Matteo Palmonari

matteo.palmonari@disco.unimib.it

Reasoning Agents:

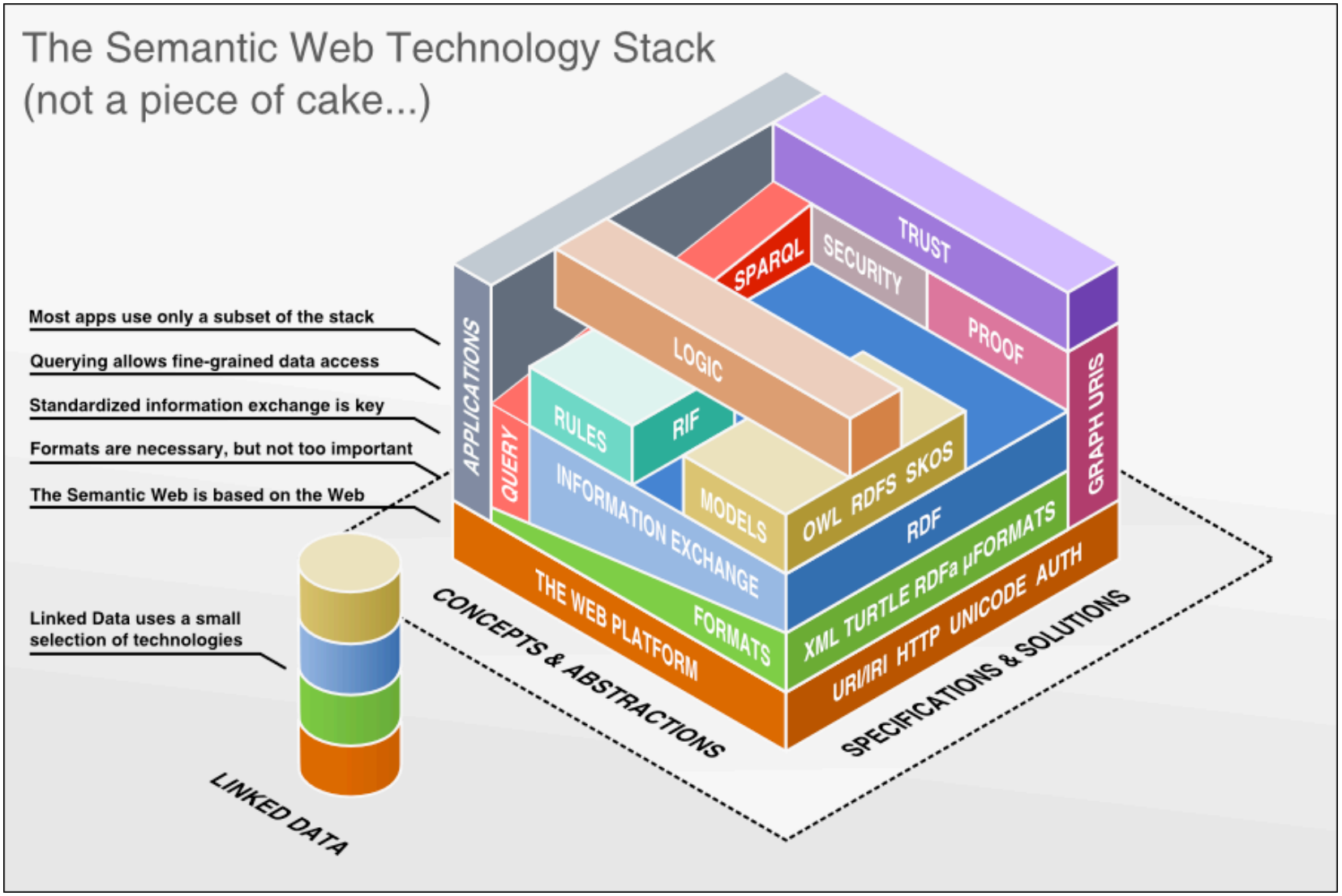
Ontologies and Reasoning for RDF Knowledge Graphs OWL and RULES

ITIS Lab – Innovative Technologies for Interaction and Services

Dipartimento di Informatica, Sistemistica e Comunicazione
Università degli Studi di Milano-Bicocca

OWL Highlights

Semantic Web Stack Revised

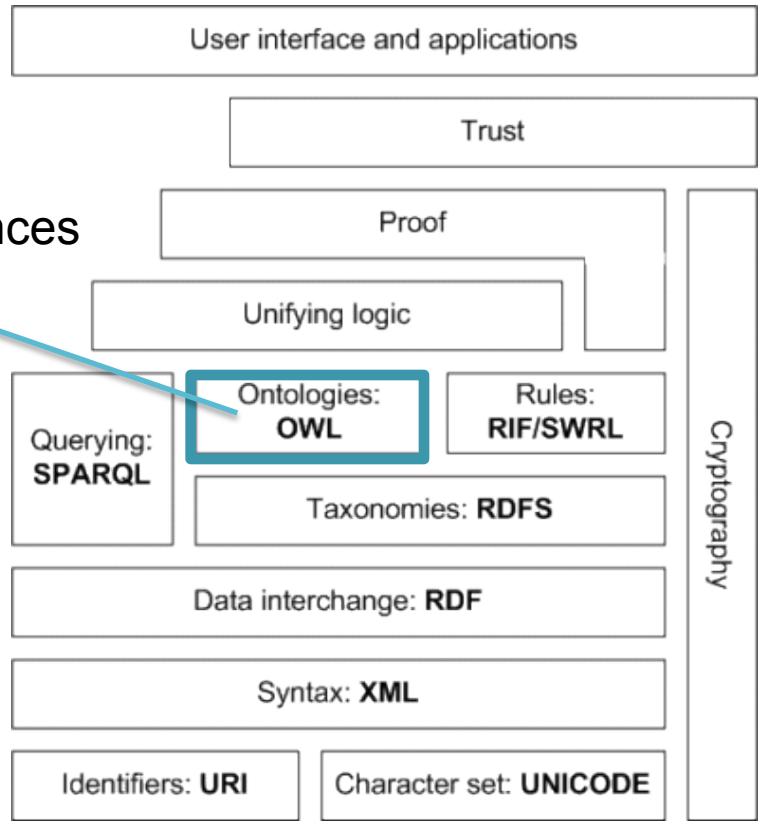


http://bnode.org/blog/2009/07/08/the-semantic-web-not-a-piece-of-cake

OWL

Web Ontology Language

Ontologies and inferences



Semantic Web Stack
Berners-Lee (2006)

Introduction to OWL

EUCLID
EdUcational Curriculum for the usage of LInked Data

- Provides more ontological constructs and avoids some of the potential confusion in RDFS
- OWL 2 is divided into sub-languages denominated *profiles*:
  - OWL 2 EL: Limited to basic classification, but with polynomial-time reasoning
  - OWL 2 QL: Designed to be translatable to relational database querying
  - OWL 2 RL: Designed to be efficiently implementable in rule-based systems

More restrictive than OWL DL

- Most triple stores concentrate on the use of RDFS with a subset of OWL features, called OWL-Horst or RDFS++

Class Axioms

EUCLID
EdUcational Curriculum for the usage of LInked Data

Axioms declare general statements about concepts which are used in logical inference (reasoning). Class axioms:

- **Sub-class** relationship (from RDF Schema)
- **Equivalent** relationship: classes have the same individuals

```
:Musician owl:equivalentClass :MusicArtist .
```

- **Disjointness**: classes have no shared individuals

```
:SoloMusicArtist owl:disjointWith :MusicGroup .
```

Axioms about Individuals

OWL Individuals represent instances of classes. They are related to their class by the **`rdf:type`** property

- We can state that two individuals are the same

```
<artist/ba550d0e-adac-4864-b88b-407cab5e76af#_> owl:sameAs
                                                dbpedia:PaulMcCartney .
```

- We can state that two individuals are different

```
:TheBeatles_band owl:differentFrom :TheBeatles_Tvseries .
```

OWL Properties

OWL distinguishes between two types of properties:

- **OWL ObjectProperties:** resources as values
- **OWL DatatypeProperties:** literals as values

```
:plays          rdf:type        owl:ObjectProperty;
                rdfs:domain     :Musician;
                rdfs:range      :Instrument .


:hasMembers     rdf:type          owl:DatatypeProperty;
                rdfs:domain       :MusicGroup
                rdfs:range        xsd:int .
```

Property Axioms

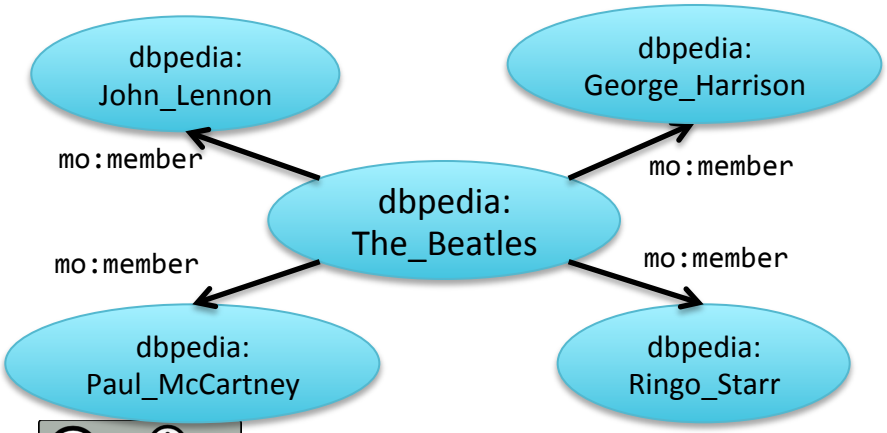
EUCLID
EdUcational Curriculum for the usage of LInked Data

- Property axioms include those from RDF Schema
- OWL allows for property **equivalence**. Example:

```
dbpedia-ont:bandMember owl:equivalentProperty mo:member.
```

Query:

```
SELECT ?x {dbpedia:The_Beatles
           dbpedia-ont:bandMember ?x.}
```



Result set:

?x

Result set with inference:

?x
dbpedia:Paul_McCartney
dbpedia:John_Lennon
dbpedia:Ringo_Starr
dbpedia:George_Harrison

Property Axioms

EUCLID
EdUcational Curriculum for the usage of LInked Data

- Property axioms include those from RDF Schema
- OWL allows for property **equivalence**. Example:

```
dbpedia-ont:bandMember owl:equivalentProperty mo:member.
```

- OWL allows for property **disjointness.** Example:

```
dbpedia-ont:length owl:propertyDisjointWith mo:duration.
```

- There is no standard for implementing inconsistency reports under SPARQL

Property Axioms (2)

EUCLID
EdUcational Curriculum for the usage of LInked Data

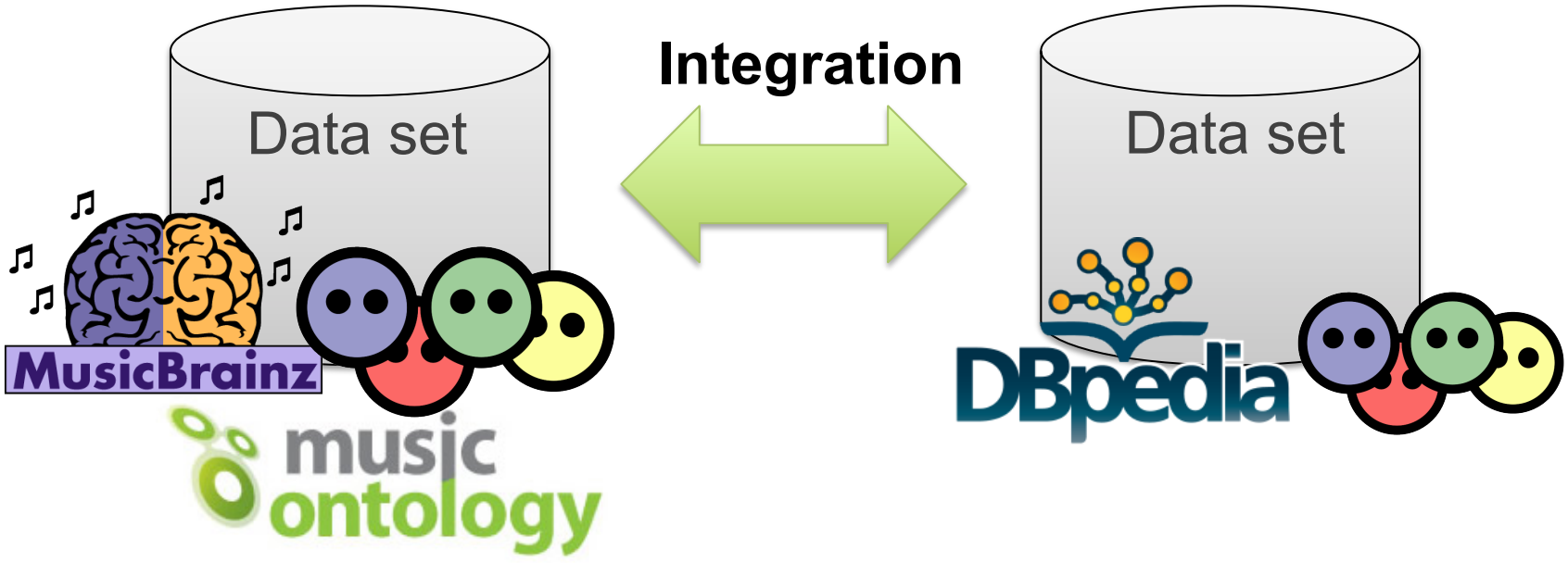
OWL allows the definition of property characteristics to infer new facts relating to instances and their properties

- Symmetry
- Transitivity
- Inverse
- Functional
- Inverse Functional

Reasoning for Linked Data Integration

EUCLID
EdUcational Curriculum for the usage of LInked Data

- **Example:** Integration of the **MusicBrainz** data set and the **DBpedia** data set

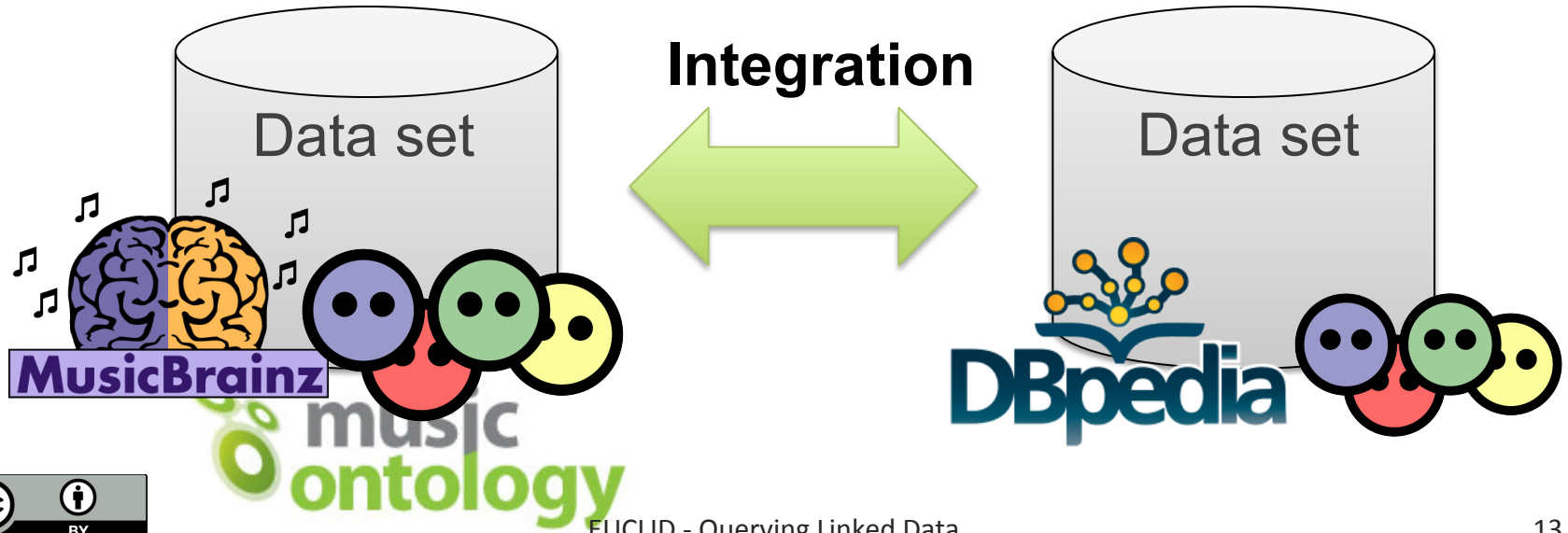


Reasoning for Linked Data Integration

EUCLID
EdUcational Curriculum for the usage of LInked Data

same

```
mo:b10bbbfc-cf9e-42e0-be17-e2c3e1d2600d ⟺ dbpedia:The_Beatles
    foaf:name The Beatles;                    dbpedia-ont:origin dbpedia:Liverpool;
    mo:member                                 dbpedia-ont:genre dbpedia:Rock_music;
     mo:ba550d0e-adac-4864-b88b-407cab5e76af; foaf:depiction <image> .
    mo:member
     mo:4d5447d7-c61c-4120-ba1b-d7f471d385b9;
    mo:member
     mo:42a8f507-8412-4611-854f-926571049fa0;
    mo:member
     mo:300c4c73-33ac-4255-9d57-4e32627f5e13.
```



