Reasoning in the Semantic Web
RDF is a logical formalism

This means that there are **formal (mathematical)** definitions of:

- A notion of **interpretation**
- A notion of **satisfaction**
RDF interpretations (simplified)

An IRI or a literal denotes a **resource**. Everything can be a resource.

An **interpretation** says what IRIs and literals denote.
Some resources (in an interpretation) are said to be properties. In this case, they are associated with a binary relation over the set of resources (called the extension of the property).

Blank nodes are not interpreted. They simply indicate the existence of a thing.
RDF Interpretation (example1)

A possible interpretation ($I_1$) could be:

- `http://example.com/batman` denotes the super hero Batman
- `http://example.com/joker` denotes the super villain The Joker
- `http://xmlns.com/foaf/0.1/knows` denotes the friendship relation. It is associated with (i.e., its extension is) the set of pairs of people that are friends with each other.
Another possible interpretation ($I_2$) could be:

http://example.com/batman denotes the Batmobile

http://example.com/joker denotes the front left wheel of the Batmobile

http://xmlns.com/foaf/0.1/knows denotes the part-of relationship. Its extension is the set of pairs of physical entities where the first element is a physical part of the other.
Satisfaction (without blank nodes)

A triple \((s, p, o)\) is **satisfied** by an interpretation if \(p\) is a property and the resource denoted by \(s\) and the resource denoted by \(o\) are in the extension of the resource denoted by \(p\).

In this case, we say that \((s, p, o)\) **is true** according to the interpretation.

An RDF Graph (without blank nodes) is satisfied by an interpretation if and only if the interpretation satisfies all triples in it.
Satisfaction (example1)

For instance, \( l_1 \) does not satisfy this RDF triple (written in Turtle):

\[
\text{ex:joker foaf:knows ex:batman}.
\]

However, \( l_2 \) does satisfy this RDF triple.
Satisfaction (with blank nodes)

An RDF interpretation does not say what a blank node denotes.

In order to define whether an interpretation satisfies an RDF graph with blank nodes or not, we use the notion of extended interpretation.

An extended interpretation of an RDF interpretation has the same denotation of IRIs and literals as the original interpretation, but also defines what blank nodes denote.
Satisfaction (with blank nodes)

Extended interpretations satisfy RDF graph in the same way as RDF interpretations.

An RDF interpretation **satisfies** an RDF Graph if and only if it can be extended into an extended interpretation that satisfies the said RDF graph.
Satisfaction (examples)

According to $I_1$, the following triple is true:

\[
[] \text{ foaf:knows } \text{ ex:batman}
\]

(because Batman has some friends) but maybe not this one:

\[
[] \text{ foaf:knows } \text{ ex:joker}
\]

However, both triples are true in $I_2$ because a wheel has some parts.
Entailment

An RDF Graph \textbf{entails} another RDF Graph if and only if all interpretations that satisfy the first graph also satisfy the second.

For instance:

\texttt{ex:joker foaf:knows ex:batman}.

entails:

\texttt{[] foaf:knows []}.

but does not entail:

\texttt{ex:batman foaf:knows ex:joker}.
How to know what IRIs denote?

For people, social conventions can help determine the intended meaning of IRIs. But machines or programmes cannot know what IRIs denote. So, machines need something more to make useful deductions.
Semantic extension

A semantic extension in RDF defines additional constraints that interpretations must satisfy for certain specific IRIs.

RDF defines several semantic extensions, one of which is RDF Schema (RDFS).
RDFS interpretation

An RDFS interpretation is an RDF interpretation where the following must hold:

● the IRIs `rdf:first`, `rdf:object`, `rdf:predicate`, `rdf:rest`, `rdf:subject`, `rdf:type`, `rdf:value`, `rdfs:comment`, `rdfs:domain`, `rdfs:isDefinedBy`, `rdfs:label`, `rdf:member`, `rdfs:range`, `rdfs:seeAlso`, `rdfs:subClassOf`, `rdfs:subPropertyOf` denote properties.

● the extensions of the properties denoted by `rdfs:subClassOf` and `rdfs:subPropertyOf` are reflexive and transitive.
RDFS interpretation (contd.)

- if a resource is a property, then it, and the resource denoted by `rdf:Property` are part of the extension of what `rdf:type` denotes
- all resources are in relation with the resource denoted by `rdfs:Resource` in the extension of the resource denoted by `rdf:type`
RDFS interpretation (contd.)

- if \((x,y)\) is in the extension of the resource denoted by `rdfs:domain`, then \(x\) is a property and for all pairs \((u,v)\) in the extension of \(x\), the \((u,y)\) belongs to the extension of the resource denoted by `rdf:type`

- if \((x,y)\) is in the extension of the resource denoted by `rdfs:range`, then \(x\) is a property and for all pairs \((u,v)\) in the extension of \(x\), the \((v,y)\) belongs to the extension of the resource denoted by `rdf:type`
RDFS interpretation (contd.)

- if \((x, y)\) is in the extension of \texttt{rdfs:subPropertyOf}, then \(x\) and \(y\) are properties and the extension \(x\) is included in the extension of \(y\)

- if \(x\) and the resource denoted by \texttt{rdfs:Class} are related in the extension of the resource denoted by \texttt{rdf:type} then \(x\) and the resource denoted by \texttt{rdfs:Resource} are in the extension of the resource denoted by \texttt{rdfs:subClassOf}
RDFS interpretation (contd.)

