M2 – DAC - LODAS - 2019

MU5IN860 Linked Open Data, Apprentisssage Symbolique

Bernd Amann

SU

8 octobre 2019

Règles et OWL : Inférence

Outline

1 - Inférence et Règles RDF

- Règles RDF : Syntaxe
- Règles RDF : Inférence

Règles RDF

Objectif

Etant donné un graphe RDF (ensembles de triplets), on veut inférer des nouvelles connaissances (triplets).

Exemple

Toutes les personnes qui ont plus que 18 ans sont des adultes.

```
ex:p1  ex:age "20"; ex:name "Susan" .
ex:p4  ex:age "15"; ex:name "Tim" .
```

Règle adultes

```
@prefix ex: http://www.asws.com/social#
[adultes:  (?s ex:age ?a) ge(?a,18) -> (?s rdf:type ex:Adult) ]
```

produit un nouveau triplet :

```
ex:p1  rdf:type ex:Adult .
```

Jena : Types de règles

règle	type de règle	LHS/RHS
$[nom : LHS < -RHS]$	backward	LHS : atom simple sans fonction RHS : conjonction d'atomes avec fonctions
$[nom : LHS- > RHS]$	forward	LHS : conjonction d'atomes avec fonctions RHS : conjonction d'atomes avec fonctions
$[nom : LHS- > [nom1 : LHS1 < -RHS1]...)$	hybrid	LHS : conjonction d'atomes avec fonctions RHS : une ou plusieurs règles backward

Règle friendsforward

```
@prefix rdf:   <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix ex: http://www.asws.com/social#
[friendsforward: (?s ex:friend ?a) (?a ex:friend ?b) (?b ex:age ?x) ge(?x,18) notEqual(?s,?b)
              -> (?s ex:friend ?b) ]
```

Règle friendsbackward

```
@prefix rdf:   <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix ex: http://www.asws.com/social#
[friendsbackward: (?s ex:friend ?b)
                  <- (?s ex:friend ?a) (?a ex:friend ?b) (?b ex:age ?x) ge(?x,18) ]
```

Règle friendshybrid

```
@prefix rdf:   <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix ex: http://www.asws.com/social#
[friendshybrid:  (?s ex:friend ?a) (?a ex:friend ?b) notEqual(?s,?b)
                -> [(?s ex:friend ?b) <- (?b ex:age ?x) ge(?x,18) ] ]
```

Jena Rules : Syntaxe Abstraite

```
Rule       :=   bare-rule .
           or   [ bare-rule ]
           or   [ ruleName : bare-rule ]

bare-rule  :=   term, ... term -> hterm, ... hterm     // forward rule
           or   term, ... term <- term, ... term       // backward rule

hterm      :=   term
           or   [ bare-rule ]

term       :=   (node, node, node)            // triple pattern
           or   (node, node, functor)         // extended triple pattern
           or   builtin(node, ... node)       // procedural primitive

functor    :=   functorName(node, ... node)   // structured literal

node       :=   uri-ref                       // e.g. http://foo.com/eg
           or   prefix:localname              // e.g. rdf:type
           or   <uri-ref>                     // e.g. <myscheme:myuri>
           or   ?varname                      // variable
           or   'a literal'                   // a plain string literal
           or   'lex'^^typeURI                // a typed literal
           or   number                        // e.g. 42 or 25.5
```

Jena : Fonctions et prédicats

`jena.apache.org/documentation/inference/#RULEbuiltins`

- isLiteral(?x), notLiteral(?x), isBNode(?x), notBNode(?x)
- bound(?x, ...), unbound(?x, ...)
- equal(?x, ?y) , notEqual(?x, ?y), lessThan(?x, ?y), greaterThan(?x, ?y), le(?x, ?y), ge(?x, ?y)
- sum(?a, ?b, ?c), difference(?a, ?b, ?c)
- strConcat(?a1, ..., ?an, ?t), uriConcat(?a1, ..., ?an, ?t)
- now(?x)
- noValue(?x, ?p), noValue(?x ?p ?v)
- remove(n, ...), drop(n, ...)
- makeSkolem(?x, ?v1, ..., ?vn)

Exemple : Fonction Skolem

Turtle test

```
@prefix ex:     <http://www.asws.com/social#> .
@prefix rdf:    <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
[] a rdf:Resource; ex:name "Tom" .
```

Règle skolem

```
@prefix ex: http://www.asws.com/social#

makeSkolem(?s, "Susan") -> (?s rdf:type rdf:Resource) (?s ex:nom "Susan") .
makeSkolem(?m, "Mary") (?t ex:name "Tom") -> (?m ex:nom "Mary")
                                             (?t ex:friend ?m) .
makeSkolem(?s, "Susan") (?t ex:name "Tom") -> (?t ex:friend ?s) .
```

Résultat de skolem

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix ex:   <http://www.asws.com/social#> .
@prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

[ a        rdf:Resource ;
  ex:friend  [ a      rdf:Resource ;
               ex:nom  "Susan"
             ] ;
  ex:friend  [ ex:nom  "Mary" ] ;
  ex:name    "Tom"
] .
```

Exemple : Noeuds Blancs

Règle blanknode

```
@prefix ex: http://www.asws.com/social#

(?s rdf:type ex:BlankNode) <- (?s rdf:type rdf:Resource) isBNode(?s) .
```

Résultat de blanknode

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix ex:   <http://www.asws.com/social#> .
@prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

[ a         rdf:Resource , ex:BlankNode ;
  ex:friend  [ ex:nom  "Mary" ] ;
  ex:friend  [ a      rdf:Resource , ex:BlankNode ;
               ex:nom  "Susan"
             ] ;
  ex:name    "Tom"
] .
```

Exemple : Suppression de triplets

Règle remove

```
@prefix ex: http://www.asws.com/social#

(?s rdf:type rdf:Resource) (?s ex:nom "Susan") -> remove(0) .
```

Résultat de remove

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix ex:   <http://www.asws.com/social#> .
@prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