Reasoning for Linked Data Integration

EUCLID
EdUcational Curriculum for the usage of LInked Data

same

```
mo:b10bbbfc-cf9e-42e0-be17-e2c3e1d2600d ⟺ dbpedia:The_Beatles
    foaf:name The Beatles;                    dbpedia-ont:origin dbpedia:Liverpool;
    mo:member                                 dbpedia-ont:genre dbpedia:Rock_music;
     mo:ba550d0e-adac-4864-b88b-407cab5e76af; foaf:depiction <image> .
    mo:member
     mo:4d5447d7-c61c-4120-ba1b-d7f471d385b9;
    mo:member
     mo:42a8f507-8412-4611-854f-926571049fa0;
    mo:member
     mo:300c4c73-33ac-4255-9d57-4e32627f5e13.
```

Query:

```
SELECT ?m ?g WHERE {
 dbpedia:The_Beatles
   dbpedia-ont:genre ?g;
   mo:member ?m.}
```

Result set:

?m	?g
mo:ba550d0e-adac-4864-b88b-407cab5e76af	dbpedia:Rock_music
mo:4d5447d7-c61c-4120-ba1b-d7f471d385b9	dbpedia:Rock_music
mo42a8f507-8412-4611-854f-926571049fa0;	dbpedia:Rock_music
mo300c4c73-33ac-4255-9d57-4e32627f5e13	dbpedia:Rock_music

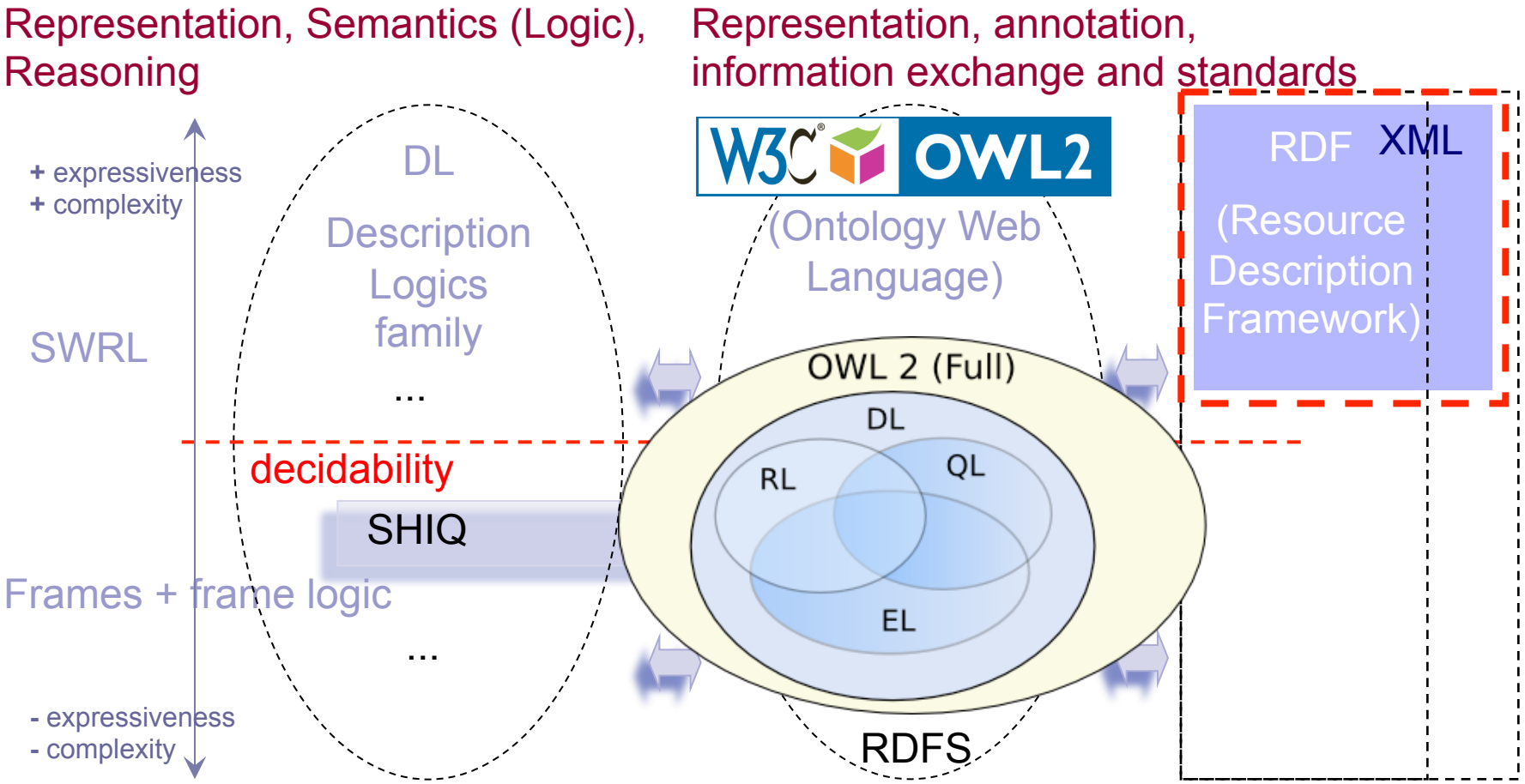
OWL & Description Logics

More limitations of RDFS

- By Fausto Giunchiglia*
  - No localised range and domain constraints
    - Cannot say that the range of teachBy is only professor when applied to professors and lecturer when applied to lecturers
  - No cardinality constraints
    - Cannot say that a course is taught by at least one professor, or persons have exactly 2 parents
  - No transitive, inverse or symmetrical properties
  - No Disjoint classes
    - Cannot say that Graduate and PhD. Students are two disjoint classes
  - No Boolean combinations of classes
    - Sometimes we may need to build new classes by combining other classes using union, intersection, and complement (e.g. person is the disjoint union of the classes male and female)

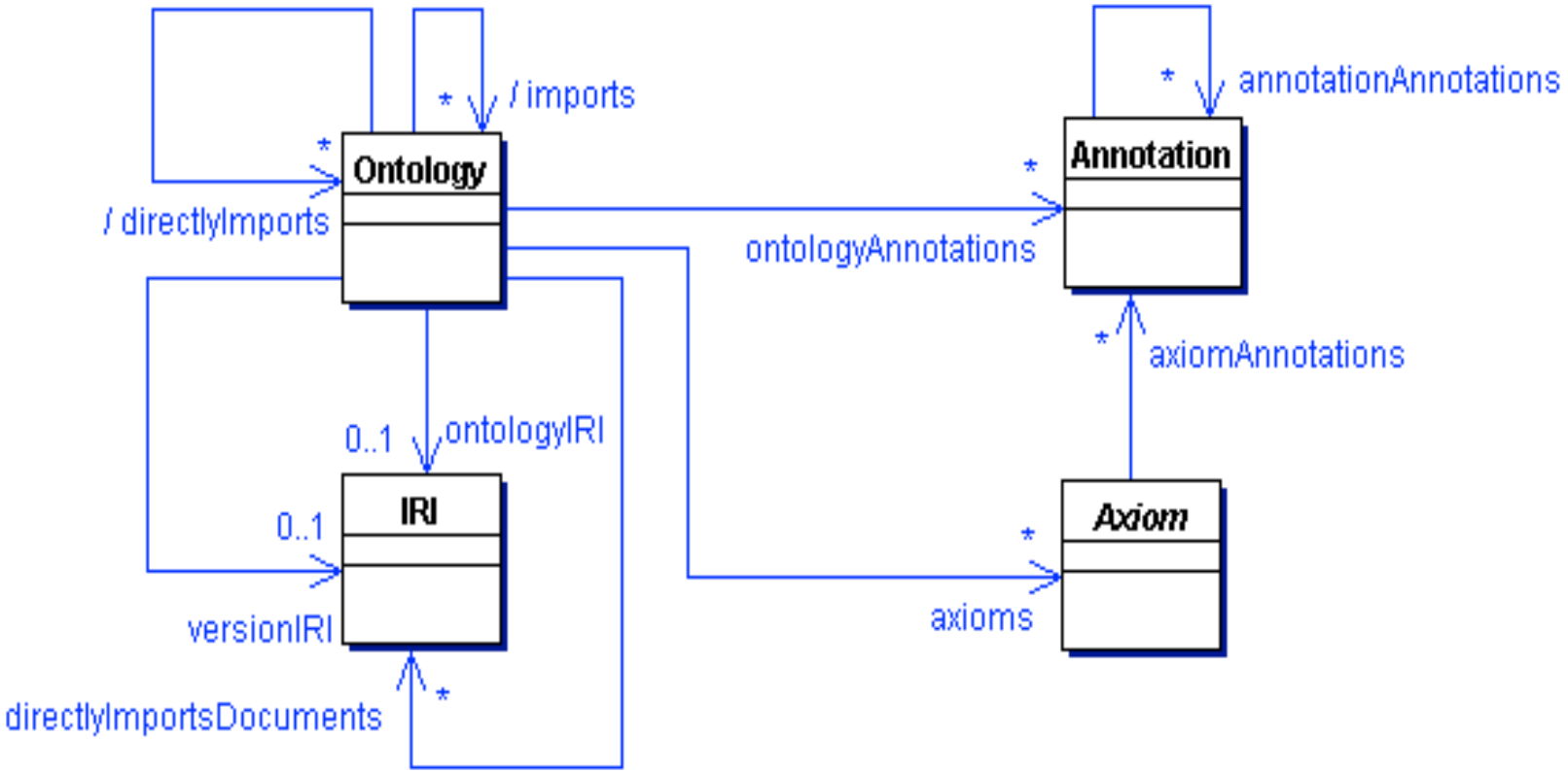
*Logics for Data and Knowledge Representation – UniTN

Ontologies as “semantic techs” (revised)



Ontologies: Editing, Reasoning, Navigation, Query

Ontologies in OWL 2*



OWL 2 Ontology Structure

*By Fausto Giunchiglia, taken from Logics for Data and Knowledge Representation – UniTN

OWL 2 vs. OWL 1*

- OWL 2 has a very similar overall structure to OWL 1
- Almost all the building blocks of OWL 2 were present in OWL 1, even thoguh possibly under different names
- The central role of RDF/XML, the role of other syntaxes, and the relationships between the Direct and RDF-Based semantics (i.e., the correspondence theorem) have not changed
- Backwards compatibility with OWL 1 is, to all intents and purposes, complete
  - Important: all OWL 1 Ontologies remain valid OWL 2 Ontologies

*By Fausto Giunchiglia, taken from Logics for Data and Knowledge Representation – UniTN

OWL 2 vs. OWL 1*

- OWL 2 adds some new functionality with respect to OWL 1. Some of the new features are:
  - Keys (identification of individuals of a given class by values of (a set of) key properties);
  - Inclusion of property chains (*allows a property to be defined as the composition of several properties, e.g., hasGrandParent , i.e., hasParent hasParent*)
  - Property characteristics (e.g., *reflexive, irreflexive, disjoint property*)
  - advanced use of cardinality restrictions (e.g., *inclusion, exclusion*);
  - complex classes (e.g., *UniTnStudent isEquvalentOf two atomic classes: Masters or PhD*);
  - Addition of new OWL constructors (object property, data property)
  - advanced use of datatypes, (*e.g., data ranges, type extension*);
  - enumeration of individuals (named individuals and anonymous individuals);
  - some of the restrictions applicable to OWL DL have been relaxed; as a result, the set of RDF Graphs that can be handled by Description Logics reasoners is slightly larger in OWL 2
  - OWL 2 defines 3 new profiles (sub-languages): OWL 2 EL, OWL 2 QL and OWL 2 RL

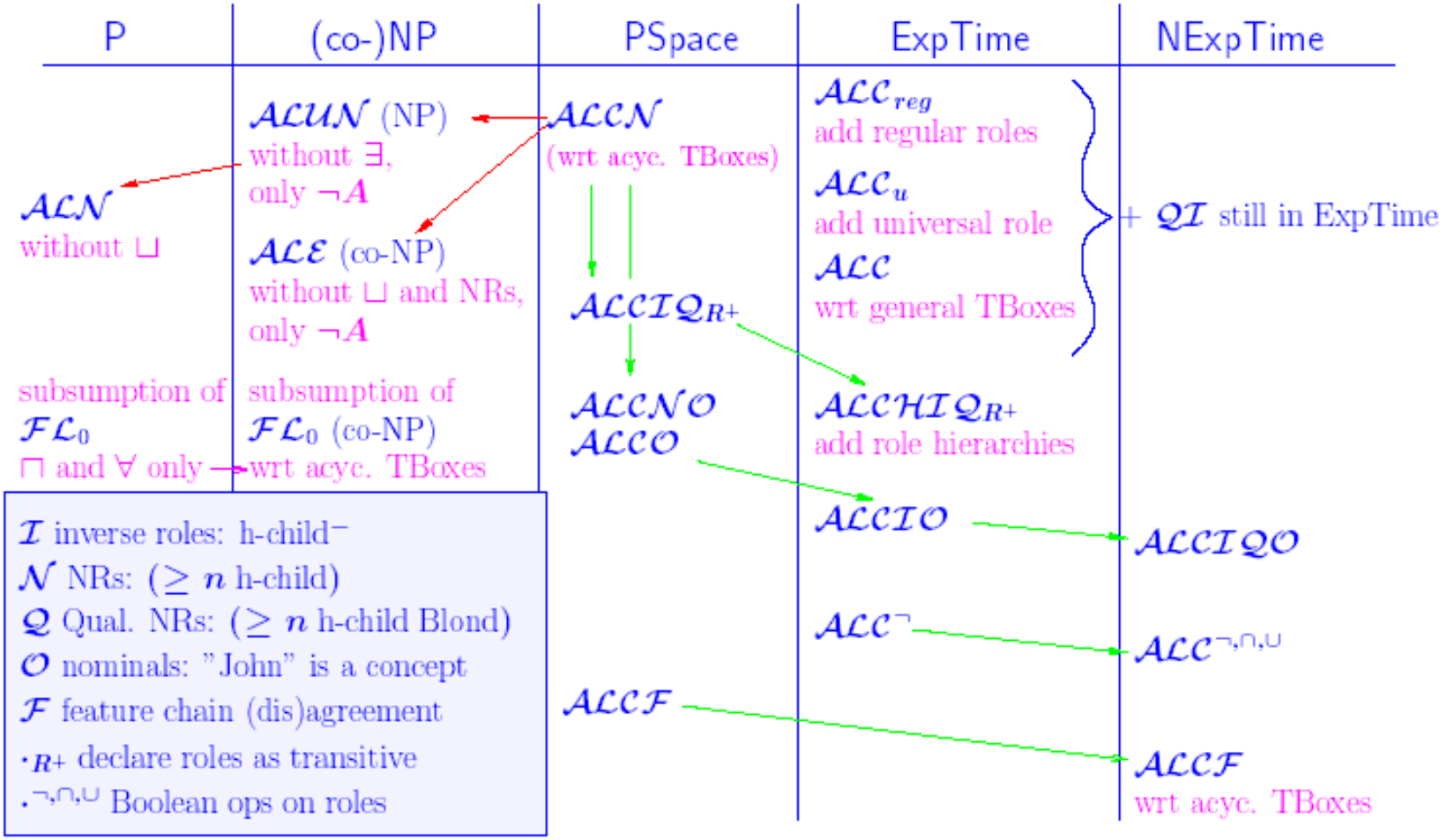
*By Fausto Giunchiglia, taken from Logics for Data and Knowledge Representation – UniTN

