Reasoning

Reasoning

- 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 = ‹T,A›

- 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 ⊨ ***X***

KB logically imply ***X*** *(**X** is a logical consequence of* **KB***)*

Logic consequence (2)

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

```
T1. PARENT ≡ PERSON ⊓ ∃parentOf
T2. parentOf:PERSON → PERSON
T3. WOMAN ≡ PERSON ⊓ FEMALE
T4. MAN ≡ PERSON ⊓ ¬FEMALE
T5. MOTHER ≡ PARENT ⊓ FEMALE
T6. FATHER ≡ PARENT ⊓ ¬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.

Logic consequence (3)

- Every mother is a person and a woman:

  $\mathrm{MOTHER} \sqsubseteq \mathrm{PERSON}$

  $\mathrm{MOTHER} \sqsubseteq \mathrm{WOMAN}$

- Every father is a person and a man:

  $\mathrm{FATHER} \sqsubseteq \mathrm{PERSON}$

  $\mathrm{FATHER} \sqsubseteq \mathrm{MAN}$

- Class of fathers and mothers are disjoint:

  $\mathrm{MOTHER} \sqcap \mathrm{FATHER} \equiv \bot$

Logic consequence (4)

- To highlight that these statements are logic consequence of **T** we write:

  $\mathbf{T} \models \mathrm{MOTHER} \sqsubseteq \mathrm{PERSON}$

- 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:

  $\mathbf{T} \not\models \mathrm{PERSON} \sqsubseteq \mathtt{=2\ parentOf\text{-}}$

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*

  $T \vDash C \equiv D$ is equivalent to $T \vDash C \sqsubseteq D$ e $T \vDash D \sqsubseteq C$
- *Soddisfacibility*

  $T \nvDash C \sqsubseteq \bot$
- *Disjunction*

  $T \vDash C \sqcap D \sqsubseteq \bot$
- This the way used to implement reasoning services for low expressive DLs

Reduction to satisfiability

- The fundamental reasoning tasks for *Tboxs can be reduced to satisfiability*
- *Subsumption* $\mathbf{T} \vDash C \sqsubseteq D$

  $\mathbf{T} \vDash C \sqcap \neg D$ `is not satisfiable`
- *Equivalence* $\mathbf{T} \vDash C \equiv D$

  $\mathbf{T} \vDash C \sqcap \neg D$ `is not satisfiable` and

  $\mathbf{T} \vDash \neg C \sqcap D$ `is not satisfiable`
- *Disjunction*

  $\mathbf{T} \vDash C \sqcap D$ `is not satisfiable`
- This is the way used to implement reasoning services for **very expressive DLs**, ex. $\mathcal{SHOIN}$

The SAT procedure

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

Reasoning tasks (ABox)

- 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)*

Reasoning tasks (ABox)

Instance check

given a *TBox* **T**, an *ABox* **A**, an arbitrary term C and a nominal a, find if **T**,**A** ⊨ 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** ⊨ C(a_k)

Reduction to satisfiability

An ***instance check*** task can be reduced to a **statisfiability 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 ∃ a t.c. T,A ⊨ 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**:

```
T1. PARENT ≡ PERSON ⊓ ∃parentOf
T2. parentOf: PERSON → PERSON,
T3. WOMAN ≡ PERSON ⊓ FEMALE
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 (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 **FEMALE** ⊓ ∃parentOf and the nominal anna we have:

?- (**FEMALE** ⊓ ∃parentOf)(anna) → yes

T,A ⊨ (**FEMALE** ⊓ ∃parentOf)(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 PARENT we have:

?- PARENT → {anna, bob}

T,A ⊨ PARENT(anna) **T,A** ⊨ PARENT(bob)

Example

TBox **T**:

T3. WOMAN ≡ PERSON ⊓ FEMALE
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

ABox **A**:

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

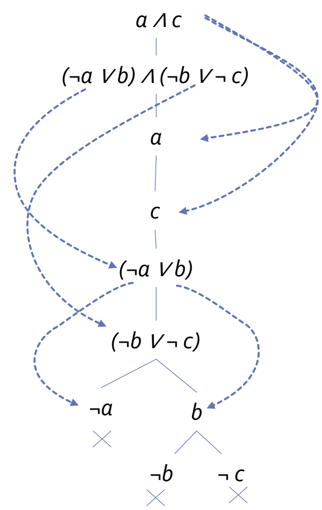
```
A6. citizenOf(anna,it)
A7. citizenOf(bob,uk)
A8. citizenOf(cecilia,it)
A9. citizenOf(cecilia,uk)
```

