

Master 2 Info & Maths-Info
Bases de données spécialisées

Cristina Sirangelo
IRIF, Université Paris Cité
cristina@irif.fr

Sémantique RDFS

Credits (matériel, transparents et exemples empruntés)

- Transparents du cours LODAS , Bernd Amman, Univ. Pierre et Marie Curie
- Transparents du cours “Semantic technologies” Univ. Of Oslo
- MOOC FUN INRIA Web Sémantique - Corby, Gandon, Faron
- Livre : Foundations of Semantic Web Technologies - Hitzler, Krotzsch, Rudolf
- Livre : Web data management - Abiteboul, Manolescu, Rousset, Senellart

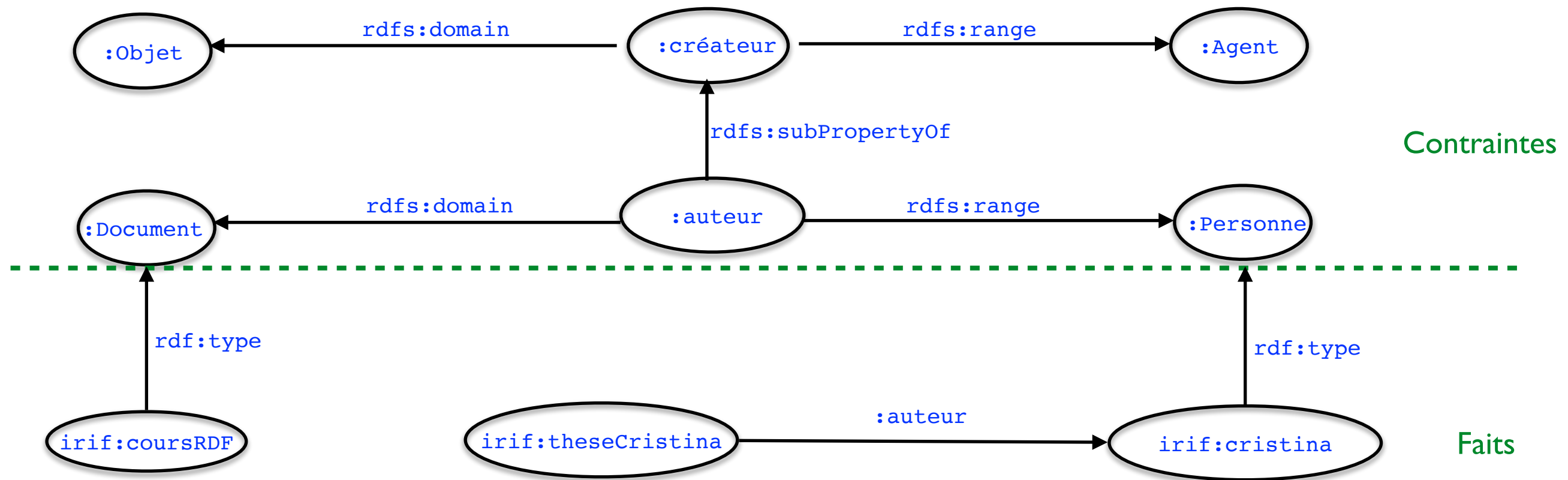
Rappel : ontologie (ou base de connaissance)

- Rappel : une **ontologie** K
 - ▶ est définie sur un **vocabulaire (ou schema)** $\langle C, P \rangle$: i.e. l'ensemble C des classes et l'ensemble P des propriétés qu'elle décrit
 - ▶ K comprends :
 - Un ensemble A de **faits**, c-a-d un ensemble d'instances des classes et des propriétés de $C \cup P$
(en RDF : des triplets $\langle s \text{ rdf:type } c \rangle$ avec $c \in C$
et des triplets $\langle s \text{ } p \text{ } o \rangle$ avec $p \in P$)
 - Un ensemble T de **contraintes** c-a-d des relations entre classes et propriété du vocabulaire $\langle C, P \rangle$
(en RDFS : des triplets $\langle s \text{ } p \text{ } o \rangle$ avec $s, o \in C \cup P$
et $p \in \{ \text{rdfs:subClassOf}, \text{rdfs:subPropertyOf}, \text{rdfs:domain}, \text{rdfs:range} \})$

Rappel : exemple d'ontologie RDFS

- Syntaxe = P = { :auteur, :createur } C = { :Objet, :Document, :Agent, :Personne }

- ▶ sous forme de graphe



- sous forme de triplets (équivalent)

Contraintes

```
:createur rdfs:domain :Objet
:createur rdfs:range :Agent
:auteur rdfs:domain :Document
:auteur rdfs:range :Personne
:auteur rdfs:subPropertyOf :createur
```

Faits

```
irif:theseCristina :auteur irif:cristina
irif:cristina rdf:type :Personne
irif:coursRDF rdf:type :Document
```

Sémantique d'une ontologie

- Une ontologie (par exemple RDFS) est une description syntaxique (un ensemble de triplets décrivant les faits et les contraintes)
- Mais on peut lui associer une **sémantique** :
 - ▶ Intuitivement une ontologie représente toutes les façon possibles de “peupler” les classes et les propriétés de telle sorte qu’elles :
 - contiennent la base de faits
 - satisfassent les contraintes
 - ▶ Chacune de ces façons possibles est appelée **modèle** de l’ontologie
 - ▶ Les modèles ne sont pas matérialisés, ils représentent des données implicites qu’on peut dériver

Sémantique d'une ontologie RDFS

- Que veut dire “satisfaire les contraintes” pour RDFS?
 - Nécessite une sémantique formelle
 - Mais **intuitivement** la sémantique attendue des contraintes RDFS est celle suggérée par leur noms, par exemple :

`:createur rdfs:range :Agent .`

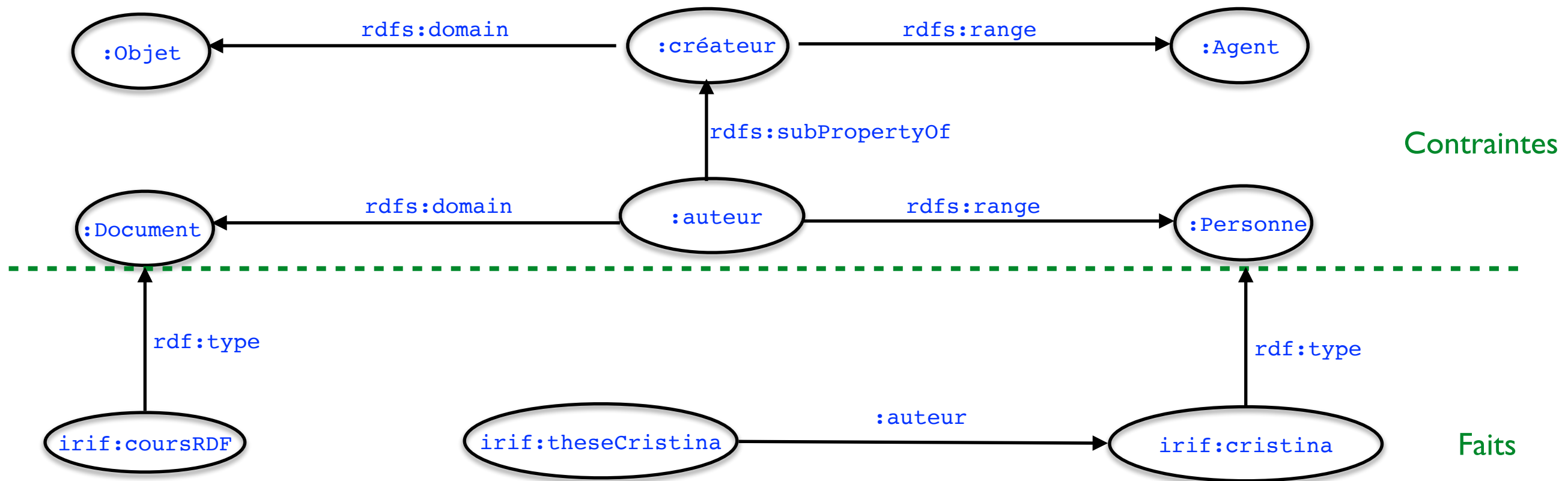
Sémantique (intuitive) :

dans tout triplet `s :createur p`, `p` doit être de classe `:Agent`

Exemple : sémantique intuitive

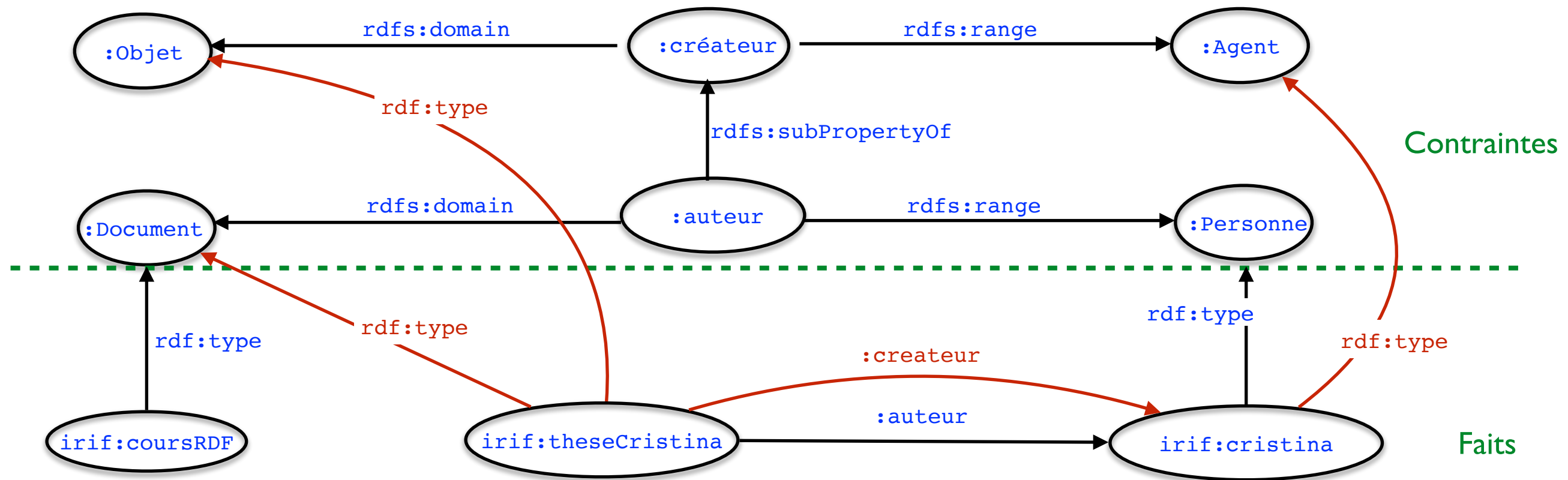
- $P = \{ :auteur, :createur \}$ $C = \{ :Objet, :Document, :Agent, :Personne \}$

K



Exemple : sémantique intuitive

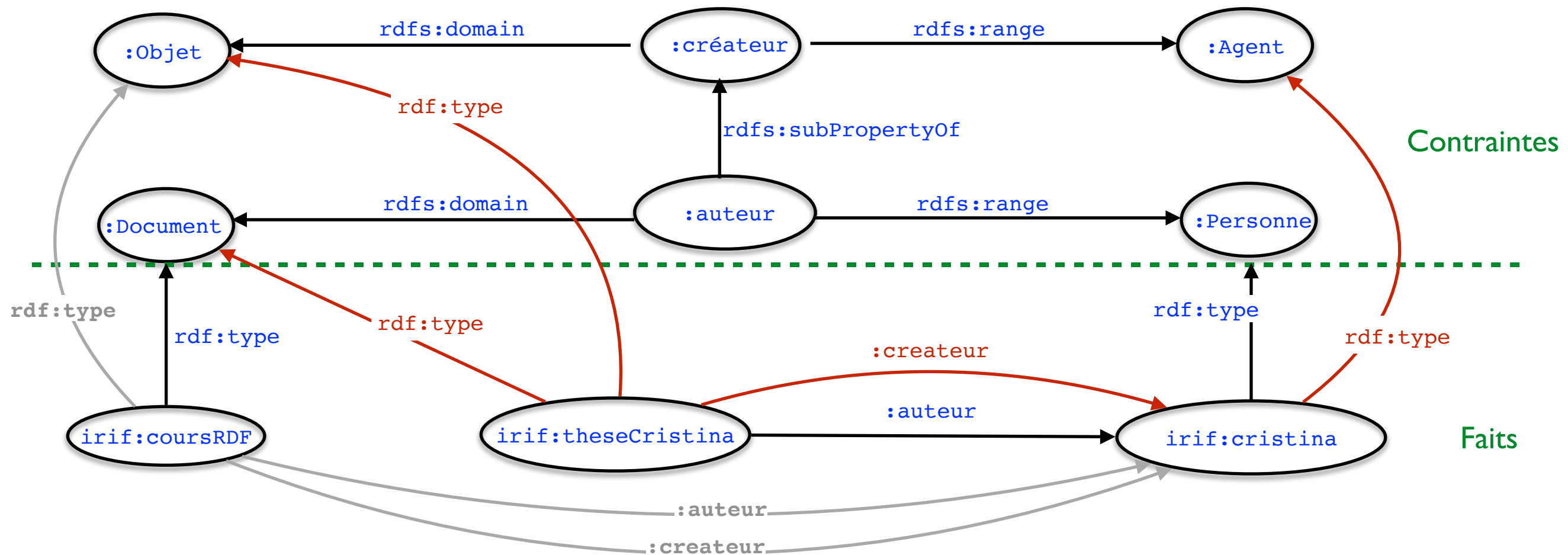
- $P = \{ :auteur, :createur \}$ $C = \{ :Objet, :Document, :Agent, :Personne \}$



tout modèle de K devra contenir au moins les nouveaux faits en rouge, en plus des fait initiaux, autrement les contraintes ne sont pas satisfaites