OWL 2 DL vs. OWL 2 Full*

- There are two alternative ways of assigning meaning to ontologies in OWL 2:
  - the direct semantics (*provides a meaning for OWL 2 in a **Description Logics** style*); and
  - the RDF-Based Semantics (*an extension of the semantics for RDFS and is based on viewing OWL 2 ontologies as RDF graphs*)
- The notion OWL 2 DL is used to refer to OWL 2 ontologies interpreted using the Direct Semantics
- The notion OWL 2 Full is used when considering the RDF-Based Semantics

*By Fausto Giunchiglia, taken from Logics for Data and Knowledge Representation – UniTN

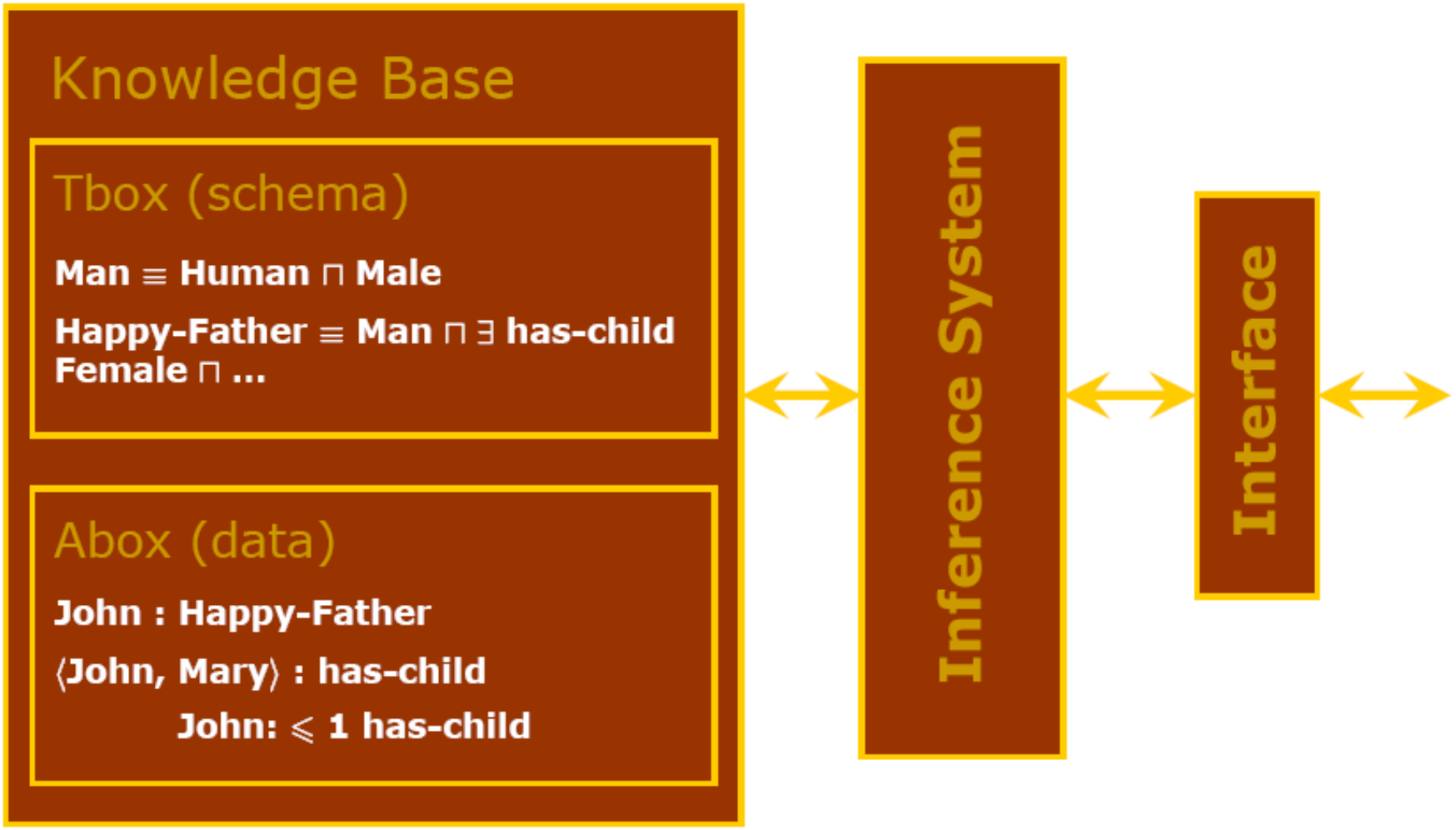
DL family's complexity MAP



Use a (Description) Logic

- OWL DL based on $\mathcal{SHIQ}$ Description Logic
  - In fact it is equivalent to $\mathcal{SHOIN}(\mathbb{D}_n)$ DL
- OWL DL Benefits from many years of DL research
  - Well defined semantics
  - Formal properties well understood (complexity, decidability)
  - Known reasoning algorithms
  - Implemented systems (highly optimised)
- In fact there are three "species" of OWL 1 (!)
  - OWL FULL is union of OWL syntax and RDF
  - OWL DL restricted OWL full to First Order fragment (≈ DAML+OIL)
  - OWL Lite is "simpler" subset of OWL DL (equiv to $\mathcal{SHIF}(\mathbb{D}_n)$)

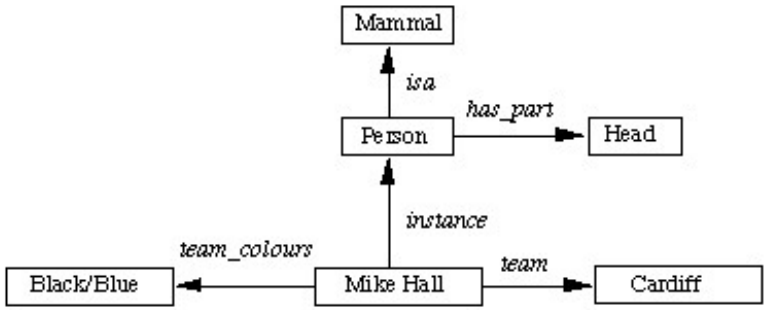
Description Logic-based System Architecture



Representation, Semantics, Reasoning: Description Logics

Representation in a Semantic Net

The physical attributes of a person can be represented as in Fig. 9.



$$
\begin{aligned}
\mathsf{Woman} &\equiv \mathsf{Person} \sqcap \mathsf{Female} \\
\mathsf{Man} &\equiv \mathsf{Person} \sqcap \neg\mathsf{Woman} \\
\mathsf{Mother} &\equiv \mathsf{Woman} \sqcap \exists \mathsf{hasChild}.\mathsf{Person} \\
\mathsf{Father} &\equiv \mathsf{Man} \sqcap \exists \mathsf{hasChild}.\mathsf{Person} \\
\mathsf{Parent} &\equiv \mathsf{Father} \sqcup \mathsf{Mother} \\
\mathsf{Grandmother} &\equiv \mathsf{Mother} \sqcap \exists \mathsf{hasChild}.\mathsf{Parent} \\
\mathsf{MotherWithManyChildren} &\equiv \mathsf{Mother} \sqcap \geqslant 3\,\mathsf{hasChild} \\
\mathsf{MotherWithoutDaughter} &\equiv \mathsf{Mother} \sqcap \forall \mathsf{hasChild}.\neg\mathsf{Woman} \\
\mathsf{Wife} &\equiv \mathsf{Woman} \sqcap \exists \mathsf{hasHusband}.\mathsf{Man}
\end{aligned}
$$

Fig. 2.2. A terminology (TBox) with concepts about family relationships.

$$
\begin{aligned}
&\mathsf{MotherWithoutDaughter}(\mathsf{MARY}) \qquad \mathsf{Father}(\mathsf{PETER}) \\
&\mathsf{hasChild}(\mathsf{MARY}, \mathsf{PETER}) \qquad \mathsf{hasChild}(\mathsf{PETER}, \mathsf{HARRY}) \\
&\mathsf{hasChild}(\mathsf{MARY}, \mathsf{PAUL})
\end{aligned}
$$

Fig. 2.4. A world description (ABox).

Class/Concept Constructors

Constructor	DL Syntax	Example	FOL Syntax
intersectionOf	$C_1 \sqcap \ldots \sqcap C_n$	Human $\sqcap$ Male	$C_1(x) \wedge \ldots \wedge C_n(x)$
unionOf	$C_1 \sqcup \ldots \sqcup C_n$	Doctor $\sqcup$ Lawyer	$C_1(x) \vee \ldots \vee C_n(x)$
complementOf	$\neg C$	$\neg$Male	$\neg C(x)$
oneOf	$\{x_1\} \sqcup \ldots \sqcup \{x_n\}$	{john} $\sqcup$ {mary}	$x = x_1 \vee \ldots \vee x = x_n$
allValuesFrom	$\forall P.C$	$\forall$hasChild.Doctor	$\forall y.P(x,y) \rightarrow C(y)$
someValuesFrom	$\exists P.C$	$\exists$hasChild.Lawyer	$\exists y.P(x,y) \wedge C(y)$
maxCardinality	$\leqslant nP$	$\leqslant$1hasChild	$\exists^{\leqslant n} y.P(x,y)$
minCardinality	$\geqslant nP$	$\geqslant$2hasChild	$\exists^{\geqslant n} y.P(x,y)$

- C is a concept (class); P is a role (property); x is an individual name
- XMLS datatypes as well as classes in ∀P.C and ∃P.C
  - Restricted form of DL concrete domains

Class/Concept Constructors

Constructor	DL Syntax	Example	Modal Syntax
intersectionOf	$C_1 \sqcap \ldots \sqcap C_n$	Human $\sqcap$ Male	$C_1 \wedge \ldots \wedge C_n$
unionOf	$C_1 \sqcup \ldots \sqcup C_n$	Doctor $\sqcup$ Lawyer	$C_1 \vee \ldots \vee C_n$
complementOf	$\neg C$	$\neg$Male	$\neg C$
oneOf	$\{x_1\} \sqcup \ldots \sqcup \{x_n\}$	{john} $\sqcup$ {mary}	$x_1 \vee \ldots \vee x_n$
allValuesFrom	$\forall P.C$	$\forall$hasChild.Doctor	$[P]C$
someValuesFrom	$\exists P.C$	$\exists$hasChild.Lawyer	$\langle P \rangle C$
maxCardinality	$\leqslant nP$	$\leqslant$1hasChild	$[P]_{n+1}$
minCardinality	$\geqslant nP$	$\geqslant$2hasChild	$\langle P \rangle_n$

- C is a concept (class); P is a role (property); x is an individual name
- XMLS datatypes as well as classes in ∀P.C and ∃P.C
  - Restricted form of DL concrete domains

Semantics (cont.)

- Interpretation function $\cdot^{\mathcal{I}}$ extends to concept (and role) expressions in the obvious way, i.e.:

$$
\begin{array}{lcl}
(C \sqcap D)^{\mathcal{I}} & = & C^{\mathcal{I}} \cap D^{\mathcal{I}} \\
(C \sqcup D)^{\mathcal{I}} & = & C^{\mathcal{I}} \cup D^{\mathcal{I}} \\
(\neg C)^{\mathcal{I}} & = & \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}} \\
\{x\}^{\mathcal{I}} & = & \{x^{\mathcal{I}}\} \\
(\exists R.C)^{\mathcal{I}} & = & \{x \mid \exists y.\langle x, y\rangle \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\} \\
(\forall R.C)^{\mathcal{I}} & = & \{x \mid \forall y.(x, y) \in R^{\mathcal{I}} \Rightarrow y \in C^{\mathcal{I}}\} \\
(\leqslant nR)^{\mathcal{I}} & = & \{x \mid \#\{y \mid \langle x, y\rangle \in R^{\mathcal{I}}\} \leqslant n\} \\
(\geqslant nR)^{\mathcal{I}} & = & \{x \mid \#\{y \mid \langle x, y\rangle \in R^{\mathcal{I}}\} \geqslant n\} \\
(R^{-})^{\mathcal{I}} & = & \{(x, y) \mid (y, x) \in R^{\mathcal{I}}\}
\end{array}
$$

From “A Description Logic Primer”*

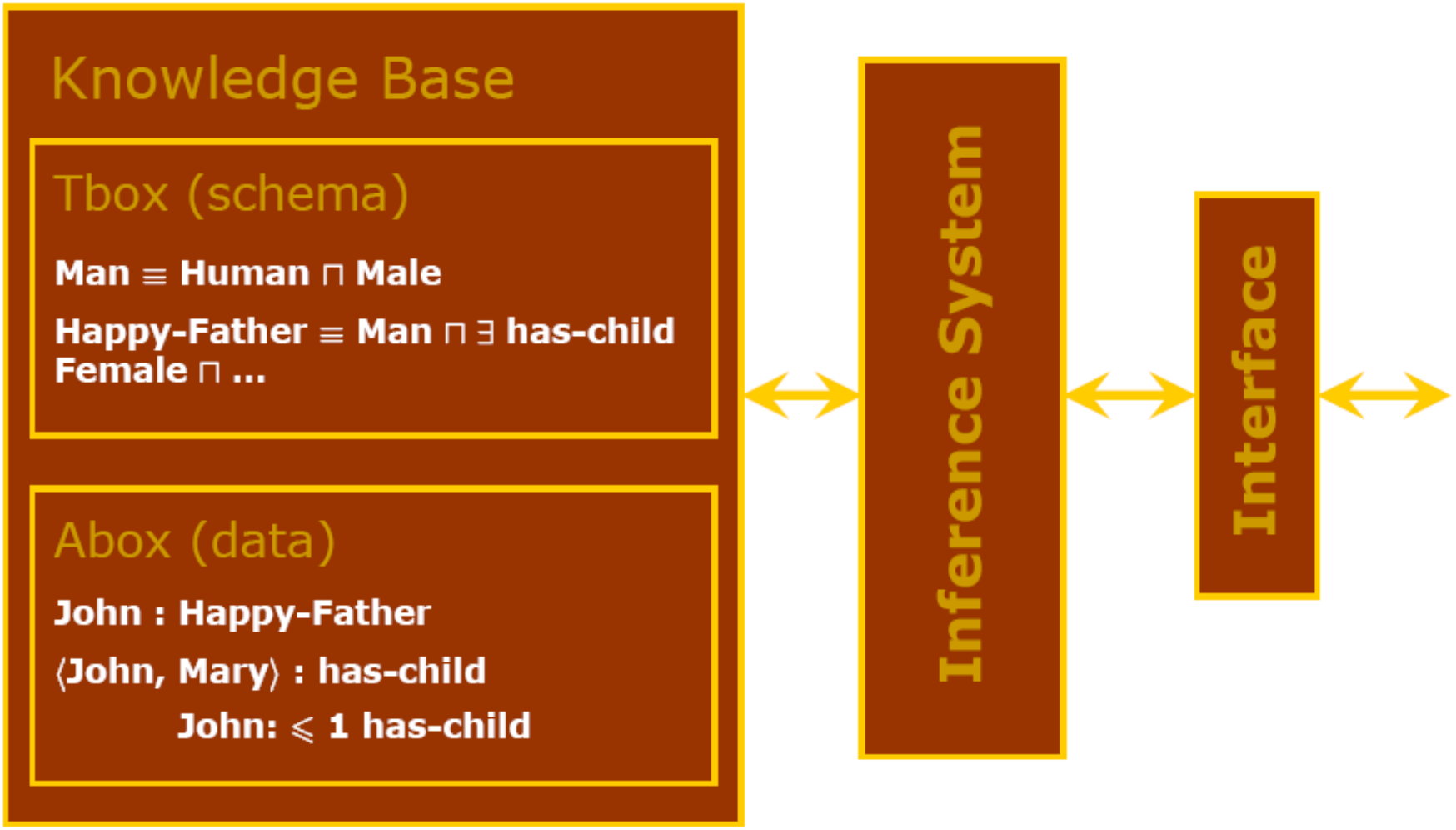
Table 1. Syntax and semantics of $\mathcal{SROIQ}$ constructors

