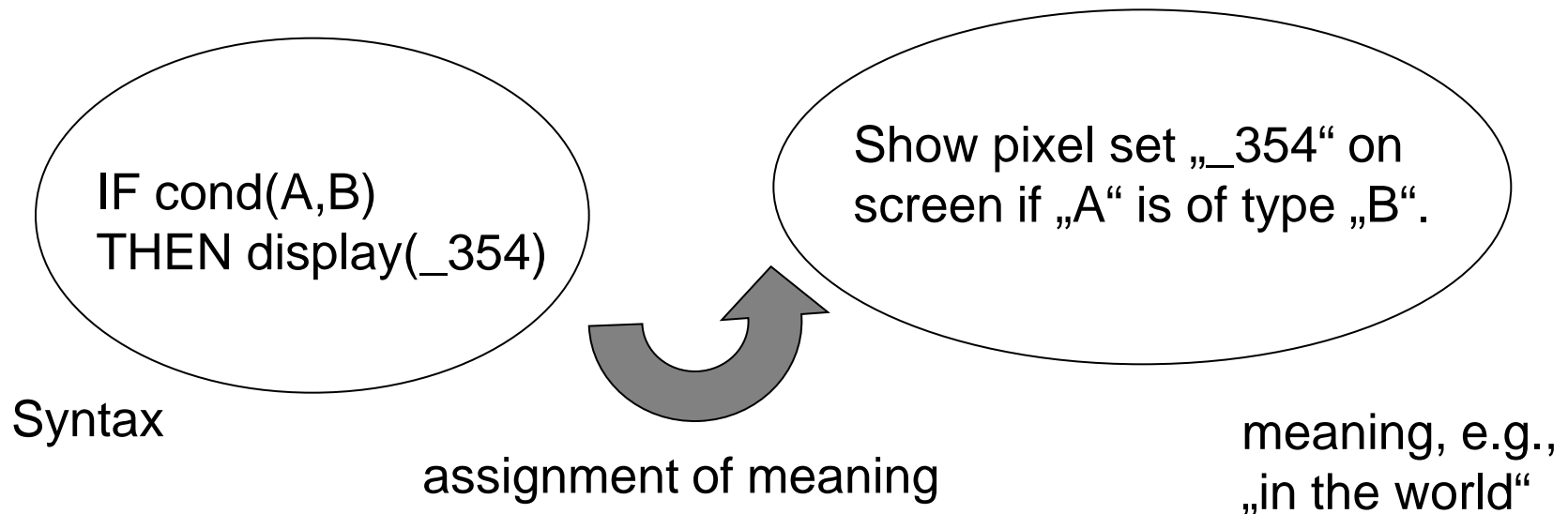
Today: RDF(S) semantics

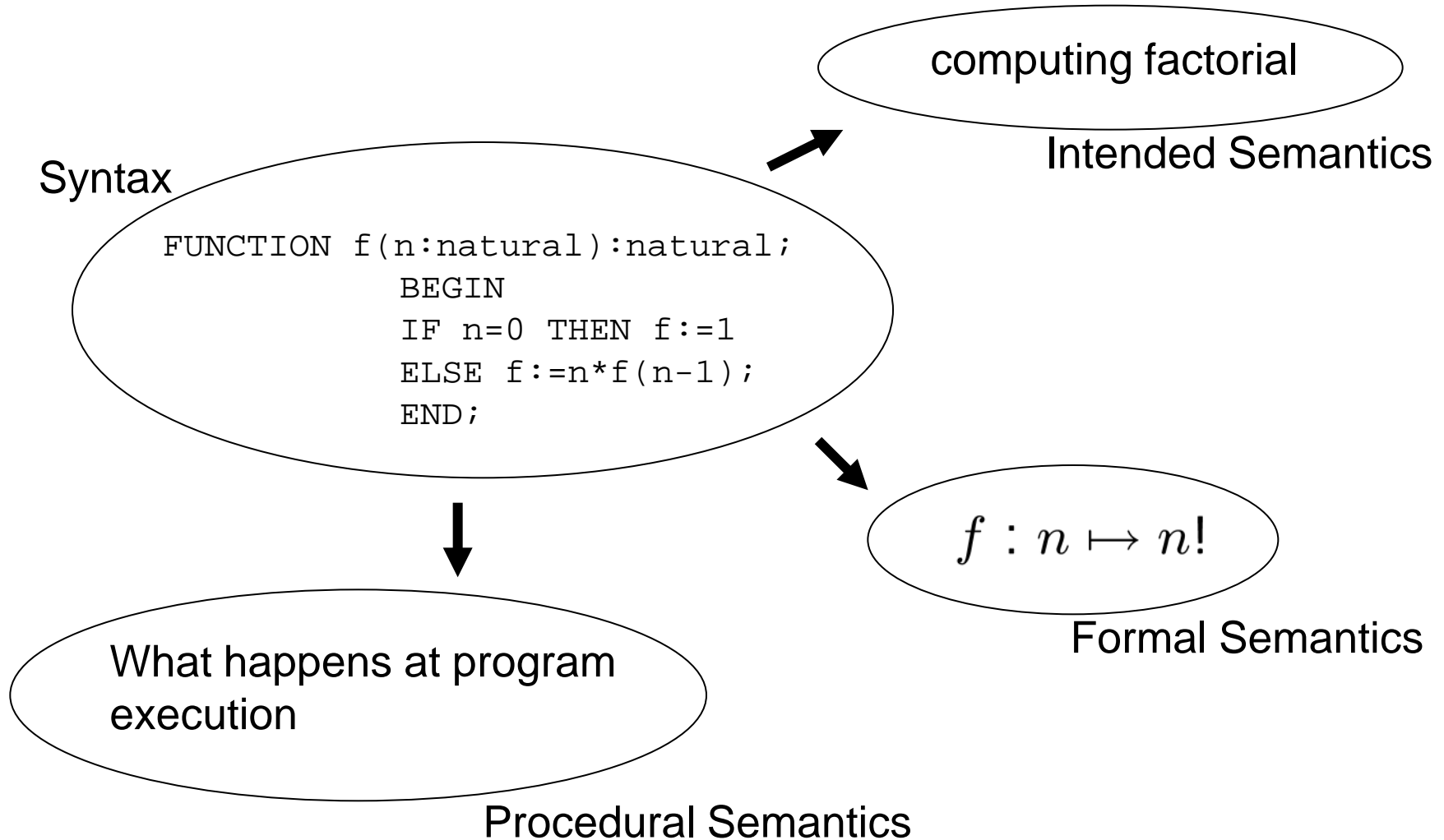


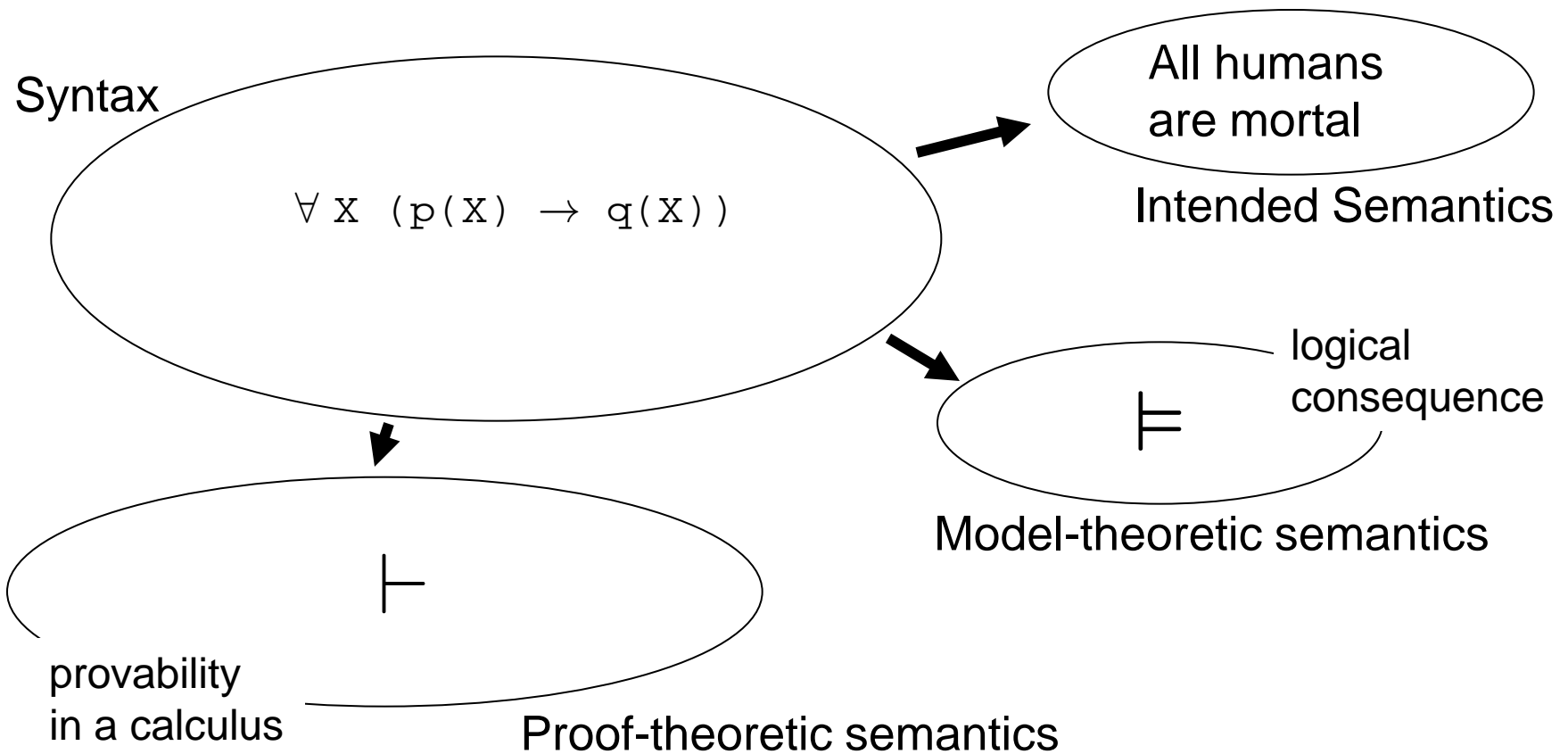
- 1. What is Semantics?**
- 2. What is Model-theoretic Semantics?**
- 3. Model-theoretic Semantics for RDF(S)**
- 4. What is Proof-theoretic Semantics?**
- 5. Proof-theoretic Semantics for RDF(S)**
- 6. Class Project**
- 7. Class Presentations**

Syntax: character strings without meaning

Semantics: meaning of the character strings







- if an RDFS document contains

```
u    rdf:type    ex:Textbook .
```

and

```
ex:Textbook  rdfs:subClassOf  ex:Book .
```

then

```
u    rdf:type    ex:Book .
```

is *implicitly* also the case: it's a *logical consequence*. (We can also say it is *deduced* (deduction) or *inferred* (inference)).

We do not have to state this explicitly.

Which statements are logical consequences is governed by the formal semantics (covered in the next session).

- **From**

```
ex:Textbook    rdfs:subClassOf    ex:Book .
```

```
ex:Book        rdfs:subClassOf    ex:PrintMedia .
```

the following is a logical consequence:

```
ex:Textbook    rdfs:subClassOf    ex:PrintMedia .
```

I.e. `rdfs:subClassOf` is *transitive*.

- **Opinions Differ. Here's my take.**
- **Semantic Web requires a shareable, declarative and *computable* semantics.**
- **I.e., the semantics must be a formal entity which is clearly defined and automatically computable.**
- **Ontology languages provide this by means of their formal semantics.**
- **Semantic Web Semantics is given by a relation – the *logical consequence relation*.**

We capture the meaning of information

**not by specifying its meaning (which is impossible)
but by specifying**

how information interacts with other information.

We describe the meaning indirectly through its effects.