Plusieurs modèles d'une même ontologie

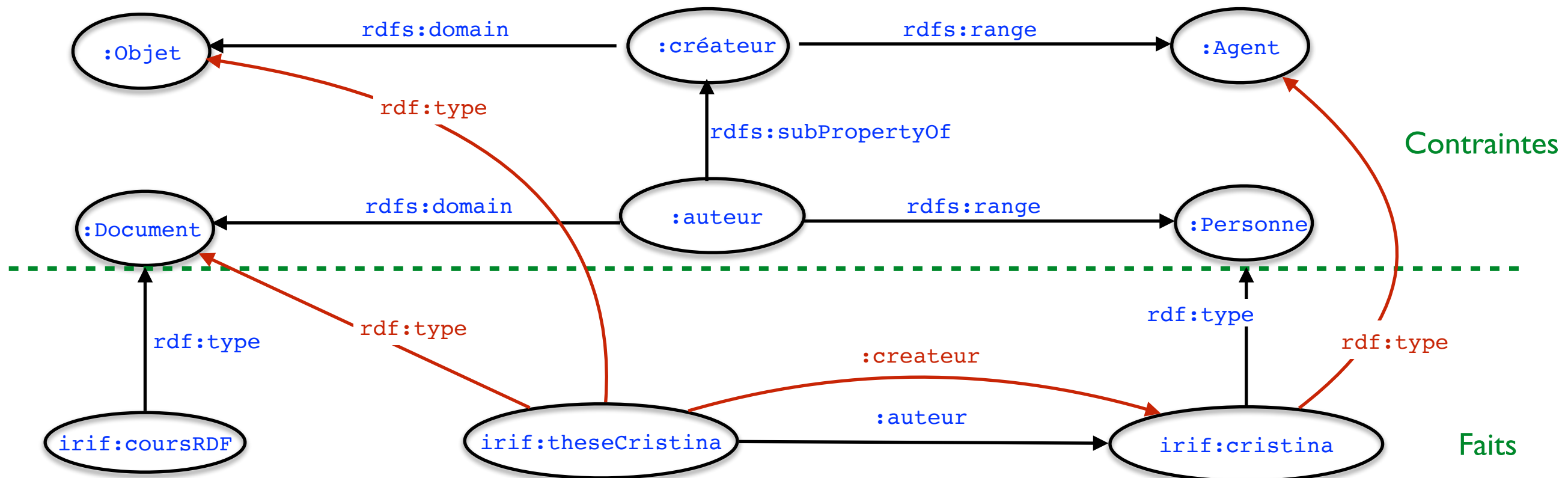
- Une ontologie admet en général plusieurs modèles. Exemple :



tout modèle de K devra contenir au moins les nouveaux faits en rouge, en plus des fait initiaux, mais peut contenir ou pas les faits en gris par exemple

Conséquences d'une ontologie

- Une ontologie admet en general plusieurs modèles
- Mais on s'intéresse à **ce qui est vrai dans tout modèle** (appelé **conséquences** de l'ontologie, ou **inférences**)
- Cela représente intuitivement ce qu'on peut derivier avec certitude de ce qui est connu (i.e de la "connaissance")
- Exemple : les fait initiaux ainsi que les faits en rouges sont des consequences



Ontologie RDFS prédéfinie

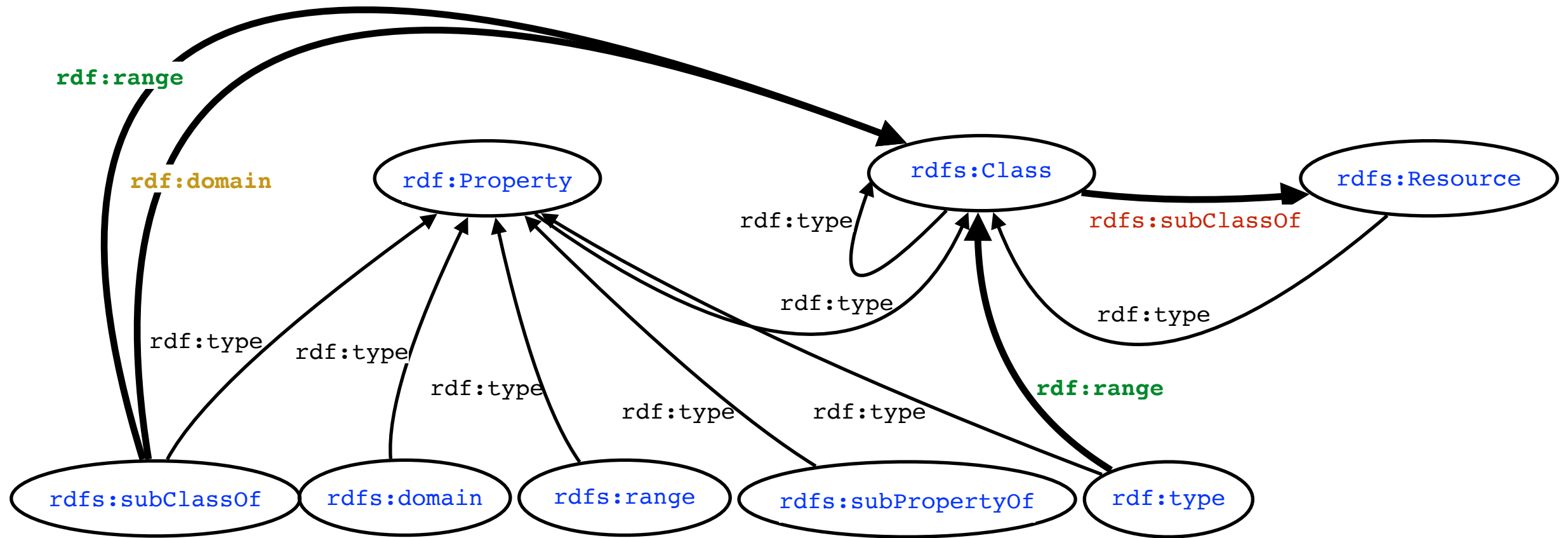
- On souhaite avoir des systèmes capables de raisonner (deriver des conséquences) en utilisant un ensemble fixé de contraintes
- C'est pour cette raison que RDFS exprime les contraintes comme données
 - ▶ C-a-d en RDFS les contraintes sur un vocabulaire $\langle C, P \rangle$ sont exprimées comme des faits, en étendant $\langle C, P \rangle$ par de nouvelles classes et propriétés d'un vocabulaire prédéfini (`rdf:` et `rdfs:`)
 - `rdfs:domain`, `rdfs:subPropertyOf`, `rdfs:Class`, `rdf:Property` etc.
- La sémantique de ces nouvelles classes et propriétés est donnée par l'**ontologie RDFS prédéfinie**

Ontologie RDFS prédéfinie - suite

- **Ontologie RDFS prédéfinie :**
 - ▶ Un ensemble de triplets de base et de contraintes fixées nécessaires à spécifier la sémantique des classes et propriétés du vocabulaire `rdf:` / `rdfs:`
 - ▶ Si K est une ontologie exprimée en RDFS, faits **et** contraintes de K sont considérés comme faits de base de l'ontologie RDFS prédéfinie
 - ▶ **Ceci permet de raisonner avec un ensemble de contraintes fixé**
- Comporte :
 - ▶ triplets de base (axiomatiques)
 - ▶ contraintes logiques
- La liste exhaustive serait longues
 - ▶ description formelle (recommandation W3C) :
<https://www.w3.org/TR/rdf11-mt/>
 - ▶ (cf. aussi “Foundations of Sem. Web Technologies” - Hitzler, Krotzsch, Rudolf)
- On en donne un extrait en guise d'exemple

Ontologie RDFS prédéfinie - triplets axiomatiques

- Quelques triplets axiomatiques



▶ e.g.

- ▶ `rdfs:Class` contient toutes les classes
- ▶ toute ressource est dans la classe `rdf:Resource`
- ▶ L'objet de tout triplet `rdf:type` est une classe
- ▶ etc.

Ontologie RDFS prédéfinie - contraintes

- Quelques contraintes :

- ▶ $(A \text{ rdfs:subClassOf } B \wedge B \text{ rdfs:subClassOf } C) \Rightarrow A \text{ rdfs:subClassOf } C$

- ▶ $(r \text{ rdf:type } A \text{ et } A \text{ rdfs:subClassOf } B) \Rightarrow r \text{ rdf:type } B$

- ▶ $S \text{ rdf:type rdfs:Class} \Rightarrow S \text{ rdfs:subClassOf rdfs:Resource}$

- ▶ $s \text{ p o} \Rightarrow p \text{ rdf:type rdf:Property}$

- ▶ $(s \text{ p o} \wedge p \text{ rdfs:range } C) \Rightarrow o \text{ rdf:type } C$

Etc. etc.

- Ensemble avec les triplets axiomatiques servent à donner la sémantique attendue aux classes et propriétés du vocabulaire `rdf:/rdfs:`

Modèle RDFS

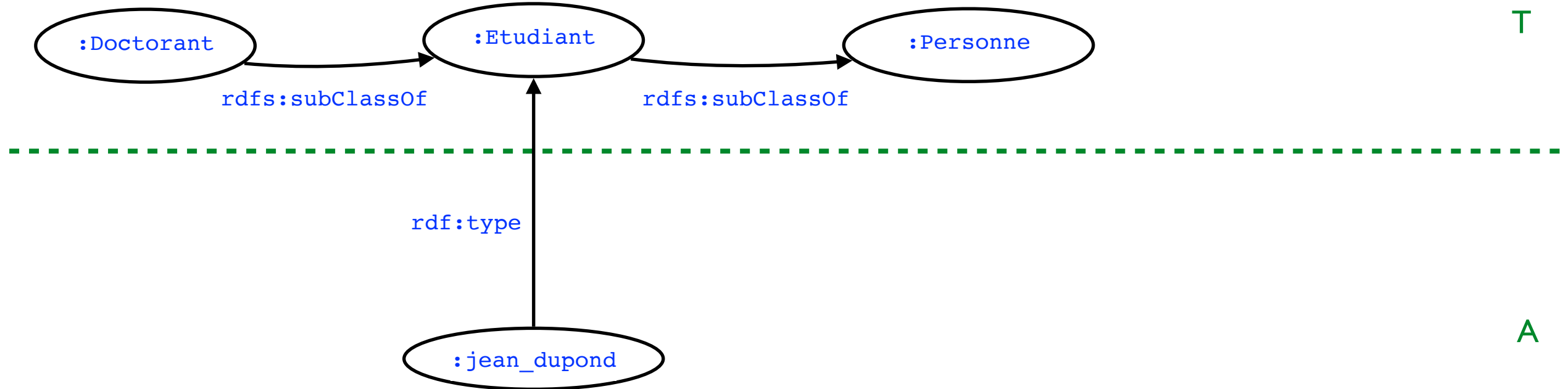
- Soit K une ontologie RDFS de schema $\langle C, P \rangle$
- Soit A l'ensemble de faits de K, et T l'ensemble des contraintes de K (triplets)
- K donne lieu une nouvelle ontologie ayant
 - ▶ **Vocabulaire** : P étendu avec les propriétés prédéfinies rdf/s (`rdf:Property`, `rdfs:domain`, etc.)
 - ▶ **Faits** : A, T et les triplets axiomatiques
 - ▶ **Contraintes** : les contraintes prédéfinies RDFS (cf slide "Ontologie RDFS prédéfinie - contraintes")

un **modèle RDFS** de K est modèle de cette nouvelle ontologie. i.e. un graphe RDF qui :

- ▶ contient les triplets A et T et les triplets axiomatiques,
- ▶ satisfait les contraintes prédéfinies RDFS

