Introduction à SWRL (Semantic Web Rule Language)

Bernard ESPINASSE

Aix-Marseille Université
LIS UMR CNRS 7020

Novembre 2019

- **De OWL à SWRL**
- **Règles SWRL**
- **Exemples d'utilisation de règles SWRL**

References

- **Livres, articles et rapports:**
  - W3C, « OWL Web Ontology Language Semantics and Abstract Syntax »
  - …
- **Web W3C:**
  - Page du W3C : http://www.w3.org/2004/OWL/
  - Reference : http://www.w3.org/TR/owl-ref/
  - Guide : http://www.w3.org/TR/owl-guide/
  - …
- **Cours/tutorials:**
  - Tutorial of *Martin Kuba, Institute of Computer Science,* *http://dior.ics.muni.cz/~makub/owl/*
  - Cours O. Papini, Aix-Marseille Université
  - …

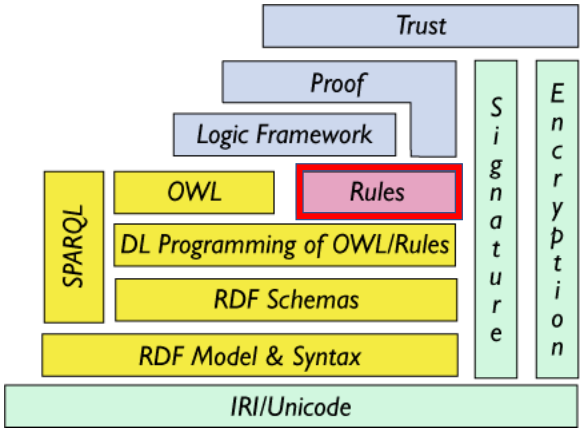
Outline

- **1. Introduction à SWRL:**
  - Intérêts des langages de règles
  - Introduction à SWRL
  - SWRL et les moteurs d'inférences
- **2. SWRL rules**
  - Prédicats SWRL
  - Principaux types de règles SWRL
- **3. Exemples d'utilisation de règles SWRL**
  - Specifications (en Protégé & SWRL syntax)
  - Inférences avec règles SWRL
- **4. Limites de SWRL (et OWL2)**

1. Introduction à SWRL

- Intérêts des langages de règles
- Introduction à SWRL
- SWRL et les moteurs d'inférences

Place de SWRL dans le gâteau du WS …



Source : W3C, T Berners-Lee, Ivan Herman

OWL et SWRL (1)

- **Le langage OWL** n'est **pas capable d'exprimer toutes les relations :**
  - Ex : Il ne peut exprimer la relation « *child of married parents* »

    -> parce qu'il n'y a aucun moyen d'exprimer dans OWL2 la relation entre les individus avec lesquels un individu a des relations.
  - L'expressivité de OWL peut être étendue en ajoutant des **REGLES** à une ontologie
- Un **langage de règles** est **utile** pour différentes raisons :
  - **Ajout d'expressivité à OWL:** *Cet ajout a un prix qui est une perte de décidabilité*
  - **La réutilisation de règles existantes**
  - **En général il est plus facile de lire et écrire des règles avec un langage de règles :** *Vrai en general mais pas toujours*

⇨ **Semantic Web Rule Language - SWRL [Horrocks & al., 2004]**

OWL and SWRL (2)

Semantic Web Rule Language - SWRL [Horrocks & al., 2004] :

- **Une proposition qui combine Ontologies et Règles**
  - Ontologies : OWL-DL
  - Règles : RuleML (a rule language)
- **Les règles SWRL** sont similaires aux règles en PROLOG ou en DATALOG
- **SWRL = OWL-DL + RuleML :**
  - OWL-DL: sans variable
  - RuleML: utilise des variables
- **Protégé OWL editor** supporte SWRL rules, comme les raisonneurs **Pellet** et **Hermit**
- **En utilisant l'OWL API, SWRL peut utiliser du code Java**

Règles SWRL (1)

- **Permet la manipulation d'instances par des variables (?x, ?y, ?z)**
- **Ne permet pas de créer des concepts** et des **relations**
- **Permet seulement d'ajouter** des **relations** selon les **valeurs** des **variables** (individus) et la **satisfaction de règles**
- **Les règles SWRL** sont construites selon le schéma suivant :

  Antécédent -> Conséquent

  Antécédent = atomes conjonctions (∧)