[ a         rdf:Resource ;
  ex:friend  [ ex:nom  "Mary" ] ;
  ex:friend  [ ex:nom  "Susan" ] ;
  ex:name    "Tom"
] .
```

Règles avec **suppression** :

- la suppression de triplets introduit une forme de négation
- logique non-monotone : il existe plusieurs solutions (modèles)

1 - Inférence et Règles RDF

- Règles RDF : Syntaxe
- Règles RDF : Inférence

Jena : Types d'Inférence

Inférence backward chaining

- inférence Programmation Logique (PL)
- résolution SLD : Datalog sans négation sur une table *triplet*(s, p, o)
- tabling : matérialisation de résultats intermédiaires (memoizing)
- *résultat virtuel* : matérialisation dynamique au moment où les données sont accédées (requêtes)

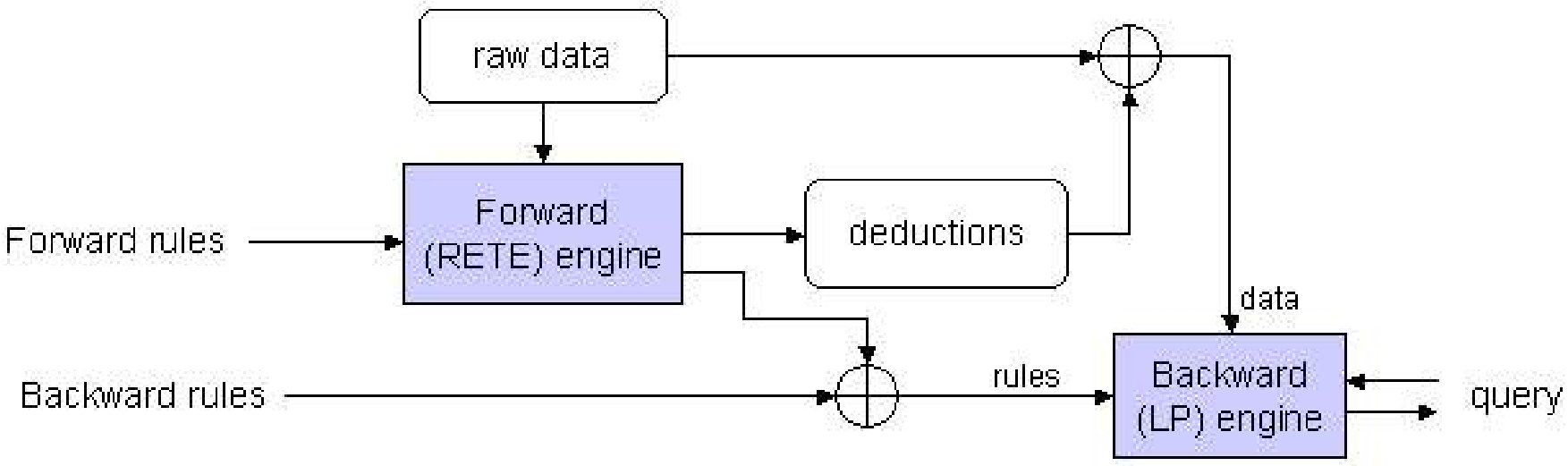
Inférence forward chaining

- algorithme RETE (moteur par défaut) :
  - graphe de déduction
  - déclenchement de règles non-deterministe
- *peut* aussi évaluer des règles *backward* (sans propriétés symétriques et reflexives)
- *résultat matérialisé :* matérialisation statique produisant l'union du graphe d'entrée et du graphe inféré

Inférence hybride

- forward chaining génère des règles backward
- permet d'équilibrer le coût de matérialisation (forward) et d'évaluation dynamique (backward)

Jena : Inférence hybride



- forward engine (RETE) : génération de faits et de règles (avec des variables) instanciées
- backward engine (LP) : évaluation des règles instanciés

Exemple : Forward et Backward Chaining

Règle friendssymforward

```
@prefix rdf:   <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix ex: http://www.asws.com/social#

[friendssymmforward: (?s ex:friend ?a) -> (?a ex:friend ?s) ]
```

- backward chaining : impossible
- forward chaining : génération de graphe de déduction et calcul de point fixe.

Règle friendssymbackward

```
@prefix rdf:   <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix ex: http://www.asws.com/social#

[friendssymmbackward: (?s ex:friend ?a) <- (?a ex:friend ?s) ]
```

- backward chaining : évaluation récursive (résolution SLD)
- forward chaining : possible, mais boucle infinie sur la relation symétrique `ex:friend` (propriété de l'algorithme RETE)

Exemple : Graphe Social

Turtle sn

```
@prefix rdfs:  <http://www.w3.org/2000/01/rdf-schema#> .
@prefix ex:    <http://www.asws.com/social#> .
@prefix rdf:   <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

ex:user3  a         ex:User ;
        ex:age      "35"^^<http://www.w3.org/2001/XMLSchema#int> ;
        ex:friend   ex:user4 ;
        ex:name     "Tom" .

ex:user1  a         ex:User ;
        ex:age      "20"^^<http://www.w3.org/2001/XMLSchema#int> ;
        ex:friend   ex:user2 , ex:user3 ;
        ex:name     "Susan" .

ex:user4  a         ex:User ;
        ex:age      "15"^^<http://www.w3.org/2001/XMLSchema#int> ;
        ex:friend   ex:user2 ;
        ex:name     "Tim" .

ex:user2  a         ex:User ;
        ex:age      "40"^^<http://www.w3.org/2001/XMLSchema#int> ;
        ex:friend   ex:user1 ;
        ex:name     "Mike" .

ex:user5  a         ex:User ;
        ex:age      "25"^^<http://www.w3.org/2001/XMLSchema#int> ;
        ex:friend   ex:user2 ;
        ex:name     "Mary" .
```

Règle forward

Résultat de friendssymforward

```
@prefix rdfs:  <http://www.w3.org/2000/01/rdf-schema#> .
@prefix ex:    <http://www.asws.com/social#> .
@prefix rdf:   <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

ex:user4  a       ex:User ;
      ex:age    "15"^^<http://www.w3.org/2001/XMLSchema#int> ;
      ex:friend  ex:user3 , ex:user2 ;
      ex:name    "Tim" .

ex:user3  a       ex:User ;
      ex:age    "35"^^<http://www.w3.org/2001/XMLSchema#int> ;
      ex:friend  ex:user1 , ex:user4 ;
      ex:name    "Tom" .

ex:user2  a       ex:User ;
      ex:age    "40"^^<http://www.w3.org/2001/XMLSchema#int> ;
      ex:friend  ex:user5 , ex:user4 , ex:user1 ;
      ex:name    "Mike" .

ex:user1  a       ex:User ;
      ex:age    "20"^^<http://www.w3.org/2001/XMLSchema#int> ;
      ex:friend  ex:user3 , ex:user2 ;
      ex:name    "Susan" .

ex:user5  a       ex:User ;
      ex:age    "25"^^<http://www.w3.org/2001/XMLSchema#int> ;
      ex:friend  ex:user2 ;
      ex:name    "Mary" .
```

Exemple : Backward, Forward, Hybrid

Règle backwardsub

```
@prefix rdfs: http://www.w3.org/2000/01/rdf-schema#
[backwardsub: (?a ?q ?b) <- (?p rdfs:subPropertyOf ?q)
                            notEqual(?p,?q) (?a ?p ?b) ]
```

→ foward chaining (avec RETE) ne termine pas (`rdfs:subProperty` est reflexive)
→ backward chaining est inefficace

Règle forwardsub