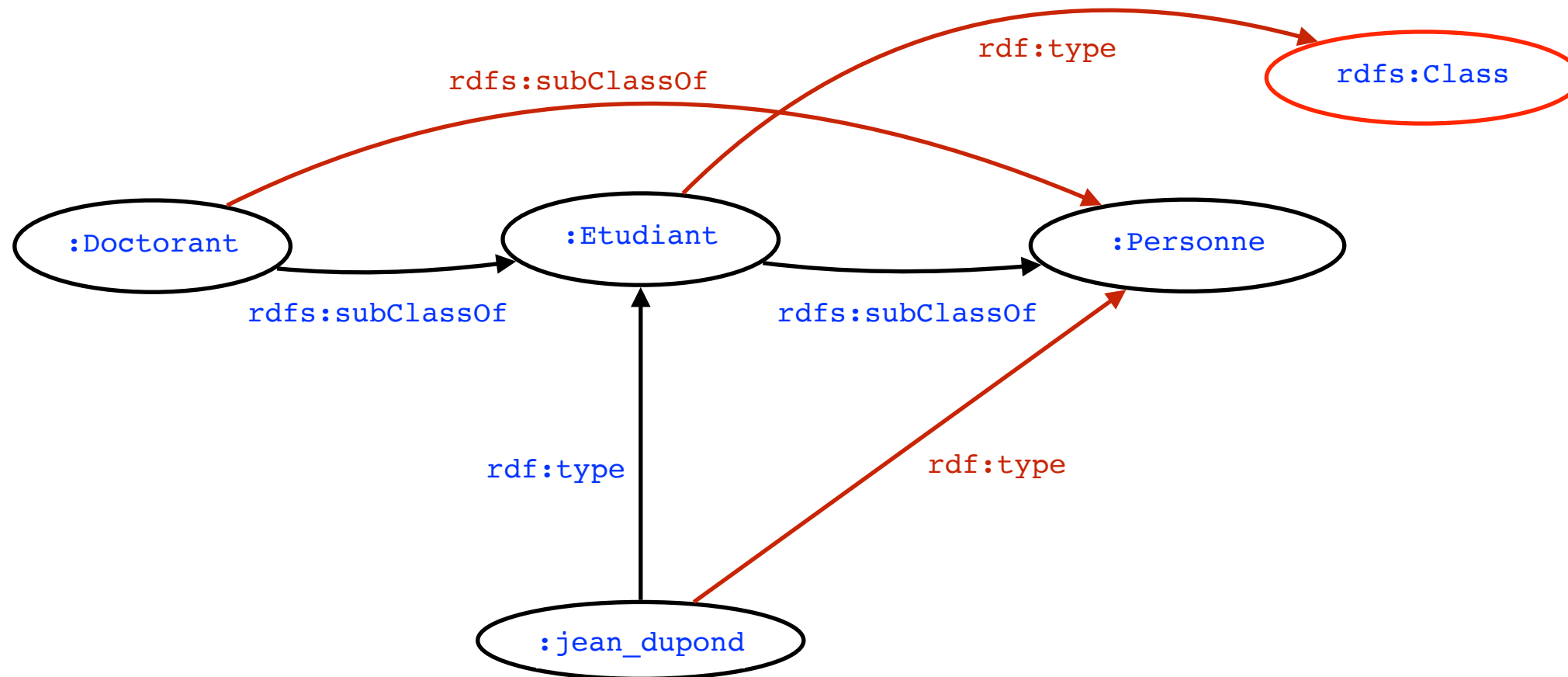
Exemple

- K



Exemple

- Les faits en rouge (entre autres) appartenant à un modèle RDFS de K



NB : Ce modèle contient bien d'autre triplets pas affichés

Remarque: `:Etudiant` `rdf:type` `rdfs:Class`

Est obtenu par le triplet axiomatique et la contrainte de range, cf. slide suivant,

Exemple - suite

- Triplet axiomatique :
 - ▶ `rdfs:subClassOf rdf:range rdfs:Class`
- Triplet dans T :
 - ▶ `:Doctorant rdfs:subClassOf :Etudiant`
- La contrainte RDFs :
 - ▶ $(s \ p \ o \wedge p \ rdfs:range \ C) \Rightarrow o \ rdf:type \ C$

Permet de dériver :

`:Etudiant rdf:type rdfs:Class`

mais ce triplet nous intéresse moins que les deux autres...
Pourquoi ?

Nouveaux faits et nouvelles contraintes de K

- Dans un modèle RDFS de K (de vocabulaire $\langle C, P \rangle$) on s'intéresse tout particulièrement aux triplets de la forme suivante :
 - **Nouveaux faits de K** : $\langle s \ p \ o \rangle$ avec $p \in P$
 $\langle s \ \text{rdf:type} \ S \rangle$ avec $S \in C$
 - **Nouvelles contraintes de K** : $\langle s \ p \ o \rangle$ avec $s, o \in C \cup P$ et $p \in \{\text{rdfs:subClassOf}, \text{rdfs:subPropertyOf}, \text{rdfs:domain}, \text{rdfs:range}\}$

`:Etudiant rdf:type rdfs:Class`

n'est ni un fait ni une contrainte du vocabulaire de K

Sémantique opérationnelle de RDFS

- Les contraintes RDFS sont toutes de la forme

$$(X \wedge Y) \Rightarrow Z$$

Exemple `(r rdf:type A et A rdfs:subClassOf B) \Rightarrow r rdf:type B`

- (appelées TDGS)
- Elle peuvent être vues comme **règles de derivation** (ou d'inférence) , et on les écrira comme

$$\frac{X \quad Y}{Z}$$

exemple

$$\frac{\begin{array}{l} <r \text{ rdf:type } A> \\ <A \text{ rdfs:subClassOf } B> \end{array}}{<r \text{ rdf:type } B>}$$

- ▶ Si un modèle contient les triplets X et Y il doit également contenir le triplet Z (autrement la contrainte n'est pas satisfaite)
- ▶ Donc l'application d'une règle de derivation génère un nouveau triplet qui est une conséquence de l'ontologie (i.e. présent dans tous les modèles RDFS)

Sémantique opérationnelle de RDFS

- Donnée une ontologie K avec faits A et contraintes T
- On part des triplets de base (qui doivent être présents dans tous les modèles RDFS)
 - ▶ triplets axiomatiques, A et T
- On peut utiliser les règles de derivations correspondantes aux contraintes RDFS comme **“générateurs” de nouveaux triplets**
- Ces nouveaux triplets doivent être obligatoirement présents dans tous les modèles RDFS de K,
 - ▶ C-a-d on ne derive **que des conséquences de K**
 - On dit alors que l'ensemble des règles de derivation est **correct** pour la derivation des consequences
 - On peut facilement voir qu'il est également **complet** (permet de derivier toutes les conséquences de K)
 - On en déduit que l'application successive des règles d'inference RDFS donne un algorithme de saturation pour calculer toutes les conséquences de l'ontologie

Algorithme de saturation

- Soit Γ l'ensemble des contraintes RDFS *

Saturation (Γ)

Input: $K = \langle A, T \rangle$ (faits A et contraintes T)

Output: toutes les conséquences de K

(1) $F \leftarrow A \cup T \cup$ triplets axiomatiques

(3) **repeat**

(4) **foreach** rule $\frac{\text{Condition}}{\text{Conclusion}}$ in Γ

(5) **if** there exists a substitution σ such that

(6) $\sigma(\text{Condition}) \in F$ and $\sigma(\text{Conclusion}) \notin F$

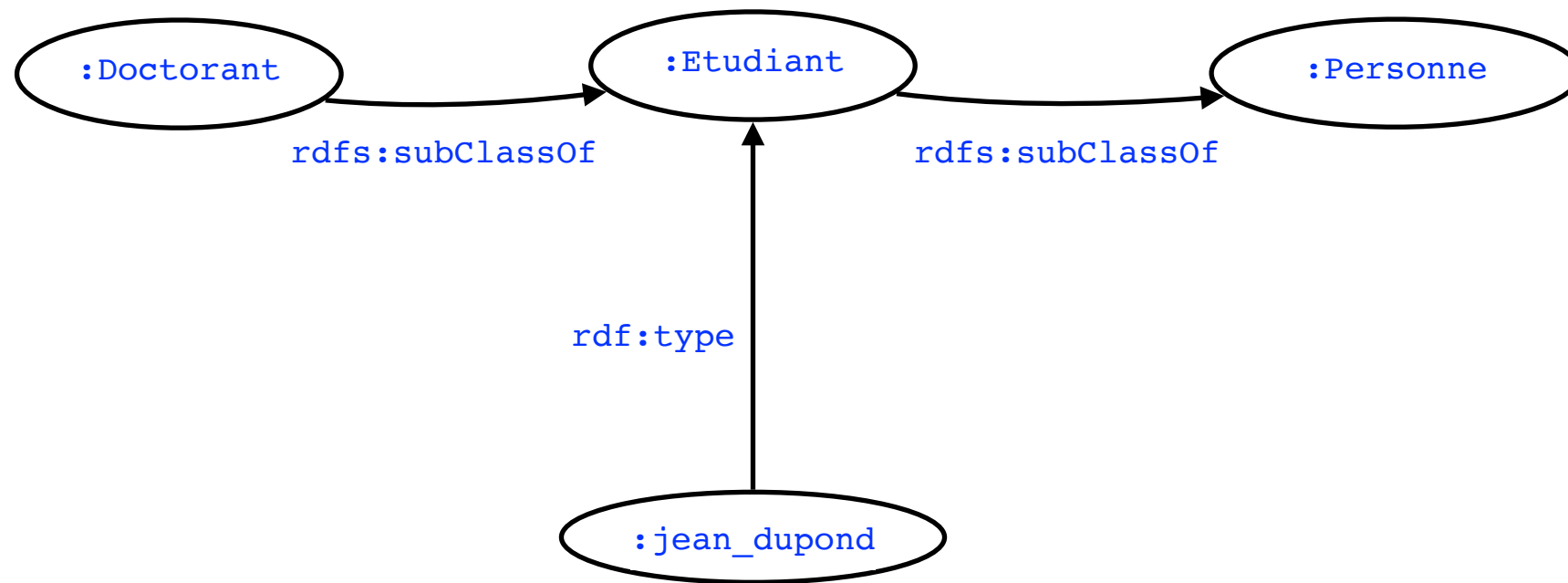
(7) add $\sigma(\text{Conclusion})$ to F

(8)

(10) **until** nothing added to F

(* Γ ici est l'ensemble des contraintes RDFS, mais l'algo est paramétré par un ensemble arbitraire de contraintes Γ)

Algorithme de saturation

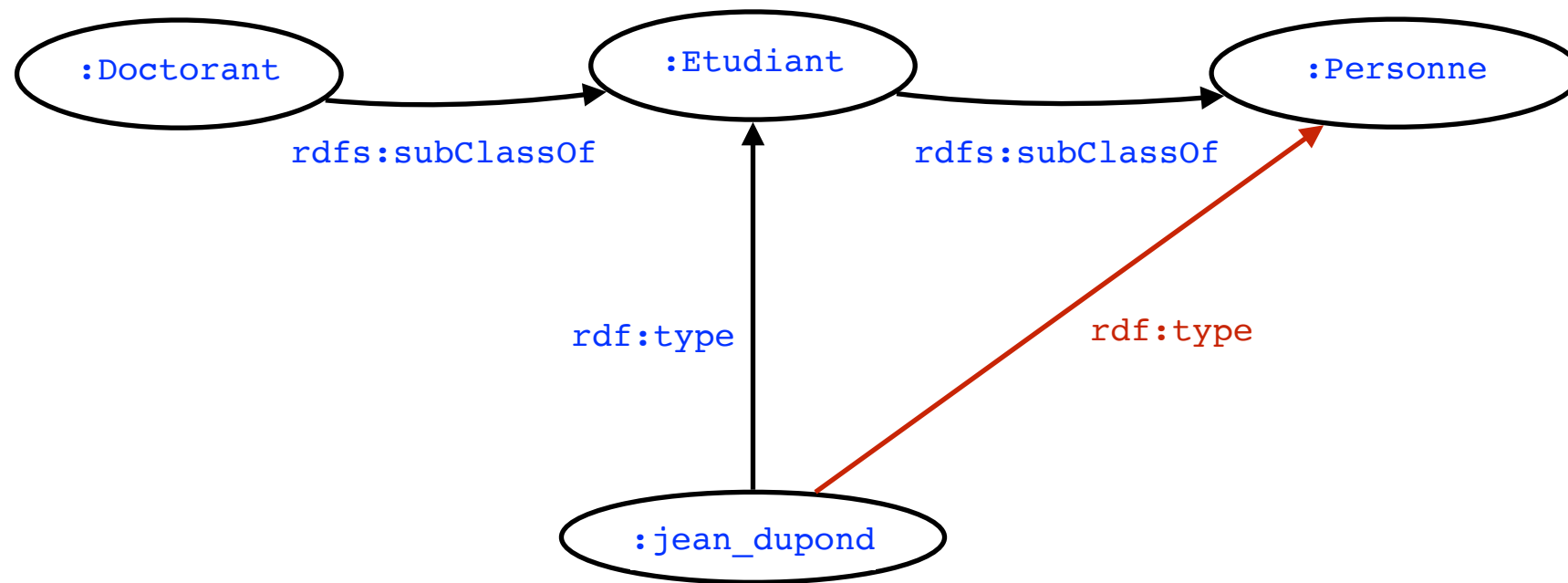


```
<r rdf:type A>  
<A rdfs:subClassOf B>
```

```
<r rdf:type B>
```

$\sigma(r) = \text{:jean_dupond}$
 $\sigma(A) = \text{:Etudiant}$
 $\sigma(B) = \text{:Personne}$