Tableau method

- used to decide sasfiability of a set of formula
- we start with propositional logic example:
  - prove unsatisfiability of

    $\{ a \wedge c, (\neg a \vee b) \wedge (\neg b \vee \neg c) \}$
- The formula have to be in negation normal form (with not applied to the letterals)

Propositional Tableaux



if all branches are closed (contain x and ¬ 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 ⊓ D)(a) → add C(a) and D(a)
- or-rule: (C ⊔ D)(a) → branch with C(a) and D(a)
- some-rule: (∃R.C)(a) → add R(a,b) and C(b) where b is a new individual
- all-rule: (∀R.C)(a) and R(a,b) → add C(b)

Example

check if **∀hasChild.Male ⊓ ∃hasChild. ¬Male** is satisfiable

1.	∀hasChild.Male ⊓ ∃hasChild.¬Male	(given)
2.	∀hasChild.Male	(1, and-rule)
3.	∃hasChild. ¬Male	(1, and-rule)
4.	hasChild(a,b)	(3, some-rule)
5.	(¬Male)(b)	(3, some-rule)
6.	Male(b)	(2,4, all-rule)
7.	Clash	(5,6)

the concept is ***unsatisfiable***

Tableau for ABox

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

1.	**(Parent ⊓ ∀ haschild.Male)(JOHN)**	**(given)**
2.	**hasChild(JOHN, MARY)**	**(given)**
3.	**(¬Male)(Mary)**	**(given)**
4.	PARENT(JOHN)	(1, and-rule)
5.	∀ haschild.Male(JOHN)	(1, and-rule)
6.	Male(MARY)	(5,2, all-rule)
7.	**Clash**	(6,3)

The Abox is unsatisfiable

Tableau for KB

- 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

Equality (1)

...	If	then
eq-ref	T(?s, ?p, ?o)	T(?s, owl:sameAs, ?s) T(?p, owl:sameAs, ?p) T(?o, owl:sameAs, ?o)
eq-sym	T(?x, owl:sameAs, ?y)	T(?y, owl:sameAs, ?x)
eq-trans	T(?x, owl:sameAs, ?y) T(?y, owl:sameAs, ?z)	T(?x, owl:sameAs, ?z)
eq-rep-s	T(?s, owl:sameAs, ?s') T(?s, ?p, ?o)	T(?s', ?p, ?o)
eq-rep-p	T(?p, owl:sameAs, ?p') T(?s, ?p, ?o)	T(?s, ?p', ?o)
eq-rep-o	T(?o, owl:sameAs, ?o') T(?s, ?p, ?o)	T(?s, ?p, ?o')
eq-diff1	T(?x, owl:sameAs, ?y) T(?x, owl:differentFrom, ?y)	False

Equality (2)

eq-diff2	T(?x, rdf:type, owl:AllDifferent) T(?x, owl:members, ?y) LIST[?y, $?z_1$, ..., $?z_n$] T($?z_i$, owl:sameAs, $?z_j$)	false	for each $1 \le i < j \le n$
eq-diff3	T(?x, rdf:type, owl:AllDifferent) T(?x, owl:distinctMembers, ?y) LIST[?y, $?z_1$, ..., $?z_n$] T($?z_i$, owl:sameAs, $?z_j$)	false	for each $1 \le i < j \le n$

Properties (1)

prp-dom	T(?p, rdfs:domain, ?c) T(?x, ?p, ?y)	T(?x, rdf:type, ?c)
prp-rng	T(?p, rdfs:range, ?c) T(?x, ?p, ?y)	T(?y, rdf:type, ?c)
prp-fp	T(?p, rdf:type, owl:FunctionalProperty) T(?x, ?p, $?y_1$) T(?x, ?p, $?y_2$)	T($?y_1$, owl:sameAs, $?y_2$)
prp-ifp	T(?p, rdf:type, owl:InverseFunctionalProperty) T($?x_1$, ?p, ?y) T($?x_2$, ?p, ?y)	T($?x_1$, owl:sameAs, $?x_2$)
prp-irp	T(?p, rdf:type, owl:IrreflexiveProperty) T(?x, ?p, ?x)	false
prp-symp	T(?p, rdf:type, owl:SymmetricProperty) T(?x, ?p, ?y)	T(?y, ?p, ?x)
prp-asyp	T(?p, rdf:type, owl:AsymmetricProperty) T(?x, ?p, ?y) T(?y, ?p, ?x)	false