	Syntax	Semantics
Individuals:		
individual name	a	$a^{\mathcal{I}}$
Roles:		
atomic role	R	$R^{\mathcal{I}}$
inverse role	R^{-}	$\{\langle x, y\rangle \mid \langle y, x\rangle \in R^{\mathcal{I}}\}$
universal role	U	$\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
Concepts:		
atomic concept	A	$A^{\mathcal{I}}$
intersection	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
union	$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
complement	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
top concept	$\top$	$\Delta^{\mathcal{I}}$
bottom concept	$\bot$	$\emptyset$
existential restriction	$\exists R.C$	$\{x \mid \text{some } R^{\mathcal{I}}\text{-successor of } x \text{ is in } C^{\mathcal{I}}\}$
universal restriction	$\forall R.C$	$\{x \mid \text{all } R^{\mathcal{I}}\text{-successors of } x \text{ are in } C^{\mathcal{I}}\}$
at-least restriction	$\geqslant n\, R.C$	$\{x \mid \text{at least } n\ R^{\mathcal{I}}\text{-successors of } x \text{ are in } C^{\mathcal{I}}\}$
at-most restriction	$\leqslant n\, R.C$	$\{x \mid \text{at most } n\ R^{\mathcal{I}}\text{-successors of } x \text{ are in } C^{\mathcal{I}}\}$
local reflexivity	$\exists R.Self$	$\{x \mid \langle x, x\rangle \in R^{\mathcal{I}}\}$
nominal	$\{a\}$	$\{a^{\mathcal{I}}\}$

where $a, b \in \mathsf{N}_I$ are individual names, $A \in \mathsf{N}_C$ is a concept name, $C, D \in \mathbf{C}$ are concepts, $R \in \mathbf{R}$ is a role

*by Krötzsch, Simancík, Horrocks → https://arxiv.org/pdf/1201.4089.pdf

DL System Architecture



Ontology/Tbox Axioms

OWL Syntax	DL Syntax	Example
subClassOf	$C_1 \sqsubseteq C_2$	Human $\sqsubseteq$ Animal $\sqcap$ Biped
equivalentClass	$C_1 \equiv C_2$	Man $\equiv$ Human $\sqcap$ Male
subPropertyOf	$P_1 \sqsubseteq P_2$	hasDaughter $\sqsubseteq$ hasChild
equivalentProperty	$P_1 \equiv P_2$	cost $\equiv$ price
transitiveProperty	$P^+ \sqsubseteq P$	ancestor$^+$ $\sqsubseteq$ ancestor

- Obvious FO/Modal Logic equivalences
  - E.g., DL: $C \sqsubseteq D$ FOL: $\forall x C(x) \rightarrow D(x)$ ML: $C \rightarrow D$
- Often distinguish two kinds of Tbox axioms
  - "Definitions" $C \sqsubseteq D$ or $C \equiv D$ where C is a concept name
  - General Concept Inclusion axioms (GCIs) where C may be complex

From “A Description Logic Primer”*

Table 2. Syntax and semantics of $\mathcal{SROIQ}$ axioms

	Syntax	Semantics
ABox:		
concept assertion	$C(a)$	$a^{\mathcal{I}} \in C^{\mathcal{I}}$
role assertion	$R(a, b)$	$\langle a^{\mathcal{I}}, b^{\mathcal{I}} \rangle \in R^{\mathcal{I}}$
individual equality	$a \approx b$	$a^{\mathcal{I}} = b^{\mathcal{I}}$
individual inequality	$a \not\approx b$	$a^{\mathcal{I}} \neq b^{\mathcal{I}}$
TBox:		
concept inclusion	$C \sqsubseteq D$	$C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$
concept equivalence	$C \equiv D$	$C^{\mathcal{I}} = D^{\mathcal{I}}$
RBox:		
role inclusion	$R \sqsubseteq S$	$R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$
role equivalence	$R \equiv S$	$R^{\mathcal{I}} = S^{\mathcal{I}}$
complex role inclusion	$R_1 \circ R_2 \sqsubseteq S$	$R_1^{\mathcal{I}} \circ R_2^{\mathcal{I}} \subseteq S^{\mathcal{I}}$
role disjointness	$Disjoint(R, S)$	$R^{\mathcal{I}} \cap S^{\mathcal{I}} = \emptyset$

*by Krötzsch, Simancík, Horrocks → https://arxiv.org/pdf/1201.4089.pdf

Ontology Facts / Abox Axioms

OWL Syntax	DL Syntax	Example
type	$a : C$	John : Happy-Father
property	$\langle a, b \rangle : R$	$\langle$John, Mary$\rangle$: has-child

- *Note: using nominals (e.g., in SHOIN), can reduce Abox axioms to concept inclusion axioms*

$$a : C \text{ equivalent to } \{a\} \sqsubseteq C$$

$$\langle a, b \rangle : R \text{ equivalent to } \{a\} \sqsubseteq \exists R.\{b\}$$

Representation, Semantics, Reasoning: Description Logics

$$
\begin{aligned}
\text{Woman} &\equiv \text{Person} \sqcap \text{Female} \\
\text{Man} &\equiv \text{Person} \sqcap \neg\text{Woman} \\
\text{Mother} &\equiv \text{Woman} \sqcap \exists\text{hasChild.Person} \\
\text{Father} &\equiv \text{Man} \sqcap \exists\text{hasChild.Person} \\
\text{Parent} &\equiv \text{Father} \sqcup \text{Mother} \\
\text{Grandmother} &\equiv \text{Mother} \sqcap \exists\text{hasChild.Parent} \\
\text{MotherWithManyChildren} &\equiv \text{Mother} \sqcap \geqslant 3\,\text{hasChild} \\
\text{MotherWithoutDaughter} &\equiv \text{Mother} \sqcap \forall\text{hasChild.}\neg\text{Woman} \\
\text{Wife} &\equiv \text{Woman} \sqcap \exists\text{hasHusband.Man}
\end{aligned}
$$

Fig. 2.2. A terminology (TBox) with concepts about family relationships.

$$
\begin{aligned}
&\text{MotherWithoutDaughter}(\text{MARY}) \qquad \text{Father}(\text{PETER}) \\
&\text{hasChild}(\text{MARY}, \text{PETER}) \qquad \text{hasChild}(\text{PETER}, \text{HARRY}) \\
&\text{hasChild}(\text{MARY}, \text{PAUL})
\end{aligned}
$$

Fig. 2.4. A world description (ABox).

Knowledge Base Semantics

- An interpretation $\mathcal{I}$ satisfies (models) an axiom A ($\mathcal{I} \models \mathrm{A}$):
  - $\mathcal{I} \models \mathrm{C} \sqsubseteq \mathrm{D}$ iff $\mathrm{C}^{\mathcal{I}} \subseteq \mathrm{D}^{\mathcal{I}}$ $\quad$ $\mathcal{I} \models \mathrm{C} \equiv \mathrm{D}$ iff $\mathrm{C}^{\mathcal{I}} = \mathrm{D}^{\mathcal{I}}$
  - $\mathcal{I} \models \mathrm{R} \sqsubseteq \mathrm{S}$ iff $\mathrm{R}^{\mathcal{I}} \subseteq \mathrm{S}^{\mathcal{I}}$ $\quad$ $\mathcal{I} \models \mathrm{R} \equiv \mathrm{S}$ iff $\mathrm{R}^{\mathcal{I}} = \mathrm{S}^{\mathcal{I}}$
  - $\mathcal{I} \models \mathrm{x} \in \mathrm{D}$ iff $\mathrm{x}^{\mathcal{I}} \in \mathrm{D}^{\mathcal{I}}$
  - $\mathcal{I} \models \langle \mathrm{x},\mathrm{y} \rangle \in \mathrm{R}$ iff $(\mathrm{x}^{\mathcal{I}},\mathrm{y}^{\mathcal{I}}) \in \mathrm{R}^{\mathcal{I}}$
  - $\mathcal{I} \models \mathrm{R}^{+} \sqsubseteq \mathrm{R}$ iff $(\mathrm{R}^{\mathcal{I}})^{+} \subseteq \mathrm{R}^{\mathcal{I}}$
- $\mathcal{I}$ satisfies a Tbox $\mathcal{T}$ ($\mathcal{I} \models \mathcal{T}$) iff $\mathcal{I}$ satisfies every axiom A in $\mathcal{T}$
- $\mathcal{I}$ satisfies an Abox $\mathcal{A}$ ($\mathcal{I} \models \mathcal{A}$) iff $\mathcal{I}$ satisfies every axiom A in $\mathcal{A}$
- $\mathcal{I}$ satisfies an KB $\mathcal{K}$ ($\mathcal{I} \models \mathcal{K}$) iff $\mathcal{I}$ satisfies both $\mathcal{T}$ and $\mathcal{A}$

RDFS Syntax

$$\text{Person} \sqcap \forall \text{hasChild}.(\text{Doctor} \sqcup \exists \text{hasChild}.\text{Doctor})$$

```
<owl:Class>
  <owl:intersectionOf rdf:parseType=" collection">
    <owl:Class rdf:about="#Person"/>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasChild"/>
      <owl:toClass>
        <owl:unionOf rdf:parseType=" collection">
          <owl:Class rdf:about="#Doctor"/>
          <owl:Restriction>
            <owl:onProperty rdf:resource="#hasChild"/>
            <owl:hasClass rdf:resource="#Doctor"/>
          </owl:Restriction>
        </owl:unionOf>
      </owl:toClass>
    </owl:Restriction>
  </owl:intersectionOf>
</owl:Class>
```

Basic Inference Tasks

Satisfiability A concept C is *satisfiable* with respect to $\mathcal{T}$ if there exists a model $\mathcal{I}$ of $\mathcal{T}$ such that $C^{\mathcal{I}}$ is nonempty. In this case we say also that $\mathcal{I}$ is a *model* of C.

Subsumption A concept C is *subsumed* by a concept D with respect to $\mathcal{T}$ if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for every model $\mathcal{I}$ of $\mathcal{T}$. In this case we write $C \sqsubseteq_{\mathcal{T}} D$ or $\mathcal{T} \models C \sqsubseteq D$.

Equivalence Two concepts C and D are *equivalent* with respect to $\mathcal{T}$ if $C^{\mathcal{I}} = D^{\mathcal{I}}$ for every model $\mathcal{I}$ of $\mathcal{T}$. In this case we write $C \equiv_{\mathcal{T}} D$ or $\mathcal{T} \models C \equiv D$.

Disjointness Two concepts C and D are *disjoint* with respect to $\mathcal{T}$ if $C^{\mathcal{I}} \cap D^{\mathcal{I}} = \emptyset$ for every model $\mathcal{I}$ of $\mathcal{T}$.

Basic Inference Tasks II

- Knowledge is correct (captures intuitions)
  - Does C subsume D w.r.t. ontology $\mathcal{O}$? ($\mathrm{C}^{\mathcal{I}} \subseteq \mathrm{D}^{\mathcal{I}}$ in *every* model $\mathcal{I}$ of $\mathcal{O}$)
- Knowledge is minimally redundant (no unintended synonyms)
  - Is C equivallent to D w.r.t. $\mathcal{O}$? ($\mathrm{C}^{\mathcal{I}} = \mathrm{D}^{\mathcal{I}}$ in *every* model $\mathcal{I}$ of $\mathcal{O}$)
- Knowledge is meaningful (classes can have instances)
  - Is C is satisfiable w.r.t. $\mathcal{O}$? ($\mathrm{C}^{\mathcal{I}} \neq \emptyset$ in *some* model $\mathcal{I}$ of $\mathcal{O}$)

- Querying knowledge
  - Is x an instance of C w.r.t. $\mathcal{O}$? ($\mathrm{x}^{\mathcal{I}} \in \mathrm{C}^{\mathcal{I}}$ in *every* model $\mathcal{I}$ of $\mathcal{O}$)
  - Is $\langle \mathrm{x},\mathrm{y} \rangle$ an instance of R w.r.t. $\mathcal{O}$? ($(\mathrm{x}^{\mathcal{I}},\mathrm{y}^{\mathcal{I}}) \in \mathrm{R}^{\mathcal{I}}$ in *every* model $\mathcal{I}$ of $\mathcal{O}$)

- Above problems can be solved using highly optimised DL reasoners

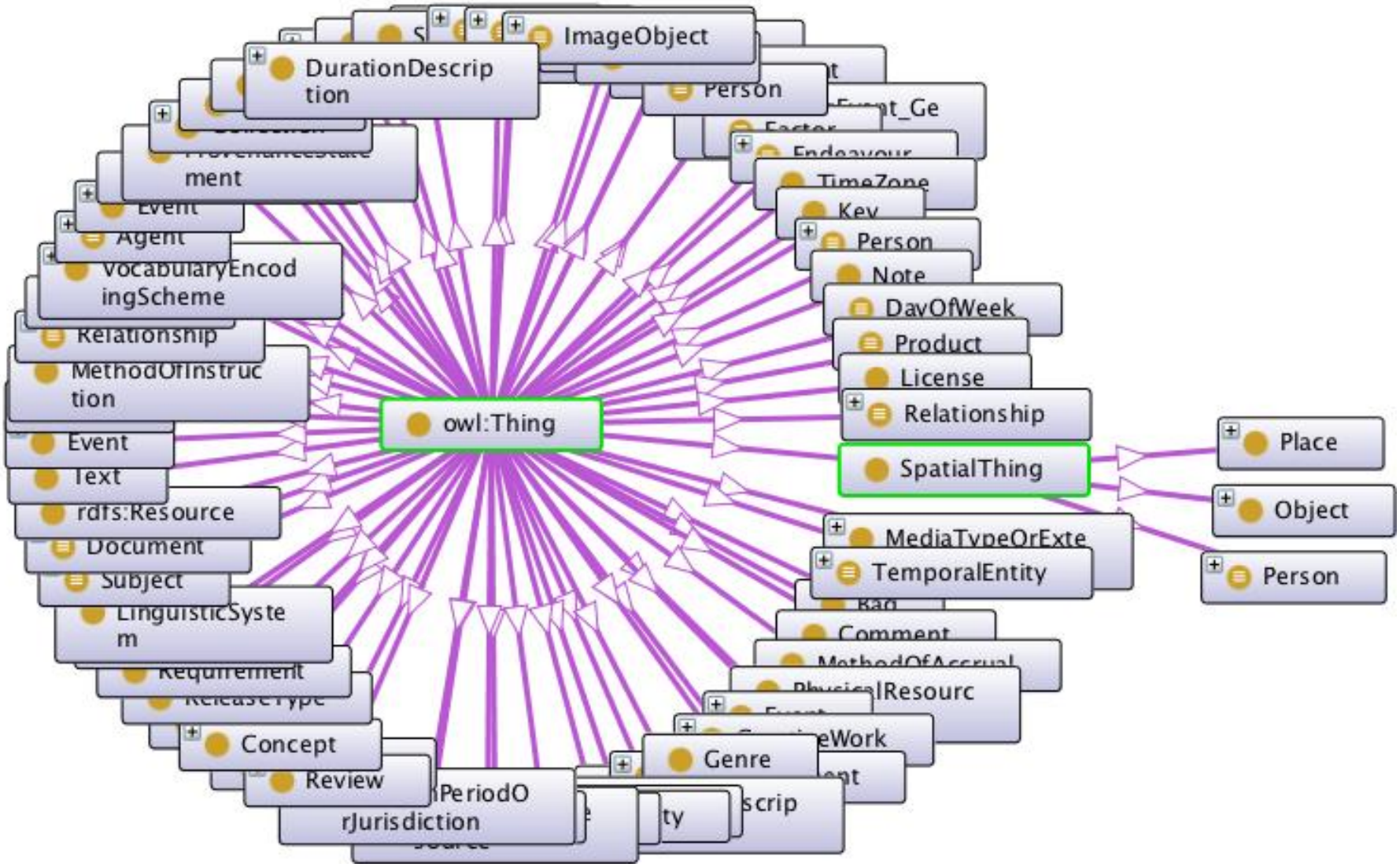
References

- Description Logic Basics
  - F. Baader, W.Nutts. **Basic Description Logics**. In F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. The Description Logic Handbook: Theory, Implementation and Applications. Cambridge University Press, 2003
- DL for reasoning about UML and ER
  - D. Calvanese, M. Lenzerini, and D. Nardi. **Description logics for conceptual data modeling**. In J. Chomicki and G. Saake, editors, *Logics for Databases and Information Systems*, pages 229-264. Kluwer Academic Publisher, 1998.
- Web:
  - Description Logics Web site: http://dl.kr.org/
  - OWL: http://www.w3.org/TR/owl-guide/