Algorithme de saturation

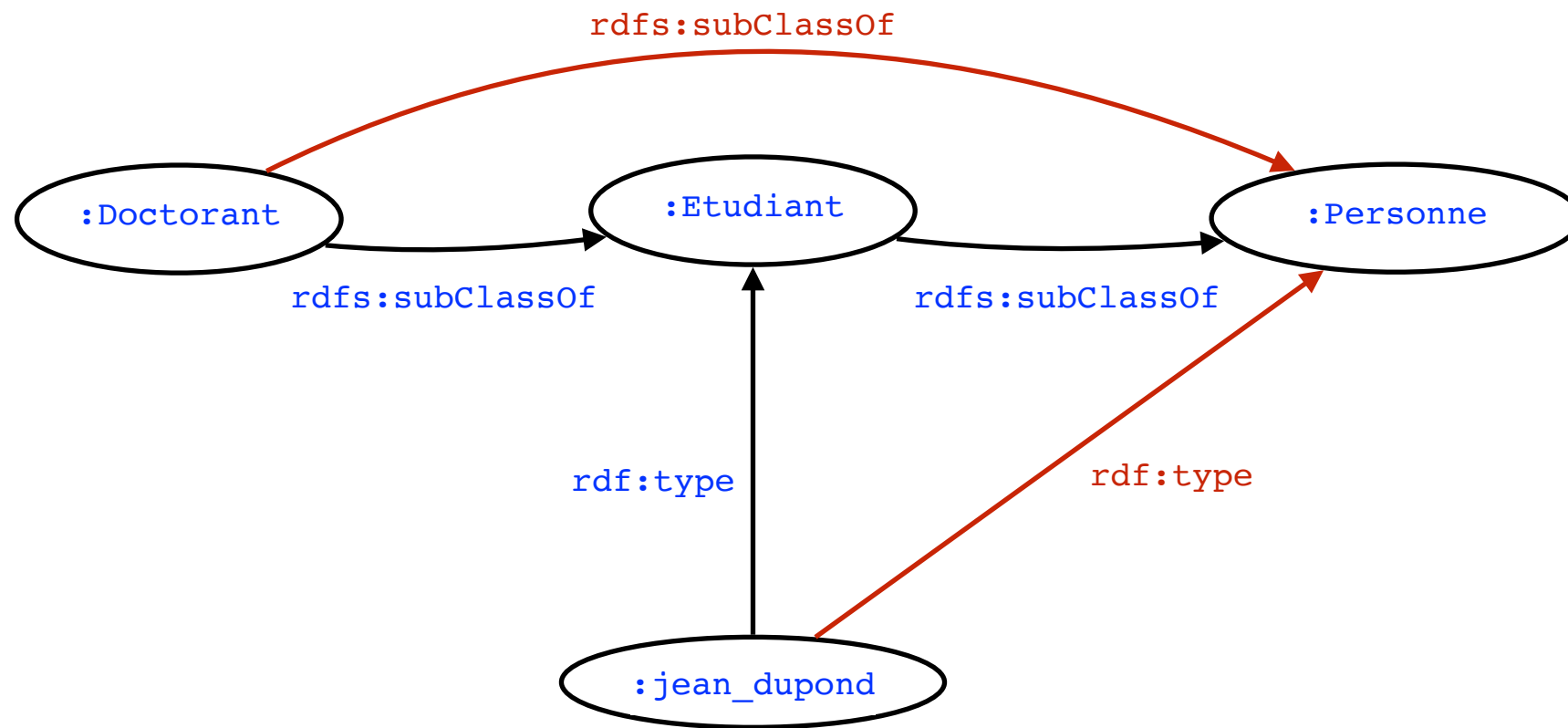


```
<r rdf:type A>  
<A rdfs:subClassOf B>
```

```
<r rdf:type B>
```

$\sigma(r) = \text{:jean_dupond}$
 $\sigma(A) = \text{:Etudiant}$
 $\sigma(B) = \text{:Personne}$

Algorithme de saturation



<A rdfs:subClassOf B>

<B rdfs:subClassOf B>

<A rdfs:subClassOf C>

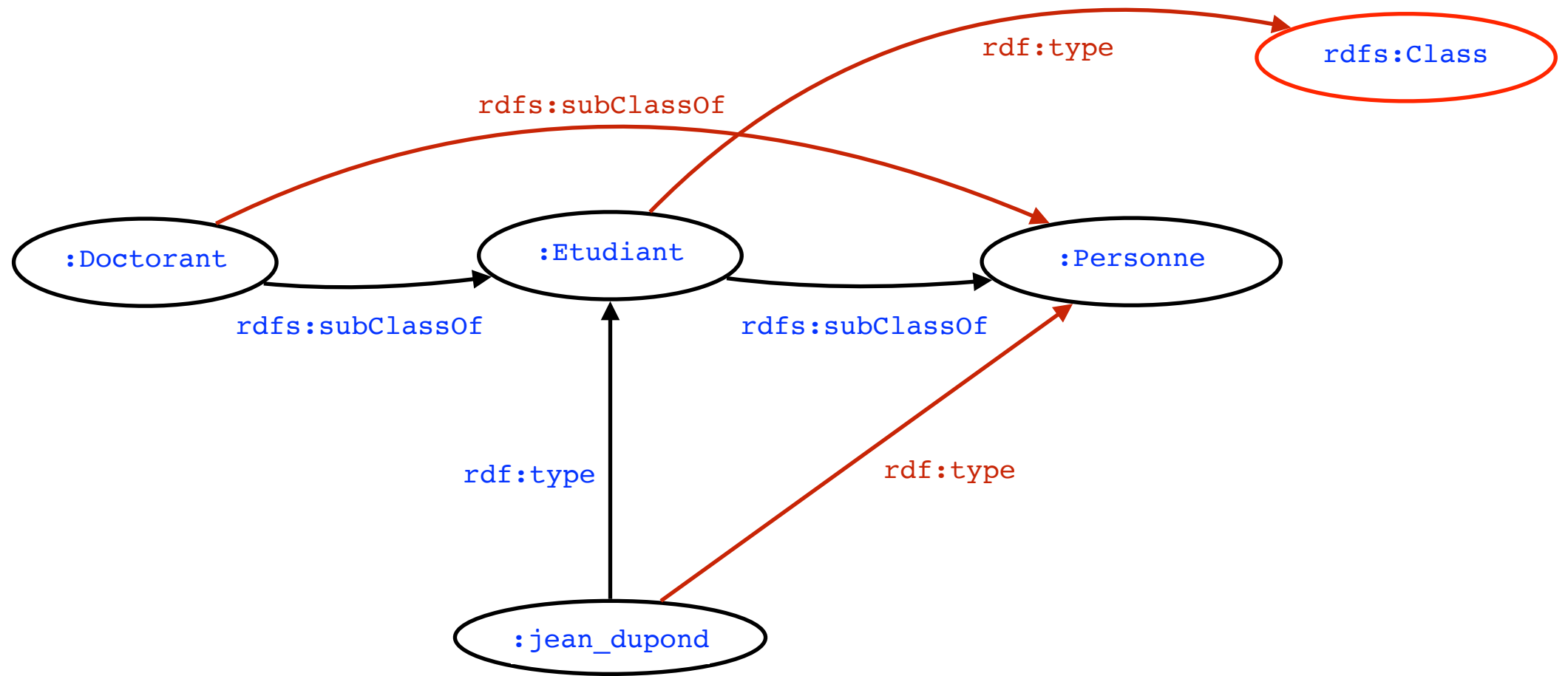
$\sigma(A) = \text{:Doctorant}$

$\sigma(B) = \text{:Etudiant}$

$\sigma(C) = \text{:Personne}$

Etc...

Algorithme de saturation



p rdf:range B
S p O

<S rdf:type B>

$\sigma(p) = rdfs:subClassOf$
 $\sigma(B) = rdfs:Class$
 $\sigma(O) = :Etudiant$
 $\sigma(S) = :Doctorant$

Etc...

Saturation partielle

- Souvent nous ne sommes pas intéressés à dériver toutes les conséquences
 - ▶ E.g : `createur rdf:type rdf:Property` moins intéressant que `irif:theseCristina :createur irif:cristina`
- On peut alors appliquer l'algorithme pour un ensemble restreints de règles de dérivation Γ
- Rappel : nous sommes souvent intéressés à dériver uniquement tous les faits et les contraintes dans le vocabulaire de K (cf. slide 19)
- On peut identifier un sous-ensemble des règles de dérivation RDFS qui est **correct et complet** pour la dérivation de tous les **faits** et toutes les **contraintes** du vocabulaire de K

Sémantique opérationnelle de RDFS : les faits

- Les règles d'inférence suivantes sont correctes et complètes pour la derivation de nouveaux **faits** * :

Γ_A

(rdfs9)

<pre><r rdf:type D> <D rdfs:subClassOf B></pre> <hr style="width: 50%; margin: 5px 0;"/> <pre><r rdf:type B></pre>
--

(rdfs7)

<pre><r P s> <P rdfs:subPropertyOf Q ></pre> <hr style="width: 50%; margin: 5px 0;"/> <pre><r Q s></pre>
--

(rdfs2)

<pre><r P s> <P rdfs:domain D ></pre> <hr style="width: 50%; margin: 5px 0;"/> <pre><r rdfs:type D></pre>

(rdfs3)

<pre><r P s> <P rdfs:range D ></pre> <hr style="width: 50%; margin: 5px 0;"/> <pre><s rdfs:type D></pre>
--

- Cela veut dire : pour une ontologie $K=(A,T)$ de schema $\langle C,P \rangle$
 Un fait (c-a-d $\langle s \ p \ o \rangle$ avec $p \in P$ ou $\langle s \ rdf:type \ S \rangle$ avec $S \in C$) est
 conséquence de K (c-a-d obligatoirement présent dans tous les modèles RDFS
 de K) ssi ce fait peut être obtenu par application successive de ces derivations à
 partir de $A \cup T$

* en l'absence de blank nodes

Sémantique opérationnelle de RDFS : les contraintes

- Les règles d'inférence suivantes sont correctes et complètes pour la derivation de nouvelles **contraintes**:

(rdfs11)

<pre><D rdfs:subClassOf B> <B rdfs:subClassOf E> ----- <D rdfs:subClassOf E></pre>
--

(rdfs5)

<pre><d rdfs:subPropertyOf b> <b rdfs:subPropertyOf e> ----- <d rdfs:subPropertyOf e></pre>

Γ_T

- Γ_T fournit la variante la plus faible de la sémantiques RDFs dans la quelle *domaine* et *co-domaine* d'une propriété sont uniquement ceux déclarés explicitement
- Des règles extensionnelles sont nécessaire si on veut imposer une sémantiques RDFS plus forte, dans la quelle domaine et co-domaine d'une propriété sont clos par super-ensemble

Sémantique opérationnelle de RDFS : les contraintes

- Les règles d'inférence suivantes sont correctes et complètes pour la derivation de nouvelles **contraintes** sous sémantiques extensionnelle:

Γ_T^{ext}

(rdfs11)

```
<D rdfs:subClassOf B>  
<B rdfs:subClassOf E>  
_____  
<D rdfs:subClassOf E>
```

(rdfs5)

```
<d rdfs:subPropertyOf b>  
<b rdfs:subPropertyOf e>  
_____  
<d rdfs:subPropertyOf e>
```

(ext1)

```
<p rdfs:domain D>  
<D rdfs:subClassOf B>  
_____  
<p rdfs:domain B>
```

(ext2)

```
<p rdfs:range D>  
<D rdfs:subClassOf B>  
_____  
<p rdfs:range B>
```

(ext3)

```
<p rdfs:domain D>  
<b rdfs:subPropertyOf p>  
_____  
<b rdfs:domain D>
```

(ext4)

```
<p rdfs:range D>  
<b rdfs:subPropertyOf p>  
_____  
<b rdfs:range D>
```

Sémantique opérationnelle de RDFS : les contraintes

- Les règles d'inférence Γ_T (Γ_T^{ext}) ou sont correctes et complètes pour la derivation de nouvelles **contraintes**

Cela veut dire : pour une ontologie $K=(A,T)$ de schema $\langle C,P \rangle$, une contrainte

$(c-a-d \langle s \ p \ o \rangle$ avec $s,o \in C \cup P$ et $p \in \{\text{rdfs:subClassOf}, \text{rdfs:subPropertyOf}, \text{rdfs:domain}, \text{rdfs:range}\})$

est consequence de K ssi

- ▶ ce triplet peut être obtenu par application successive des derivations Γ_T (Γ_T^{ext}) à partir de $A \cup T$
- ▶ ou bien s'il est une contrainte reflexive :

$D \text{ rdfs:subClassOf } D$ pour tout $D \in C$
 $p \text{ rdfs:subPropertyOf } p$ pour tout $p \in P$

Algorithme de saturation pour les faits et les contraintes

- On en derive que l'algorithme de saturation restraint aux règles d'inference Γ_A et Γ_T (ou Γ_T^{ext}) et ignorant les triplets axiomatiques, calcule **tous les faits et contraintes** impliquées par (i.e conséquences de) une ontologie RDFS

Algorithme de saturation RDFS pour les faits et contraintes

Saturation (Γ_T, Γ_A)

Input: $K = \langle A, T \rangle$ (faits A et contraintes T)

Output: Les faits et contraintes conséquences de K

(1) $F \leftarrow A \cup T$

(3) **repeat**

(4) **foreach** rule $\frac{\text{Condition}}{\text{Conclusion}}$ in $\Gamma_A \cup \Gamma_T$

(5) **if** there exists a substitution σ such that

(6) $\sigma(\text{Condition}) \in F$ and $\sigma(\text{Conclusion}) \notin F$

(7) add $\sigma(\text{Conclusion})$ to F

(8)

