

Université d'Angers

Master 1 Informatique

Soutenance de stage

Représentation de l'inférence OWL 2 en CHR++

Nom : Arina Abed

Encadrants : Vincent Barichard, David Genest, Eric Monfroy

Lieu : LERIA – Université d'Angers

Période : 14 avril – 30 juillet 2025

- **Sujet** : représenter l'inférence OWL 2 en CHR++.
- **Motivation** : les raisonneurs OWL classiques sont :
 - des boîtes noires
 - algorithmes complexes
 - inférences « on-the-fly »
- **Proposition** : un moteur transparent, modulable, saturant explicitement la base de faits.
- **Résultat** : un prototype fonctionnel.

- **Web sémantique :**

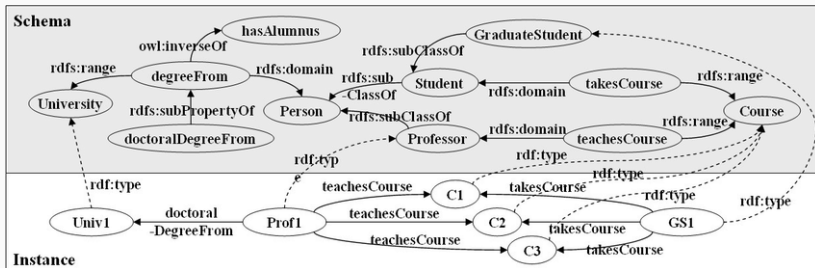
Donner un sens aux données pour qu'elles soient comprises et utilisées par les machines (pas seulement affichées).

- **Ontologie :**

- Structurer les connaissances d'un domaine.
- Décrire les concepts et leurs relations.
- Pour : *raisonner, échanger, rechercher efficacement.*

- **OWL** : Langage standard du W3C pour définir des ontologies sémantiques
- **Principales caractéristiques** :
 - Classes et instances.
 - Relations : Propriétés d'objet (owl:ObjectProperty) et de données (owl:DatatypeProperty).
 - Capacités de raisonnement : sous-classes, équivalences, restrictions de cardinalité, transitivité, ...
- **OWL 2** : version enrichie et normalisée (2009) d'OWL, offrant une expressivité accrue.

OWL : exemple d'ontologie



- **CHR (Constraint Handling Rules)** : langage déclaratif de réécriture de contraintes.
- **Types de règles CHR** :
 - **Simplification** tête $\Leftrightarrow$ garde | corps
 - **propagation** tête $\Rightarrow$ garde | corps
 - **simpagation** tête+ \tête- $\Leftrightarrow$ garde | corps
- **CHR++** : réimplémentation moderne de CHR en C++.
- **Avantages** : flexibilité et expressivité, exécution incrémentale, optimisation des ressources.

Exemple en CHR++ pour la relation d'ordre `leq(X, Y)` :

- Réflexivité : `leq(X, X)  $\Leftrightarrow$  success()`
- Antisymétrie : `leq(X, Y), leq(Y, X)  $\Leftrightarrow$  X = Y`
- Idempotence : `leq(X, Y) \setminus leq(X, Y)  $\Leftrightarrow$  success()`
- Transitivité : `leq(X, Y), leq(Y, Z)  $\Rightarrow$  leq(X, Z)`

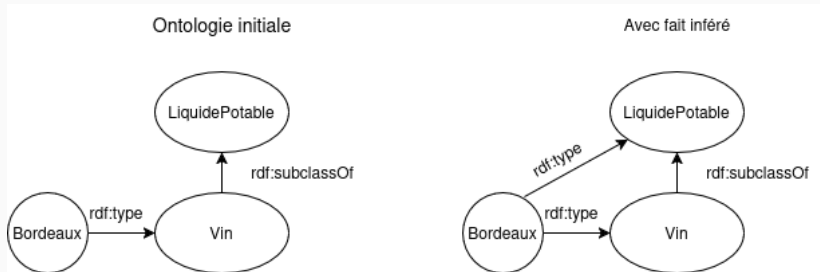
Déroulement avec les faits `leq(2, 4), leq(4, 6)` :

`leq(2, 4), leq(4, 6)`
 $\Rightarrow$ `leq(2, 6)` (par transitivité)

- Modéliser les axiomes OWL 2 en contraintes CHR++.
- Formaliser les mécanismes d'inférence OWL 2 via des règles CHR++ :
 - inférer de nouveaux faits
 - assurer la cohérence et la validité des faits initiaux
 - suppression de doublons
- Définir un moyen d'interroger la base construite.
- Implémenter un prototype de raisonneur OWL 2 basé sur CHR++.