- if \((x,y)\) is in the extension of the resource denoted by `rdfs:subClassOf` then \(x\) and \(y\) are related to the resource denoted by `rdfs:Class` in the extension of the resource denoted by `rdf:type` and anything that is in relation with \(x\) in the extension of the resource denoted by `rdf:type` is also in relation with \(y\)
RDFS interpretation (contd.)

- if $x$ is in relation with the resource denoted by $\text{rdfs:ContainerMembershipProperty}$ in the extension of the resource denoted by $\text{rdfs:type}$ then $x$ is in relation with the resource denoted by $\text{rdfs:member}$ in the extension of the resource denoted by $\text{rdfs:subPropertyOf}$

- if $x$ is in relation with the resource denoted by $\text{rdfs:Datatype}$ in the extension of the resource denoted by $\text{rdfs:type}$ then $x$ is in relation with the resource denoted by $\text{rdfs:Literal}$ in the extension of the resource denoted by $\text{rdfs:subClassOf}$
RDFS interpretation (contd.)

● The following triples must be satisfied:

    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 .
    rdfs:member rdfs:domain rdfs:Resource .
    rdf:first rdfs:domain rdf:List .
RDFS interpretation (contd.)

```turtle
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: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 .
```
RDFS interpretation (contd.)

```
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 interpretation (contd.)

```
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 RDF Graph **RDFS-entails** another RDF Graph if and only if all RDFS interpretations that satisfy the first graph also satisfy the second.

For instance:

```text
ex:joker foaf:knows ex:batman .
foaf:knows rdfs:domain foaf:Person .
```

entails:

```text
ex:joker rdf:type foaf:Person .
```
Practically useful RDFS entailments

If an RDF graph contains RDF triples of the form:

\[
\begin{align*}
  x \text{ rdf:type } y. \\
  y \text{ rdfs:subClassOf } z.
\end{align*}
\]

where \( x \) and \( y \) are IRIs or blank nodes and \( z \) then it entails:

\[
  x \text{ rdf:type } z.
\]

this can be written:

\[
\begin{align*}
  x \text{ rdf:type } y. \\
  y \text{ rdfs:subClassOf } z. \\
  x \text{ rdf:type } z.
\end{align*}
\]
Practically useful RDFS entailments

\( x \text{ rdfs:subClassOf } y. \)
\( y \text{ rdfs:subClassOf } z. \)
\( \frac{}{x \text{ rdfs:subClassOf } z.} \)
\hline
\( x \text{ rdfs:subPropertyOf } y. \)
\( y \text{ rdfs:subPropertyOf } z. \)
\( \frac{}{x \text{ rdfs:subPropertyOf } z.} \)
\hline
\( x \text{ rdfs:subClassOf } y. \)
\( a \text{ rdf:type } x. \)
\( \frac{}{a \text{ rdf:type } y.} \)
\hline
\( x \text{ rdfs:subPropertyOf } y. \)
\( a \times b. \)
\( \frac{}{a \times b.} \)
Practically useful RDFS entailments

\[ x \text{ rdfs:domain } y. \]
\[ a \ x \ b. \]
\[ a \text{ rdf:type } y. \]

\[ x \text{ rdfs:range } y. \]
\[ a \ x \ b. \]
\[ b \text{ rdf:type } y. \]

\[ x \text{ rdfs:domain } y. \]
\[ z \text{ rdfs:subPropertyOf } x. \]
\[ a \ z \ b. \]
\[ a \text{ rdf:type } y. \]

\[ x \text{ rdfs:range } y. \]
\[ z \text{ rdfs:subPropertyOf } x. \]
\[ a \ z \ b. \]
\[ b \text{ rdf:type } y. \]
Web Ontology Language (OWL)
Extends RDFS with more expressive features

Class equivalence:

```xml
ex:Human  owl:equivalentClass  foaf:Person .
```

Class disjointness:

```xml
```

Union / intersection:

```xml
ex:Person  owl:equivalentClass  [
    a  owl:Class;
    owl:unionOf  ( ex:Man  ex:Woman )
] .
```
OWL features (contd.)

Class complement:

```owl
ex:AbstractEntity owl:complementOf ex:ConcreteEntity .
```

Inverse property:

```owl
ex:isChildOf owl:inverseOf ex:isParentOf .
```

Typing attributes:

```owl
ex:Person rdfs:subClassOf [ a owl:Restriction;
 owl:onProperty ex:hasParent;
 owl:allValuesFrom ex:Person ] .
```
OWL features (contd.)

Existing attributes:

\[
\text{ex:Person} \text{ rdfs:subClassOf [ a owl:Restriction; owl:onProperty ex:hasParent; owl:someValuesFrom ex:Person ] .}
\]

Cardinality restrictions:

\[
\text{ex:Person} \text{ rdfs:subClassOf [ a owl:Restriction; owl:onProperty ex:hasParent; owl:maxCardinality 2 ] .}
\]
OWL features (contd.)

Equality / difference:

dbpedia:Tim_Berners-Lee owl:sameAs w3c:timbl .
w3c:timbl owl:differentFrom emse:az .

Property chains:

ex:uncle owl:propertyChainAxiom (ex:parent ex:brother) .

Functional / Inverse functional property:

ex:ssn rdf:type owl:InverseFunctionalProperty .

and more...
OWL Entailment

OWL defines a semantic extension that allows complex entailment to be computed.

Reasoning in OWL is very costly for big ontologies. Standard subsets of OWL are defined to guarantee better efficiency:

- OWL 2 EL for efficient classification
- OWL 2 QL for efficient queries over big datasets
- OWL 2 RL for efficient rule-based inference generation