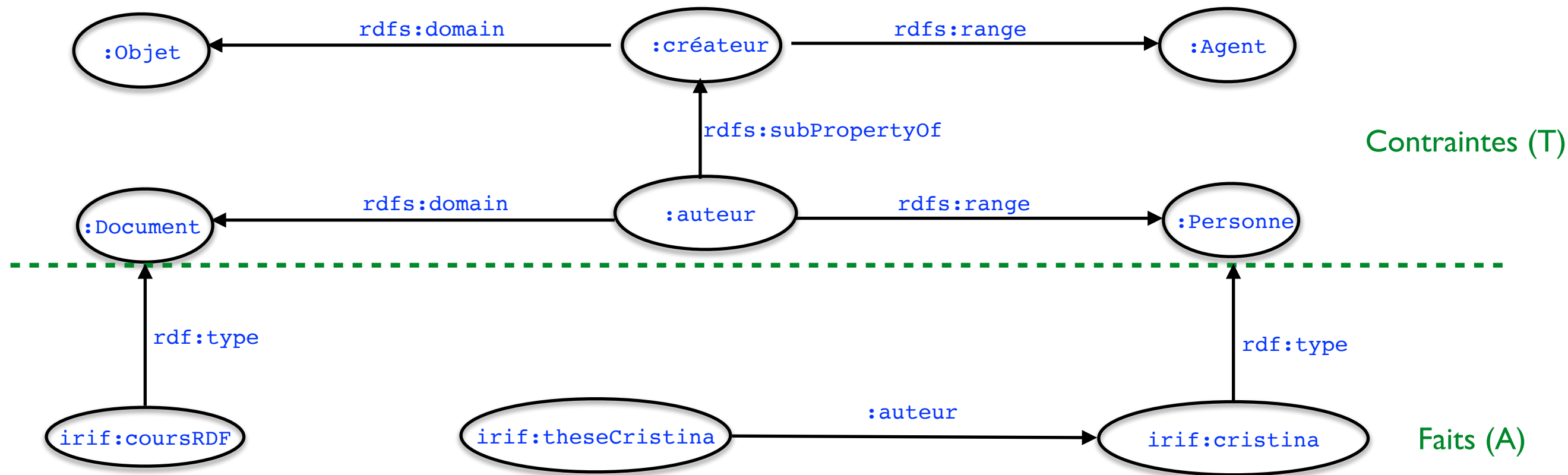
(10) **until** nothing added to F

Pour utiliser la sémantique extensionnelle utiliser Saturation ($\Gamma_T^{\text{ext}}, \Gamma_A$)

Exemple : algorithme de saturation pour les faits et les contraintes

- $P = \{ :auteur, :createur \}$ $C = \{ :Objet, :Document, :Agent, :Personne \}$

F

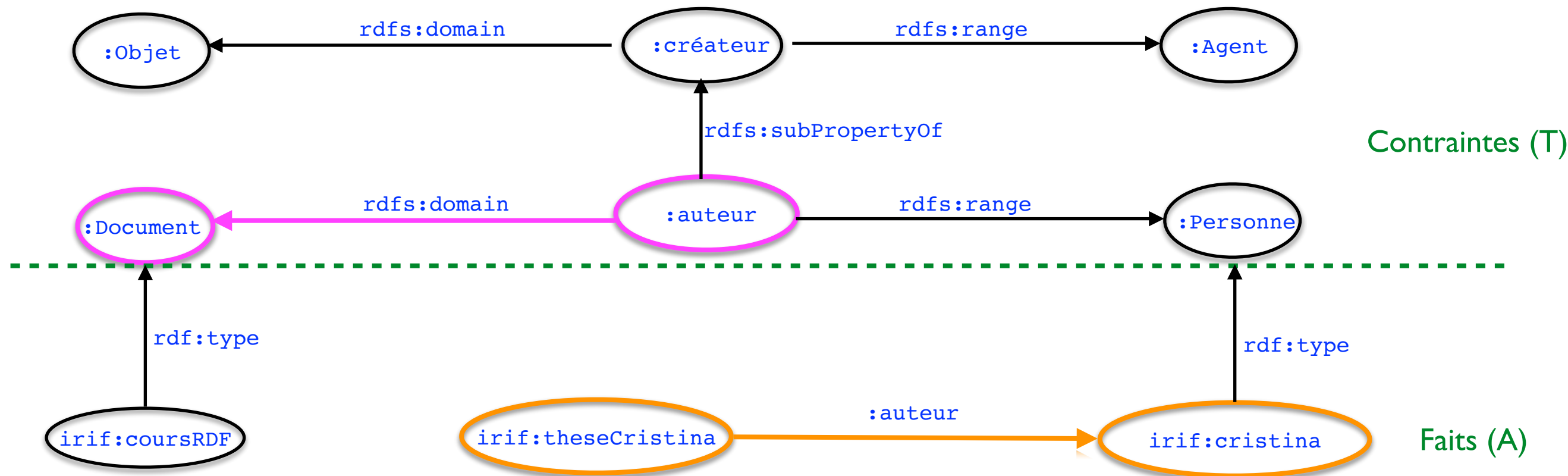


$F := A \cup T$

Exemple : algorithme de saturation pour les faits et les contraintes

- $P = \{ :auteur, :createur \}$ $C = \{ :Objet, :Document, :Agent, :Personne \}$

F



$F := A \cup T$

I. La regle

(rdfs2)

$\langle r \ P \ s \rangle$
$\langle P \ rdfs:domain \ D \rangle$
$\langle r \ rdfs:type \ D \rangle$

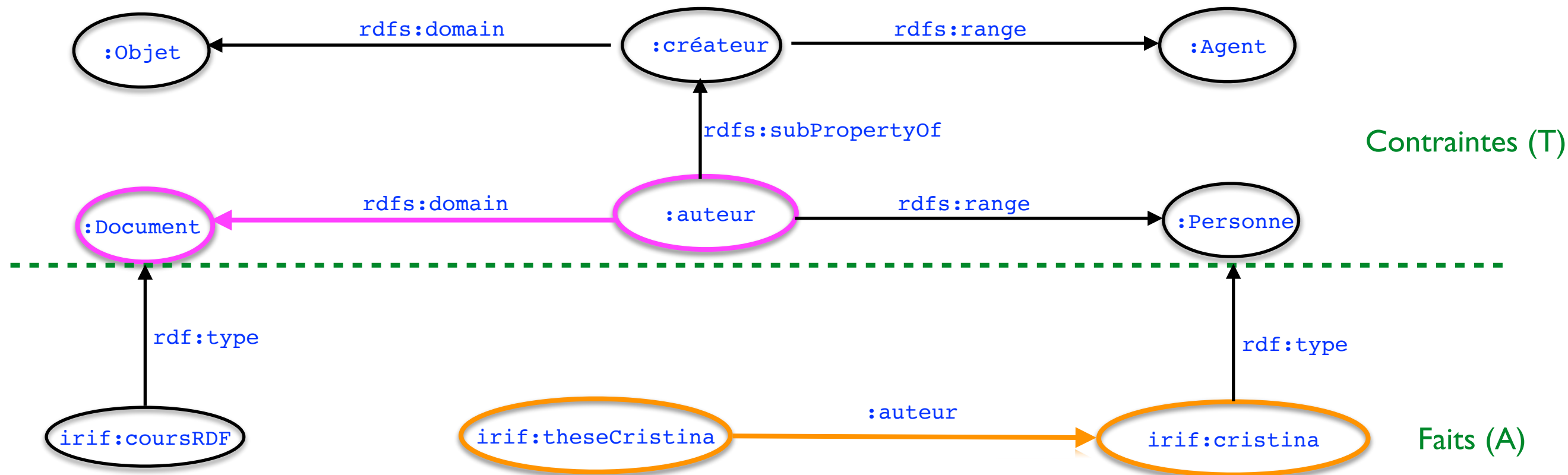
a une substitution σ qui inclut les premisses dans F :

$\sigma : r \rightarrow irif:theseCristina$
 $P \rightarrow :auteur$
 $s \rightarrow irif:cristina$
 $D \rightarrow :Document$

Exemple : algorithme de saturation pour les faits et les contraintes

- $P = \{ :auteur, :createur \}$ $C = \{ :Objet, :Document, :Agent, :Personne \}$

F



$F := A \cup T$

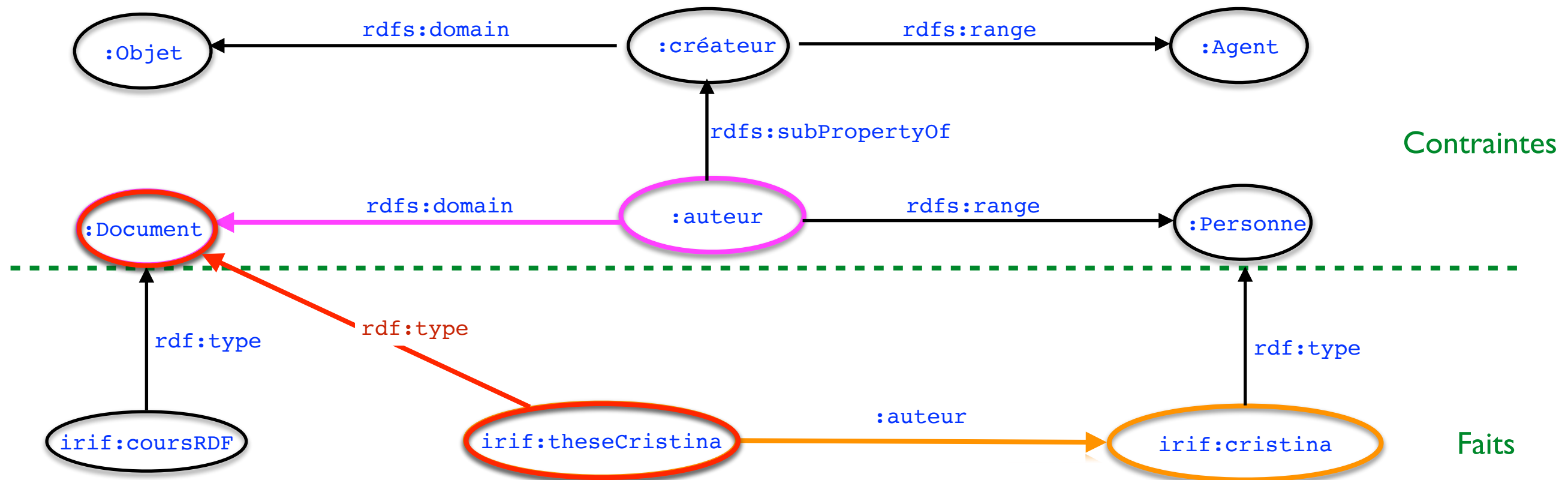
1.

```
<irif:theseCristina :auteur irif:Cristina>
<:auteur rdfs:domain :Document >
____
<irif:theseCristina rdfs:type :Document>
```

Exemple : algorithme de saturation pour les faits et les contraintes

- $P = \{ :auteur, :createur \}$ $C = \{ :Objet, :Document, :Agent, :Personne \}$

F



$F := A \cup T \cup \{ irif:theseCristina \ rdfs:type \ :Document \}$

1.

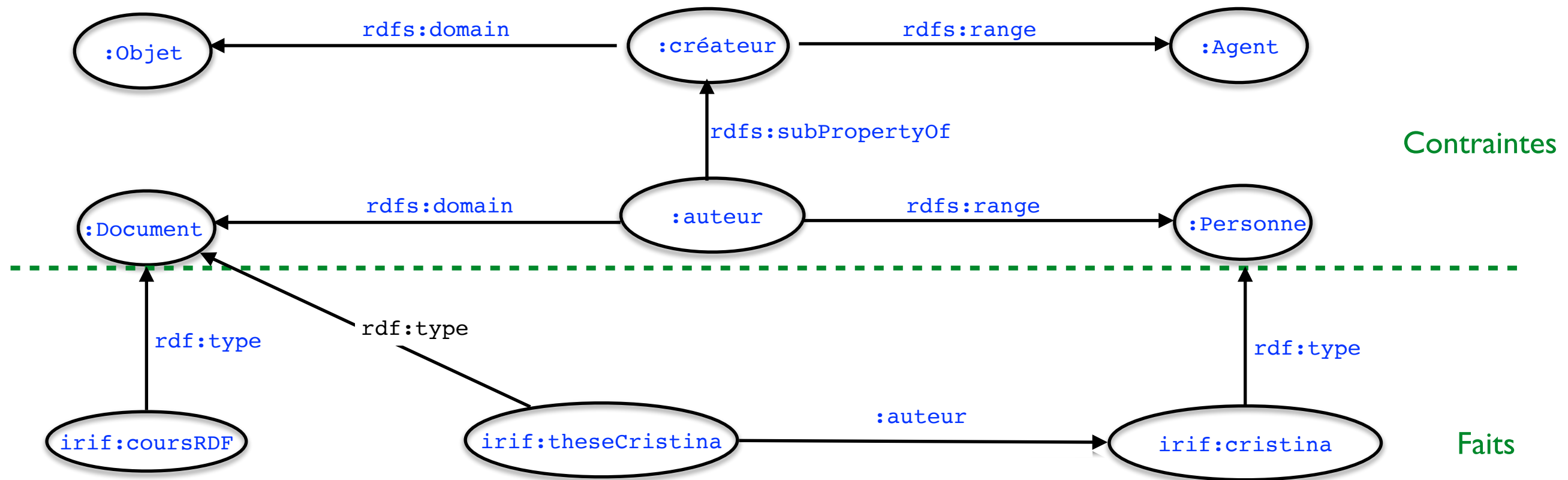
```
<irif:theseCristina :auteur irif:Cristina>  
<:auteur rdfs:domain :Document >  
_____  
<irif:theseCristina rdfs:type :Document>
```