```
@prefix rdfs: http://www.w3.org/2000/01/rdf-schema#
[forwardsub: (?p rdfs:subPropertyOf ?q)
             notEqual(?p,?q) (?a ?p ?b) -> (?a ?q ?b) ]
```

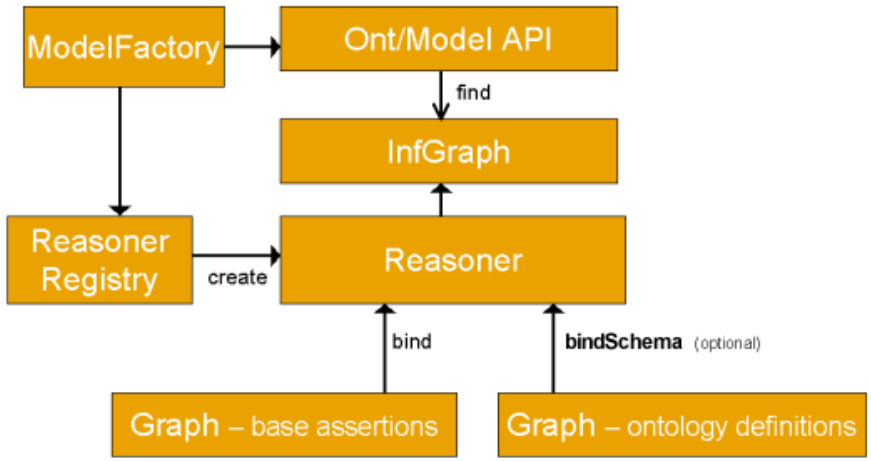
→ forward chaining génère beaucoup de triplets ...

Règle hybridsub

