What is different with KBs from DBs is the possibility of automatic reasoning.

Because a KB is made of a TBox $T$ (terminological box) and an ABox $A$ (assertional box) we write:

$$KB = \langle T, A \rangle$$

In logic when we talk about "reasoning" we refer to deductive reasoning or simply deductions.

In general, a reasoning is a procedure that allows to verify if a statement $X$ (example equivalence or subsumption between two terms) is logic consequence of a KB.
Logic consequence (1)

- Intuitively a statement X is logic consequence of a KB when X is true in every situation where are true the terminological axioms and assertions in the KB.

- More precisely a statement X is the logic consequence of a KB when X is true in every model of terminological axioms and assertions in KB.

- In this case we write:

  \[ KB \models X \]

  \[ KB \text{ logically imply } X \text{ (X is a logical consequence of KB)} \]

Logic consequence (2)

- Let's consider the TBox T with the following axioms:

  T1. \( \text{PARENT} \equiv \text{PERSON} \sqcap \exists \text{parentOf} \)
  T2. \( \text{parentOf} : \text{PERSON} \rightarrow \text{PERSON} \)
  T3. \( \text{WOMAN} \equiv \text{PERSON} \sqcap \neg \text{FEMALE} \)
  T4. \( \text{MAN} \equiv \text{PERSON} \sqcap \neg \text{FEMALE} \)
  T5. \( \text{MOTHER} \equiv \text{PARENT} \sqcap \text{FEMALE} \)
  T6. \( \text{FATHER} \equiv \text{PARENT} \sqcap \neg \text{FEMALE} \)

- The T axioms logically imply some statements that are not present in T but are necessarily true in the hypothesis that T is true.
• Every mother is a person and a woman:
  \[
  \text{MOTHER} \sqsubseteq \text{PERSON} \\
  \text{MOTHER} \sqsubseteq \text{WOMAN}
  \]

• Every father is a person and a man:
  \[
  \text{FATHER} \sqsubseteq \text{PERSON} \\
  \text{FATHER} \sqsubseteq \text{MAN}
  \]

• Class of fathers and mothers are disjoint:
  \[
  \text{MOTHER} \cap \text{FATHER} \equiv \bot
  \]

To highlight that these statements are logic consequence of \( T \) we write:

\[
T \models \text{MOTHER} \sqsubseteq \text{PERSON}
\]

Other statements are not logic consequence of \( T \). For example the previous TBox does not logically imply that a person have 2 parents. To state this we write:

\[
T \not\models \text{PERSON} \sqsubseteq \equiv 2 \: \text{parentOf-}
\]
Reasoning types

Reasoning task
Is characterized with the type of statements to be inferred

Reasoning procedure
The algorithm used for reasoning

Reasoning service
A service implemented by a tool, usable from applications accessing to the KB

Reduction to subsumption

- It can be easily seen that fundamental reasoning tasks for TBox can be reduced to subsumption
- Equivalence
  \[ T \models C \equiv D \text{ is equivalent to } T \models C \sqsubseteq D \text{ e } T \models D \sqsubseteq C \]
- Soddisfacibility
  \[ T \not\models C \sqsubseteq \bot \]
- Disjunction
  \[ T \models C \sqcap D \sqsubseteq \bot \]
- This the way used to implement reasoning services for low expressive DLs
Reduction to satisfiability

- The fundamental reasoning tasks for Tboxes can be reduced to satisfiability

- **Subsumption** $\mathcal{T} \models C \sqsubseteq D$
  
  $\mathcal{T} \models C \sqcap \neg D$ is not satisfiable

- **Equivalence** $\mathcal{T} \models C \equiv D$
  
  $\mathcal{T} \models C \sqcap \neg D$ is not satisfiable and
  
  $\mathcal{T} \models \neg C \sqcap D$ is not satisfiable

- **Disjunction**
  
  $\mathcal{T} \models C \sqcup D$ is not satisfiable

- This is the way used to implement reasoning services for very expressive DLs, ex. SHOIN

The SAT procedure

- For decidable DLs – as SHOIN – we can find a procedure that given an arbitrary TBox $\mathcal{T}$ and a complex term $C$ and, in a finite number of steps, states if $C$ is or not satisfiable (considering the definitions in $\mathcal{T}$)

- In the most diffuse versions this procedure, that we will call SAT, is based on the tableaux method, already studied and applied for FOL.
We will consider now reasoning services that use not only terminological axioms from TBox but also assertions from ABox.