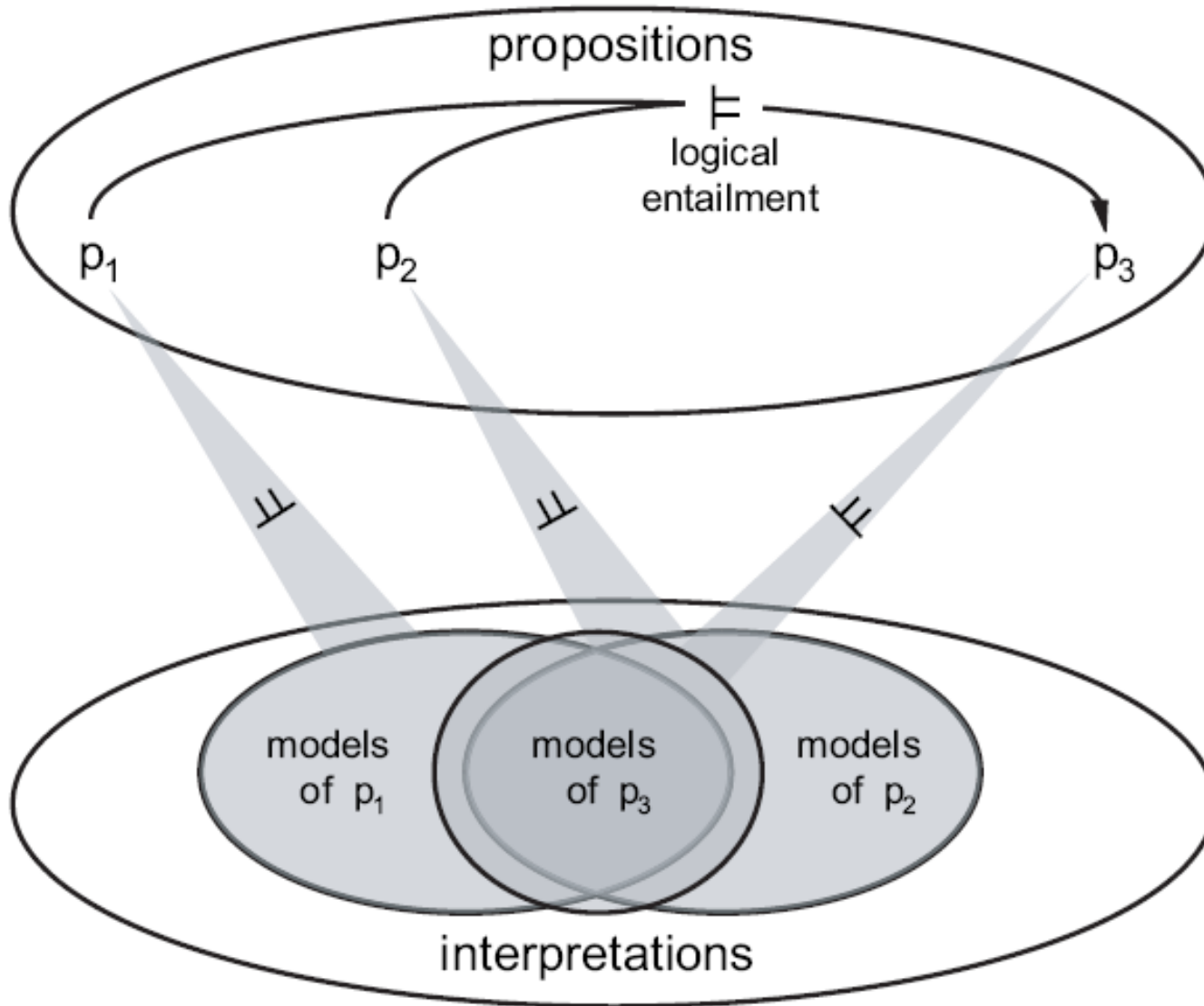
1. What is Semantics?
2. **What is Model-theoretic Semantics?**
3. Model-theoretic Semantics for RDF(S)
4. What is Proof-theoretic Semantics?
5. Proof-theoretic Semantics for RDF(S)
6. Class Project
7. Class Presentations

- You need:
 - a language/syntax
 - a notion of *model* for sentences in the language
- Models
 - are made such that each sentence is either true or false in each model
 - If a sentence α is true in a model M , then we write $M \models \alpha$
- Logical consequence:
 - β is a logical consequence of α (written $\alpha \models \beta$), if for all M with $M \models \alpha$, we also have $M \models \beta$
 - If K is a set of sentences, we write $K \models \beta$ if $M \models \beta$ for each $M \models K$
 - If J is another set of sentences, we write $K \models J$ if $K \models \beta$ for each $\beta \in J$

(note that the notation \models is overloaded)



Logical Consequence



- **Language:**
variables ...,w,x,y,z,...
symbol η
allowed sentences: $a \eta b$ (for a, b any variables)

- **We want to know:**

What are the logical consequences of the set

$\{x \eta y, y \eta z\}$

- **To answer this, we must say what the models in our semantics are.**

- Say, a model I of a set K of sentences consists of
 - a set C of cars and
 - a function $I(\cdot)$ which maps each variable to a car in C such that, for each sentence $a \eta b$ in K we have that $I(a)$ has more horsepower than $I(b)$.
- We now claim that $\{x \eta y, y \eta z\} \models x \eta z$.
- Proof: Consider any model M of $\{x \eta y, y \eta z\}$. Since $M \models \{x \eta y, y \eta z\}$, we know that
 - $M(x)$ has more horsepower than $M(y)$ and
 - $M(y)$ has more horsepower than $M(z)$.Hence, $M(x)$ has more horsepower than $M(z)$, i.e. $M \models x \eta z$.

This argument holds for all models of $\{x \eta y, y \eta z\}$, therefore $\{x \eta y, y \eta z\} \models x \eta z$.