```
@prefix rdfs: http://www.w3.org/1999/02/22-rdf-syntax-ns#
[hybridsub: (?p rdfs:subPropertyOf ?q)
             notEqual(?p,?q) -> [ (?a ?q ?b) <- (?a ?p ?b) ] ]
```

→ génère une règle backward pour chaque relation `rdfs:subPropertyOf`

Architecture Générale



- `ModelFactory` : création d'associations entre un ou plusieurs **moteurs d'inférence** (`Reasoner`) et des `Graph`es RDF
- `ReasonerRegistry` : *catalogue* de moteurs d'inférence connus
- `OntModel API` : identification automatique d'un moteur d'inférence pour un type d'ontologie donnée
- `InfGraph` : **graphe inféré**

Moteurs d'inférence existants

- **Transitive reasoner** : matérialisation des relations transitives et reflexives `rdfs:subPropertyOf` et `rdfs:subClassOf` d'un schéma RDFS (commande `infer`).
- `RDF/RDFS rule reasoner:` matérialisation d'un sous-ensemble configurable de règles RDFS (RDF/RDFS entailment rules, voir `sémantique formelle`).
- `OWL, OWL Mini, OWL Micro Reasoners:` matérialisation « *utile mais incomplète* » de OWL/Lite.
- **Generic rule reasoner** : matérialisation de règles d'utilisateurs (inférence forward chaining, tabled backward chaining et hybride)

Outline

Logiques de Description

Une logique de description permet

- la définition de concepts par des contraintes logiques sur leur interprétation (“ensemble d’instances”) :
  - relations sémantiques entre concepts (incusion, equivalence)
  - composition logique / ensembliste de concepts (union, intersection, complément)
- le raisonnement logique sur les concept :
  - satisfiabilité des concepts : est-ce qu’il peut exister une instance (interprétation non-vide) pour un concept ?
  - cohérence de plusieurs concepts : est-ce qu’il existe une interprétation non-vide pour un ensemble de concepts ?
  - inclusion de concepts (subsomption) : est-ce que l’ensemble des instances d’un concept est incluse dans l’ensemble des instance d’un autre concept (pour toutes les interprétations) ?

L’ensemble des instances (l’interprétation) d’un concept peut être infini (définition et raisonnement intensionnels).

Exemples d'expressions

- Tous les artistes sont des personnes :

$$Artiste \sqsubseteq Personne$$

- Un sculpteur est une personne qui a créé une sculpture (définition) :

$$Sculpteur \equiv Personne \sqcap \exists a\_cree.Sculpture$$

- Toutes les personnes qui ont créé une peinture sont des peintres :

$$Personne \sqcap \exists a\_cree.Peinture \sqsubseteq Peintre$$

- Un artefact est un objet créé par une personne (définition) :

$$Artefact \equiv Objet \sqcap \exists cree\_par.Personne$$

- Toutes les sculptures et peintures sont des artefacts créés par des artistes :

$$Sculpture \sqcup Peinture \sqsubseteq Artefact \sqcap \exists cree\_par.Artiste$$

Interprétation de concepts et de rôles

Interprétation

Soit donné un ensemble de concepts C et de rôles R. Une interprétation $\mathcal{I}$ est un couple $(\Delta^{\mathcal{I}}, .^{\mathcal{I}})$ où :

- $\Delta^{\mathcal{I}}$ est un domaine non-vide d'instances
- $.^{\mathcal{I}}$ est une fonction d'interprétation qui associe à
  - chaque concept $A \in C$ un ensemble d'instances dans $\Delta^{\mathcal{I}}$:

$$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$$

  - chaque rôle $r \in R$ une relation binaire entre des instances du domaine $\Delta^{\mathcal{I}}$:

$$r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$$

Interprétation et sémantique)

Expr.	Fonction d'interprétation	Commentaire
a	$a^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$	concept a
r	$r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$	rôle r
$c \sqcap d$	$c^{\mathcal{I}} \cap d^{\mathcal{I}}$	intersection
$\forall r.c$	$\{x \in \Delta^{\mathcal{I}} \mid \forall y : (x,y) \in r^{\mathcal{I}} \rightarrow y \in c^{\mathcal{I}}\}$	quant. univ. *qualifié*
$\exists r$	$\{x \in \Delta^{\mathcal{I}} \mid \exists y \in \Delta^{\mathcal{I}} : (x,y) \in r^{\mathcal{I}}\}$	quant. exist. *non-qualifié*
$\neg c$	$\Delta^{\mathcal{I}} \setminus c^{\mathcal{I}}$	négation/complément
$c \sqcup d$	$c^{\mathcal{I}} \cup d^{\mathcal{I}}$	union
$\exists r.c$	$\{x \in \Delta^{\mathcal{I}} \mid \exists y \in \Delta^{\mathcal{I}} : (x,y) \in r^{\mathcal{I}} \wedge y \in c^{\mathcal{I}}\}$	quantif. exist. *qualifiée*

Satisfiabilité et Subsomption

Satisfiabilité

Un concept c est satisfiable, noté $sat(c)$, si et seulement si (ssi) il existe une interprétation $\mathcal{I}$ telle que $c^{\mathcal{I}}$ n'est pas vide :

$$sat(c) \Leftrightarrow \exists \mathcal{I} : c^{\mathcal{I}} \neq \emptyset$$

Subsomption

c subsume d, noté $d \sqsubseteq c$, ssi pour toute interprétation $(\Delta^{\mathcal{I}}, .^{\mathcal{I}})$, $d^{\mathcal{I}}$ est un sous-ensemble de $c^{\mathcal{I}}$:

$$d \sqsubseteq c \Leftrightarrow \forall \mathcal{I} : d^{\mathcal{I}} \subseteq c^{\mathcal{I}}$$

La subsomption peut être réduite à la satisfiabilité :

$$d \sqsubseteq c \Leftrightarrow \neg sat(\neg c \sqcap d) \Leftrightarrow \nexists \mathcal{I} : (\neg c \sqcap d)^{\mathcal{I}} \neq \emptyset$$

2 - Logiques de Description

- TBox et ABox
- La famille des Logiques de Description

TBox

TBox et terminologie

- Une TBox est un ensemble de définitions de concepts nommés.
- L'ensemble de noms de concepts est appelé une terminologie.

Services d'inférence sur une TBox

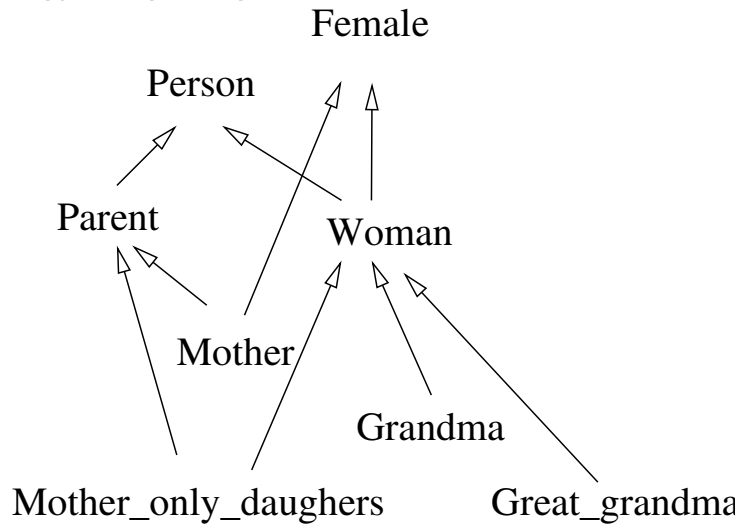
- Test de cohérence : trouver tous les concepts insatisfiables
- Extraction de taxinomie : calculer l'hiérarchie de subsomption ou la taxinomie des concepts nommés.

TBox : Construction de Taxinomie

Terminologie :

1. $\text{Woman} \equiv \text{Person} \sqcap \text{Female}$
2. $\text{Parent} \equiv \text{Person} \sqcap \exists \text{has\_child.Person}$
3. $\text{Mother} \equiv \text{Parent} \sqcap \text{Female} \equiv \text{Parent} \sqcap \text{Woman}$
4. $\text{Mother\_only\_daughters} \equiv \text{Woman} \sqcap \text{Parent} \sqcap \forall\ \text{has\_child.Woman} \equiv \text{Mother} \sqcap \forall\ \text{has\_child.Woman}$
5. $\text{Grandma} \equiv \text{Woman} \sqcap \exists \text{has\_child.Parent} \equiv \text{Mother} \sqcap \exists \text{has\_child.Parent}$
6. $\text{Great\_grandma} \equiv \text{Woman} \sqcap \exists \text{has\_child.}\exists \text{has\_child.Parent} \equiv \text{Grandma} \sqcap \exists \text{has\_child.}\exists \text{has\_child.Parent}$

Taxinomie :

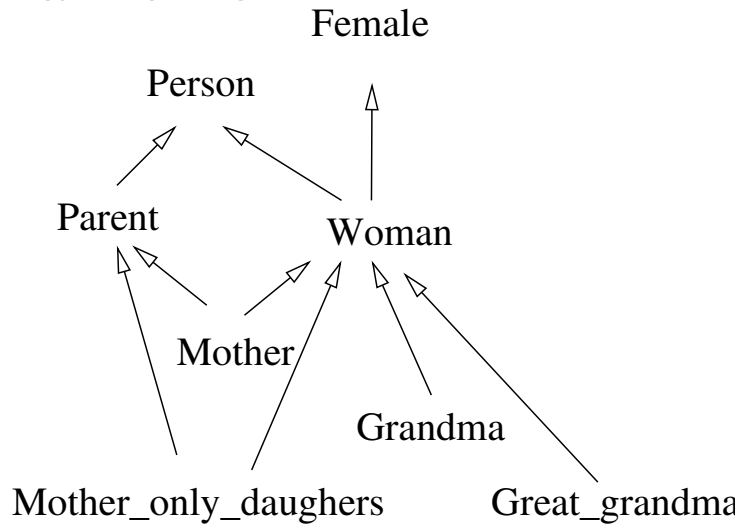


TBox : Construction de Taxinomie

Terminologie :

1. $\text{Woman} \equiv \text{Person} \sqcap \text{Female}$
2. $\text{Parent} \equiv \text{Person} \sqcap \exists \text{has\_child.Person}$