Properties (2)

prp-trp	T(?p, rdf:type, owl:TransitiveProperty) T(?x, ?p, ?y) T(?y, ?p, ?z)	T(?x, ?p, ?z)
prp-spo1	T(?p_1, rdfs:subPropertyOf, ?p_2) T(?x, ?p_1, ?y)	T(?x, ?p_2, ?y)
prp-eqp1	T(?p_1, owl:equivalentProperty, ?p_2) T(?x, ?p_1, ?y)	T(?x, ?p_2, ?y)
prp-eqp2	T(?p_1, owl:equivalentProperty, ?p_2) T(?x, ?p_2, ?y)	T(?x, ?p_1, ?y)
prp-pdw	T(?p_1, owl:propertyDisjointWith, ?p_2) T(?x, ?p_1, ?y) T(?x, ?p_2, ?y)	false
prp-inv1	T(?p_1, owl:inverseOf, ?p_2) T(?x, ?p_1, ?y)	T(?y, ?p_2, ?x)
prp-inv2	T(?p_1, owl:inverseOf, ?p_2) T(?x, ?p_2, ?y)	T(?y, ?p_1, ?x)

Classes

cls-thing		T(owl:Thing, rdf:type, owl:Class)
cls-nothing1		T(owl:Nothing, rdf:type, owl:Class)
cls-nothing2	T(?x, rdf:type, owl:Nothing)	false

cax-sco	T(?c_1, rdfs:subClassOf, ?c_2) T(?x, rdf:type, ?c_1)	T(?x, rdf:type, ?c_2)
cax-eqc1	T(?c_1, owl:equivalentClass, ?c_2) T(?x, rdf:type, ?c_1)	T(?x, rdf:type, ?c_2)
cax-eqc2	T(?c_1, owl:equivalentClass, ?c_2) T(?x, rdf:type, ?c_2)	T(?x, rdf:type, ?c_1)
cax-dw	T(?c_1, owl:disjointWith, ?c_2) T(?x, rdf:type, ?c_1) T(?x, rdf:type, ?c_2)	false

Classes (2)

cls-int1	T(?c, owl:intersectionOf, ?x) LIST[?x, $?c_1$, ..., $?c_n$] T(?y, rdf:type, $?c_1$) T(?y, rdf:type, $?c_2$) ... T(?y, rdf:type, $?c_n$)	T(?y, rdf:type, ?c)
cls-int2	T(?c, owl:intersectionOf, ?x) LIST[?x, $?c_1$, ..., $?c_n$] T(?y, rdf:type, ?c)	T(?y, rdf:type, $?c_1$) T(?y, rdf:type, $?c_2$) ... T(?y, rdf:type, $?c_n$)

Classe (2)

cls-uni	T(?c, owl:unionOf, ?x) LIST[?x, $?c_1$, ..., $?c_n$] T(?y, rdf:type, $?c_i$)	T(?y, rdf:type, ?c)
cls-com	T($?c_1$, owl:complementOf, $?c_2$) T(?x, rdf:type, $?c_1$) T(?x, rdf:type, $?c_2$)	false
cls-svf1	T(?x, owl:someValuesFrom, ?y) T(?x, owl:onProperty, ?p) T(?u, ?p, ?v) T(?v, rdf:type, ?y)	T(?u, rdf:type, ?x)
cls-svf2	T(?x, owl:someValuesFrom, owl:Thing) T(?x, owl:onProperty, ?p) T(?u, ?p, ?v)	T(?u, rdf:type, ?x)
cls-avf	T(?x, owl:allValuesFrom, ?y) T(?x, owl:onProperty, ?p) T(?u, rdf:type, ?x) T(?u, ?p, ?v)	T(?v, rdf:type, ?y)
cls-hv1	T(?x, owl:hasValue, ?y) T(?x, owl:onProperty, ?p) T(?u, rdf:type, ?x)	T(?u, ?p, ?y)
cls-hv2	T(?x, owl:hasValue, ?y) T(?x, owl:onProperty, ?p) T(?u, ?p, ?y)	T(?u, rdf:type, ?x)

Classes (4)

ls-maxc1	T(?x, owl:maxCardinality, "0"^^xsd:nonNegativeInteger) T(?x, owl:onProperty, ?p) T(?u, rdf:type, ?x) T(?u, ?p, ?y)	false
cls-maxc2	T(?x, owl:maxCardinality, "1"^^xsd:nonNegativeInteger) T(?x, owl:onProperty, ?p) T(?u, rdf:type, ?x) T(?u, ?p, $?y_1$) T(?u, ?p, $?y_2$)	T($?y_1$, owl:sameAs, $?y_2$)
cls-maxqc1	T(?x, owl:maxQualifiedCardinality, "0"^^xsd:nonNegati..) T(?x, owl:onProperty, ?p) T(?x, owl:onClass, ?c) T(?u, rdf:type, ?x) T(?u, ?p, ?y) T(?y, rdf:type, ?c)	false
cls-maxqc2	T(?x, owl:maxQualifiedCardinality, "0"^^xsd:nonNegativ..) T(?x, owl:onProperty, ?p) T(?x, owl:onClass, owl:Thing) T(?u, rdf:type, ?x) T(?u, ?p, ?y)	false