  Conséquent = seulement un atome
- **Un atome** peut être :
  - Soit une **instance de concept** : $C_i(z)$ = ***unary*** *predicat*
  - Soir une **relation OWL :** $R_i(x, y)$ = ***binary*** *predicat*
  - Soit des **relations SWRL**: **same-as(?x, ?y)** ou **different-from (?x, ?y)**

Ex:

$$R_1(x, y) \wedge \text{different-from}(x, y) \wedge C_1(z) \wedge \ldots \rightarrow R_n(x, z)$$

Antécédent *Conséquent*

Règles SWRL (2)

- Une règle SWRL fonctionne selon le **principe de satisfaction** de **l'Antécédent** ou du **Conséquent :**
  - Si **l'Antécédent** et le **Conséquent** sont **définis** :

    *-> SI **l'Antécédent** est **satisfait** ALORS le **Conséquent** est aussi **satisfait***
  - Si l'**Antécédent** est **vide**, cela correspond à un **Antécédent satisfait** :

    *-> Permet de **définir des faits***
  - Si le **Conséquent** est **vide**, cela correspond à un **Conséquent satisfait** :

    *-> **L'Antécédent** ne doit pas être **satisfait***

SWRL Rules (3)

Exemple :

- Soit en OWL le concept de *Uncle* définit ainsi:

  intersectionOf(SubClassOf(Man), isBrotherOf(Father))

  et la relation *isUncleOf*
- Nous savons définir qu'une personne est un oncle, mais OWL ne permet pas de définir la relation *isUncleOf* représentant le fait d'être un oncle d'une personne donnée
- SWRL permet de définir cette relation *isUncleOf* en la reliant aux instances concernées :

aChild(?x,?y) ∧ isBrotherOf(?z,?x) -> isUncleOf(?z,?y)

SWRL et moteurs d'inference (Reasoners)

- La plupart des **moteurs d'inférences** (raisonneurs) **supportent SWRL** :
  - Pellet, Bossam, Hoolet, KAON2, RacerPro, R2ML (REWERSE Rule Markup Language) et Sesame.
- **Selon 3 approches différentes :**
  - **Traduire SWRL en logique du 1er ordre** (Hoolet, …)
  - **Traduire OWL-DL en règles** et appliquer un **algorithme de chaînage avant** (forward chaining) (Bossam, …)
  - **Intégrer les règles SWRL** dans le moteur d'inférences **OWL-DL** basé sur la méthode des tableaux (Pellet, …)

Cependant, les implémentations SWRL actuelles ont besoin de **beaucoup de calculs**, elles ne sont seulement utilisable qu'avec des **ontologies petites** ou **moyennes**

Règles SWRL et décidabilité: DL-safe rules

- Arbitrairement les règles SWRL devraient conduire à une **indécidabilité**

  => aussi seulement des règles appelées **DL-safe** rules are sont implémentées dans les raisonneurs.
- **DL-safe rules** sont des règles appliquées seulement à des named individuals (individus nommés)
- Elles **ne s'appliquent pas** à des **individus qui ne sont pas nommés**, mais connus comme existant.

Règles SWRL & règles DL-Safe SWRL (1)

- Considérons l'ontologie (Turtle syntaxe):

```
BrotherOfJoe  a  owl:Class;
     rdfs:subClassOf  [
          a  owl:Restriction;
          owl:onProperty  :hasBrother;
          owl:hasValue  :joe
     ] .
:Dick  a  [
          a  owl:Restriction;
          owl:onProperty  :hasParent;
          owl:someValuesFrom  :BrotherOfJoe
     ] .
:Joe  a  :Person .
```

- La première partie spécifie qu'un frère de Joe (:BrotherOfJoe) a nécessairement un frère qui est Joe
- La seconde partie spécifie que Dick a un parent qui est un frère de Joe

⇨ **On peut conclure que Joe est l'oncle de Dick**

Règles SWRL et règles SWRL DL-Safe (2)

- **On peut écrire la règle SWRL :**

  :hasParent(?x,?y), :hasBrother(?y,?z) -> :hasUncle(?x,?z)