On en déduit un nouveau fait dans F

Exemple : algorithme de saturation pour les faits et les contraintes

- $P = \{ :auteur, :createur \}$ $C = \{ :Objet, :Document, :Agent, :Personne \}$

F



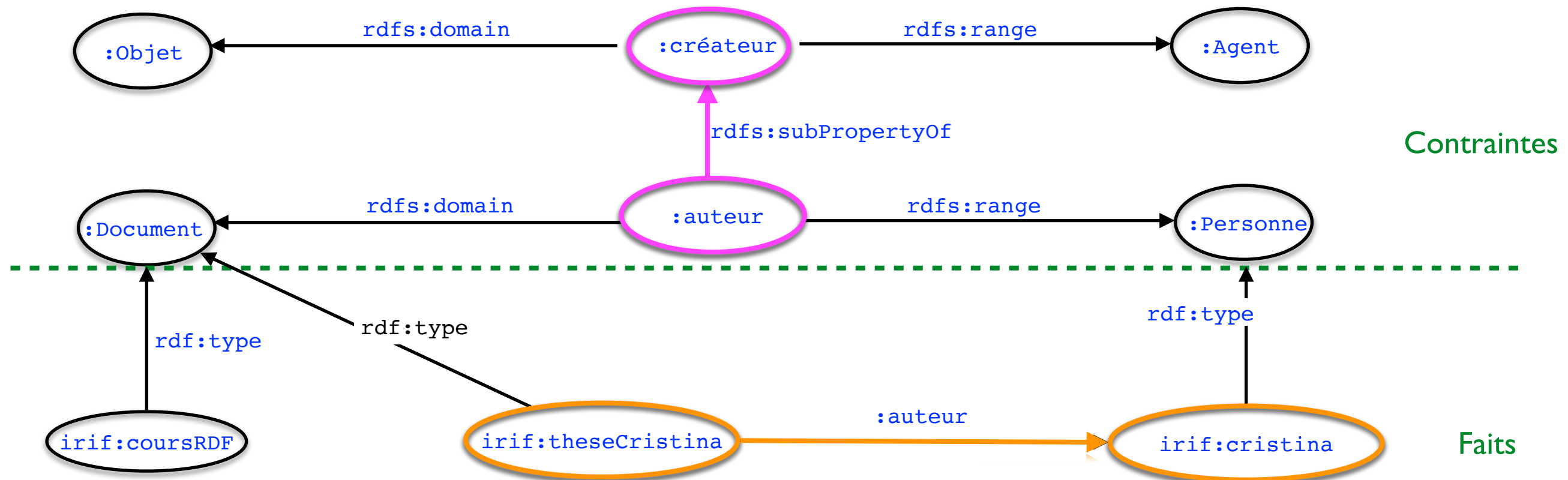
$F := \text{AUT} \cup \{ \text{irif:theseCristina rdfs:type :Document} \}$

2.

Exemple : algorithme de saturation pour les faits et les contraintes

- $P = \{ :auteur, :createur \}$ $C = \{ :Objet, :Document, :Agent, :Personne \}$

F



$F := \text{AUT} \cup \{ \text{irif:theseCristina rdfs:type :Document} \}$

2.

```
<irif:theseCristina :auteur irif:Cristina>
<:auteur rdfs:subPropertyOf :createur >

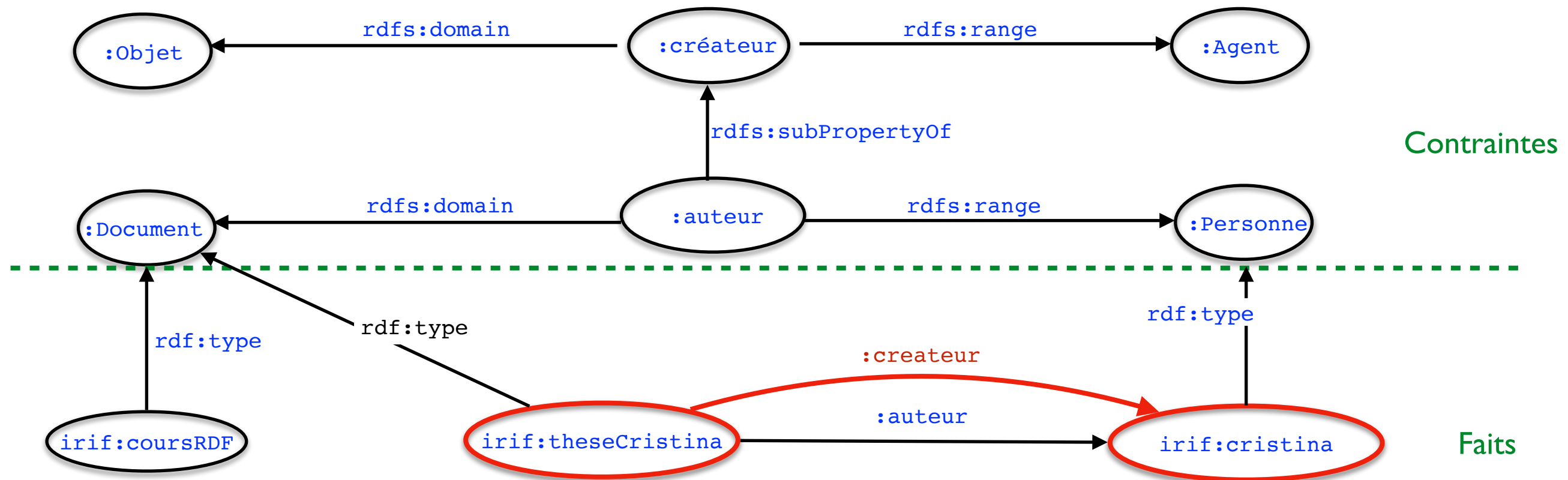

---


<irif:theseCristina :createur irif:cristina>
```

Exemple : algorithme de saturation pour les faits et les contraintes

- $P = \{ :auteur, :createur \}$ $C = \{ :Objet, :Document, :Agent, :Personne \}$

F



$F := \text{AUT} \cup \{ \text{irif:theseCristina rdfs:type :Document, irif:theseCristina :createur irif:cristina} \}$

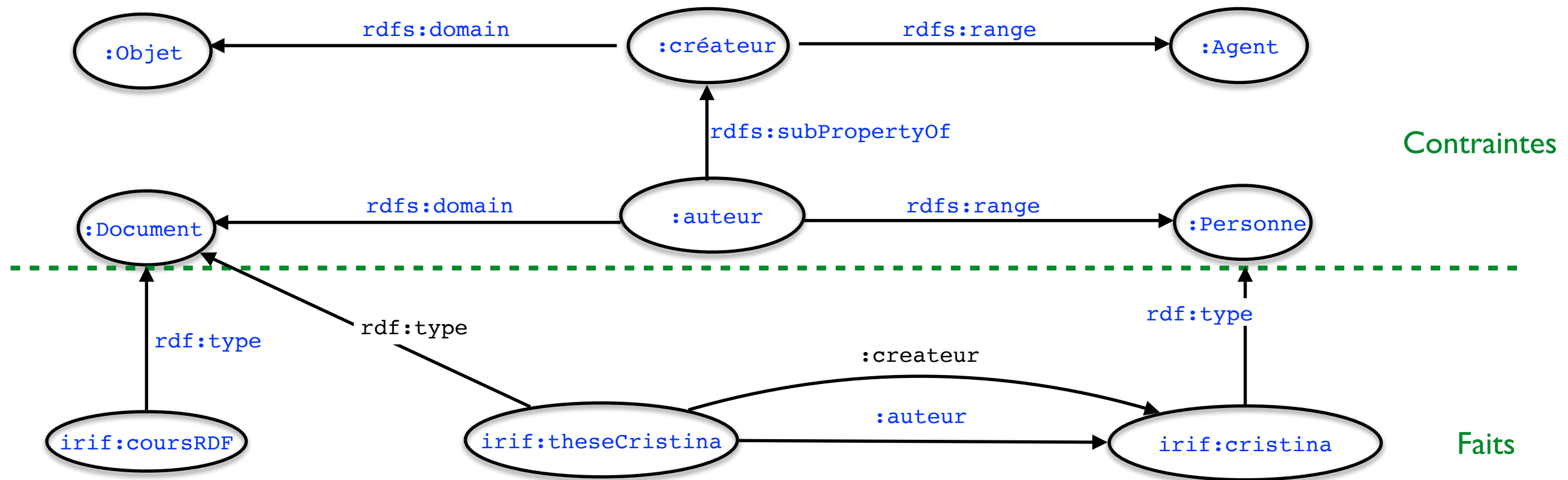
2.

```
<irif:theseCristina :auteur irif:Cristina>
<:auteur rdfs:subPropertyOf :createur >
____
<irif:theseCristina :createur irif:cristina>
```


Exemple : algorithme de saturation pour les faits et les contraintes

- $P = \{ :auteur, :createur \}$ $C = \{ :Objet, :Document, :Agent, :Personne \}$

F



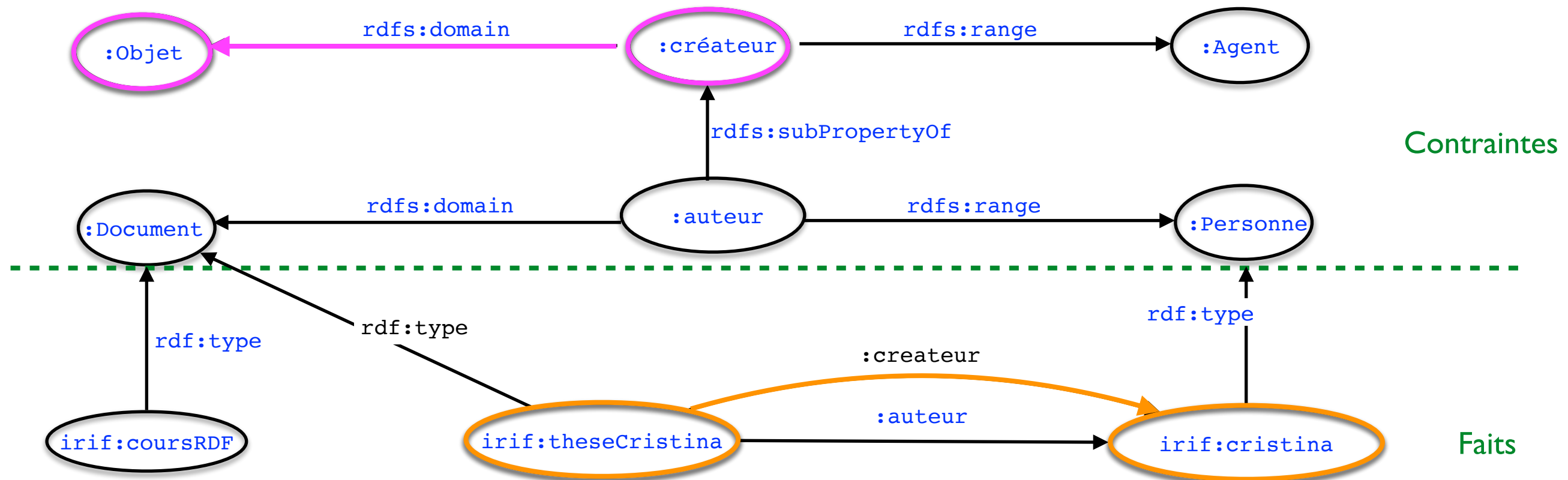
$F := \text{AUT} \cup \{ \text{irif:theseCristina rdfs:type :Document, irif:theseCristina :createur irif:cristina} \}$

3.

Exemple : algorithme de saturation pour les faits et les contraintes

- $P = \{ :auteur, :createur \}$ $C = \{ :Objet, :Document, :Agent, :Personne \}$

F



$F := \text{AUT} \cup \{ \text{irif:theseCristina rdfs:type :Document, irif:theseCristina :createur irif:cristina} \}$

3.