3. $\text{Mother} \equiv \text{Parent} \sqcap \text{Female} \equiv \text{Parent} \sqcap \text{Woman}$
4. $\text{Mother\_only\_daughters} \equiv \text{Woman} \sqcap \text{Parent} \sqcap \forall\ \text{has\_child.Woman} \equiv \text{Mother} \sqcap \forall\ \text{has\_child.Woman}$
5. $\text{Grandma} \equiv \text{Woman} \sqcap \exists \text{has\_child.Parent} \equiv \text{Mother} \sqcap \exists \text{has\_child.Parent}$
6. $\text{Great\_grandma} \equiv \text{Woman} \sqcap \exists \text{has\_child.}\exists \text{has\_child.Parent} \equiv \text{Grandma} \sqcap \exists \text{has\_child.}\exists \text{has\_child.Parent}$

Taxinomie :

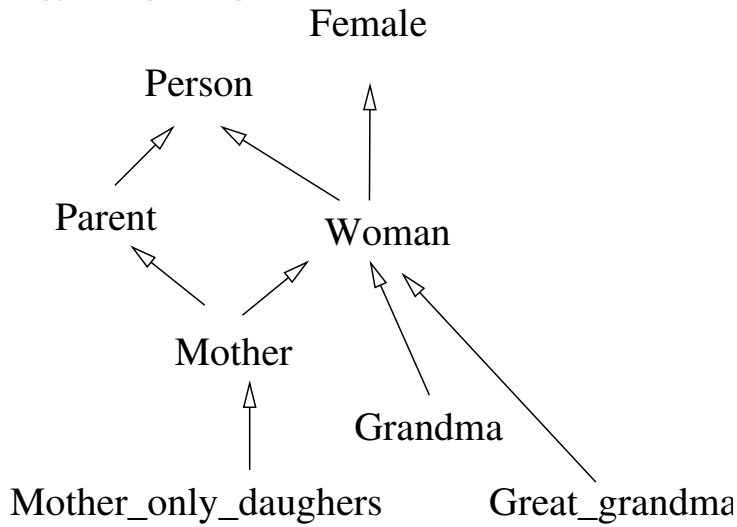


TBox : Construction de Taxinomie

Terminologie :

1. $\text{Woman} \equiv \text{Person} \sqcap \text{Female}$
2. $\text{Parent} \equiv \text{Person} \sqcap \exists \text{has\_child.Person}$
3. $\text{Mother} \equiv \text{Parent} \sqcap \text{Female} \equiv \text{Parent} \sqcap \text{Woman}$
4. $\text{Mother\_only\_daughters} \equiv \text{Woman} \sqcap \text{Parent} \sqcap \forall\ \text{has\_child.Woman} \equiv \text{Mother} \sqcap \forall\ \text{has\_child.Woman}$
5. $\text{Grandma} \equiv \text{Woman} \sqcap \exists \text{has\_child.Parent} \equiv \text{Mother} \sqcap \exists \text{has\_child.Parent}$
6. $\text{Great\_grandma} \equiv \text{Woman} \sqcap \exists \text{has\_child.}\exists \text{has\_child.Parent} \equiv \text{Grandma} \sqcap \exists \text{has\_child.}\exists \text{has\_child.Parent}$

Taxinomie :

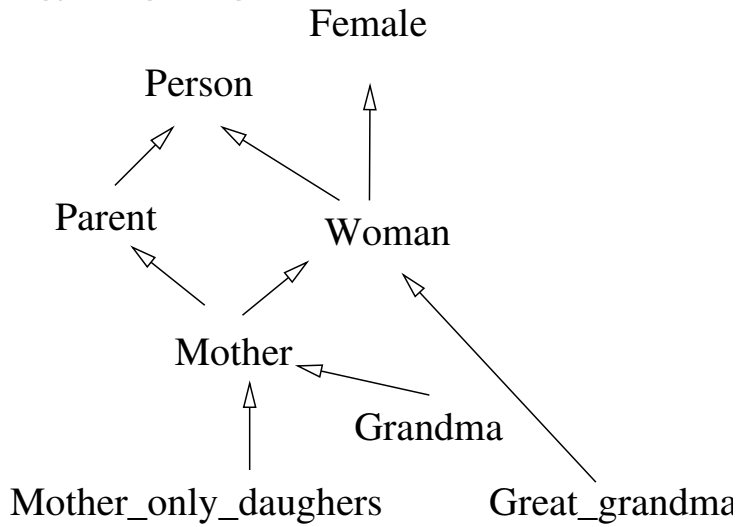


TBox : Construction de Taxinomie

Terminologie :

1. $\text{Woman} \equiv \text{Person} \sqcap \text{Female}$
2. $\text{Parent} \equiv \text{Person} \sqcap \exists \text{has\_child.Person}$
3. $\text{Mother} \equiv \text{Parent} \sqcap \text{Female} \equiv \text{Parent} \sqcap \text{Woman}$
4. $\text{Mother\_only\_daughters} \equiv \text{Woman} \sqcap \text{Parent} \sqcap \forall\ \text{has\_child.Woman} \equiv \text{Mother} \sqcap \forall\ \text{has\_child.Woman}$
5. $\text{Grandma} \equiv \text{Woman} \sqcap \exists \text{has\_child.Parent} \equiv \text{Mother} \sqcap \exists \text{has\_child.Parent}$
6. $\text{Great\_grandma} \equiv \text{Woman} \sqcap \exists \text{has\_child.}\exists \text{has\_child.Parent} \equiv \text{Grandma} \sqcap \exists \text{has\_child.}\exists \text{has\_child.Parent}$

Taxinomie :

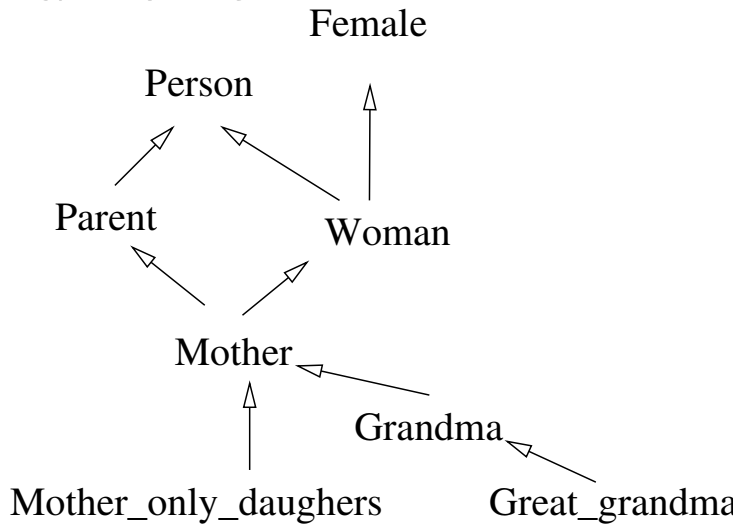


TBox : Construction de Taxinomie

Terminologie :

1. $\text{Woman} \equiv \text{Person} \sqcap \text{Female}$
2. $\text{Parent} \equiv \text{Person} \sqcap \exists \text{has\_child.Person}$
3. $\text{Mother} \equiv \text{Parent} \sqcap \text{Female} \equiv \text{Parent} \sqcap \text{Woman}$
4. $\text{Mother\_only\_daughters} \equiv \text{Woman} \sqcap \text{Parent} \sqcap \forall\ \text{has\_child.Woman} \equiv \text{Mother} \sqcap \forall\ \text{has\_child.Woman}$
5. $\text{Grandma} \equiv \text{Woman} \sqcap \exists \text{has\_child.Parent} \equiv \text{Mother} \sqcap \exists \text{has\_child.Parent}$
6. $\text{Great\_grandma} \equiv \text{Woman} \sqcap \exists \text{has\_child.}\exists \text{has\_child.Parent} \equiv \text{Grandma} \sqcap \exists \text{has\_child.}\exists \text{has\_child.Parent}$

Taxinomie :



Individus nommés et assertions

Individus nommés

Soit donné un ensemble d'individus $\mathcal{N}$:

- La fonction d'interprétation $.^{\mathcal{I}}$ associe à chaque nom dans $\mathcal{N}$ une instance dans $\Delta^{\mathcal{I}}$: $a \in \mathcal{N} : a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$
- Supposition du nom unique (unique name assumption) : $a \neq b \Rightarrow a^{\mathcal{I}} \neq b^{\mathcal{I}}$

Assertions sur les individus

Une assertion définit des contraintes sur les individus :

- Assertions de concepts "$a : c$" où $a \in \mathcal{N}$ et $c \in C$:
  - L'assertion $a : c$ est satisfaite ssi $a^{\mathcal{I}} \in c^{\mathcal{I}}$;
  - Exemple : *pierre* : *etudiant*
- Assertions de rôles "$(a, b) : r$" où $a, b \in \mathcal{N}$ et $r \in R$:
  - L'assertion $(a, b) : r$ est satisfaite ssi $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in r^{\mathcal{I}}$;
  - Exemple : (*pierre*, *bdia*) : *inscrit*

ABox

ABox

Une ABox est ensemble d'assertions de concepts et de rôles. La satisfiabilité d'assertions est définie par rapport à une ABox et une TBox données au départ.

Services d'inférence sur une ABox

- Est-ce qu'une ABox A est *satisfiable* étant donné une TBox T : $asat(A)$?
- Est-ce que a est une *instance* du concept c : $instance(a, c, A)$?

Tous les services d'inférence peuvent être réduits à *asat*

- a est une instance de c dans A : $instance(a, c, A) \equiv \neg asat(A \cup \{a : \neg c\})$
- c est satisfiable : $sat(c) \equiv \exists a : asat(\{a : c\})$
- c est subsumé par d : $c \sqsubseteq d \equiv \neg sat(\neg c \sqcap d) \equiv \neg asat(\{a : \neg c \sqcap d\})$

Suppposition du monde ouvert (OWA)

Exemple

- $C = \{Homme, Femme\}, R = \{enfant\}$
- $A = \{andre : Homme, charles : Homme, (charles, andre) : enfant\}$

CWA (Closed-World Assumption)

Il n'existe que les instances définies dans A :

- *andre* est l'enfant de *charles*.
- il y a que deux hommes et pas de femmes (compter).
- *andre* n'a pas d'enfant (negation by failure)

OWA (Open-World Assumption)

D'autres instances que celles définies dans A peuvent exister :

- il peut y avoir d'autres hommes, femmes et enfants

OWA $\rightarrow$ CWA

Pour obtenir la sémantique CWA (BD) avec OWA, il faut « fermer » les concepts et les rôles pour chaque individu :

- La contrainte $femme \sqsubseteq \bot$ indique qu'il n'y a pas de femmes ($\bot^{\mathcal{I}} = \emptyset$)
- L'assertion $andre : \nexists a\_enfant$ indique qu'*andre* est une instance du concept des individus qui n'ont pas d'enfants.
- L'assertion $charles : \leq 1 a\_enfant$ indique que *charles* à au maximum un enfant.

Suppposition du monde ouvert (OWA)

Exemple

- $C = \{Homme, Femme\}, R = \{enfant\}$
- $A = \{andre : Homme, charles : Homme, (charles, andre) : enfant\}$

CWA (Closed-World Assumption)

Il n'existe que les instances définies dans A :

- *andre* est l'enfant de *charles*.
- il y a que deux hommes et pas de femmes (compter).
- *andre* n'a pas d'enfant (negation by failure)

OWA (Open-World Assumption)

D'autres instances que celles définies dans A peuvent exister :

- il peut y avoir d'autres hommes, femmes et enfants

OWA $\rightarrow$ CWA

Pour obtenir la sémantique CWA (BD) avec OWA, il faut « fermer » les concepts et les rôles pour chaque individu :