  - Si cette règle est une règle **SWRL normale**, on obtient la conclusion attendue
  - Si cette règle est une règle **SWRL DL-safe**, on peut l'appliquer seulement **si on peut remplacer les variables ?x, ?y et ?z par une instance nommée** (named individual)
- **Mais l'ontologie ne spécifie pas le rôle de ?y**
  - En fait une règle SWRL DL-safe fonctionne comme si nous avions une autre classe *:Nom*, dont les instances sont toutes des individus nommées (named individuals)
  - La règles SWRL DL-safe est ***équivalente*** à une règle SWRL normale comme :

  :hasParent(?x,?y), :hasBrother(?y,?z), :Nom(?x), :Nom(?y), :Nom(?z)

  -> :hasUncle(?x,?z)

2. Règles SWRL

- **Prédicats SWRL**
- **Principaux types de règles SWRL**

Prédicats SWRL

- Les règles SWRL peuvent utiliser comme prédicats **predicates** : des **class** ou des **property names**
- Les règles SWRL peuvent aussi utiliser **d'autres prédicats** comme :
  - **class expressions:** des expression de classes arbitraires, pas seulement des classes nommées.
  - **property expressions:** le seul opérateur disponible dans OWL 2 pour créer des expressions de propriétés est *inverse of object property*
  - **data range restrictions:** spécifie le type de la valeur de la donnée, comme *integer, date*, union quelques XML Schema types, enumerated type
  - **sameIndividual** et **differentIndividuals:** pour spécifier des individus identiques ou différents

  et :
  - ***core SWRL built-ins:*** *predicates spéciaux définis dans SWRL qui peuvent manipuler les valeurs de données, par exemple pour additionner des nombres*
  - ***custom SWRL built-ins:*** *on peut définir ses propres en utilisant du code Java*

Principaux types de règles SWRL

Avec ces prédicats possibles, les principaux types de règles SWRL sont :

- **SWRL rules with class expressions**
- **SWRL rules with data range restrictions**
- SWRL rules with SWRL core built-ins
- SWRL rules with custom SWRL built-ins (using Java code)

SWRL Rules **Syntaxes**

- Syntaxe Protégé:

aChild(?x,?y) ∧ isBrotherOf(?z,?x) -> isUncleOf(?z,?y)

- Syntaxe SWRL (functional syntax):

```
DLSafeRule(
  Annotation(rdfs:comment "Uncle rule")
  Body(
     ClassAtom( :aChild Variable(var:x) Variable(var :y) )
     ClassAtom( :isBrotherOf Variable(var:z) Variable(var :x) )
  )
  Head(
     ClassAtom( :isUncleOf Variable(var:z) Variable(var :y))
  )
)
```


Example of SWRL Rules **with class expressions**

- **Syntaxe Protégé :**

Person(?x), hasChild min 1 Person(?x) -> Parent(?x)

- **Signification :** les individus de la classe *Person*, qui ont au moins un enfant un individu de la classe *Person* doivent alors faire partie de la classe *Parent* (en d'autres termes, les personnes ayant au moins un enfant sont des parents)
- **Syntaxe SWRL:**

```
DLSafeRule(
  Annotation(rdfs:comment "Rule with class expression")
  Body(
     ClassAtom( :Person Variable(var:x) )
     ClassAtom(ObjectMinCardinality( 1 :hasChild :Person )
           Variable(var:x)
     )
  )
  Head(
     ClassAtom( :Parent Variable(var:x) )
  )
)
```


Example of SWRL Rules with data restrictions

- **Syntaxe Protégé :**

Person(?p), integer[>= 18 , <= 65](?age), hasAge(?p, ?age) -> hasDriverAge(?p, true)

- **Meaning :** La « data range restriction » est satisfait lorsque la vairable *?age* a pour valeur un entier entre 18 et 65 inclus.
- **In SWRL syntax (functional syntax):**

```
DLSafeRule
    (Annotation(rdfs:comment "Rule with data range restriction")
    Body(
      ClassAtom(:Person Variable(var:p))
      DataPropertyAtom(:hasAge Variable(var:p) Variable(var:age))
      DataRangeAtom(
        DatatypeRestriction(xsd:integer xsd:minInclusive "18"^^xsd:integer
              xsd:maxInclusive "65"^^xsd:integer) Variable(var:age)) )
    Head(
      DataPropertyAtom(:hasDriverAge Variable(var:p) "true"^^xsd:boolean)
    )
)
```