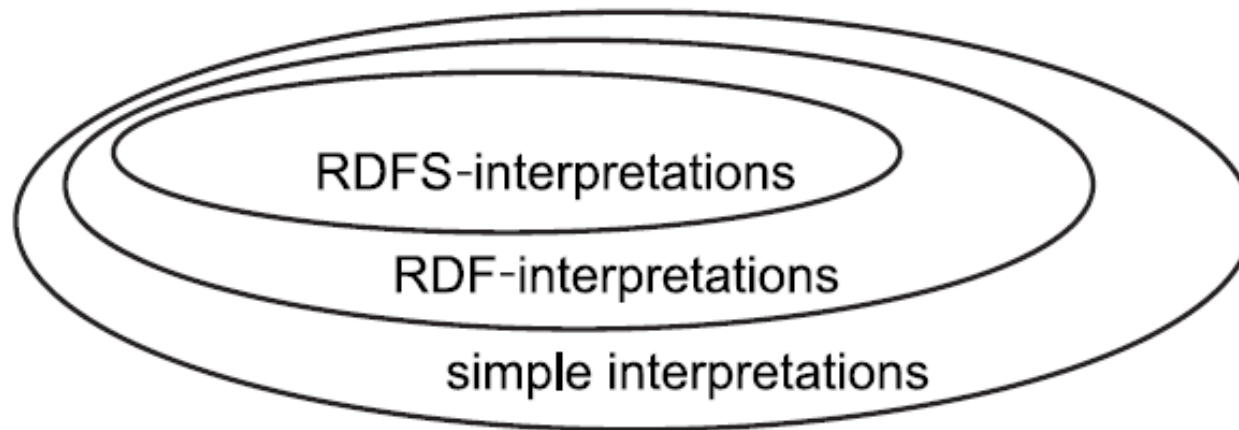
- Say, a model I of a set K of sentences consists of
 - a set C of cars and
 - a function $I(\cdot)$ which maps each variable to a car in C such that, for each sentence $a \eta b$ in K we have that $I(a)$ has more horsepower than $I(b)$.
- An *interpretation* I for a our language consists of
 - a set C of cars and
 - a function $I(\cdot)$ which maps each variable to a car in C .

(and that's it, i.e. no information whether a sentence is true or false with respect to I).

1. What is Semantics?
2. What is Model-theoretic Semantics?
3. **Model-theoretic Semantics for RDF(S)**
4. What is Proof-theoretic Semantics?
5. Proof-theoretic Semantics for RDF(S)
6. Class Project
7. Class Presentations

Now let's do this for RDF(S)

- **Language: Whatever is valid RDF(S).**
- **Sentences are triples. (Graphs are sets of triples.)**
- **Interpretations are given via sets and functions from language vocabularies to these sets.**
- **Models are defined such that they capture the intended meaning of the RDF(S) vocabulary.**
- **And there are three different notions:**



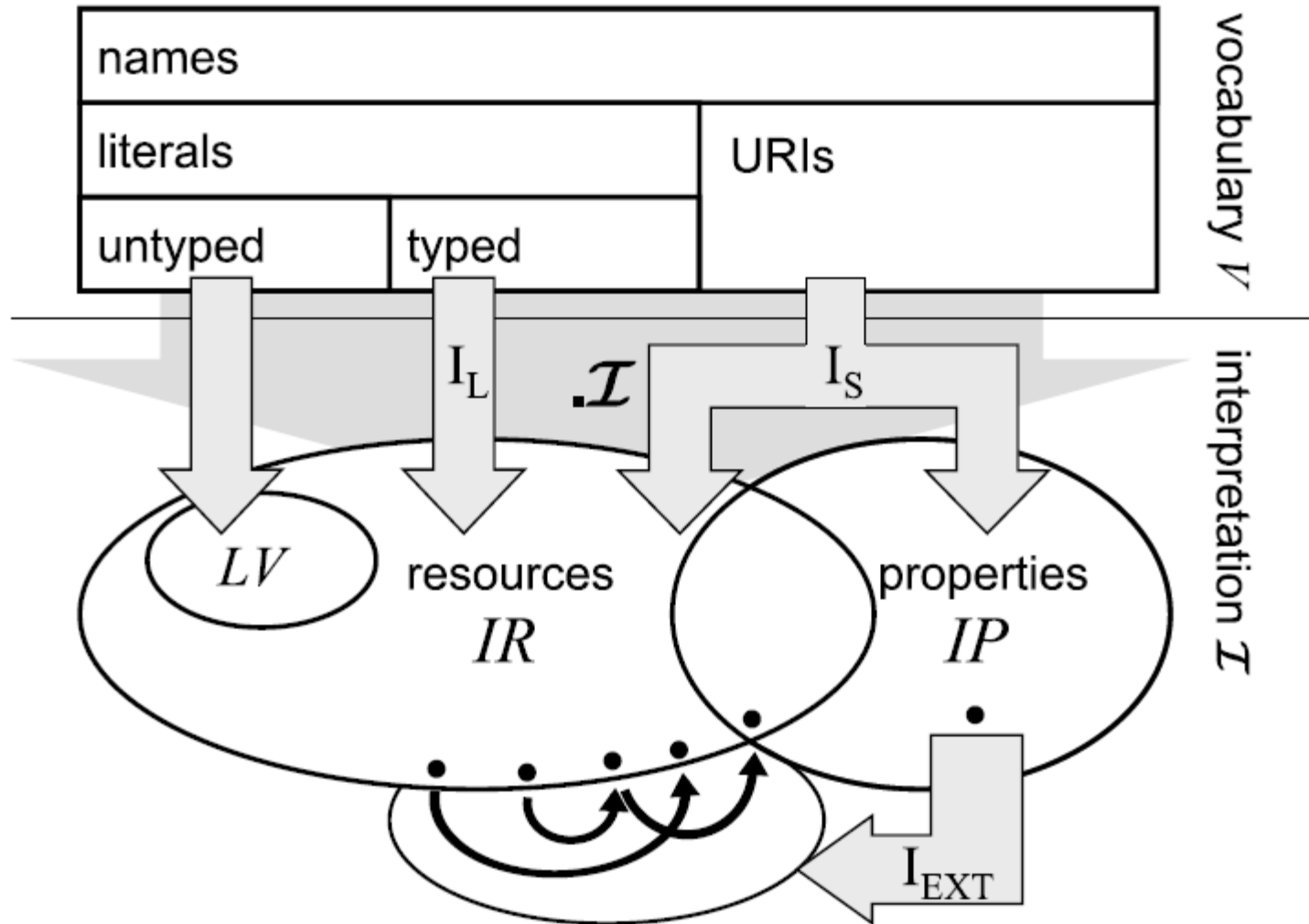
So we define: a *simple interpretation* \mathcal{I} of a given vocabulary V consists of