OWL-DL // DL Contructs Explained

Simple Named Classes

RDFS ✓



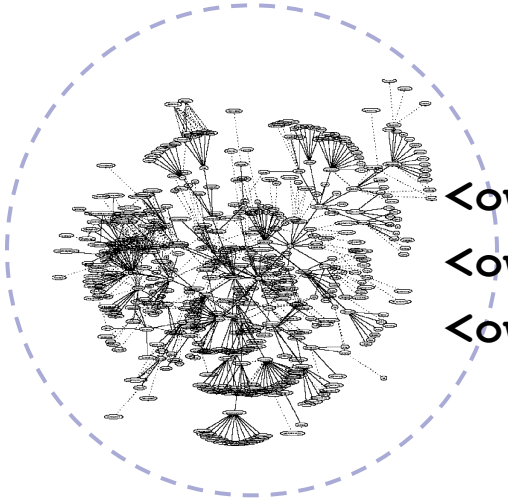
Simple Named Classes

RDFS ✓

The most basic concepts in a domain should correspond to ***classes that are the roots of various taxonomic trees***.
Every individual in the OWL world is a member of the class **`owl:Thing`**. Thus each user-defined class is implicitly a subclass of **`owl:Thing`**. Domain specific root classes are defined by simply declaring a named class. OWL also defines the empty class, **`owl:Nothing`**.

`owl:Thing`

Medium



Name Specification

```
<owl:Class rdf:ID="MusicalItem"/>
<owl:Class rdf:ID="Medium"/>
<owl:Class rdf:ID="CD"/>
```

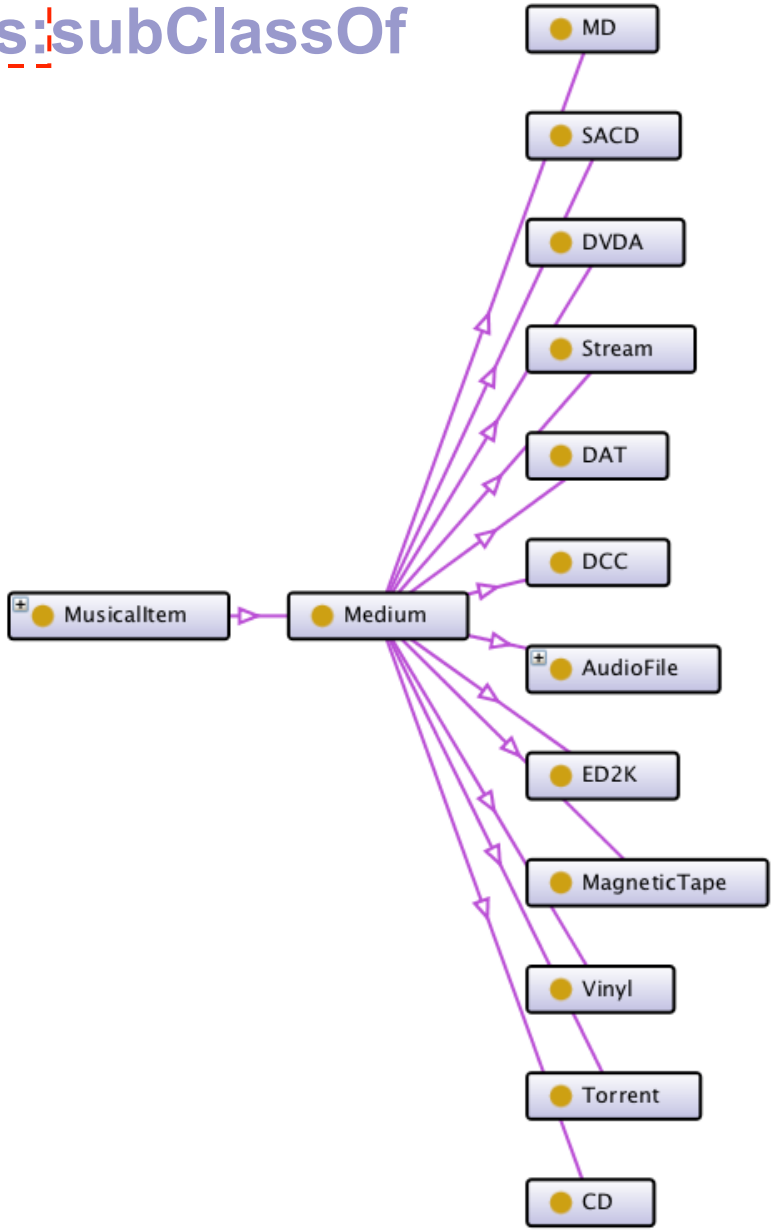
`owl:Nothing`

And while the classes exist, they may have **no members**.
For all we know at the moment, these classes might as well have been called `Thing1`, `Thing2`, and `Thing3`

Taxonomic Constructor for Classes

RDFS ✓

rdfs:subClassOf



Medium ⊑ MusicalItem

Taxonomic Constructor for Classes

rdfs:subClassOf

RDFS ✓

The fundamental taxonomic constructor for classes is `rdfs:subClassOf`.

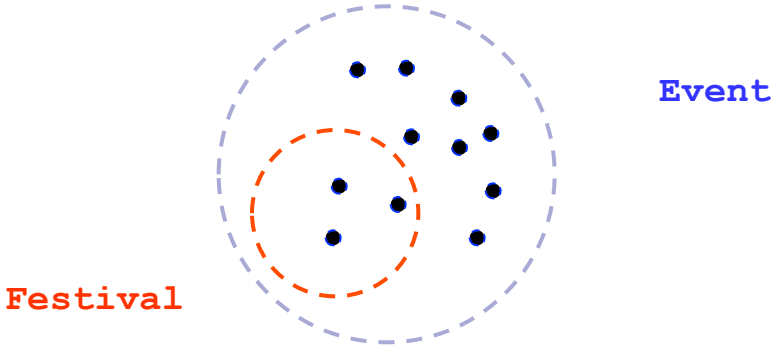
It relates a more specific class to a more general class. If X is a subclass of Y, then every instance of X is also an instance of Y. The `rdfs:subClassOf` relation is ***transitive*** (*if X is a subclass of Y and Y a subclass of Z then X is a subclass of Z*

Event ⊑ Thing

Festival ⊑ Event

```
<owl:Class rdf:ID="Festival">
<rdfs:subClassOf
 rdf:resource="Event"/>
...
</owl:Class>
```

We define IncomeService as a set of services which is subcset of the AbstractService set.



A class definition has two parts: a ***name introduction*** or reference and ***a list of restrictions***. Each of the immediate contained expressions in the class definition further restricts the instances of the defined class. Instances of the class belong to the ***intersection*** of the restrictions.

Reasoning about Individuals

Instance-of Relationship

RDFS ✓

A class is simply a name and collection of properties that describe a set of individuals. Individuals are the members of those sets. Classes should correspond to naturally occurring ***sets of things*** in a domain of discourse, and individuals should correspond to actual entities that can be grouped into these classes. An individual is minimally introduced by declaring it ***to be a member of a class***.

```
<Festival rdf:ID="Glastonbury"/>
```

```
<owl:Class rdf:ID="Event">
...
</owl:Class>
```

Glastonbury: Festival

```
<owl:Class rdf:ID="Festival">
        <rdfs:subClassOf rdf:resource="Event" />
</owl:Class>

<Event rdf:ID="Glastonbury" />
```

Levels of representation: In certain contexts something that is obviously a class can itself be considered an instance of something else. For example, in the serviceontology we have the notion of a Actor, which is intended to denote the set of all *Actor varietals.* CabernetSauvingonActor is an example instance of this class, as it denotes the actual Actor varietal called Cabernet Sauvignon. However, INPS could itself be considered a class, the set of all actual Cabernet Sauvignon Actors.

Subclass vs. instance: It is very easy to confuse the *instance-of* relationship with the *subclass* relationship. For example, it may seem arbitrary to choose to make INPS an individual that is an instance of Actor, as opposed to a subclass of Actor. This is not an arbitrary decision. The Actor class denotes the set of all *Actor varietals*, and therefore any subclass of Actor should denote a subset of these varietals. Thus, INPS should be considered an *instance of* Actor, and not a subclass. It does not describe a subset of Actor varietals, it *is* a Actor varietal.

Object and Datatype Properties

RDFS ✓

Properties let us assert general facts about the members of classes and specific facts about individuals.
A property is a ***binary relation***.

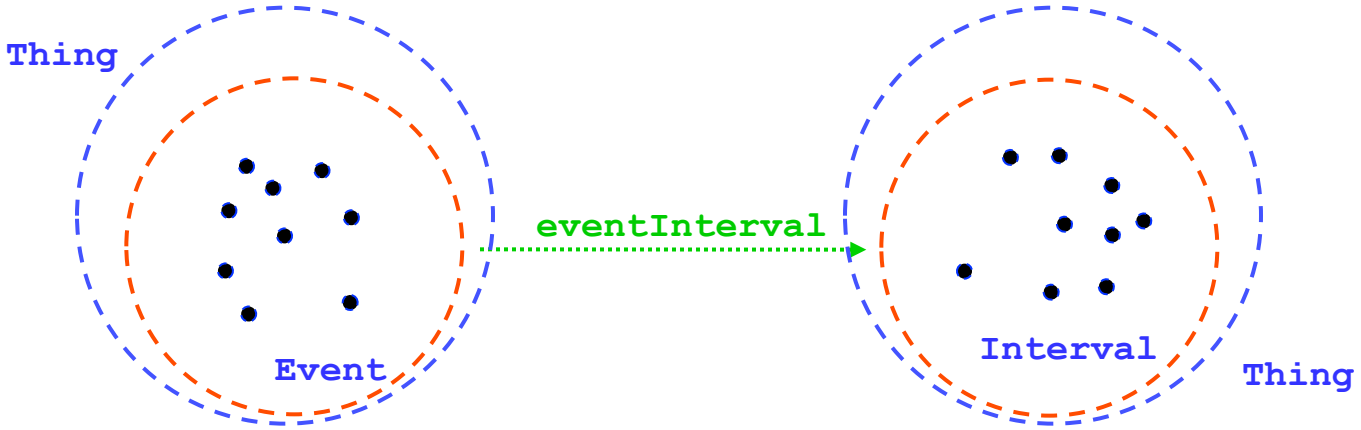
[1] ***object properties***, relations between instances of two classes (*individuals to individuals*)
[2] ***datatype properties***, relations between instances of classes (*individuals to datatypes*)

```
<owl:ObjectProperty rdf:ID="eventInterval">
        <rdfs:domain rdf:resource="Event"/>
        <rdfs:range rdf:resource="Interval"/>
</owl:ObjectProperty>
```

$\exists$.eventInterval $\sqsubseteq$ Event

$\exists$.eventInterval^{--} $\sqsubseteq$ Interval

```
<owl:ObjectProperty rdf:ID="track">
        <rdfs:domain rdf:resource="Record" />
        <rdfs:range rdf:resource="Track" />
</owl:ObjectProperty>
```

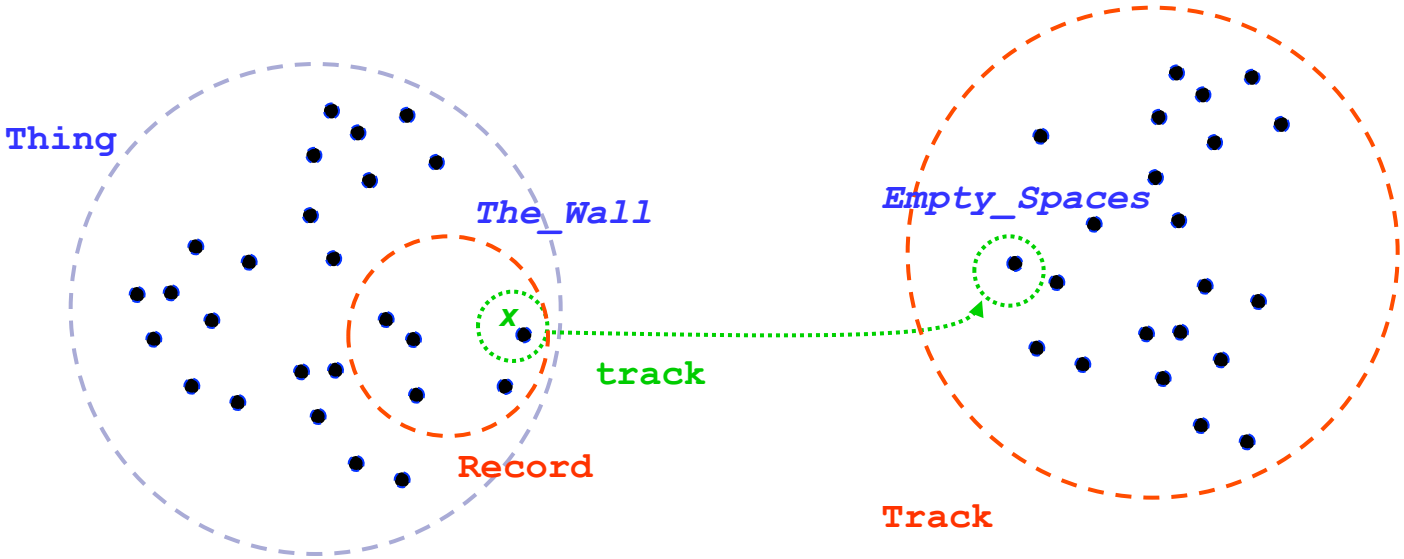


Properties and Inference

RDFS ✓

The use of range and domain information in OWL is different from type information in a programming language. Among other things, types are used to check consistency in a programming language. In OWL, ***a range may be used to infer a type***.
For example, given

```
<owl:Thing rdf:ID="The_Wall">
        <track rdf:resource="Empty_Spaces" />
</owl:Thing>
```



We can ***infer*** that `The_Wall` is a Record
because the domain of `track` is `Record`