- Axiome OWL : SubClassOf
- En CHR++ :
 - contrainte :
owlSubclassOf(String,String)
 - règle :
owlSubclassOf(C1, C2), owlClassAssertion(x, C1)
 $\Rightarrow$ different(C1,C2) | owlClassAssertion(x, C2)

Test : Evolution du graph d'une ontologie



Architecture interne prototype



Interrogation de la base

Contraintes-but :

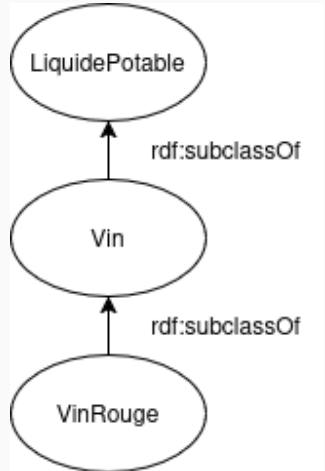
- querySuperClassOf
- querySubClassOf
- queryClassification

Exemple :

$querySuperClassOf(X), owlSubclassOf(X, Y) \Rightarrow X \neq Y \mid res(Y), querySuperClassOf(Y)$

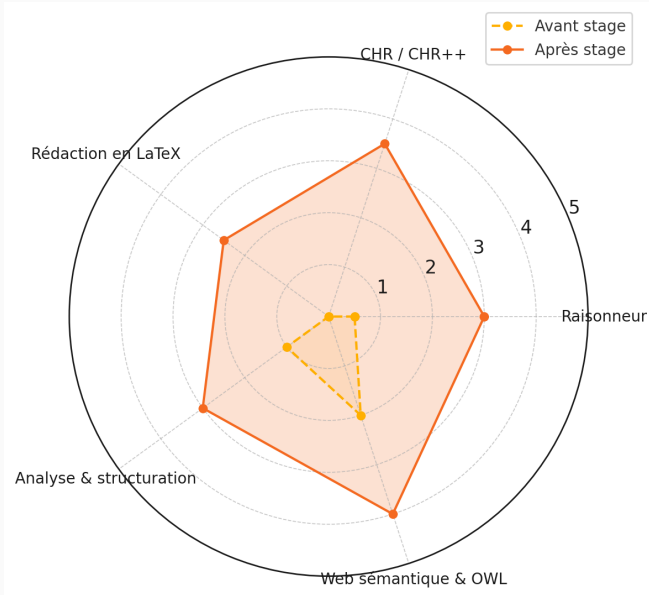
$querySuperClassOf(X) \Rightarrow res(X)$

$res(X) \Rightarrow afficher(X)$



- Bilan :
 - prototype fonctionnel
 - transparence et modularité
 - contraintes-but
- Perspectives :
 - couverture complète d'OWL 2 (cardinalités, énumérations, chaînes de propriétés)
 - définition d'autres contraintes-but
 - requêtes complexes (multi-variables comme SPARQL)

Enseignements personnels



Définitions C++ :

```
bool different(const std::string& A, const std::string& B){  
    return A != B;  
}  
void afficher(std::string x){  
    std::cout << x << std::endl;  
}
```

Contraintes CHR++ :

```
<CHR name="OWL2">  
    // === Contraintes ===  
    owlClass(+std::string),  
    owlSubclassOf(+std::string,+std::string),  
    owlClassAssertion(+std::string,+std::string),  
    owlDisjointClass(+std::string,+std::string),  
    querySuperClassOf(+std::string),  
    res(+std::string)  
</CHR>
```

```

// Suppression des doublons
owlClass(C) #passive \ owlClass(C) <=> true ;;
// Détection 'dincohérences
DisjointClass @ owlDisjointClass(A,B),owlClassAssertion(X,A),
                owlClassAssertion(X,B)<=> failure();;
// Inférence du schéma
owlClass(A) ==> owlSubclassOf(A,A);;
owlClass(C) ==> owlSubclassOf(C,owlThing);;
// Inférence sur les individus
subclass @ owlSubclassOf(A,B),owlClassAssertion(X,A)
           ==> different(A,B) | owlClassAssertion(X,B);;
// Requêtes
querySuperClassOf(X), owlSubclassOf(X,Y)
  ==> different(X,Y) | res(Y), querySuperClassOf(Y);;
querySuperClassOf(X) ==> res(X);;
res(X) ==> afficher(X);;

```

```
int main()
{
    TRACE(chr::Log::register_flags(chr::Log::ALL);)
    auto space = OWL2::create();
    using namespace std;
    ParserCowl<decltype(*space)> parser("example2.ofn",*space)
    // === Injection de faits OWL ===
    CHR_RUN(
        space->querySuperClasses(std::string("VinRouge"),
    )
    print(*space);
    chr::Statistics::print(std::cout);
    return 0;
}
```