- IR , a non-empty set of *resources*, alternatively called domain or universe of discourse of \mathcal{I} ,
- IP , the set of *properties* of \mathcal{I} (which may overlap with IR),
- I_{EXT} , a function assigning to each property a set of pairs from IR , i.e. $I_{\text{EXT}} : IP \rightarrow 2^{IR \times IR}$, where $I_{\text{EXT}}(p)$ is called the *extension* of the property p ,
- I_S , a function, mapping URIs from V into the union of the sets IR and IP , i.e. $I_S : V \rightarrow IR \cup IP$,
- I_L , a function from the typed literals from V into the set IR of resources and
- LV , a particular subset of IR , called the set of *literal values*, containing (at least) all untyped literals from V .

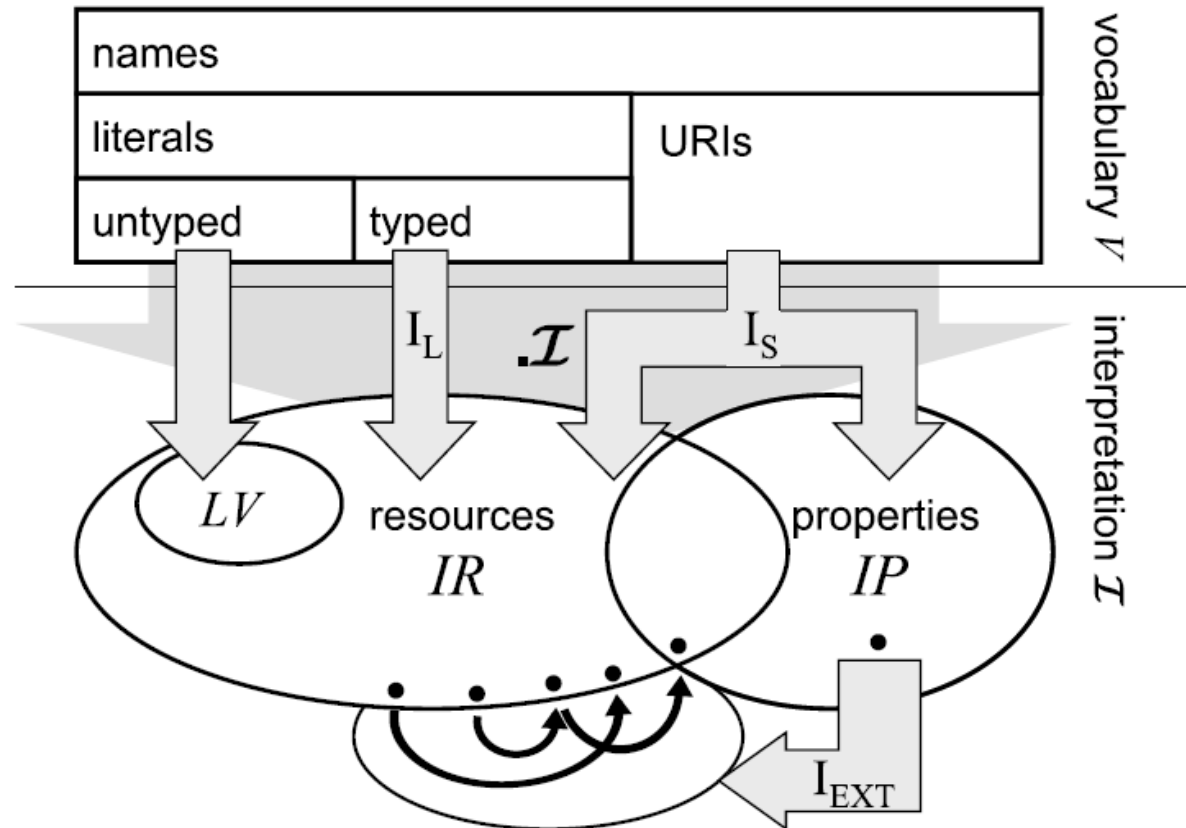
Now define an interpretation function $\cdot^{\mathcal{I}}$ (written as exponent).

- every untyped literal " a " is mapped to a , formally: $(\text{"}a\text{"})^{\mathcal{I}} = a$,
- every untyped literal carrying language information " a "@ t is mapped to the pair $\langle a, t \rangle$, i.e. $(\text{"}a\text{"}@t)^{\mathcal{I}} = \langle a, t \rangle$,
- every typed literal l is mapped to $I_L(l)$, formally: $l^{\mathcal{I}} = I_L(l)$, and
- every URI u is mapped to $I_S(u)$, i.e. $u^{\mathcal{I}} = I_S(u)$.