As already noted the assertion in the ABox can be based on terms or based on roles; that is, the assertions can be in the following two forms:

- \( C(a) \)  (\( C \) complex term; \( a \) nominal)
- \( R(a, b) \)  (\( R \) role; \( a, b \) nominals)

Instance check

- given a TBox \( T \), an ABox \( A \), an arbitrary term \( C \) and a nominal \( a \), find if \( T, A \models C(a) \)

Retrieval

- given a TBox \( T \), an ABox \( A \) and an arbitrary term \( C \), among all nominals present in the KB find all nominals \( a_1, ..., a_n \) so that \( T, A \models C(a_k) \)
Reduction to satisfiability

An instance check task can be reduced to a satisfiability problem

(An arbitrary term \( C \) is satisfiable if exists at least a model of \( T,A \) where is not empty the set of individuals that satisfy \( C \), in other words \( \exists a \text{ t.c. } T,A \models C(a) \))

A retrieval task can be reduced to an instance check for each nominal in the KB

In principle, all reasoning tasks can be reduced to satisfiability problems.

Example (1)

Define the following TBox \( T \):

\[
\begin{align*}
T1. \text{PARENT} & \equiv \text{PERSON} \land \exists \text{parentOf} \\
T2. \text{parentOf: PERSON} & \rightarrow \text{PERSON}, \\
T3. \text{WOMAN} & \equiv \text{PERSON} \land \text{FEMALE} \\
T4. \text{MAN} & \equiv \text{PERSON} \land \lnot \text{FEMALE} \\
T5. \text{MOTHER} & \equiv \text{PARENT} \land \text{FEMALE} \\
T6. \text{FATHER} & \equiv \text{PARENT} \land \lnot \text{FEMALE} \\
T7. \text{STATE} & \equiv \{ \text{au, ch, de, es, fr, it, uk} \}, \\
T8. \text{citizenOf: PERSON} & \rightarrow \text{STATE}, \\
T9. \text{ITAL} & \equiv \text{PERSON} \land \exists \text{citizenOf.\{it\}}, \\
T10. \text{BRIT} & \equiv \text{PERSON} \land \exists \text{citizenOf.\{uk\}}.
\end{align*}
\]
Example (2)

Define the ABox $A$:

- A1. WOMAN(anna)
- A2. WOMAN(cecilia)
- A3. MAN(bob)
- A4. parentOf(anna, cecilia)
- A5. parentOf(bob, cecilia)
- A6. citizenOf(anna, it)
- A7. citizenOf(bob, uk)
- A8. citizenOf(cecilia, it)
- A9. citizenOf(cecilia, uk)

Example (3)

Instance check

Given term $\text{FEMALE} \sqcap \exists \text{parentOf}$ and the nominal $\text{anna}$ we have:

$?- (\text{FEMALE} \sqcap \exists \text{parentOf})(\text{anna}) \rightarrow \text{yes}$

$\text{T,A} \models (\text{FEMALE} \sqcap \exists \text{parentOf})(\text{anna})$
Example

ABox A:

A2. WOMAN(cecilia)
A3. MAN(bob)
A5. parentOf(bob, cecilia)
A6. citizenOf(anna, it)
A7. citizenOf(bob, uk)
A8. citizenOf(cecilia, it)
A9. citizenOf(cecilia, uk)

Example

TBox T:

T1. PARENT ≡ PERSON ⊓ ∃parentOf
T2. parentOf: PERSON → PERSON,
T4. MAN ≡ PERSON ⊓ ¬FEMALE
T5. MOTHER ≡ PARENT ⊓ FEMALE
T6. FATHER ≡ PARENT ⊓ ¬FEMALE
T7. STATE ≡ {au, ch, de, es, fr, it, uk},
T8. citizenOf: PERSON → STATE,
T9. ITAL ≡ PERSON ⊓ ∃citizenOf.{it},
T10. BRIT ≡ PERSON ⊓ ∃citizenOf.{uk}. 
Example

Retrieval

Given term \textbf{PARENT} we have:


\begin{align*}
\text{?-} & \text{ PARENT } \rightarrow \{ \text{anna, bob} \} \\
T,A & \models \text{ PARENT (anna)} \quad T,A & \models \text{ PARENT (bob)}
\end{align*}

Example

\textit{TBox T:}

\begin{align*}
\text{T3.} & \text{ WOMAN } \equiv \text{ PERSON } \cap \text{ FEMALE} \\
\text{T4.} & \text{ MAN } \equiv \text{ PERSON } \cap \neg \text{FEMALE} \\
\text{T5.} & \text{ MOTHER } \equiv \text{ PARENT } \cap \text{ FEMALE} \\
\text{T6.} & \text{ FATHER } \equiv \text{ PARENT } \cap \neg \text{FEMALE} \\
\text{T7.} & \text{ STATE } \equiv \{ \text{au,ch,de,es,fr,it,uk} \}, \\
\text{T8.} & \text{ citizenOf: PERSON } \rightarrow \text{ STATE}, \\
\text{T9.} & \text{ ITAL } \equiv \text{ PERSON } \cap \exists \text{ citizenOf.\{it\}}, \\
\text{T10.} & \text{ BRIT } \equiv \text{ PERSON } \cap \exists \text{ citizenOf.\{uk\}}.
\end{align*}
Example

**ABox A:**

A1. WOMAN(anna)
A2. WOMAN(cecilia)
A3. MAN(bob)

A6. citizenOf(anna,it)
A7. citizenOf(bob,uk)
A8. citizenOf(cecilia,it)
A9. citizenOf(cecilia,uk)

Tableau method

- used to decide satsifiability of a set of formula
- we start with propositional logic example:
  - prove unsatisfiability of
    \[ \{ a \land c, (\neg a \lor b) \land (\neg b \lor \neg c) \} \]
- The formula have to be in negation normal form (with not applied to the letterals)
Propositional Tableaux

- $a \land c$
- $(\neg a \lor b) \land (\neg b \lor \neg c)$
- $a$
- $c$
- $(\neg a \lor b)$
- $(\neg b \lor \neg c)$
- $\neg a$
- $b$
- $\neg b$
- $\neg c$

If all branches are closed (contain $x$ and $\neg x$) the formula is **unsatisfiable**

Tableau for ALC concepts satisfiability

- **Algorithm to check if complex concept C is satisfiable:**
  - C should be in negation normal form
  - start with C(a)
  - apply transformation rules, they can be deterministic or nondeterministic (branch)
  - continue until (i) there is a contradiction in all branches or (ii) there is a branch where no rule is applicable
  - In case (i) the concept C is **unsatisfiable**, in case (ii) C is **satisfiable**
**Rules**

- **and-rule:** \((C \land D)(a) \rightarrow \text{add } C(a) \text{ and } D(a)\)
- **or-rule:** \((C \lor D)(a) \rightarrow \text{branch with } C(a) \text{ and } D(a)\)
- **some-rule:** \((\exists R.C)(a) \rightarrow \text{add } R(a,b) \text{ and } C(b)\)
  where \(b\) is a new individual
- **all-rule:** \((\forall R.C)(a) \text{ and } R(a,b) \rightarrow \text{add } C(b)\)

**Example**

check if \(\forall \text{hasChild.Male} \land \exists \text{hasChild}. \neg \text{Male}\)

is satisfiable

1. \(\forall \text{hasChild.Male} \land \exists \text{hasChild}. \neg \text{Male}\) (given)
2. \(\forall \text{hasChild.Male}\) (1, and-rule)
3. \(\exists \text{hasChild.} \neg \text{Male}\) (1, and-rule)
4. \(\text{hasChild}(a,b)\) (3, some-rule)
5. \(\neg \text{Male}(b)\) (3, some-rule)
6. \(\text{Male}(b)\) (2,4, all-rule)
7. \(\text{Clash}\) (5,6)

the concept is **unsatisfiable**
The same rules can be used to check if the Abox is satisfiable