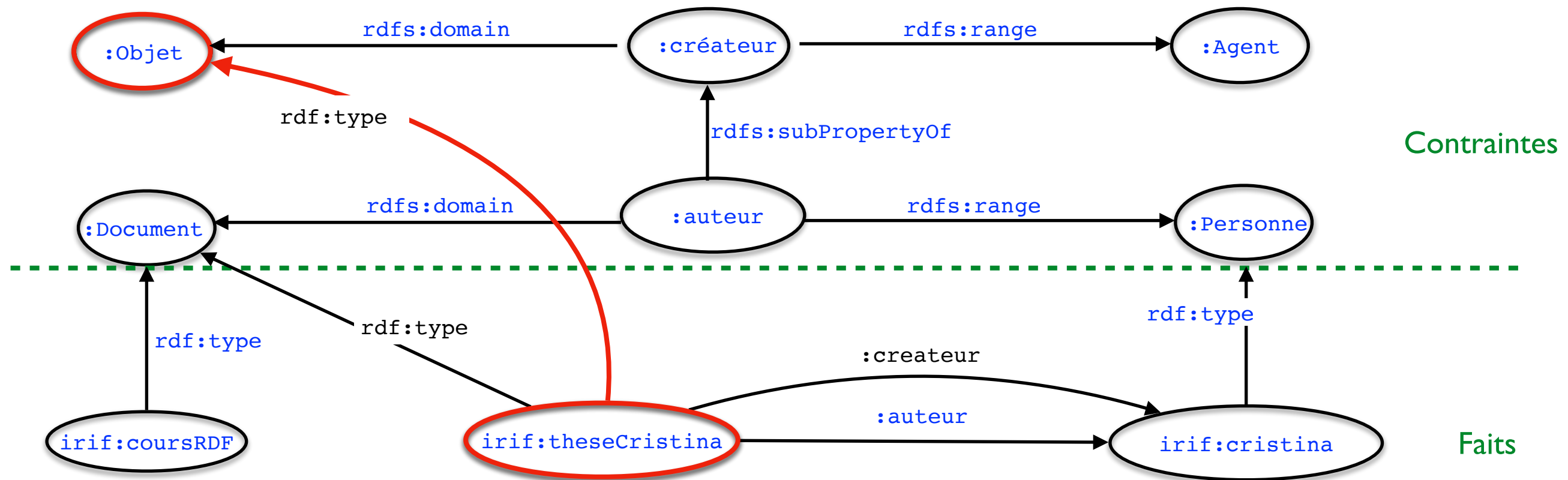
```
<irif:theseCristina :createur irif:cristina>
<:createur rdfs:domain :Objet >
_____
<irif:theseCristina :rdf:type :Objet>
```

Remarque : on peut utiliser les inférences comme base pour de nouvelles inférences

Exemple : algorithme de saturation pour les faits et les contraintes

- $P = \{ :auteur, :createur \}$ $C = \{ :Objet, :Document, :Agent, :Personne \}$

F



$F := \text{AUT} \cup \{ \text{irif:theseCristina rdfs:type :Document, irif:theseCristina :createur irif:cristina, irif:theseCristina :rdf:type :Objet} \}$

3.

```
<irif:theseCristina :createur irif:cristina>
<:createur rdfs:domain :Objet >

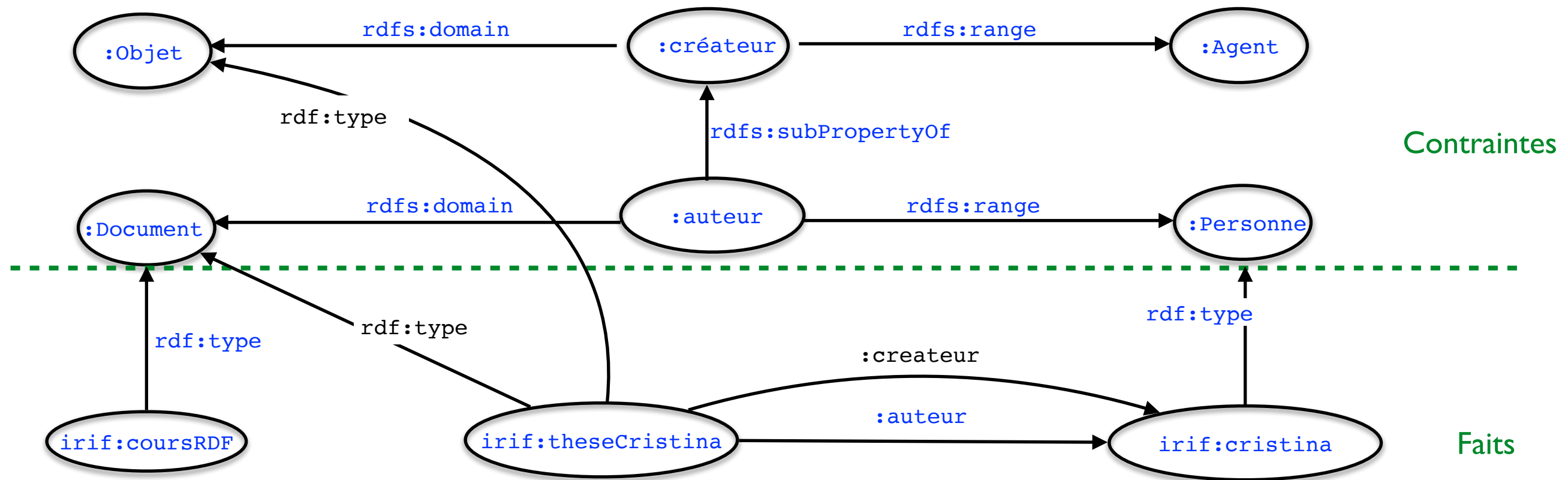

---


<irif:theseCristina :rdf:type :Objet>
```

Exemple : algorithme de saturation pour les faits et les contraintes

- $P = \{ :auteur, :createur \}$ $C = \{ :Objet, :Document, :Agent, :Personne \}$

F



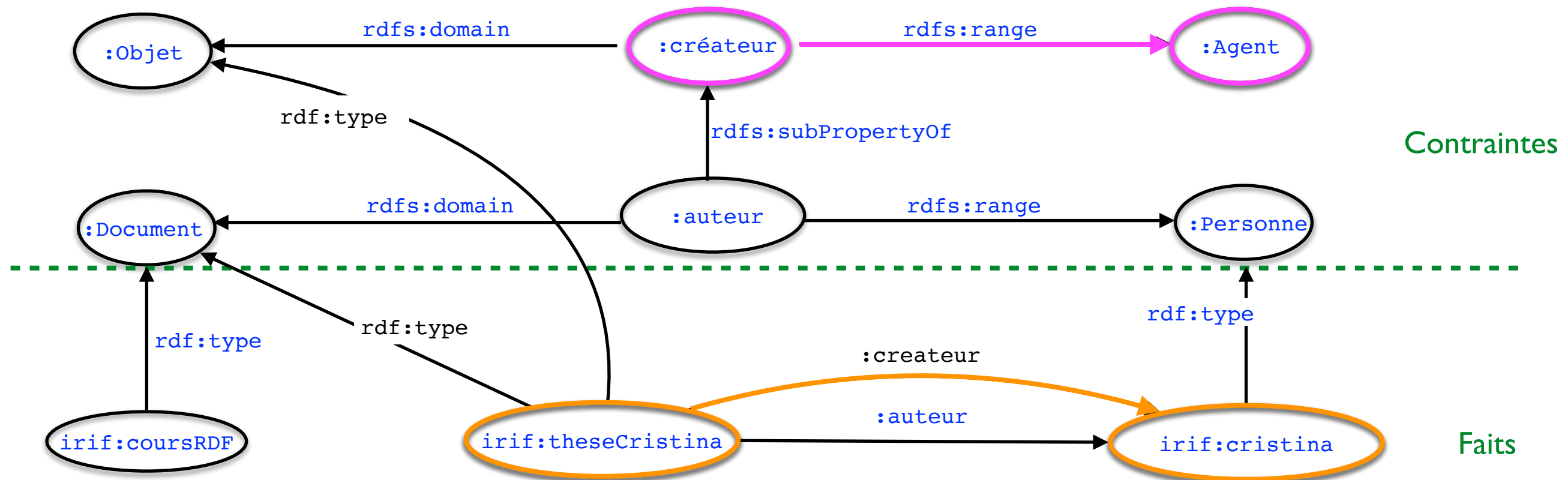
$F := \text{AUT} \cup \{ \text{irif:theseCristina rdfs:type :Document, irif:theseCristina :createur irif:cristina, irif:theseCristina :rdf:type :Objet} \}$

4.

Exemple : algorithme de saturation

- $P = \{ :auteur, :createur \}$ $C = \{ :Objet, :Document, :Agent, :Personne \}$

F



$F := \text{AUT} \cup \{ \text{irif:theseCristina rdfs:type :Document, irif:theseCristina :createur irif:cristina, irif:theseCristina :rdf:type :Objet} \}$

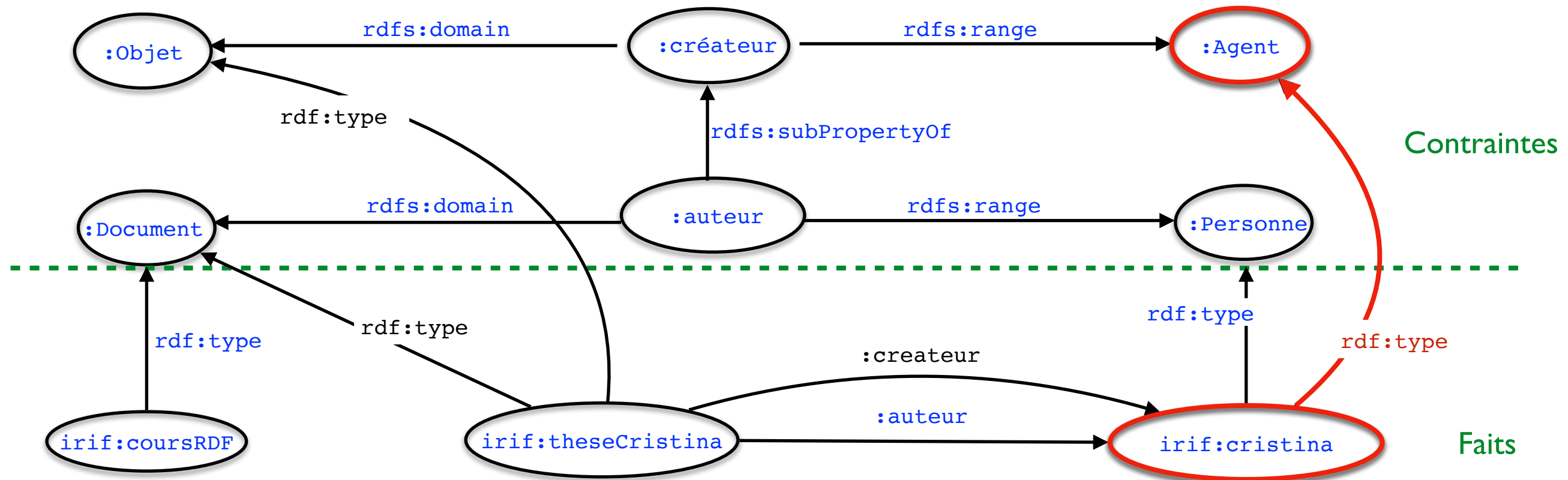
4.

```
<irif:theseCristina :createur irif:cristina>
<:createur rdfs:range :Agent >
_____
<irif:theseCristina rdf:type :Agent>
```

Exemple : algorithme de saturation pour les faits et les contraintes

- $P = \{ :auteur, :createur \}$ $C = \{ :Objet, :Document, :Agent, :Personne \}$

F



$F := \text{AUT} \cup \{ \text{irif:theseCristina rdfs:type :Document, irif:theseCristina :createur irif:cristina, irif:theseCristina :rdf:type :Objet, irif:theseCristina rdf:type :Agent} \}$

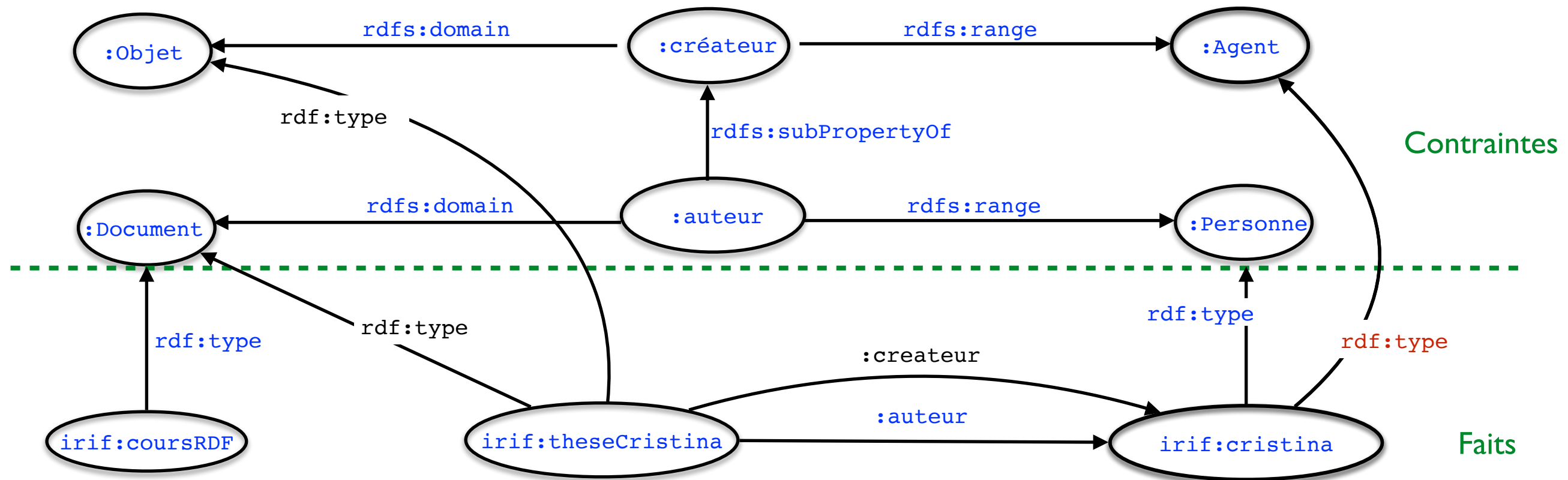
4.

```
<irif:theseCristina :createur irif:cristina>
<:createur rdfs:range :Agent >
_____
<irif:theseCristina rdf:type :Agent>
```