- La contrainte $femme \sqsubseteq \bot$ indique qu'il n'y a pas de femmes ($\bot^{\mathcal{I}} = \emptyset$)
- L'assertion $andre : \nexists a\_enfant$ indique qu'*andre* est une instance du concept des individus qui n'ont pas d'enfants.
- L'assertion $charles : \leq 1 a\_enfant$ indique que *charles* à au maximum un enfant.

Suppposition du monde ouvert (OWA)

Exemple

- $C = \{Homme, Femme\}, R = \{enfant\}$
- $A = \{andre : Homme, charles : Homme, (charles, andre) : enfant\}$

CWA (Closed-World Assumption)

Il n'existe que les instances définies dans A :

- *andre* est l'enfant de *charles*.
- il y a que deux hommes et pas de femmes (compter).
- *andre* n'a pas d'enfant (negation by failure)

OWA (Open-World Assumption)

D'autres instances que celles définies dans A peuvent exister :

- il peut y avoir d'autres hommes, femmes et enfants

OWA $\rightarrow$ CWA

Pour obtenir la sémantique CWA (BD) avec OWA, il faut « fermer » les concepts et les rôles pour chaque individu :

- La contrainte $femme \sqsubseteq \bot$ indique qu'il n'y a pas de femmes ($\bot^{\mathcal{I}} = \emptyset$)
- L'assertion $andre : \nexists a\_enfant$ indique qu'*andre* est une instance du concept des individus qui n'ont pas d'enfants.
- L'assertion $charles :\leq 1 a\_enfant$ indique que *charles* à au maximum un enfant.

2 - Logiques de Description

- TBox et ABox
- La famille des Logiques de Description

$\mathcal{FL}^-$ et $\mathcal{ALC}$

Les familles de logiques de description qui se distinguent par les *constructeurs de concepts* qui peuvent être utilisés :

$\mathcal{FL}^-$

Syntaxe	Sémantique	Commentaire
a	$a^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$	concept a
r	$r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$	rôle r
$c \sqcap D$	$c^{\mathcal{I}} \cap D^{\mathcal{I}}$	intersection
$\forall r.c$	$\{x \in \Delta^{\mathcal{I}} \mid \forall y : (x,y) \in r^{\mathcal{I}} \rightarrow y \in c^{\mathcal{I}}\}$	quant. univ. *qualifié*
$\exists r$	$\{x \in \Delta^{\mathcal{I}} \mid \exists y \in \Delta^{\mathcal{I}} : (x,y) \in r^{\mathcal{I}}\}$	quant. exist. *non-qualifié*

$\mathcal{ALC}$

$\mathcal{ALC}$ est une extension de $\mathcal{FL}^-$ avec :

Syntaxe	Sémantique	Commentaire
$\neg c$	$\Delta^{\mathcal{I}} \setminus c^{\mathcal{I}}$	négation/complément
$c \sqcup D$	$c^{\mathcal{I}} \cup D^{\mathcal{I}}$	union
$\exists r.c$	$\{x \in \Delta^{\mathcal{I}} \mid \exists y \in \Delta^{\mathcal{I}} : (x,y) \in r^{\mathcal{I}} \wedge y \in c^{\mathcal{I}}\}$	quantif. exist. *qualifiée*

Autres constructeurs DL

Restrictions numériques sur les rôles ($\mathcal{N}$ resp. $\mathcal{Q}$)

- simples : $\geq 3 a\_enfant$, $\leq 1 a\_enfant$
- qualifiés : $\geq 3 a\_enfant.Homme$, $\leq 1 a\_enfant.Femme$

Hiérarchies de rôles ($\mathcal{H}$)

a_fils est une sous-propriété de *a_enfant* :

- $a\_fils \sqsubseteq a\_enfant$

Rôles transitives ($\mathcal{R}_+$)

- *ancetre* est transitif : $ancetre \equiv ancetre.ancetre$

Autres familles de DL

- $\mathcal{S} = \mathcal{ALCR}_+$: intersection, union, négation, quantification existentielle, rôle transtivies
- $\mathcal{SHIQ}$: $\mathcal{S}$ plus réstriction numérique qualifié, hiérarchie de rôles, rôles inverses

Autres constructeurs DL (2)

Rôles inverses ($\mathcal{I}$)

- $a\_enfant \equiv parent^{-}$

Cycles terminologiques

- $Humain \sqsubseteq a\_parent.Humain$

Axioms généraux

Axioms qui n'ont pas de nom de concepts à gauche ou $\top$ à gauche :

- domaine d'un rôle : $\exists has\_child.\top \sqsubseteq Parent$
- co-domaine d'un rôle : $\top \sqsubseteq \forall has\_child.Person$

Compléxité et expressivité

Logique	Expressivité	$C \sqsubseteq D$	$C(a)$
$\mathcal{FL}^-$	$C \sqcap D$, $\forall R.C$, $\exists R$	P	P
$\mathcal{AL}$	$\neg A$	P	P
$\mathcal{ALE}$	$\exists R.C$	NP	PSPACE
$\mathcal{ALC}$	$\neg C$	PSPACE	PSPACE
$\mathcal{ALCO}$	$\{a_1, ...\}$	PSPACE	PSPACE
$\mathcal{SHIQ}$ (OWL-DL)		EXPTIME	EXPTIME
KL-ONE		non-décidable	non-décidable

Bibliographie

- Transparents d'Enrico Franconi, Department of Computer Science, University of Manchester http ://www.cs.man.ac.uk/ franconi
- Description Logics and Semantic Web Description Logics : A Logical Foundation of the Semantic Web and its Applications Volker Haarslev Concordia University
- Description Logics : Basics, Applications, and More, Ian Horrocks,Information Management Group University of Manchester, UK Ulrike Sattler, Institut für Theoretische Informatik TU Dresden, Germany

Outline

3 - OWL

- Web Ontology Language
- Définition de classes
- Définition de propriétés
- Définition d'individus

Rappel de RDF/RDFS

- RDF/RDFS permet de créer un graphe par l'*union* de descriptions de ressources.
- Une classe RDFS (un concept) est défini par un nom, un ensemble de propriétés et sa position dans une hiérarchie définie par la relation rdfs:subClassOf.
- L'intégration et le raisonnement sont limités à l'utilisation des types de propriétés et des relations rdfs:subClassOf et rdfs:subPropertyOf.

Web Ontology Language : OWL

Recommandation W3C qui définit une famille de langages d'ontologies qui

- facilitent l'intégration et l'évolution d'ontologies et de métadonnées,
- permettent de définir des concepts "complexes",
- de vérifier la cohérence de ces définitions et
- d'inférer des relations sémantiques (inclusion, équivalence) entre les concepts.

RDF Schema et OWL

- OWL et RDF Schema utilisent *le modèle et la syntaxe RDF* pour définir des ontologies
- RDF Schema = ontologies “simples” :
  - classes, ressources, litéraux
  - sous-classes, sous-propriétés
- OWL = ontologies “complexes”
  - classes et propriétés équivalentes,
  - identité d’objet,
  - propriétés symétriques et transitives,
  - contraintes de cardinalité

Full OWL, OWL-DL, OWL Lite

OWL définit une famille de langages :

- OWL est un langage de représentation de connaissances puissant qui permet de décrire formellement des concepts complexes à partir d'autres concepts.
- "Full OWL" permet de manipuler des classes comme des objets et dépasse le pouvoir d'expression de la logique du premier ordre.
- OWL-DL est un fragment dont la sémantique peut-être formalisée sous forme d'une logique de description (OWL DL).

OWL-DL

- OWL-Lite correspond à la logique de description $\mathcal{SHIQ}$ (restriction numérique des rôles, hiérarchie de rôles, rôles transitifs et inverses).
- OWL-DL = $\mathcal{SHIQO}$ ($\mathcal{SHIQ}$ avec individus nommés)
- Expression DL :

$$Student = Person \sqcap >= 1 enrolledIn$$

- Expression OWL :