1. \((\text{Parent} \land \forall \text{haschild.Male})(\text{JOHN})\) (given)
2. \(\text{hasChild}(\text{JOHN}, \text{MARY})\) (given)
3. \(\neg \text{Male}(\text{Mary})\) (given)
4. \(\text{PARENT}({\text{JOHN}})\) (1, and-rule)
5. \(\forall \text{haschild.Male}({\text{JOHN}})\) (1, and-rule)
6. \(\text{Male}({\text{MARY}})\) (5,2, all-rule)
7. \(\text{Clash}\) (6,3)

The Abox is unsatisfiable

Similar rules can be applied for the satisfiability of a KB made of Tbox and Abox
Rule based reasoning

- A set of rules IF...THEN...
- Used to produce new triples on the basis of the current triples
- Applied iteratively until no more applicable or found a contradiction

<table>
<thead>
<tr>
<th>...</th>
<th>if</th>
<th>then</th>
</tr>
</thead>
<tbody>
<tr>
<td>eq-ref</td>
<td>T(?s, ?p, ?o)</td>
<td>T(?s, owl:sameAs, ?s) T(?p, owl:sameAs, ?p) T(?o, owl:sameAs, ?o)</td>
</tr>
<tr>
<td>eq-sym</td>
<td>T(?x, owl:sameAs, ?y)</td>
<td>T(?y, owl:sameAs, ?x)</td>
</tr>
<tr>
<td>eq-trans</td>
<td>T(?x, owl:sameAs, ?y) T(?y, owl:sameAs, ?z)</td>
<td>T(?x, owl:sameAs, ?z)</td>
</tr>
<tr>
<td>eq-rep-s</td>
<td>T(?s, owl:sameAs, ?s') T(?s, ?p, ?o)</td>
<td>T(?s', ?p, ?o)</td>
</tr>
<tr>
<td>eq-rep-o</td>
<td>T(?o, owl:sameAs, ?o') T(?s, ?p, ?o)</td>
<td>T(?s, ?p, ?o')</td>
</tr>
<tr>
<td>eq-diffs</td>
<td>T(?x, owl:sameAs, ?y) T(?x, owl:differentFrom, ?y)</td>
<td>False</td>
</tr>
</tbody>
</table>
### Equality (2)

<table>
<thead>
<tr>
<th>eq-diff2</th>
<th>(T(?x, \text{rdf:type, owl:AllDifferent}))</th>
<th>(T(?x, \text{owl:members}, ?y))</th>
<th>(\text{LIST}[?y, ?z_1, ..., ?z_n])</th>
<th>(T(?z_i, \text{owl:sameAs}, ?z_j))</th>
<th>false</th>
<th>for each (1 \leq i &lt; j \leq n)</th>
</tr>
</thead>
</table>

| eq-diff3 | \(T(?x, \text{rdf:type, owl:AllDifferent})\) | \(T(?x, \text{owl:distinctMembers}, ?y)\) | \(\text{LIST}[?y, ?z_1, ..., ?z_n]\) | \(T(?z_i, \text{owl:sameAs}, ?z_j)\) | false | for each \(1 \leq i < j \leq n\) |