Taxonomy of Properties

RDFS ✓

```
<owl:Class rdf:ID="Thing" />
```

```
<owl:Class rdf:ID="license">
        <rdfs:subPropertyOf rdf:resource="rights" />
        ...
</owl:Class>
```

license ⊑ rights

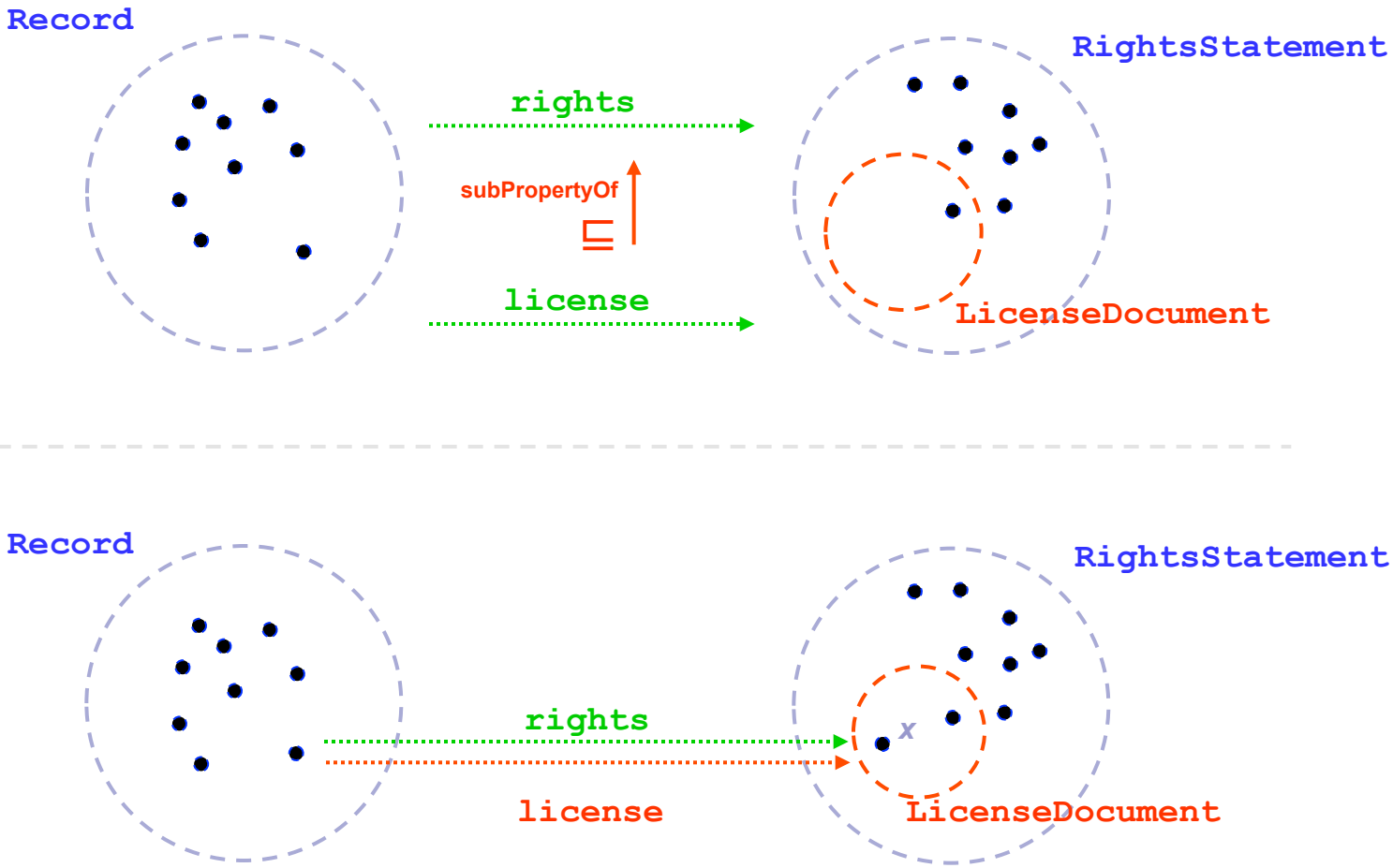
```
<owl:ObjectProperty rdf:ID="rights">
        <rdfs:domain rdf:resource="Record" />
        <rdfs:range rdf:resource="RightsStatement" />
</owl:ObjectProperty>
```

```
<owl:ObjectProperty rdf:ID="license">
        <rdfs:subPropertyOf rdf:resource="rights" />
        <rdfs:range rdf:resource="LicenseDocument" />
        ...
</owl:ObjectProperty>
```

`license` is a subproperty of the `rights` property, with its range further restricted to LicenseDocument

Taxonomy of Properties and Inference

RDFS ✓



Anything with a `license` property with value ***x***
also has a `rights` property with value ***x***

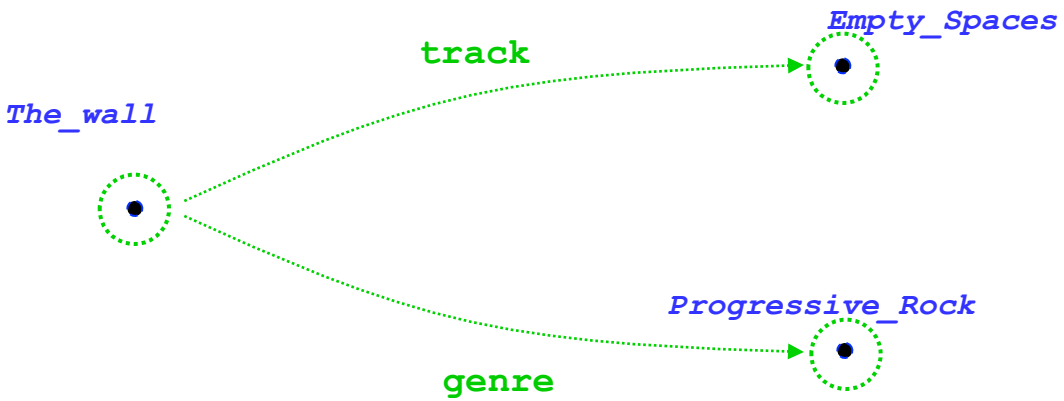
Properties of Individuals

RDF ✓

```
<Record rdf:ID="The_Wall">
        <track rdf:resource="Empty_Spaces" />
</Record>

<Channel rdf:ID="Progressive_Rock" />
```

```
<Record rdf:ID="The_wall" >
        <track rdf:resource="Empty_Spaces"/>
        <genre rdf:resource="Progressive_Rock" /> </Record>
```

<The_Wall, Empty_Spaces> : track



Property Existential/Universal Restrictions

allValuesFrom

The restriction allValuesFrom is stated on a property with respect to a class. It means that this property on this particular class has a local range restriction associated with it. Thus **if** an instance of the class is related by the property to a second individual, **then** the second individual can be inferred to be an instance of the local range restriction class

```
<owl:Class rdf:ID="Record">
          <rdfs:subClassOf rdf:resource="MusicManifestation" />
          ...
          <rdfs:subClassOf>
                    <owl:Restriction>
                    <owl:onProperty rdf:resource="producer" />
                    <owl:allValuesFrom rdf:resource="MusicArtist" />
          </owl:Restriction>
          </rdfs:subClassOf>
          ...
</owl:Class>
```

$\text{Record} \sqsubseteq \forall\ \text{producer}.\text{MusicArtist}$

Every release has only producer of type MusicArtist

someValuesFrom

The restriction *someValuesFrom* is stated on a property with respect to a class. A particular class may have a restriction on a property that at least one value for that property is of a certain type

```
<owl:Class rdf:ID="Release">
          <rdfs:subClassOf rdf:resource="MusicManifestation" />
          <rdfs:subClassOf>
                    <owl:Restriction>
                    <owl:onProperty rdf:resource="record" />
          <owl:someValuesFrom rdf:resource="Record" />                    </
owl:Restriction>
          </rdfs:subClassOf>
          ...
</owl:Class>
```

$\text{Release} \sqsubseteq \exists\ \text{record}.\text{Record}$

Every release has at least one record of type Record

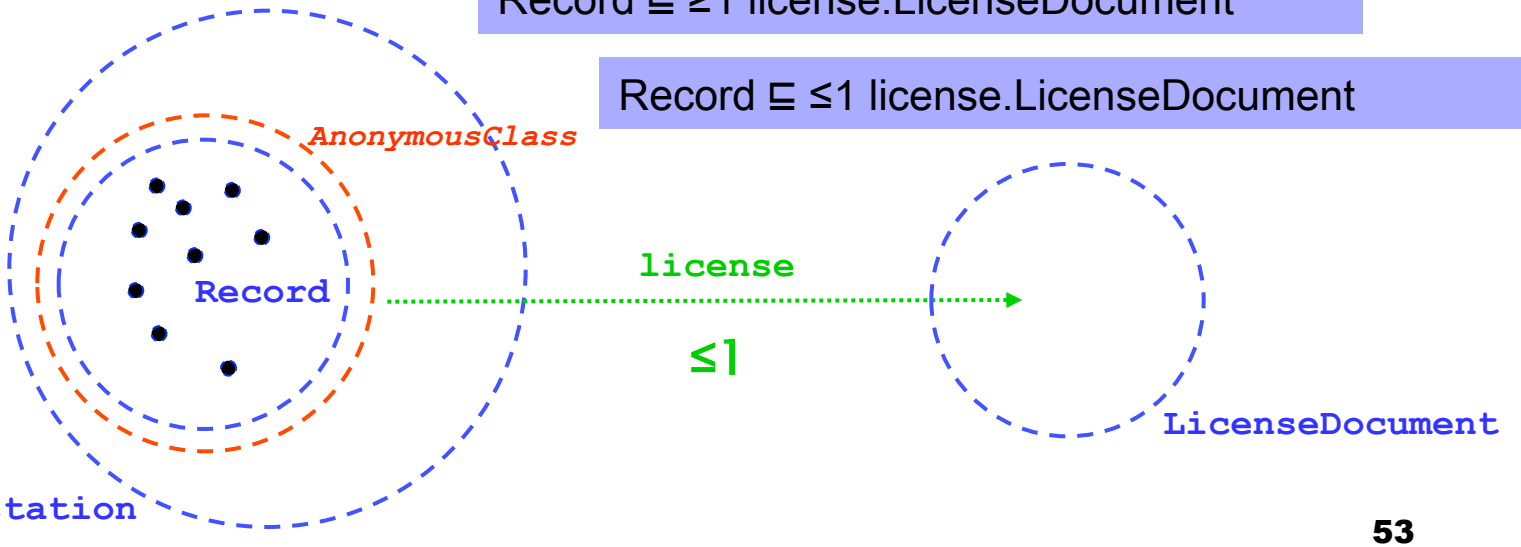
Cardinality restrictions

```
<owl:Class rdf:ID="Record">
    <rdfs:subClassOf rdf:resource="MusicalManifestation"/>

    <rdfs:subClassOf>
        <owl:Restriction>
        <owl:onProperty rdf:resource="license"/>
        <owl:minCardinality rdf:datatype="nonNegativeInteger">
        1
        </owl:minCardinality>
        </owl:Restriction>
    </rdfs:subClassOf>

    ...
</owl:Class>
```

A record has *at least* **one** license document attached

Record ⊑ ≥1 license.LicenseDocument

Record ⊑ ≤1 license.LicenseDocument



Properties Formal Characteristics

inverseOf

One property may be stated to be the inverse of another property. If the property P1 is stated to be the inverse of the property P2, then if X is related to Y by the P2 property, then Y is related to X by the P1 property

TransitiveProperty

Properties may be stated to be transitive. If a property is transitive, then if the pair (x,y) is an instance of the transitive property P, and the pair (y,z) is an instance of P, then the pair (x,z) is also an instance of P

OWL Lite (and OWL DL) impose the side condition that transitive properties (and their superproperties) cannot have a maxCardinality 1 restriction. Without this side-condition, OWL Lite and OWL DL would become undecidable languages.

SymmetricProperty

Properties may be stated to be symmetric. If a property is symmetric, then if the pair (x,y) is an instance of the symmetric property P, then the pair (y,x) is also an instance of P

FunctionalProperty

P(x,y) and P(x,z) implies y = z

Properties may be stated to have a unique value. If a property is a FunctionalProperty, then it has no more than one value for each individual (it may have no values for an individual). This characteristic has been referred to as having a unique property. FunctionalProperty is shorthand for stating that the property's minimum cardinality is zero and its maximum cardinality is 1.

InverseFunctionalProperty

P(y,x) and P(z,x) implies y = z

Properties may be stated to be inverse functional. If a property is inverse functional then the inverse of the property is functional. Thus the inverse of the property has at most one value for each individual. This characteristic has also been referred to as an unambiguous property

SymmetricProperty

```
<owl:ObjectProperty rdf:ID="adjacentTerritory">
          <rdf:type rdf:resource="SymmetricProperty" />
          <rdfs:domain rdf:resource="Territory" />
          <rdfs:range rdf:resource="Territory" />
</owl:ObjectProperty>


<Territory rdf:ID="Tuscany">
          <locatedIn rdf:resource="Italy" />
          <adjacentTerritory rdf:resource="Umbria" />
</Territory>
```

Tuscany is adjacent to *Umbria* and vice-versa.
Tuscany is located in the *Italy* but not vice versa.

TransitiveProperty

```
<owl:ObjectProperty rdf:ID="locatedIn">
          <rdf:type rdf:resource="TransitiveProperty" />
          <rdfs:domain rdf:resource="owl:Thing" />
          <rdfs:range rdf:resource="Territory" />
</owl:ObjectProperty>

<Territory rdf:ID="Florence">
          <locatedIn rdf:resource="Tuscany" />
</Territory>

<Territory rdf:ID="Tuscany">
          <locatedIn rdf:resource="Italy" />
</Territory>
```

$locatedIn^{+} \sqsubseteq locatedIn$

Because the *Florence* is locatedIn *Tuscany*,
then it must also be locatedIn the *Italy*, since locatedIn is transitive

Equality and Inequality

`equivalentClass`

Two ***classes*** may be stated to be equivalent. Equivalent classes have the same instances. Equality can be used to create synonymous classes.

For example, Car can be stated to be *equivalentClass* to Automobile. From this a reasoner can deduce that any individual that is an instance of Car is also an instance of Automobile and vice versa.

`equivalentProperty`

Two ***properties*** may be stated to be equivalent. Equivalent properties relate one individual to the same set of other individuals. Equality may be used to create synonymous properties.

`sameAs`

Two ***individuals*** may be stated to be the same. These constructs may be used to create a number of different names that refer to the same individual.

For example, the individual Deborah may be stated to be the same individual as DeborahMcGuinness.

`differentFrom`

An ***individual*** may be stated to be different from other individuals.

For example, the individual Frank may be stated to be different from the individuals Deborah and Jim. Thus, if the individuals Frank and Deborah are both values for a property that is stated to be functional (thus the property has at most one value), then there is a contradiction. Explicitly stating that individuals are different can be important in when using languages such as OWL (and RDF) that do not assume that individuals have one and only one name. For example, with no additional information, a reasoner will not deduce that Frank and Deborah refer to distinct individuals.

`AllDifferent`

A number of ***individuals*** may be stated to be mutually distinct in one AllDifferent statement.

For example, Frank, Deborah, and Jim could be stated to be mutually distinct using the AllDifferent construct. Unlike the differentFrom statement above, this would also enforce that Jim and Deborah are distinct (not just that Frank is distinct from Deborah and Frank is distinct from Jim). The AllDifferent construct is particularly useful when there are sets of distinct objects and when modelers are interested in enforcing the ***unique names assumption*** within those sets of objects. It is used in conjunction with `distinctMembers` to state that all members of a list are distinct and pairwise disjoint.

Concept definitions and “roles”

$\text{Student} \sqsubseteq \exists \text{enrolledIn}.\text{EducationalInstitution}$

$\text{Student} \equiv \text{Person} \sqcap \exists \text{enrolledIn}.\text{EducationalInstitution}$

- In order to be considered a student one needs to be enrolled in at least in an educational institution
  - Stating a **necessary condition**
- A student IS a person that is enrolled at least in an educational institution
  - Stating a **necessary and sufficient condition**
  - Student identity: if a person has an enrollment code can be **classified** as a student

Advanced OWL DL

`oneOf` (enumerated classes)

Classes can be described by enumeration of the individuals that make up the class. The members of the class are exactly the set of enumerated individuals; no more, no less.

For example, the class of daysOfTheWeek can be described by simply enumerating the individuals Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday. From this a reasoner can deduce the maximum cardinality (7) of any property that has daysOfTheWeek as its allValuesFrom restriction.

`hasValue` (property values)

A ***property*** can be required to have a certain individual as a value (also sometimes referred to as property values).

For example, instances of the class of dutchCitizens can be characterized as those people that have theNetherlands as a value of their nationality. (The nationality value, theNetherlands, is an instance of the class of Nationalities).

`disjointWith`

Classes may be stated to be disjoint from each other.

For example, Man and Woman can be stated to be disjoint classes. From this disjointWith statement, a reasoner can deduce an inconsistency when an individual is stated to be an instance of both and similarly a reasoner can deduce that if A is an instance of Man, then A is *not* an instance of Woman.