```
asws:minEn   a                    owl:Restriction  ;
             owl:minCardinality   "1^^xsd:integer" ;
             owl:onProperty       asws:enrolledIn  .

asws:Student  a                   owl:Class  ;
             owl:intersectionOf   ( asws:Person asws:minEn )  .

asws:Person  a  owl:Class  .
```

3 - OWL

- Web Ontology Language
- Définition de classes
- Définition de propriétés
- Définition d'individus

OWL : Définition de classes

- Une classe OWL peut être anonyme ou identifiée par un nom (référence URI).
- Il est possible de définir des contraintes sur l'l'extension (l'ensemble des instances) d'une classe :
  - par l'énumération de ses instances,
  - par la définition de contraintes sur les propriétés des instances (définition intensionnelle),
  - comme union, intersection, complément, inclusion, équivalence, exclusion des extensions d'autres classes.

Définition par énumération

Définition par énumération des instances :

- owl:oneOf
- DL (assertions) :

$$\begin{aligned} \mathit{beline} &: \mathit{InscritsASWS14} \\ \mathit{chamakhi} &: \mathit{InscritsASWS14} \\ \mathit{essayan} &: \mathit{InscritsASWS14} \\ \mathit{feuillerat} &: \mathit{InscritsASWS14} \end{aligned}$$

- OWL :

```
asws:inscritsASWS14
      a owl:Class ;
      owl:oneOf (asws:beline asws:chamakhi
                 asws:essayan asws:feuillerat ) .
```

Définition d'une classe par les propriétés de ses instances

Définition d'une classe par des contraintes sur les propriétés de ses instances :

- contraintes sur les valeurs : owl:hasValue
- contraintes sur les types des valeurs : owl:allValuesFrom, owl:someValuesFrom
- contraintes sur la cardinalité des propriétés : owl:maxCardinality, owl:minCardinality, owl:Cardinality

Définition par propriétés : Exemple

Exemple : La classe GroupeMixte contient tous les groupes qui contiennent au moins un étudiant DAC et un étudiant ANDROIDE :

- DL :

$$GroupeMixte \sqsubseteq Groupe \sqcap \exists membre.InscritsDAC \sqcap \exists membre.InscritsANDROIDE$$

- OWL :

```
asws:GroupeMixte  a          owl:Class ;
    rdfs:subClassOf  asws:Groupe ;
    rdfs:subClassOf   [ a                    owl:Restriction ;
                      owl:onProperty         asws:membre ;
                      owl:someValuesFrom     asws:InscritsDAC
                    ] ;
    rdfs:subClassOf   [ a                    owl:Restriction ;
                      owl:onProperty         asws:membre ;
                      owl:someValuesFrom     asws:InscritsANDROIDE
                    ] .
```

Contraintes de cardinalité

Restrictions numériques sur les rôles :

- owl:minCardinality, owl:maxCardinality
- DL :
  - simples ($\mathcal{N}$) : $\geq 1 a\_enfant$
  - qualifiés ($\mathcal{Q}$) : $\leq 3 a\_enfant.Homme$
- OWL :

```
[ a                                  owl:Restriction ;
  owl:maxQualifiedCardinality        "3"^^xsd:integer ;
  owl:minCardinality                 "1"^^xsd:integer ;
  owl:onClass                        asws:Homme ;
  owl:onProperty                     asws:a_enfants
] .
```

Intersection, union, complément

Définition d'une classe comme l'intersection, l'union et le complément d'autres classes :

- owl:intersectionOf, owl:unionOf, owl:complementOf
- DL : $A \sqcap B$, $A \sqcup B$, $\neg A$
- OWL :