### Properties (1)

<table>
<thead>
<tr>
<th>prp-dom</th>
<th>(T(?p, \text{rdfs:domain}, ?c))</th>
<th>(T(?x, \text{rdf:type}, ?c))</th>
</tr>
</thead>
<tbody>
<tr>
<td>prp-rng</td>
<td>(T(?p, \text{rdfs:range}, ?c))</td>
<td>(T(?y, \text{rdf:type}, ?c))</td>
</tr>
<tr>
<td>prp-fp</td>
<td>(T(?p, \text{rdf:type, owl:FunctionalProperty}))</td>
<td>(T(?y_u, \text{owl:sameAs}, ?y_u))</td>
</tr>
<tr>
<td>prp-ifp</td>
<td>(T(?p, \text{rdf:type, owl:InverseFunctionalProperty}))</td>
<td>(T(?x_u, \text{owl:sameAs}, ?x_u))</td>
</tr>
<tr>
<td>prp-irp</td>
<td>(T(?p, \text{rdf:type, owl:IrreflexiveProperty}))</td>
<td>false</td>
</tr>
<tr>
<td>prp-symp</td>
<td>(T(?p, \text{rdf:type, owl:SymmetricProperty}))</td>
<td>(T(?y, \text{?p, ?x}))</td>
</tr>
<tr>
<td>prp-asyp</td>
<td>(T(?p, \text{rdf:type, owl:AsymmetricProperty}))</td>
<td>false</td>
</tr>
</tbody>
</table>
### Properties (2)

<table>
<thead>
<tr>
<th>Property</th>
<th>Rule</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ppp-trp</td>
<td>(T(?p, \text{rdf:}\text{type}, \text{owl:TransitiveProperty}))</td>
<td>(T(?x, ?p, ?y)) (T(?y, ?p, ?z)) (T(?x, ?p, ?z))</td>
</tr>
<tr>
<td>ppp-sposs</td>
<td>(T(?p_u, \text{rdfs:subPropertyOf}, ?p_s))</td>
<td>(T(?x, ?p_u, ?y))</td>
</tr>
<tr>
<td>ppp-eqps</td>
<td>(T(?p_u, \text{owl:equivalentProperty}, ?p_s))</td>
<td>(T(?x, ?p_u, ?y))</td>
</tr>
<tr>
<td>ppp-eqps2</td>
<td>(T(?p_u, \text{owl:equivalentProperty}, ?p_s))</td>
<td>(T(?x, ?p_u, ?y))</td>
</tr>
<tr>
<td>ppp-pdw</td>
<td>(T(?p_u, \text{owl:propertyDisjointWith}, ?p_s))</td>
<td>(T(?x, ?p_u, ?y))</td>
</tr>
<tr>
<td>ppp-inv1</td>
<td>(T(?p_u, \text{owl:inverseOf}, ?p_s))</td>
<td>(T(?x, ?p_u, ?y)) (T(?y, ?p_s, ?x))</td>
</tr>
<tr>
<td>ppp-inv2</td>
<td>(T(?p_u, \text{owl:inverseOf}, ?p_s))</td>
<td>(T(?x, ?p_u, ?y)) (T(?y, ?p_s, ?x))</td>
</tr>
</tbody>
</table>

### Classes

<table>
<thead>
<tr>
<th>Class</th>
<th>Rule</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>cls-thing</td>
<td>(T(\text{owl:}\text{Thing}, \text{rdf:}\text{type}, \text{owl:Class}))</td>
<td></td>
</tr>
<tr>
<td>cls-nothing1</td>
<td>(T(\text{owl:}\text{Nothing}, \text{rdf:}\text{type}, \text{owl:Class}))</td>
<td></td>
</tr>
<tr>
<td>cls-nothing2</td>
<td>(T(?x, \text{rdf:}\text{type}, \text{owl:Nothing}))</td>
<td>(\text{false})</td>
</tr>
<tr>
<td>cax-sc0</td>
<td>(T(?c_u, \text{rdfs:subClassOf}, ?c_s))</td>
<td>(T(?x, \text{rdf:}\text{type}, ?c_s))</td>
</tr>
<tr>
<td>cax-eqc1</td>
<td>(T(?c_u, \text{owl:equivalentClass}, ?c_s))</td>
<td>(T(?x, \text{rdf:}\text{type}, ?c_s))</td>
</tr>
<tr>
<td>cax-eqc2</td>
<td>(T(?c_u, \text{owl:equivalentClass}, ?c_s))</td>
<td>(T(?x, \text{rdf:}\text{type}, ?c_s))</td>
</tr>
<tr>
<td>cax-dw</td>
<td>(T(?c_u, \text{owl:disjointWith}, ?c_s))</td>
<td>(\text{false})</td>
</tr>
</tbody>
</table>
### Classes (2)