Simple Interpretations



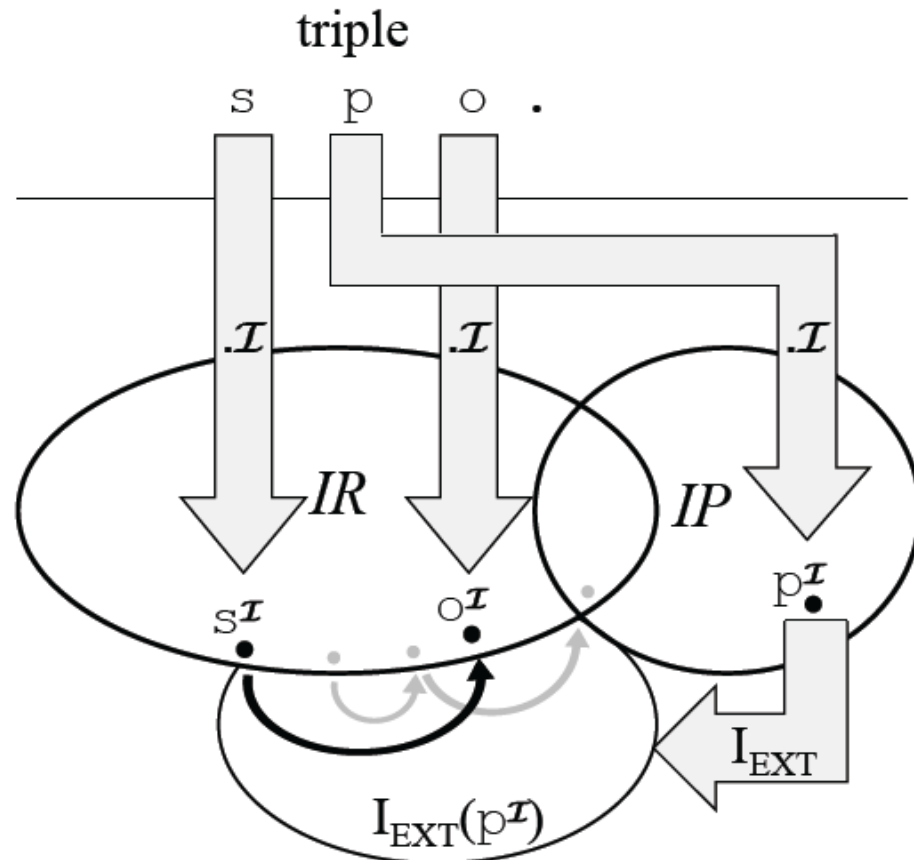
- The truth value $s p o.^{\mathcal{I}}$ of a (grounded*) triple $s p o.$ is true exactly if (s, p, o are contained in V) and $\langle s^{\mathcal{I}}, o^{\mathcal{I}} \rangle \in I_{\text{EXT}}(p^{\mathcal{I}})$.



* A grounded triple does not contain a blank node.

- The truth value $s \ p \ o.^{\mathcal{I}}$ of a (grounded*) triple $s \ p \ o.$ is true exactly if (s, p, o are contained in V) and $\langle s^{\mathcal{I}}, o^{\mathcal{I}} \rangle \in I_{\text{EXT}}(p^{\mathcal{I}})$.

* A grounded triple does not contain a blank node.



- Say, A is a function from blank nodes to URIs.
[these URIs need not be contained in the graph we're looking at]
- If, in a graph G , we replace each blank node x by $A(x)$, then we obtain a graph G' which we call a *grounding* of G .
- We know how to do the semantics for the grounded graphs.
- So define:
 $I \models G$ if and only if $I \models G'$ for **at least one** grounding G' of G .

- A graph G *simply entails* a graph G' if every simple interpretation that is a model of G is also a model of G' .
- (Recall that a simple interpretation is a model of a graph G if it is a model of each triple in G .)

- **Basically, $G \models G'$ if and only if G' can be obtained from G by replacing some nodes in G by blank nodes.**

- **It's really *simple* entailment.**

An *RDF-interpretation* of a vocabulary V is a simple interpretation of the vocabulary $V \cup V_{\text{RDF}}$ that additionally satisfies the following conditions:

- $x \in IP$ exactly if $\langle x, \text{rdf:Property}^{\mathcal{I}} \rangle \in I_{\text{EXT}}(\text{rdf:type}^{\mathcal{I}})$.
- if `"s"^^rdf:XMLLiteral` is contained in V and s is a well-typed XML-Literal, then
 - $I_L(\text{"s"^^rdf:XMLLiteral})$ is the XML value of s ;
 - $I_L(\text{"s"^^rdf:XMLLiteral}) \in LV$;
 - $\langle I_L(\text{"s"^^rdf:XMLLiteral}), \text{rdf:XMLLiteral}^{\mathcal{I}} \rangle \in I_{\text{EXT}}(\text{rdf:type}^{\mathcal{I}})$
- if `"s"^^rdf:XMLLiteral` is contained in V and s is an ill-typed XML literal, then
 - $I_L(\text{"s"^^rdf:XMLLiteral}) \notin LV$ and
 - $\langle I_L(\text{"s"^^rdf:XMLLiteral}), \text{rdf:XMLLiteral}^{\mathcal{I}} \rangle \notin I_{\text{EXT}}(\text{rdf:type}^{\mathcal{I}})$.

- In addition, each RDF-interpretation has to evaluate all the following triples to true:

<code>rdf : type</code>	<code>rdf : type</code>	<code>rdf : Property.</code>
<code>rdf : subject</code>	<code>rdf : type</code>	<code>rdf : Property.</code>
<code>rdf : predicate</code>	<code>rdf : type</code>	<code>rdf : Property.</code>
<code>rdf : object</code>	<code>rdf : type</code>	<code>rdf : Property.</code>
<code>rdf : first</code>	<code>rdf : type</code>	<code>rdf : Property.</code>
<code>rdf : rest</code>	<code>rdf : type</code>	<code>rdf : Property.</code>
<code>rdf : value</code>	<code>rdf : type</code>	<code>rdf : Property.</code>
<code>rdf : <i>_i</i></code>	<code>rdf : type</code>	<code>rdf : Property.</code>
<code>rdf : nil</code>	<code>rdf : type</code>	<code>rdf : List.</code>

- **Define (for a given RDF-interpretation \mathcal{I}):**
 - $I_{\text{CEXT}} : IR \rightarrow 2^{IR}$: We define $I_{\text{CEXT}}(y)$ to contain exactly those elements x for which $\langle x, y \rangle$ is contained in $I_{\text{EXT}}(\text{rdf:type}^{\mathcal{I}})$. The set $I_{\text{CEXT}}(y)$ is then also called the *(class) extension* of y .
 - $IC = I_{\text{CEXT}}(\text{rdfs:Class}^{\mathcal{I}})$.
- $IR = I_{\text{CEXT}}(\text{rdfs:Resource}^{\mathcal{I}})$
- $LV = I_{\text{CEXT}}(\text{rdfs:Literal}^{\mathcal{I}})$
- If $\langle x, y \rangle \in I_{\text{EXT}}(\text{rdfs:domain}^{\mathcal{I}})$ and $\langle u, v \rangle \in I_{\text{EXT}}(x)$, then $u \in I_{\text{CEXT}}(y)$.
- If $\langle x, y \rangle \in I_{\text{EXT}}(\text{rdfs:range}^{\mathcal{I}})$ and $\langle u, v \rangle \in I_{\text{EXT}}(x)$, then $v \in I_{\text{CEXT}}(y)$.
- $I_{\text{EXT}}(\text{rdfs:subPropertyOf}^{\mathcal{I}})$ is reflexive and transitive on IP .