`unionOf, complementOf, intersectionOf` (Boolean combinations)

OWL DL and OWL Full allow arbitrary Boolean combinations of classes and restrictions: unionOf, complementOf, and intersectionOf.

For example, using unionOf, we can state that a class contains things that are either USCitizens or DutchCitizens. Using complementOf, we could state that children are *not* SeniorCitizens. (i.e. the class Children is a subclass of the complement of SeniorCitizens). Citizenship of the European Union could be described as the union of the citizenship of all member states.

`minCardinality, maxCardinality, cardinality` (full cardinality)

While in OWL Lite, cardinalities are restricted to at least, at most or exactly 1 or 0, full OWL allows cardinality statements for arbitrary non-negative integers.

Open (vs. Close) World Semantics

- OWL and DLs assume Open World Semantics

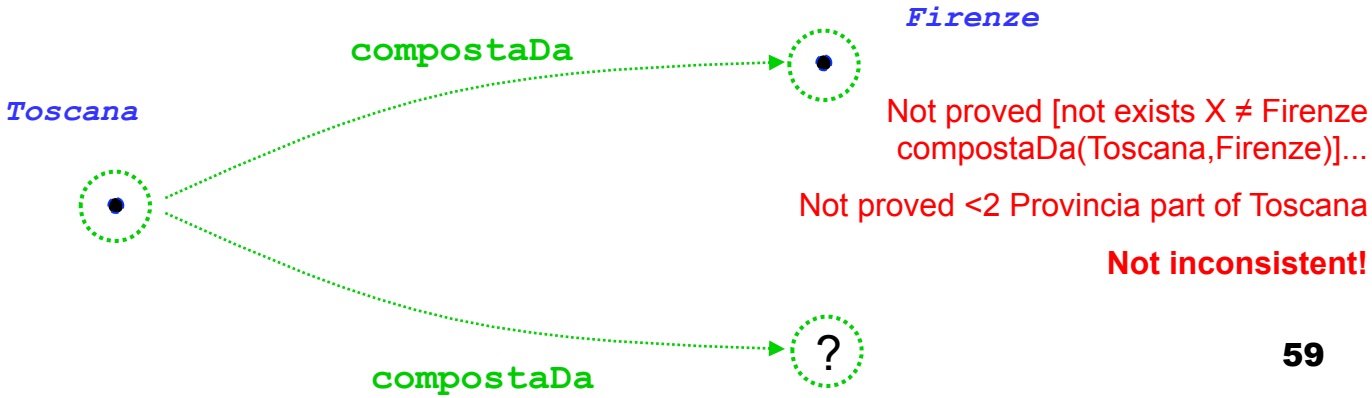
<Florence, Tuscany> : locatedIn

<Florence, USA> : locatedIn

not in the KB

  - Is Florence located in USA?
    - Close World: NO
    - Open World: Can't tell!
      - It is not proved that it does not hold!
  - Open World: what is not proved from the KB is not assumed to be false
  - Close World (DBs): what is not proved to be true is assumed to be false

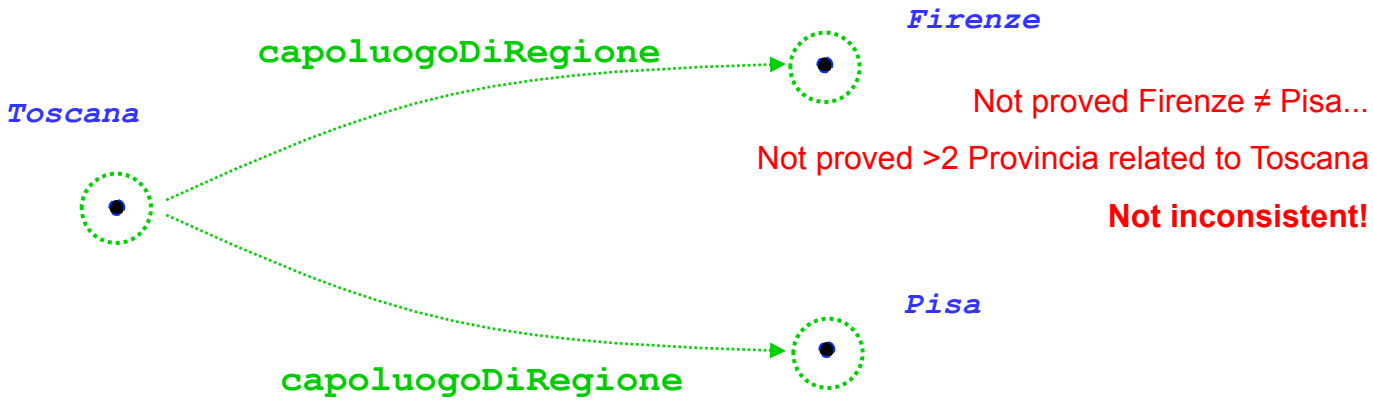
Regione ⊑ ≥2 compostaDa.Provincia



Unique Name Assumption (UNA)

- UNA: different names are assumed to refer to different entities
  - Firenze ≠ Pisa
- OWL and DL does not assume UNA
  - Some reasoners allow to switch between UNA and not UNA
- Be careful: consequences on reasoning about cardinalities!

Regione ⊑ =1capoluogoDiRegione.Provincia



DL based reasoning in OWL-DL

Basic Inference Tasks

- Knowledge is correct (captures intuitions)
  - Does C subsume D w.r.t. ontology $\mathcal{O}$? ($\mathrm{C}^{\mathcal{I}} \subseteq \mathrm{D}^{\mathcal{I}}$ in *every* model $\mathcal{I}$ of $\mathcal{O}$)
- Knowledge is minimally redundant (no unintended synonyms)
  - Is C equivallent to D w.r.t. $\mathcal{O}$? ($\mathrm{C}^{\mathcal{I}} = \mathrm{D}^{\mathcal{I}}$ in *every* model $\mathcal{I}$ of $\mathcal{O}$)
- Knowledge is meaningful (classes can have instances)
  - Is C is satisfiable w.r.t. $\mathcal{O}$? ($\mathrm{C}^{\mathcal{I}} \neq \emptyset$ in *some* model $\mathcal{I}$ of $\mathcal{O}$)

- Querying knowledge
  - Is x an instance of C w.r.t. $\mathcal{O}$? ($\mathrm{x}^{\mathcal{I}} \in \mathrm{C}^{\mathcal{I}}$ in *every* model $\mathcal{I}$ of $\mathcal{O}$)
  - Is $\langle \mathrm{x},\mathrm{y} \rangle$ an instance of R w.r.t. $\mathcal{O}$? ($(\mathrm{x}^{\mathcal{I}},\mathrm{y}^{\mathcal{I}}) \in \mathrm{R}^{\mathcal{I}}$ in *every* model $\mathcal{I}$ of $\mathcal{O}$)

- Above problems can be solved using highly optimised DL reasoners

DL Reasoning: Basics

- Tableau algorithms used to test satisfiability (consistency)

$$C \sqsubseteq D \text{ iff } C \sqcap \neg D \text{ is unsatisfiable.}$$

  - E.g.: to check whether $(\exists R.A) \sqcap (\exists R.B) \sqsubseteq \exists R.(A \sqcap B)$
  - Check satisfiability of: $C = (\exists R.A) \sqcap (\exists R.B) \sqcap \neg(\exists R.(A \sqcap B))$
- Try to build a tree-like model of the input concept C
- Decompose C syntactically
  - Apply tableau expansion rules
  - Infer constraints on elements of model
- Tableau rules correspond to constructors in logic (u, t etc)
  - Some rules are nondeterministic (e.g., t, 6)
  - In practice, this means search
- Stop when no more rules applicable or clash occurs
  - Clash is an obvious contradiction, e.g., A(x), :A(x)
- Cycle check (blocking) may be needed for termination
- C satisfiable iff rules can be applied such that a fully expanded clash free tree is constructed

DL Reasoning: Basics

The $\rightarrow_{\sqcap}$-rule
Condition: $\mathcal{A}$ contains $(C_1 \sqcap C_2)(x)$, but it does not contain both $C_1(x)$ and $C_2(x)$.
Action: $\mathcal{A}' = \mathcal{A} \cup \{C_1(x), C_2(x)\}$.

The $\rightarrow_{\sqcup}$-rule
Condition: $\mathcal{A}$ contains $(C_1 \sqcup C_2)(x)$, but neither $C_1(x)$ nor $C_2(x)$.
Action: $\mathcal{A}' = \mathcal{A} \cup \{C_1(x)\}$, $\mathcal{A}'' = \mathcal{A} \cup \{C_2(x)\}$.

The $\rightarrow_{\exists}$-rule
Condition: $\mathcal{A}$ contains $(\exists R.C)(x)$, but there is no individual name z such that $C(z)$ and $R(x, z)$ are in $\mathcal{A}$.
Action: $\mathcal{A}' = \mathcal{A} \cup \{C(y), R(x, y)\}$ where y is an individual name not occurring in $\mathcal{A}$.

The $\rightarrow_{\forall}$-rule
Condition: $\mathcal{A}$ contains $(\forall R.C)(x)$ and $R(x, y)$, but it does not contain $C(y)$.
Action: $\mathcal{A}' = \mathcal{A} \cup \{C(y)\}$.

The $\rightarrow_{\geqslant}$-rule
Condition: $\mathcal{A}$ contains $(\geqslant n\, R)(x)$, and there are no individual names $z_1, \ldots, z_n$ such that $R(x, z_i)$ $(1 \leq i \leq n)$ and $z_i \neq z_j$ $(1 \leq i < j \leq n)$ are contained in $\mathcal{A}$.
Action: $\mathcal{A}' = \mathcal{A} \cup \{R(x, y_i) \mid 1 \leq i \leq n\} \cup \{y_i \neq y_j \mid 1 \leq i < j \leq n\}$, where $y_1, \ldots, y_n$ are distinct individual names not occurring in $\mathcal{A}$.

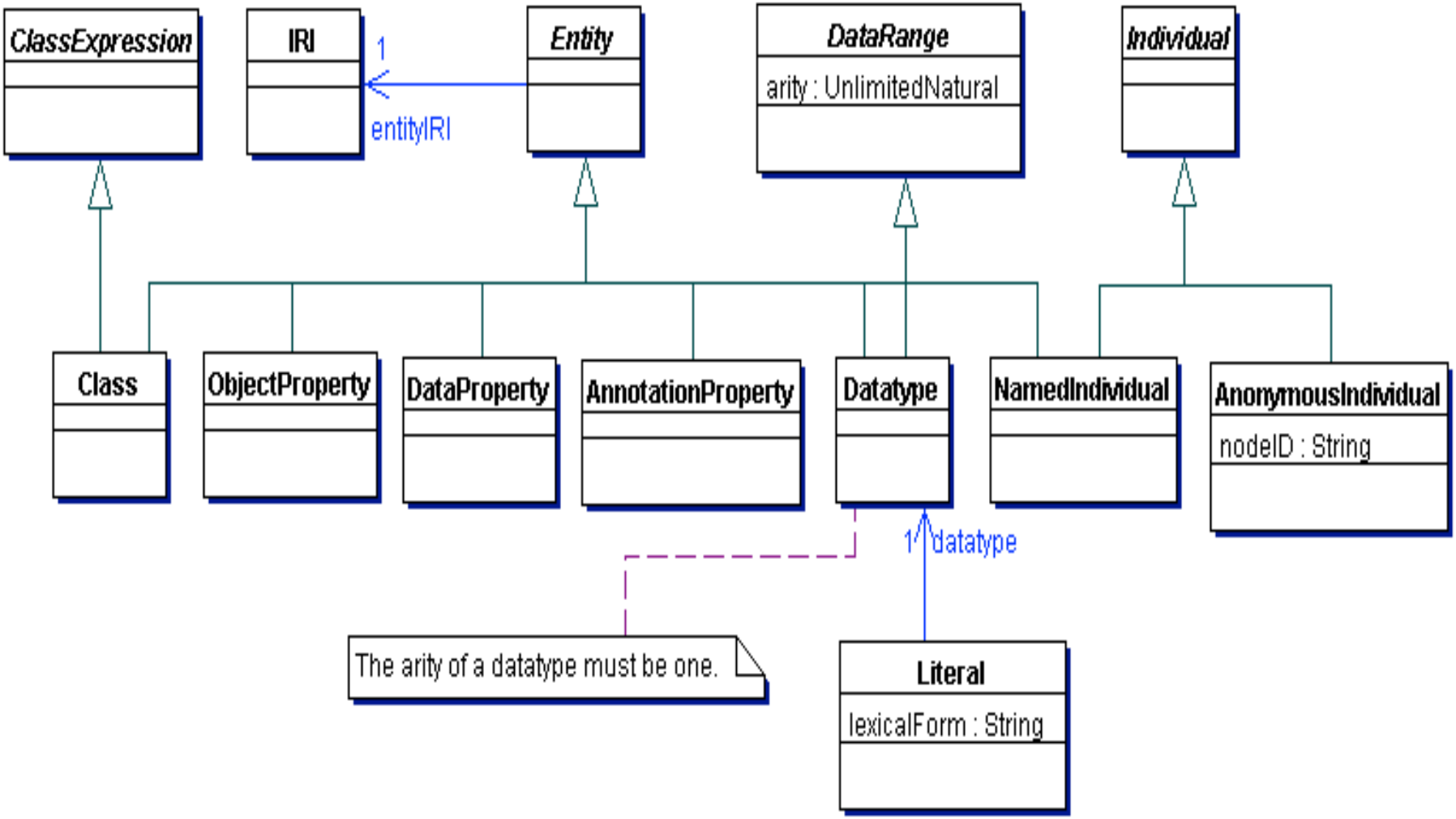
The $\rightarrow_{\leqslant}$-rule
Condition: $\mathcal{A}$ contains distinct individual names $y_1, \ldots, y_{n+1}$ such that $(\leqslant n\, R)(x)$ and $R(x, y_1), \ldots, R(x, y_{n+1})$ are in $\mathcal{A}$, and $y_i \neq y_j$ is not in $\mathcal{A}$ for some $i \neq j$.
Action: For each pair y_i, y_j such that $i > j$ and $y_i \neq y_j$ is not in $\mathcal{A}$, the ABox $\mathcal{A}_{i,j} = [y_i/y_j]\mathcal{A}$ is obtained from $\mathcal{A}$ by replacing each occurrence of y_i by y_j.

Fig. 2.6. Transformation rules of the satisfiability algorithm.

Basic Inference Tasks: More Pragmatically..