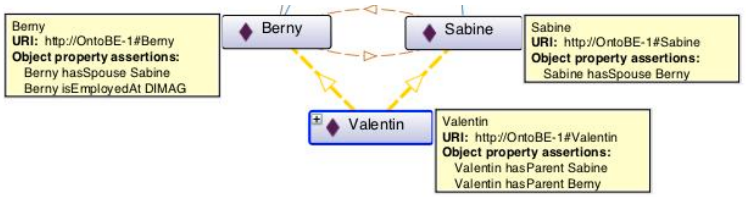
3. Exemples d'utilisation de règles SWRL

- **Spécifications (en Protégé et syntaxe SWRL)**
- **Inférences avec règles**

Exemple de règle SWRL (1)

Ex :

- *Valentin* est le fils de *Berny* et *Sabine*
- La propriété symétrique *hasSpouse* relie *Berny* et *Sabine*:




```
SymmetricObjectProperty(:hasSpouse)
ObjectPropertyAssertion(:hasSpouse :Berny :Sabine)
ObjectPropertyAssertion(:hasParent :Valentin :Berny)
ObjectPropertyAssertion(:hasParent :Valentin :Sabine)
```


Exemple de règle SWRL (2)

- **Syntaxe Protégé :**

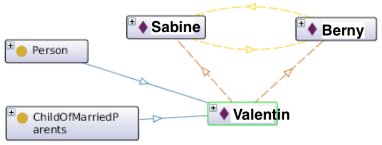
  Person(?x), hasParent(?x, ?y), hasParent(?x, ?z), hasSpouse(?y, ?z) -> ChildOfMarriedParents(?x)

  (In Protege, in Active Ontology onglet, in the Rule window, we add this expression)
- **Signification :** un individu X de la classe *Person* qui a des parents Y et Z tels que Y a l'épouse Z, appartient à la nouvelle classe *ChildOfMarriedParents*
- **Syntaxe SWRL (Functional syntax) :**

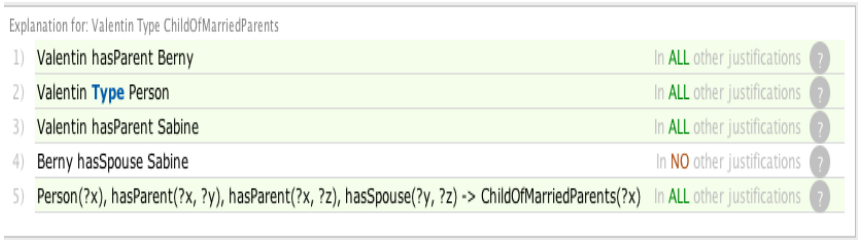
```
Prefix(var:=<urn:swrl#>)
Declaration( Class( :ChildOfMarriedParents ) )
SubClassOf( :ChildOfMarriedParents :Person )
DLSafeRule(
  Body(
    ClassAtom( :Person Variable(var:x))
    ObjectPropertyAtom(:hasParent Variable(var:x) Variable(var:y))
    ObjectPropertyAtom(:hasParent Variable(var:x) Variable(var:z))
    ObjectPropertyAtom(:hasSpouse Variable(var:y) Variable(var:z))
  )
  Head(
    ClassAtom( :ChildOfMarriedParents Variable(var:x))
  )
)
```


Exemple de règle SWRL (3)

Quand on utilise le raisonneur, il infère que *Valentin* appartient à la *ChildOfMarriedParents*, et même explique pourquoi :

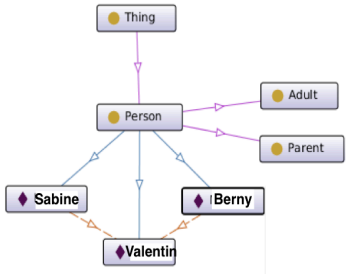



```
ClassAssertion( :ChildOfMarriedParents :Valentin )
```



Exemple de règle SWRL (4)