- If $\langle x, y \rangle \in I_{\text{EXT}}(\text{rdfs:subPropertyOf}^{\mathcal{I}})$,
then $x, y \in IP$ and $I_{\text{EXT}}(x) \subseteq I_{\text{EXT}}(y)$.
- If $x \in IC$,
then $\langle x, \text{rdfs:Resource}^{\mathcal{I}} \rangle \in I_{\text{EXT}}(\text{rdfs:subClassOf}^{\mathcal{I}})$.
- If $\langle x, y \rangle \in I_{\text{EXT}}(\text{rdfs:subClassOf}^{\mathcal{I}})$,
then $x, y \in IC$ and $I_{\text{CEXT}}(x) \subseteq I_{\text{CEXT}}(y)$.
- $I_{\text{EXT}}(\text{rdfs:subClassOf}^{\mathcal{I}})$ is reflexive and transitive on IC .
- If $x \in I_{\text{CEXT}}(\text{rdfs:ContainerMembershipProperty}^{\mathcal{I}})$,
then $\langle x, \text{rdfs:member}^{\mathcal{I}} \rangle \in I_{\text{EXT}}(\text{rdfs:subPropertyOf}^{\mathcal{I}})$.
- If $x \in I_{\text{CEXT}}(\text{rdfs:Datatype}^{\mathcal{I}})$,
then $\langle x, \text{rdfs:Literal}^{\mathcal{I}} \rangle \in I_{\text{EXT}}(\text{rdfs:subClassOf}^{\mathcal{I}})$

- Furthermore, all of the following must be satisfied.

<code>rdf:type</code>	<code>rdfs:domain</code>	<code>rdfs:Resource</code> .
<code>rdfs:domain</code>	<code>rdfs:domain</code>	<code>rdf:Property</code> .
<code>rdfs:range</code>	<code>rdfs:domain</code>	<code>rdf:Property</code> .
<code>rdfs:subPropertyOf</code>	<code>rdfs:domain</code>	<code>rdf:Property</code> .
<code>rdfs:subClassOf</code>	<code>rdfs:domain</code>	<code>rdfs:Class</code> .
<code>rdf:subject</code>	<code>rdfs:domain</code>	<code>rdf:Statement</code> .
<code>rdf:predicate</code>	<code>rdfs:domain</code>	<code>rdf:Statement</code> .
<code>rdf:object</code>	<code>rdfs:domain</code>	<code>rdf:Statement</code> .
<code>rdfs:member</code>	<code>rdfs:domain</code>	<code>rdfs:Resource</code> .
<code>rdf:first</code>	<code>rdfs:domain</code>	<code>rdf:List</code> .
<code>rdf:rest</code>	<code>rdfs:domain</code>	<code>rdf:List</code> .
<code>rdfs:seeAlso</code>	<code>rdfs:domain</code>	<code>rdfs:Resource</code> .
<code>rdfs:isDefinedBy</code>	<code>rdfs:domain</code>	<code>rdfs:Resource</code> .

- **Furthermore, all of the following must be satisfied.**

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

- Furthermore, all of the following must be satisfied.

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

rdfs:isDefinedBy      rdfs:subPropertyOf  rdfs:seeAlso .

rdf:XMLLiteral
    rdf:type            rdfs:Datatype .
rdf:XMLLiteral
    rdfs:subClassOf      rdfs:Literal .
rdfs:Datatype
    rdfs:subClassOf      rdfs:Class .

rdf:_i
    rdf:type
        rdfs:ContainerMembershipProperty .