<table>
<thead>
<tr>
<th>cls-int1</th>
<th>T(?c, owl:intersectionOf, ?x)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LIST[?x, ?c₁, ..., ?cₙ]</td>
</tr>
<tr>
<td></td>
<td>T(?y, rdf:type, ?c₂)</td>
</tr>
<tr>
<td></td>
<td>T(?y, rdf:type, ?cₙ)</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>T(?y, rdf:type, ?cₙ)</td>
</tr>
<tr>
<td></td>
<td>T(?y, rdf:type, ?c)</td>
</tr>
<tr>
<td>cls-int2</td>
<td>T(?c, owl:intersectionOf, ?x)</td>
</tr>
<tr>
<td></td>
<td>LIST[?x, ?c₁, ..., ?cₙ]</td>
</tr>
<tr>
<td></td>
<td>T(?y, rdf:type, ?c₂)</td>
</tr>
<tr>
<td></td>
<td>T(?y, rdf:type, ?cₙ)</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>T(?y, rdfetype, ?cₙ)</td>
</tr>
<tr>
<td></td>
<td>T(?y, rdf:type, ?c)</td>
</tr>
</tbody>
</table>

### Classe (2)

<table>
<thead>
<tr>
<th>cls-uni</th>
<th>T(?c, owl:unionOf, ?x)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LIST[?x, ?c₁, ..., ?cₙ]</td>
</tr>
<tr>
<td></td>
<td>T(?y, rdf:type, ?c)</td>
</tr>
<tr>
<td>cls-com</td>
<td>T(?c, owl:complementOf, ?c₂)</td>
</tr>
<tr>
<td></td>
<td>T(?x, rdf:type, ?c₂)</td>
</tr>
<tr>
<td>cls-svfs</td>
<td>T(?x, owl:someValuesFrom, ?y)</td>
</tr>
<tr>
<td></td>
<td>T(?x, owl:onProperty, ?p)</td>
</tr>
<tr>
<td></td>
<td>T(?u, ?p, ?v)</td>
</tr>
<tr>
<td></td>
<td>T(?y, rdf:type, ?y)</td>
</tr>
<tr>
<td>cls-svfa</td>
<td>T(?x, owl:someValuesFrom, owl:Thing)</td>
</tr>
<tr>
<td></td>
<td>T(?x, owl:onProperty, ?p)</td>
</tr>
<tr>
<td></td>
<td>T(?u, ?p, ?v)</td>
</tr>
<tr>
<td>cls-avf</td>
<td>T(?x, owl:allValuesFrom, ?y)</td>
</tr>
<tr>
<td></td>
<td>T(?x, owl:onProperty, ?p)</td>
</tr>
<tr>
<td></td>
<td>T(?u, rdf:type, ?y)</td>
</tr>
<tr>
<td>cls-hvs</td>
<td>T(?x, owl:hasValue, ?y)</td>
</tr>
<tr>
<td></td>
<td>T(?x, owl:onProperty, ?p)</td>
</tr>
<tr>
<td></td>
<td>T(?u, rdf:type, ?x)</td>
</tr>
<tr>
<td>cls-hvfa</td>
<td>T(?x, owl:hasValue, ?y)</td>
</tr>
<tr>
<td></td>
<td>T(?x, owl:onProperty, ?p)</td>
</tr>
<tr>
<td></td>
<td>T(?u, rdf:type, ?x)</td>
</tr>
</tbody>
</table>
### Classes (4)