Soit une ontologie définissant leur anniversaire :




```
Declaration(Class(:Person))
Declaration(ObjectProperty(:hasChild))
Declaration(Class(:Parent)) Declaration(Class(:Adult))
SubClassOf(:Adult :Person) SubClassOf(:Parent :Person)
Declaration(DataProperty(:bornOnDate))
Declaration(DataProperty(:bornInYear))
Declaration(DataProperty(:hasAge))
FunctionalDataProperty(:hasAge)
Declaration(DataProperty(:hasDriverAge))

Declaration(NamedIndividual(:Berny))
ClassAssertion(:Person :Berny)
DataPropertyAssertion(:bornOnDate :Berny "1954-06-
09"^^xsd:date)

Declaration(NamedIndividual(:Sabine))
ClassAssertion(:Person :Sabine)
DataPropertyAssertion(:bornOnDate :Sabine "1962-07-
08"^^xsd:date)

Declaration(NamedIndividual(:Valentin))
ClassAssertion(:Person :Valentin)
DataPropertyAssertion(:bornOnDate :Valentin "1998-08-
17"^^xsd:date)

ObjectPropertyAssertion(:hasChild :Berny :Valentin)
ObjectPropertyAssertion(:hasChild :Sabine :Valentin)
```

Cette ontologie est écrite en functional syntax que Protégé peut charger directement

Examples of SWRL rules (5)

L'ontologie défini aussi quelques règles SWRL :

Rule 1:
DLSafeRule(Annotation(rdfs:comment "**Rule computing hasAge from bornInYear**") **Body**(ClassAtom(:Person Variable(var:p)) DataPropertyAtom(:bornInYear Variable(var:p) Variable(var:year)) BuiltInAtom(my:thisYear Variable(var:nowyear)) BuiltInAtom(swrlb:subtract Variable(var:age) Variable(var:nowyear) Variable(var:year)))
Head(DataPropertyAtom(:hasAge Variable(var:p) Variable(var:age))))

Rule 2:
DLSafeRule(Annotation(rdfs:comment "**Rule with core builtin swrlb:greaterThan**") **Body**(ClassAtom(:Person Variable(var:p)) DataPropertyAtom(:hasAge Variable(var:p) Variable(var:age)) BuiltInAtom(swrlb:greaterThan Variable(var:age) "18"^^xsd:integer))**Head**(ClassAtom(:Adult Variable(var:p))))

Rule 3:
DLSafeRule(Annotation(rdfs:comment "**Rule with data range restriction**") **Body**(ClassAtom(:Person Variable(var:p)) DataPropertyAtom(:hasAge Variable(var:p) Variable(var:age)) DataRangeAtom(DatatypeRestriction(xsd:integer xsd:minInclusive "18"^^xsd:integer xsd:maxInclusive "65"^^xsd:integer) Variable(var:age)))**Head**(DataPropertyAtom(:hasDriverAge Variable(var:p) "true"^^xsd:boolean)))

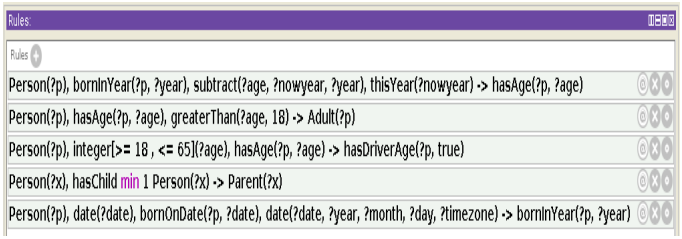
Rule 4:
DLSafeRule(Annotation(rdfs:comment "**Rule with class expression**") **Body**(ClassAtom(ObjectMinCardinality(1 :hasChild :Person) Variable(var:x)) ClassAtom(:Person Variable(var:x)))**Head**(ClassAtom(:Parent Variable(var:x))))

Rule 5:
DLSafeRule(Annotation(rdfs:comment "**Rule computing bornInYear from bornOnDate**")
Body(ClassAtom(:Person Variable(var:p)) DataPropertyAtom(:bornOnDate Variable(var:p) Variable(var:date)) DataRangeAtom(xsd:date Variable(var:date)) BuiltInAtom(swrlb:date Variable(var:date) Variable(var:year) Variable(var:month) Variable(var:day) Variable(var:timezone)))**Head**(DataPropertyAtom(:bornInYear Variable(var:p) Variable(var:year))))

Exemples de règle SWRL (6)

Dans Protégé, ces règles sont spécifiées selon une variante de la syntaxe *Manchester* sans *namespace qualifiers :*