```
asws:InscritsIMA  a  owl:Class .
asws:InscritsDAC  a  owl:Class .
asws:InscritsANDROIDE        a        owl:Class .
asws:Inscrits  a     owl:Class ;
        owl:unionOf  ( asws:InscritsANDROIDE
                       asws:InscritsDAC
                       asws:InscritsIMA ) .
```

Relations entre classes

Définition d'une classe comme l'inclusion, l'équivalence, l'exclusion d'autres classes :

- rdfs:subClassOf, owl:equivalentClass, owl:disjointWith
- DL : $A \sqsubseteq B$, $A \equiv B$, $A \sqcap B \sqsubseteq \bot$
- OWL :

```
asws:Students   a                  owl:Class   ;
          rdfs:subClassOf          asws:Person   ;
          owl:disjointWith         asws:Teachers   ;
          owl:equivalentClass      asws:Etudiant   .
```

3 - OWL

- Web Ontology Language
- Définition de classes
- Définition de propriétés
- Définition d'individus

Types de propriétés I

Types de propriétés

- Propriétés d'objets : owl:ObjectProperty
- Propriétés de valeurs : owl:DatatypeProperty

Les deux types de propriétés sont des sous-classes de la classe rdf :Property.

Contraintes RDFS I

Hiérarchies de rôles : l'extension d'une propriété est incluse dans l'extension d'une autre propriété

- rdfs:subPropertyOf
- DL ($\mathcal{H}$) : *fils* $\sqsubseteq$ *child*
- RDF/OWL :

```
asws:fils   a  owl:ObjectProperty  ;
            rdfs:subPropertyOf   asws:child  .
```

Contraintes RDFS II

Contrainte de domaine

- rdfs:domain
- DL : $\exists enfant.\top \sqsubseteq Pere \sqcup Mere$
- OWL :

```
asws:enfant  a          owl:ObjectProperty ;
  dfs:domain  [ a owl:Class ;
              owl:unionOf  ( asws:Pere
                             asws:Mere ) ] .

asws:Mere  a    owl:Class .
asws:Pere  a    owl:Class .
```

Avec RDFS il faut définir une super-classe de Pere et Mere.

Contraintes RDFS III

Contrainte de co-domaine

- RDFS/OWL : rdfs:range : types des objets
- DL : $\top \sqsubseteq \forall enfant.(Fils \sqcup Fille)$

```
asws:enfant  a       owl:ObjectProperty  ;
 rdfs:range  [ a                 owl:Class  ;
                owl:unionOf   ( asws:Fils
                                asws:Fille  ) ]  .

asws:Fille  a   owl:Class  .
asws:Fils   a   owl:Class  .
```

Contraintes « inter-propriétés » I

Equivalence

L'extension d'une propriété est équivalente à l'extension de l'autre propriété

- owl:equivalentProperty
- DL : $enfant \equiv child$
- OWL :

```
asws:enfant  owl:equivalentProperty  asws:child  .
```

Rôles inverses

L'extension d'une propriété est égal à l'inverse de l'extension de l'autre

- owl:inverseOf
- DL ($\mathcal{I}$) : $enfant \equiv parent^{-}$

```
asws:enfant  owl:inverseOf  asws:parent  .
```

Contraintes « inter-propriétés » II

Propriété symmetrique

L'extension est une relation symmetrique

- owl:SymmetricProperty
- DL : $sibling = sibling^{-}$
- OWL :

```
asws:sibling  a  owl:SymmetricProperty .
```

Rôles transitives

L'extension est une relation transitive

- owl:TransitiveProperty
- DL ($\mathcal{R}_{+}$) : $ancetre \equiv ancetre.ancetre$
- OWL :

```
asws:ancetre  a  owl:TransitiveProperty .
```

Properiétés fonctionnelles I

Contraintes de cardinalité du domaine

Un seul objet par sujet (extension = fonction)

- owl:FunctionalProperty
- DL : $(= 1)a\_pere$

```
asws:a_pere  a  owl:FunctionalProperty .
```

Contraintes de cardinalité du co-domaine

- owl:InverseFunctionalProperty : un seul sujet par objet
- DL : $(= 1)pere\_de^{-}$
- OWL :

```
asws:pere_de  a  owl:InverseFunctionalProperty .
```

Exemples

- La propriété `enfant` est **l'inverse** de la propriété `parent` :

```
asws:parent   a            owl:ObjectProperty  ;
              rdfs:domain  asws:personne       ;
              rdfs:range   asws:personne       .

asws:enfant   a              owl:ObjectProperty  ;
              owl:inverseOf  asws:parent         .
```

  Le domaine et le co-domaine de la propriété enfant peuvent être déduits à partir de la propriété parent.

- La propriété `époux_de` est **mono-valuée (fonctionnelle) et symétrique** :

```
asws:epoux_de   a   owl:FunctionalProperty  ,
                    owl:SymmetricProperty   ;
                rdfs:domain  asws:personne  ;
                rdfs:range   asws:personne  .
```

  Le domaine est égal au co-domaine.

Exemples

- Toutes les personnes ont exactement une mère → La propriété `mère_de` est **mono-valuée inverse** :

```
asws:mere_de   a  owl:InverseFunctionalProperty  ;
          rdfs:domain   asws:femme  ;
          rdfs:range    asws:personne  .
```

- **Transitivité :** si *a* est un ancêtre de *b* et *b* est un ancêtre de *c*, alors *a* est un ancêtre de *c* :

```
asws:ancetre_de
          a                owl:TransitiveProperty  ;
          rdfs:domain      asws:personne  ;
          rdfs:range       asws:personne  .
```

Le domaine est égal au co-domaine.

3 - OWL

- Web Ontology Language
- Définition de classes
- Définition de propriétés
- Définition d'individus

Propriétés d'individus (faits)

Les faits sont exprimés sous forme de descriptions RDF :

```
asws:Tosca   a                    asws:Opera  ;
       asws:hasComposer           asws:Giacomo_Puccini  ;
       asws:hasLibrettist         asws:Victorien_Sardou  ,
                                        asws:Giuseppe_Giacosa  ,
                                        asws:Luigi_Illica  ;
       asws:numberOfActs          "3^^xsd:positiveInteger"  ;
       asws:premiereDate          "1900-01-14^^xsd:Date"  ;
       asws:premierePlace         asws:Roma  .
```

Contraintes sur l'identité des ressources

- owl:sameAs : deux ressources (faits, classes, propriétés) sont identiques (désignent le même objet)
- OWL :

```
asws:William_Jefferson_Clinton
        a              asws:President ;
        owl:sameAs     asws:BillClinton .
```

- owl:differentFrom : deux ressources (faits, classes, propriétés) sont distinctes (désignent un objet différent)
- OWL :

```
asws:Meurtrier  a              asws:Opera ;
        owl:differentFrom      asws:Jardinier .
```

La conclusion que le jardinier est le meurtrier mène à une contradiction.