Classes (5)

s-maxqc3	T(?x, owl:maxQualifiedCardinality, "1"...) T(?x, owl:onProperty, ?p) T(?x, owl:onClass, ?c) T(?u, rdf:type, ?x) T(?u, ?p, $?y_1$) T($?y_1$, rdf:type, ?c) T(?u, ?p, $?y_2$) T($?y_2$, rdf:type, ?c)	T($?y_1$, owl:sameAs, $?y_2$)
cls-maxqc4	T(?x, owl:maxQualifiedCardinality, "1"...) T(?x, owl:onProperty, ?p) T(?x, owl:onClass, owl:Thing) T(?u, rdf:type, ?x) T(?u, ?p, $?y_1$) T(?u, ?p, $?y_2$)	T($?y_1$, owl:sameAs, $?y_2$)
cls-oo	T(?c, owl:oneOf, ?x) LIST[?x, $?y_1$, ..., $?y_n$]	T($?y_1$, rdf:type, ?c) ... T($?y_n$, rdf:type, ?c)

Vocabulary (1)

scm-cls	T(?c, rdf:type, owl:Class)	T(?c, rdfs:subClassOf, ?c) T(?c, owl:equivalentClass, ?c) T(?c, rdfs:subClassOf, owl:Thing) T(owl:Nothing, rdfs:subClassOf, ?c)
scm-sco	T($?c_1$, rdfs:subClassOf, $?c_2$) T($?c_2$, rdfs:subClassOf, $?c_3$)	T($?c_1$, rdfs:subClassOf, $?c_3$)
scm-eqc1	T($?c_1$, owl:equivalentClass, $?c_2$)	T($?c_1$, rdfs:subClassOf, $?c_2$) T($?c_2$, rdfs:subClassOf, $?c_1$)
scm-eqc2	T($?c_1$, rdfs:subClassOf, $?c_2$) T($?c_2$, rdfs:subClassOf, $?c_1$)	T($?c_1$, owl:equivalentClass, $?c_2$)
scm-op	T(?p, rdf:type, owl:ObjectProperty)	T(?p, rdfs:subPropertyOf, ?p) T(?p, owl:equivalentProperty, ?p)
scm-dp	T(?p, rdf:type, owl:DatatypeProperty)	T(?p, rdfs:subPropertyOf, ?p) T(?p, owl:equivalentProperty, ?p)
scm-spo	T($?p_1$, rdfs:subPropertyOf, $?p_2$) T($?p_2$, rdfs:subPropertyOf, $?p_3$)	T($?p_1$, rdfs:subPropertyOf, $?p_3$)

Vocabulary (2)

scm-eqp1	T($?p_1$, owl:equivalentProperty, $?p_2$)	T($?p_1$, rdfs:subPropertyOf, $?p_2$) T($?p_2$, rdfs:subPropertyOf, $?p_1$)
scm-eqp2	T($?p_1$, rdfs:subPropertyOf, $?p_2$) T($?p_2$, rdfs:subPropertyOf, $?p_1$)	T($?p_1$, owl:equivalentProperty, $?p_2$)
scm-dom1	T(?p, rdfs:domain, $?c_1$) T($?c_1$, rdfs:subClassOf, $?c_2$)	T(?p, rdfs:domain, $?c_2$)
scm-dom2	T($?p_2$, rdfs:domain, ?c) T($?p_1$, rdfs:subPropertyOf, $?p_2$)	T($?p_1$, rdfs:domain, ?c)
scm-rng1	T(?p, rdfs:range, $?c_1$) T($?c_1$, rdfs:subClassOf, $?c_2$)	T(?p, rdfs:range, $?c_2$)
scm-rng2	T($?p_2$, rdfs:range, ?c) T($?p_1$, rdfs:subPropertyOf, $?p_2$)	T($?p_1$, rdfs:range, ?c)

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:
  - $\exists R.C$, $\exists R.\{v\}$, $\exists R.\ Self$, $\{v\}$, $C \sqcap D$
  - 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)

W3C Semantic Web Stack