- R1- Person(?p), swrlb:subtract(?age, ?nowyear, ?year), thisYear(?nowyear), bornInYear(?p, ?year) -> hasAge(?p, ?age)
- R2 - Person(?p), hasAge(?p, ?age), swrlb:greaterThan(?age, 18) -> Adult(?p)
- R3 - Person(?p), hasAge(?p, ?age), xsd:integer[>= 18 , <= 65](?age) -> hasDriverAge(?p, true)
- **R4 - Person(?x), (hasChild min 1 Person)(?x) -> Parent(?x)**
- R5 - Person(?p), swrlb:date(?date, ?year, ?month, ?day, ?timezone), bornOnDate(?p, ?date), xsd:date(?date) -> bornInYear(?p, ?year)



Exemples de règle SWRL (7)

Rule 4:

- **Syntaxe Protege :**

```
Person(?x), hasChild min 1 Person(?x) -> Parent(?x)
```

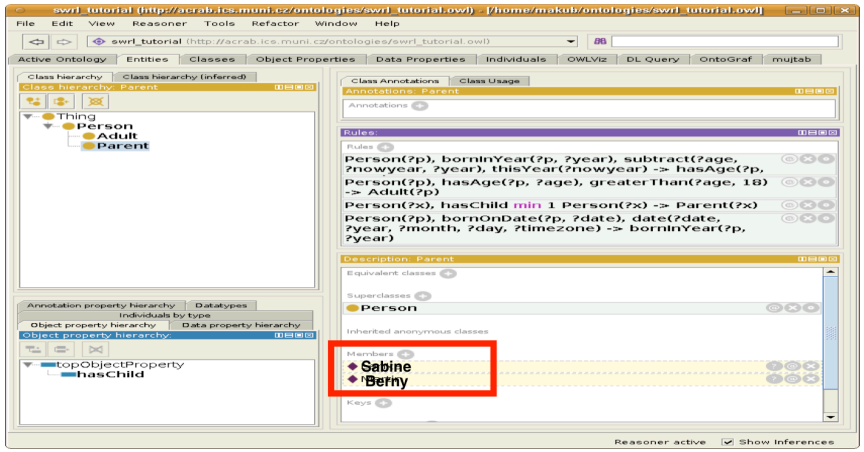
- **Signification** : les individus qui sont de la classe *Person*, et qui ont au moins une propriété *hasChild* avec un individu de la classe *Person*, doivent aussi être de la classe *ParentClass (personnes qui ont au moins un enfant sont des parents)*

- **Functional syntax:**

```
DLSafeRule(
    Annotation(rdfs:comment "Rule with class expression")
    Body(
          ClassAtom( :Person Variable(var:x) )
          ClassAtom(
                ObjectMinCardinality( 1 :hasChild :Person )
                Variable(var:x)
          )
    )
    Head(
         ClassAtom( :Parent Variable(var:x) )
    )
)
```

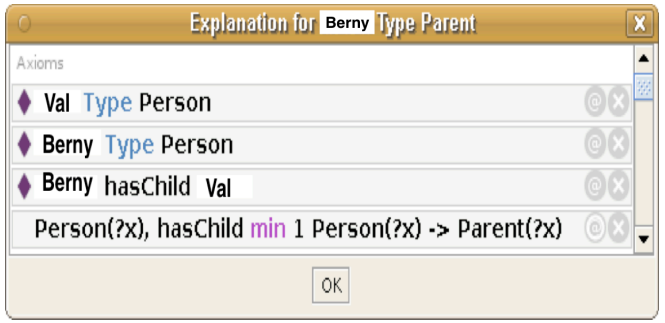
Exemples de règle SWRL (8)

- Dans Protégé, le **raisonneur** (Pellet) déduira que *Berny* et *Sabine* sont dans la classe *Parent*, comme on peut le voir dans le **coin inférieur droit** :



Exemples de règle SWRL (9)

- Dans Protégé, lorsque vous cliquez sur le point d'interrogation **(?)** à droite de l'information déduite, le **raisonneur fournit même des informations sur la manière dont cette information a été déduite** :



4. Limite de SWRL

- Limite de SWRL

Limite de SWRL : décidabilité

- Des règles SWRL arbitraires peuvent conduire à de l'indécidabilité.

  ⇨Aussi seulement des règles SWRL appelées **DL-safe** sont implémentées dans des raisonneurs

- **Règles DL-safe** :
  - Sont des règles **appliquées** seulement **à des individus nommés**
  - Elles ne **s'appliquent pas** à **des individus qui ne sont pas nommés**, mais connu comme existant