<table>
<thead>
<tr>
<th>cls-maxc1</th>
<th>T(?x, owl:maxCardinality, &quot;0&quot;^^xsd:nonNegativeInteger)</th>
<th>T(?x, owl:onProperty, ?p)</th>
<th>T(?u, rdf:type, ?x)</th>
<th>T(?u, ?p, ?y)</th>
<th>false</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>T(?y, owl:sameAs, ?y)</td>
<td></td>
</tr>
<tr>
<td>cls-maxc1</td>
<td>T(?x, owl:maxQualifiedCardinality, &quot;0&quot;^^xsd:nonNegative...)</td>
<td>T(?x, owl:onProperty, ?p)</td>
<td>T(?x, owl:onClass, ?c)</td>
<td>T(?u, rdf:type, ?x)</td>
<td>T(?u, ?p, ?y)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>T(?u, ?p, ?y)</td>
<td></td>
<td>T(?y, rdf:type, ?c)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>T(?y, owl:sameAs, ?y)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>T(?u, ?p, ?y)</td>
<td></td>
<td>T(?y, rdf:type, ?c)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>T(?y, owl:sameAs, ?y)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>T(?y, owl:sameAs, ?y)</td>
<td></td>
</tr>
</tbody>
</table>

### Classes (5)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>cls-oo</td>
<td>T(?c, owl:oneOf, ?x)</td>
<td>LIST[?x, ?y, ..., ?y]</td>
<td>T(?y, rdf:type, ?c)</td>
<td>...</td>
<td>T(?y, rdf:type, ?c)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Vocabulary (1)

<table>
<thead>
<tr>
<th>scm-rdfs</th>
<th>T((?c, \text{rdf:type, owl:Class}))</th>
<th>T((?c, \text{rdfs:subClassOf, } ?c)) T((?c, \text{owl:equivalentClass, } ?c)) T((?c, \text{rdfs:subClassOf, owl:Thing}) T((\text{owl:Nothing, rdfs:subClassOf, } ?c))</th>
</tr>
</thead>
<tbody>
<tr>
<td>scm-sco</td>
<td>T((?c_1, \text{rdfs:subClassOf, } ?c_2)) T((?c_2, \text{rdfs:subClassOf, } ?c_3))</td>
<td>T((?c_2, \text{rdfs:subClassOf, } ?c_2))</td>
</tr>
<tr>
<td>scm-eqcl</td>
<td>T((?c_1, \text{owl:equivalentClass, } ?c_1))</td>
<td>T((?c_1, \text{rdfs:subClassOf, } ?c_2))</td>
</tr>
<tr>
<td>scm-eqcl</td>
<td>T((?c_1, \text{rdfs:subClassOf, } ?c_2)) T((?c_2, \text{rdfs:subClassOf, } ?c_2))</td>
<td>T((?c_1, \text{owl:equivalentClass, } ?c_2)) T((?c_2, \text{owl:equivalentClass, } ?c_2))</td>
</tr>
<tr>
<td>scm-op</td>
<td>T((?p, \text{rdf:type, owl:ObjectProperty}))</td>
<td>T((?p, \text{rdfs:subPropertyOf, } ?p)) T((?p, \text{owl:equivalentProperty, } ?p))</td>
</tr>
<tr>
<td>scm-dp</td>
<td>T((?p, \text{rdf:type, owl:DatatypeProperty}))</td>
<td>T((?p, \text{rdfs:subPropertyOf, } ?p)) T((?p, \text{owl:equivalentProperty, } ?p))</td>
</tr>
<tr>
<td>scm-spo</td>
<td>T((?p_1, \text{rdfs:subPropertyOf, } ?p_2)) T((?p_2, \text{rdfs:subPropertyOf, } ?p_3))</td>
<td>T((?p_1, \text{rdfs:subPropertyOf, } ?p_2))</td>
</tr>
</tbody>
</table>