rdf:_i
    rdfs:domain          rdfs:Resource .
rdf:_i
    rdfs:range           rdfs:Resource .
```

1. What is Semantics?
2. What is Model-theoretic Semantics?
3. Model-theoretic Semantics for RDF(S)
4. **What is Proof-theoretic Semantics?**
5. Proof-theoretic Semantics for RDF(S)
6. Class Project
7. Class Presentations

- Say, a model I of a set K of sentences consists of
 - a set C of cars and
 - a function $I(\cdot)$ which maps each variable to a car in C such that, for each sentence $a \eta b$ in K we have that $I(a)$ has more horsepower than $I(b)$.
- Can we find an algorithm to compute all logical consequences of a set of sentences?
- Algorithm Input: set K of sentences
 1. The algorithm non-deterministically selects two sentences from K . If the first sentence is $a \eta b$, and the second sentence is $b \eta c$, then add $a \eta c$ to K .

IF $a \eta b \in K$ and $b \eta c \in K$ THEN $K \leftarrow \{a \eta c\}$
 2. Repeat step 1 until no selection results in a change of K .
 3. Output: K

- The algorithm produces only logical consequences: it is **sound** with respect to the model-theoretic semantics.
- The algorithm produces all logical consequences: it is **complete** with respect to the model-theoretic semantics.
- The algorithm always terminates.
- The algorithm is non-deterministic.
- What is the computational complexity of this algorithm?

- **Recall:**
 - β is a logical consequence of α (written $\alpha \models \beta$), if for **all** M with $M \models \alpha$, we also have $M \models \beta$ are
- **Implementing model-theoretic semantics directly is not feasible:** We would have to deal with *all* models of a knowledge base. Since there are a lot of cars in this world, we would have to check a lot of possibilities.
- **Proof theory reduces model-theoretic semantics to symbol manipulation!** It removes the models from the process.

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

is a so-called *deduction rule*. Such rules are usually written schematically as

$$\frac{a \eta b \quad b \eta c}{a \eta c}$$

1. What is Semantics?
2. What is Model-theoretic Semantics?
3. Model-theoretic Semantics for RDF(S)
4. What is Proof-theoretic Semantics?
5. **Proof-theoretic Semantics for RDF(S)**
6. Class Project
7. Class Presentations

- a and b can refer to arbitrary URIs (i.e. anything admissible for the predicate position in a triple),
- $_:n$ will be used for the ID of a blank node,
- u and v refer to arbitrary URIs or blank node IDs (i.e. any possible subject of a triple),
- x and y can be used for arbitrary URIs, blank node IDs or literals (i.e. anything admissible for the object position in a triple), and
- l may be any literal.

$$\frac{u \quad a \quad x \quad .}{u \quad a \quad _ : n \quad .} \text{ se1}$$

$$\frac{u \quad a \quad x \quad .}{_ : n \quad a \quad x \quad .} \text{ se2}$$

$_ : n$ must not be contained in the graph the rule is applied to

$\frac{}{u \ a \ x} \text{rdfax}$ for all RDF axiomatic triples $u \ a \ x$.

$\frac{u \ a \ l \ .}{u \ a \ _ : n \ .} \text{lg}$ where $_ : n$ does not yet occur in the graph

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

$\frac{u \ a \ l \ .}{_ : n \ \text{rdf:type} \ \text{rdf:XMLLiteral}} \text{rdf2}$

where $_ : n$ does not yet occur in the graph,
unless it has been introduced by a
preceding application of the lg rule

Additional RDFS-entailment Rules - I

$\frac{}{u \ a \ x} \text{rdfax}$ for all RDFS axiomatic triples $u \ a \ x$.

$\frac{u \ a \ l \ .}{_ : n \ \text{rdf:type} \ \text{rdfs:Literal} \ .} \text{rdfs1}$ with $_ : n$ as usual

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

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

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

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

Additional RDFS-entailment Rules - II

$$\frac{u \text{ rdfs:subPropertyOf } v . \quad v \text{ rdfs:subPropertyOf } x .}{u \text{ rdfs:subPropertyOf } x .} \text{ rdfs5}$$
$$\frac{u \text{ rdf:type rdf:Property } .}{u \text{ rdfs:subPropertyOf } u .} \text{ rdfs6}$$
$$\frac{a \text{ rdfs:subPropertyOf } b . \quad u \ a \ y .}{u \ b \ y .} \text{ rdfs7}$$
$$\frac{u \text{ rdf:type rdfs:Class } .}{u \text{ rdfs:subClassOf rdfs:Resource } .} \text{ rdfs8}$$
$$\frac{u \text{ rdfs:subClassOf } x . \quad v \text{ rdf:type } u .}{v \text{ rdf:type } x .} \text{ rdfs9}$$
$$\frac{u \text{ rdf:type rdfs:Class } .}{u \text{ rdfs:subClassOf } u .} \text{ rdfs10}$$

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

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

$$\frac{u \text{ rdf:type rdfs:Datatype } .}{u \text{ rdfs:subClassOf rdfs:Literal } .} \text{ rdfs13}$$

$$\frac{u \text{ a } _ : n .}{u \text{ a } l .} \text{ g1}$$

where $_ : n$ identifies a blank node introduced by an earlier “weakening” of the literal l via the rule lg

- The deduction rules for simple and RDF entailment are sound and complete.
- The deduction rules for RDFS entailment are sound.

The spec says, they are also complete, but they are not:

```
ex:isHappilyMarriedTo    rdfs:subPropertyOf    _:bnode .
_:bnode                  rdfs:domain           ex:Person .
ex:markus                ex:isHappilyMarriedTo ex:anja .
```

has as logical consequence

```
ex:markus                rdf:type              ex:Person .
```

but this is not derivable using the deduction rules.

Simple, RDF, and RDFS entailment are NP-complete problems.

If we disallow blank nodes, all three entailment problems are polynomial.

Does

```
ex:speaksWith    rdfs:domain    ex:Homo .  
ex:Homo          rdfs:subClassOf  ex:Primates .
```

entail

```
ex:speaksWith    rdfs:domain    ex:Primates .
```

?

A new W3C working group has just been chartered and should continue work shortly:

<http://www.w3.org/2011/01/rdf-wg-charter>

1. What is Semantics?
2. What is Model-theoretic Semantics?
3. Model-theoretic Semantics for RDF(S)
4. What is Proof-theoretic Semantics?
5. Proof-theoretic Semantics for RDF(S)
6. **Class Project**
7. Class Presentations

- keep bugfixing
- find, for your RDF Schema ontology, each of the following:
 - a triple which is RDFS-entailed, but not RDF-entailed
 - a triple which is RDF-entailed, but not simply entailed
 - a triple which is simply entailed
- For each of them, write down a justification why it is entailed.

- **send to me by Wednesday 18th of January**
 - **the current version of your Turtle RDF Schema document**
 - **the three entailed triples with explanations.**

1. What is Semantics?
2. What is Model-theoretic Semantics?
3. Model-theoretic Semantics for RDF(S)
4. What is Proof-theoretic Semantics?
5. Proof-theoretic Semantics for RDF(S)
6. Class Project
7. **Class Presentations**

<nothing yet>

Thursday 13th of January: RDFS Part I
Tuesday 18th of January: Exercise Session
Thursday 20th of January: RDF and RDFS Semantics
Tuesday 25th of January: RDF and RDFS Semantics
Thursday 27th of January: Description Logics
Tuesday 1st of March: Description Logic Semantics

Estimated breakdown of sessions:

Intro + XML: 2 RDF: 4 OWL and Logic: 6
SPARQL and Querying: 2 Class Presentations: 3
Exercise sessions: 3

Class planning (tentative)

Tuesday 10th of January: RDF Schema

Thursday 12th of January: RDF and RDFS Semantics

Tuesday 17th of January: Exercise Session

Then several OWL sessions.