- Consistency check (TBox/ABox)
- Infer subclass / subproperty relations (via subsumption)
- Inheritance from superclasses
- Infer type of individuals (based on axioms)
- Infer new relations (e.g., from transitive, symmetric or inverse properties)

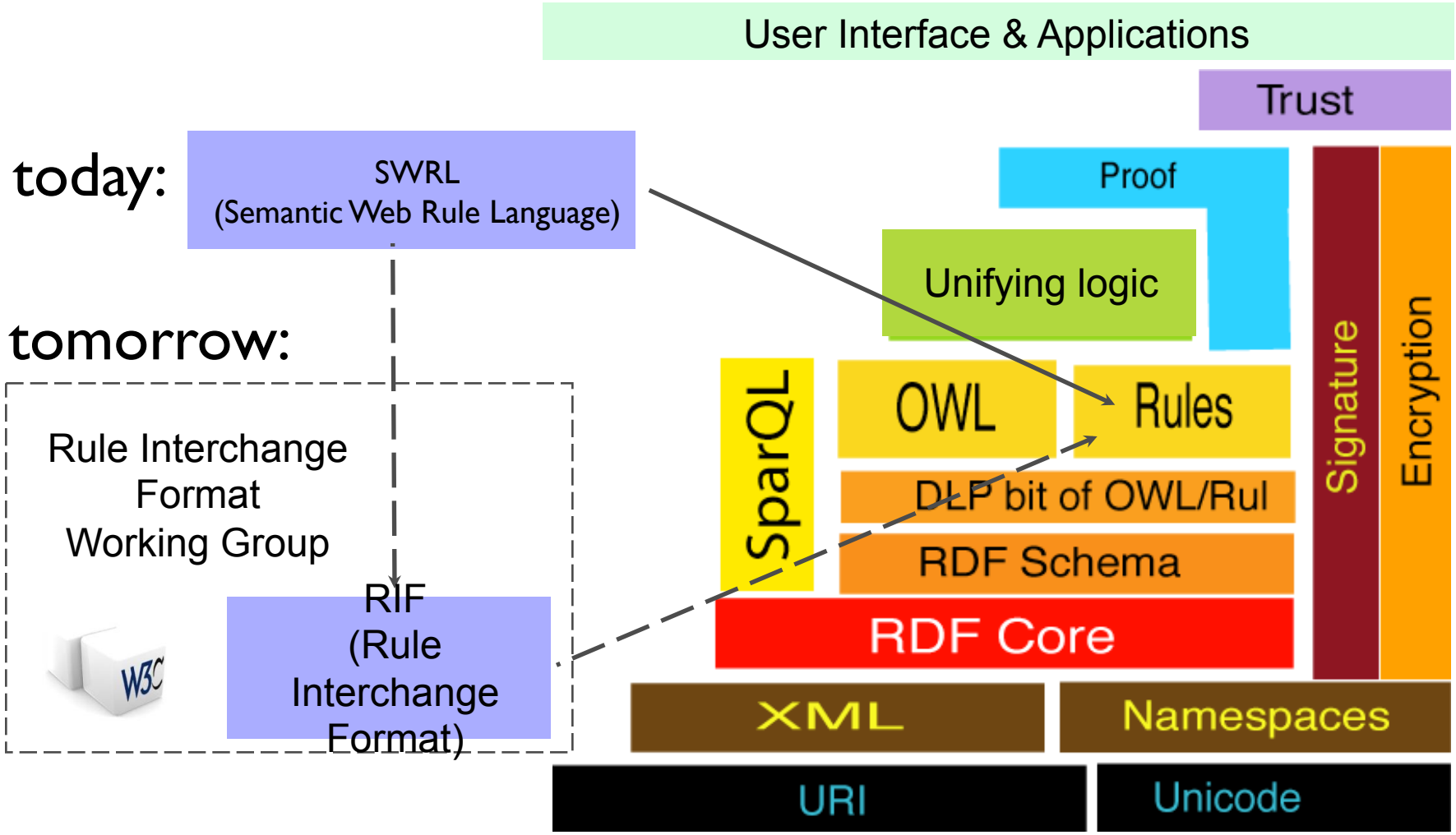
OWL 2: Entities, Literals, and Anonymous Individuals*



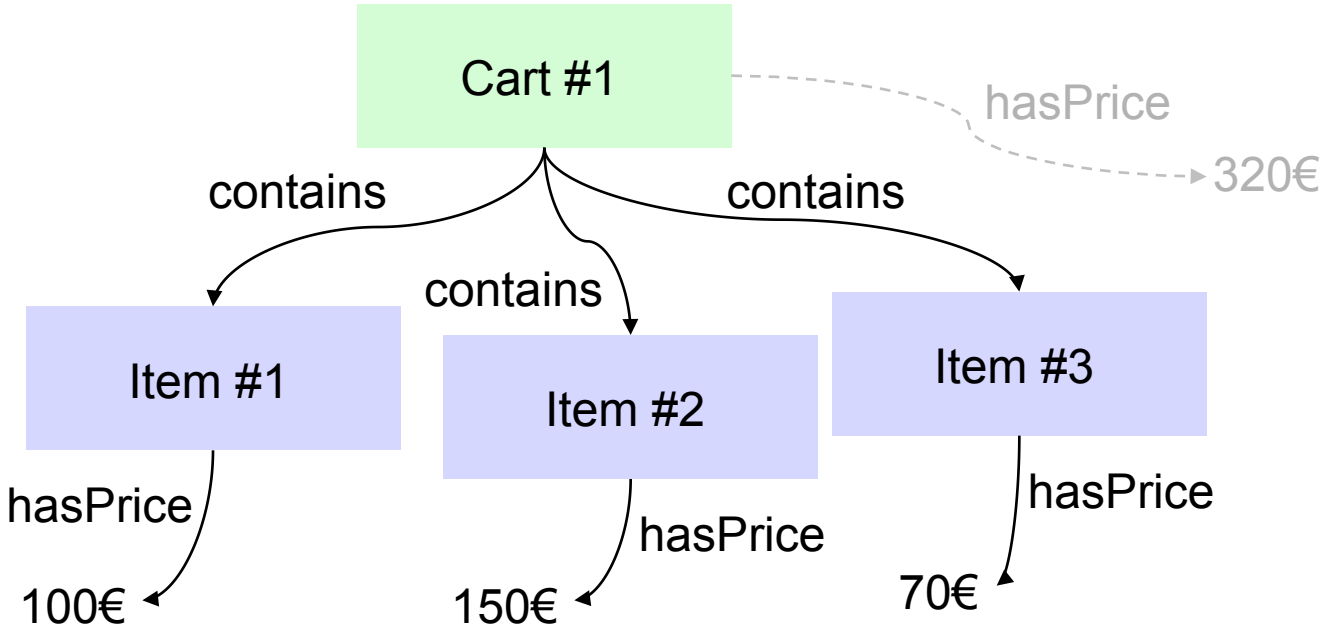
*By Fausto Giunchiglia, taken from Logics for Data and Knowledge Representation – UniTN

Rule-based Reasoning

Semantic Web Layer Cake



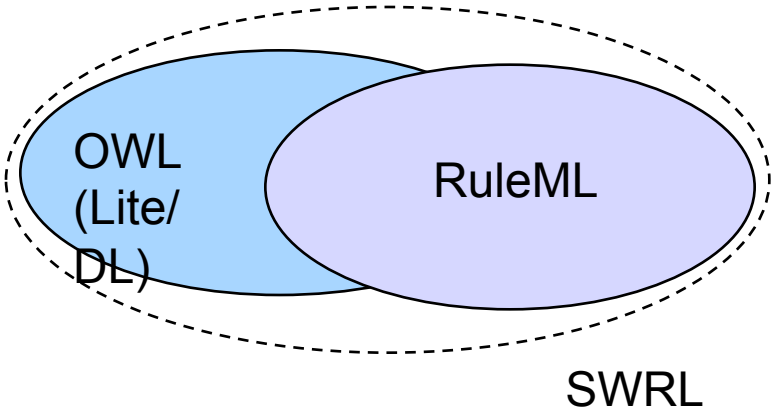
OWL Reasoning



- "The total price is the sum of the price of the items in the cart"
- OWL lacks of support for this kind of reasoning !

SWRL (Semantic Web Rule Language)

SWRL is a a rule language, combining OWL (Lite or DL) with RuleML (Rule Markup Language)



Example (*human readable*)

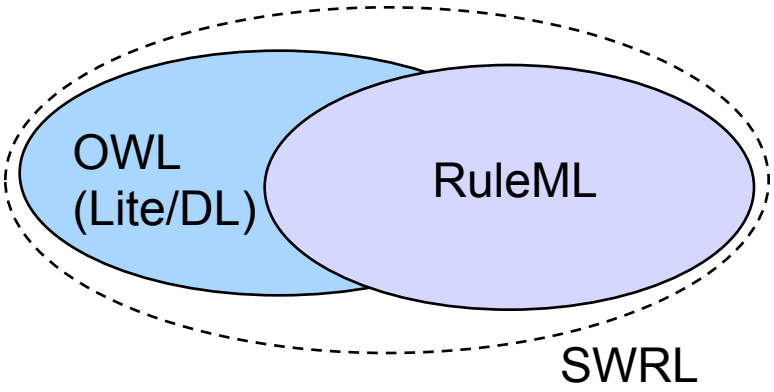
hasPrice(?x1,?p1) ∧ hasPrice(?x2,?p2) ∧

∧ lessThan (?p1,?p2) → isCheaper(?x1,?x2)

http://www.w3.org/Submission/SWR

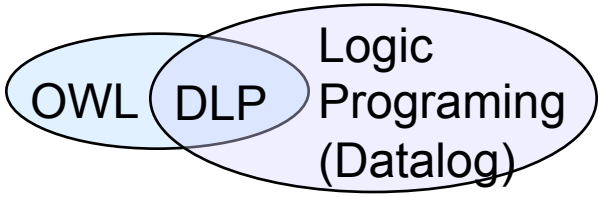
Decidability

Decidability:
Terminates after a finite amount of time and correctly decides



Decidability ? No

SWRL Alternative: DLP



Decidability ? YES

SWRL Rule Example

hasPrice(?x1,?p1) ∧ hasPrice(?x2,?p2) ∧ lessThan (?p1,?p2) ⟶ isCheaper(?x1,?x2)

In the abstract syntax the rule would be written like:

```
Implies (
        Antecedent(
                hasPrice(I-variable(x1) I-variable(p1))
                hasPrice(I-variable(x2) I-variable(p2))
                swrlb:lessThan(I-variable(p1) I-variable(p2)))
                Consequent(
                        isCheaper(I-variable(x1) I-variable(x2))))
```

swrlb: namespace for the SWRL Built-ins

Rule Example in RDF Concrete Syntax

```
<swrl:Variable rdf:ID="x1"/>
<swrl:Variable rdf:ID="x2"/>
<swrl:Variable rdf:ID="p1"/>
<swrl:Variable rdf:ID="p2"/>
<ruleml:Imp>
  <ruleml:body rdf:parseType="Collection">
    <swrl:IndividualPropertyAtom>
      <swrl:propertyPredicate rdf:resource="&eg;hasPrice"/>
      <swrl:argument1 rdf:resource="#x1" />
      <swrl:argument2 rdf:resource="#p1" />
    </swrl:IndividualPropertyAtom>
    <swrl:IndividualPropertyAtom>
      <swrl:propertyPredicate rdf:resource="&eg;hasPrice"/>
      <swrl:argument1 rdf:resource="#x2" />
      <swrl:argument2 rdf:resource="#p2" />
    </swrl:IndividualPropertyAtom>
    <swrl:IndividualPropertyAtom>
      <swrl:propertyPredicate rdf:resource="&eg;swrlb:lessThen"/>
      <swrl:argument1 rdf:resource="#p2" />
      <swrl:argument2 rdf:resource="#p2" />
    </swrl:IndividualPropertyAtom>
  </ruleml:body>
  <ruleml:head rdf:parseType="Collection">
    <swrl:IndividualPropertyAtom>
      <swrl:propertyPredicate rdf:resource="&eg;isCheaper"/>
      <swrl:argument1 rdf:resource="#x1" />
      <swrl:argument2 rdf:resource="#x2" />
    </swrl:IndividualPropertyAtom>
  </ruleml:head>
</ruleml:Imp>
```

swrl: namespace for the SWRL

ruleml: namespace for the SWRL Built-ins

RDF Triples.

Datalog

Example Datalog program:
parent(bill,mary).
parent(mary,john).
ancestor(X,Y) :- parent(X,Y).
ancestor(X,Y) :- ancestor(X,Z),parent(Z,Y).

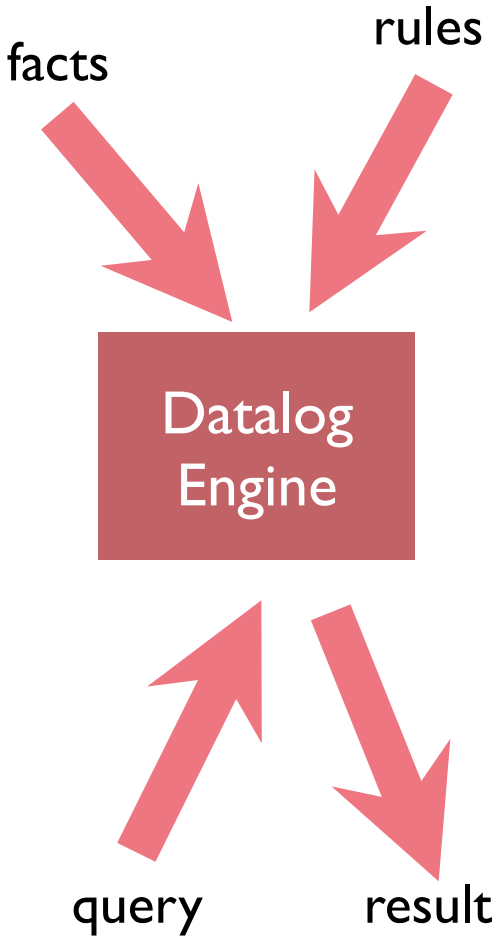
facts

rules

Query:

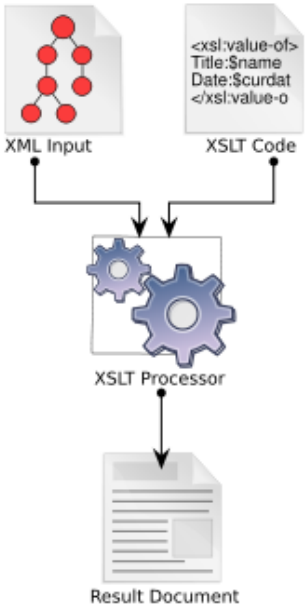
?- ancestor(bill,X).

ancestor(bill,mary).
ancestor(bill,john).



RuleML (Rule Markup Language)

- RuleML is a family of languages developed to express both forward and backward rules in XML for deduction, rewriting, and further inferential-transformational tasks



- Example:
  XSLT (Extensible Stylesheet Language Transformations) is a restricted term-rewriting system of rules, written in XML, for transforming XML documents into other documents

http://www.ruleml.org/

Tool Support for OWL

- Ontology editors
  - Protege (http://protege.stanford.edu/)
  - ... Many commercial tools
- APIs
  - OWL-API (http://owlapi.sourceforge.net)
  - Jena (http://jena.sourceforge.net)
  - …
- OWL makes use of the reasoners such as,
  - FaCT++ (http://owl.man.ac.uk/factplusplus/)
  - Pellet (http://clarkparsia.com/pellet/)
  - HermiT (http://hermit-reasoner.com/)
  - Jena rules
  - … including commercial reasonsers (e.g., Oracle for RDFS++)
  - For other reasoners check http://owl.cs.manchester.ac.uk/tools/list-of-reasoners/

*By Fausto Giunchiglia, taken from Logics for Data and Knowledge Representation – UniTN

Ontology Editing

- With state of the art editors (e.g. Protegè):
  - Graphical support for design and editing (for TBox and ABox)
    - Editing and representation with Description Logic syntax and automatic generation of OWL and RDF code
    - Basic checks (e.g. OWL dialect)
    - Visualization Tools
  - Integration with state of the art reasoners, rule based systems and query systems

Further Readings

- OWL Web Ontology Language
  Semantics and Abstract Syntax. http://www.w3.org/TR/2004/REC-owl-semantics-20040210/
- Horrocks, I., Patel-Schneider, Peter F., McGuinness, D. L., and Welty, C. A. OWL: a description logic based ontology language for the semantic web. Deborah L. McGuinness and Peter F. Patel-Schneider. From Description Logic Provers to Knowledge Representation Systems. In *The Description Logic Handbook: Theory, Implementation and Applications*, ed. Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider. Cambridge University Press, 2nd edition, August 2007, pp. 458--486.
- Antoniou, G. and Harmelen, F.V. A semantic web primer. http://www.emu.edu.tr/aelci/Courses/D-588/MIT.Press.A.Semantic.Web.Primer.eBook-TLFeBOOK.pdf
- http://www.w3.org/TR/2004/REC-owl-guide-20040210/
- http://www.w3.org/TR/owl2-overview/
- http://www.w3.org/TR/2009/REC-owl2-primer-20091027/
- http://www.w3.org/TR/2009/REC-owl2-quick-reference-20091027/
- http://www.w3.org/TR/2009/REC-owl2-syntax-20091027/
- http://www.w3.org/TR/2009/REC-owl2-new-features-20091027/

*By Fausto Giunchiglia, taken from Logics for Data and Knowledge Representation – UniTN