### Vocabulary (2)

<table>
<thead>
<tr>
<th>scm-eqpl</th>
<th>T((?p_1, \text{owl:equivalentProperty, } ?p_2))</th>
<th>T((?p_1, \text{rdfs:subPropertyOf, } ?p_2)) T((?p_1, \text{rdfs:subPropertyOf, } ?p_2))</th>
</tr>
</thead>
<tbody>
<tr>
<td>scm-eqpl</td>
<td>T((?p_1, \text{rdfs:subPropertyOf, } ?p_2)) T((?p_2, \text{rdfs:subPropertyOf, } ?p_2))</td>
<td>T((?p_1, \text{owl:equivalentProperty, } ?p_2))</td>
</tr>
<tr>
<td>scm-domd</td>
<td>T((?p, \text{rdfs:domain, } ?c_1)) T((??c_1, \text{rdfs:subClassOf, } ?c_2))</td>
<td>T((?p, \text{rdfs:domain, } ?c_2))</td>
</tr>
<tr>
<td>scm-domd</td>
<td>T((?p_2, \text{rdfs:domain, } ?c)) T((??c_1, \text{rdfs:subPropertyOf, } ?p_2))</td>
<td>T((?p_2, \text{rdfs:domain, } ?c))</td>
</tr>
<tr>
<td>scm-rngs</td>
<td>T((?p, \text{rdfs:range, } ?c_1)) T((??c_1, \text{rdfs:subClassOf, } ?c_2))</td>
<td>T((?p, \text{rdfs:range, } ?c_2))</td>
</tr>
<tr>
<td>scm-rngs</td>
<td>T((?p, \text{rdfs:range, } ?c)) T((??c_1, \text{rdfs:subPropertyOf, } ?p_2))</td>
<td>T((?p, \text{rdfs:range, } ?c))</td>
</tr>
</tbody>
</table>
OWL2 profile EL

- is particularly suitable for applications employing ontologies that define very large numbers of classes and/or properties (e.g. SNOMED-CT medical ontology with about 292,000 logical axioms),
- captures the expressive power used by many such ontologies, and consistency, class expression subsumption, and instance checking can be decided in polynomial time
- Allows operations:
  - ∃R.C, ∃R.{v}, ∃R. Self, {v}, C ⊓ D
  - class inclusion, class equivalence, class disjointness, object property inclusion with or without property chains, property equivalence, transitive object properties, reflexive object properties, domain restrictions, range restrictions, functional data properties, assertions, keys.

OWL2 profile QL

- designed so that data (assertions) that is stored in a relational database system can be queried through an ontology by rewriting the query into an SQL query, without any changes to the data.
- Allowed
  - <subclass expression> subClassOf <super class expression>
  - where <subclass expressions> can be:
    - a class, unqualified existential quantification, existential quantification to a data range.
  - and <super class expression> can be:
    - a class, intersection, negation, qualified existential quantification, existential quantification to a data range
  - subclass axioms, class expression equivalence, class expression disjointness, inverse object properties, property inclusion (not involving property chains), property equivalence, property domain, property range, disjoint properties, symmetric properties, reflexive properties, irreflexive properties, asymmetric properties, assertions other than individual equality assertions and negative property assertions
OWL2 profile RL

- aimed at applications that require scalable reasoning without sacrificing too much expressive power
- Allowed
  - `<subclass expression> subClassOf <super class expression>`
  - where `<subclass expression>` can be:
    - a class other than `owl:Thing`, an enumeration of individuals, intersection of class expressions, union of class expressions, existential quantification to a class expression, existential quantification to a data range, existential quantification to an individual, existential quantification to a literal.
  - and `<superclass expression>` can be:
    - a class other than `owl:Thing`, intersection of classes, negation, universal quantification to a class expression, existential quantification to an individual, at-most 0/1 cardinality restriction to a class expression, universal quantification to a data range, existential quantification to a literal, at-most 0/1 cardinality restriction to a data range

W3C RIF – Rule Interchange Format

- allows to represent additional inference rules that are specific for a domain and cannot be derived with OWL

```
Document(
  Prefix(rdfs <http://www.w3.org/2000/01/rdf-schema#>)
  Prefix(imdbrel <http://example.com/imdbrelations#>)
  Prefix(dbpedia <http://dbpedia.org/ontology/>)
  Group( Forall ?Actor ?Film ?Role ( 
      If And(imdbrel:playsRole(?Actor ?Role) imdbrel:roleInFilm(?Role ?Film))
      Then dbpedia:starring(?Film ?Actor) 
    )
  )
)
```
RIF

- RIF was designed for interchange, to allow the transformation of rules in other languages (e.g. SWRL, RuleML)