Exemple : algorithme de saturation pour les faits et les contraintes

- $P = \{ :auteur, :createur \}$ $C = \{ :Objet, :Document, :Agent, :Personne \}$

F



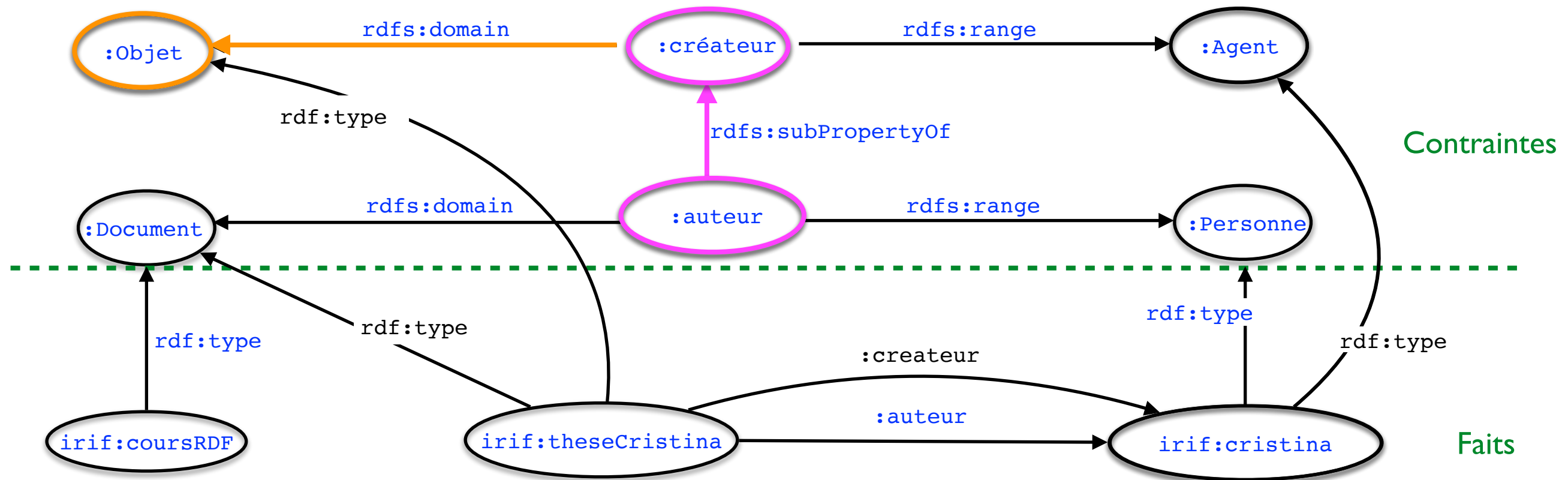
$F := AUT \cup \{ \text{irif:theseCristina rdfs:type :Document, irif:theseCristina :createur irif:cristina, irif:theseCristina :rdf:type :Objet, irif:theseCristina rdf:type :Agent} \}$

Exemple : algorithme de saturation pour les faits et les contraintes

- $P = \{ :auteur, :createur \}$ $C = \{ :Objet, :Document, :Agent, :Personne \}$

F

En plus si sémantique extensionnelle



$F := \text{AUT} \cup \{ \text{irif:theseCristina rdfs:type :Document}, \text{irif:theseCristina :createur irif:cristina}, \text{irif:theseCristina :rdf:type :Objet}, \text{irif:theseCristina rdf:type :Agent} \}$

5.

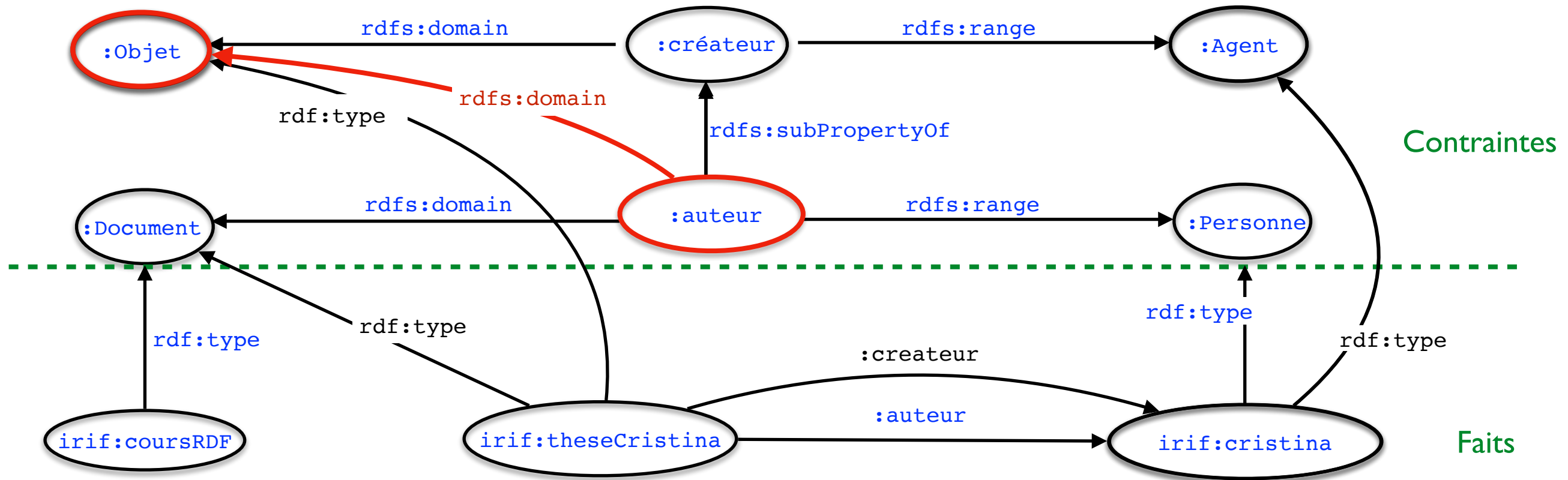
```
<:createur rdfs:domain :Objet>
<:auteur rdfs:subPropertyOf :createur >
____
<:auteur rdfs:domain :Objet>
```


Exemple : algorithme de saturation pour les faits et les contraintes

- $P = \{ :auteur, :createur \}$ $C = \{ :Objet, :Document, :Agent, :Personne \}$

F

En plus si sémantique extensionnelle



```
F:= AuT U { irif:theseCristina rdfs:type :Document, irif:theseCristina :createur
irif:cristina, irif:theseCristina :rdf:type :Object, irif:theseCristina rdf:type
:Agent, :auteur rdfs:domain :Object }
```

5.

```
<:createur rdfs:domain :Object>
<:auteur rdfs:subPropertyOf :createur >


---

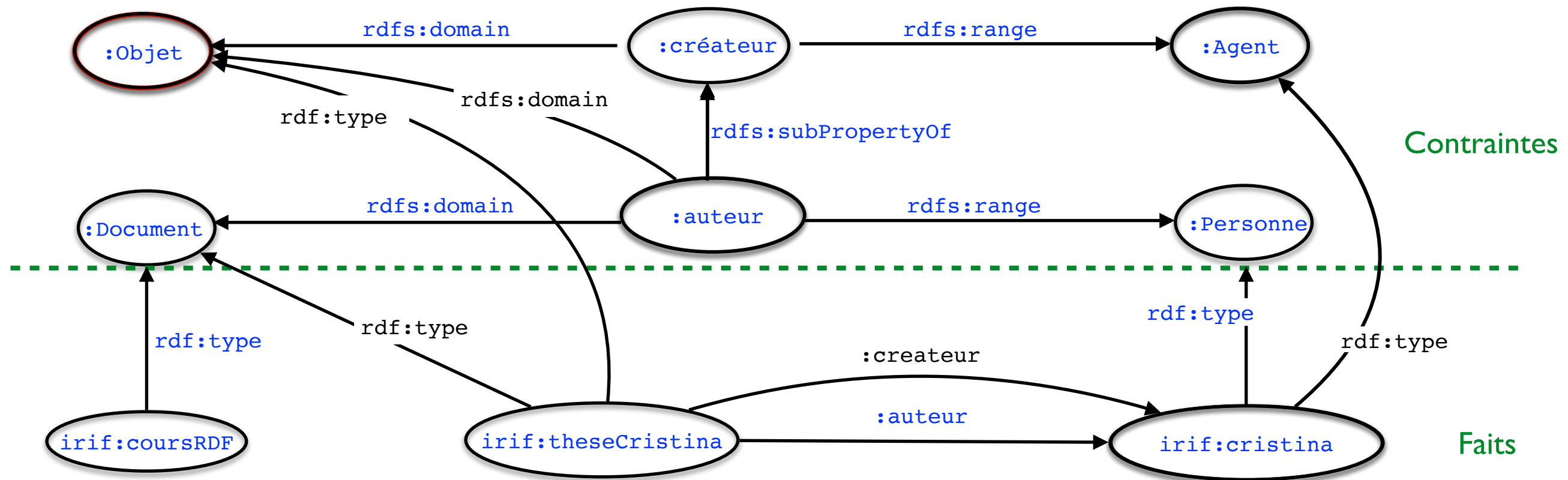

<:auteur rdfs:domain :Object>
```

Exemple : algorithme de saturation pour les faits et les contraintes

- $P = \{ :auteur, :createur \}$ $C = \{ :Objet, :Document, :Agent, :Personne \}$

F

En plus si sémantique extensionnelle



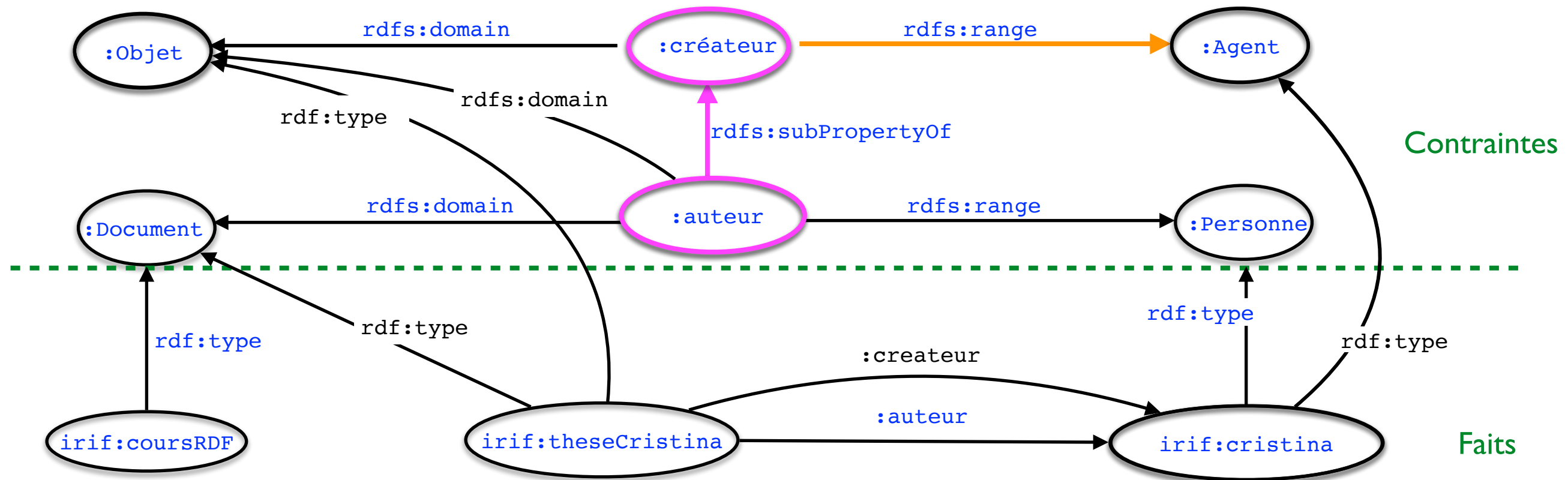
$F := AUT \cup \{ \text{irif:theseCristina rdfs:type :Document, irif:theseCristina :createur irif:cristina, irif:theseCristina :rdf:type :Objet, irif:theseCristina rdf:type :Agent, :auteur rdfs:domain :Objet} \}$

Exemple : algorithme de saturation pour les faits et les contraintes

- $P = \{ :auteur, :createur \}$ $C = \{ :Objet, :Document, :Agent, :Personne \}$

F

En plus si sémantique extensionnelle



$F := AUT \cup \{ \text{irif:theseCristina rdfs:type :Document, irif:theseCristina :createur irif:cristina, irif:theseCristina :rdf:type :Objet, irif:theseCristina rdf:type :Agent, :auteur rdfs:domain :Objet} \}$

6.

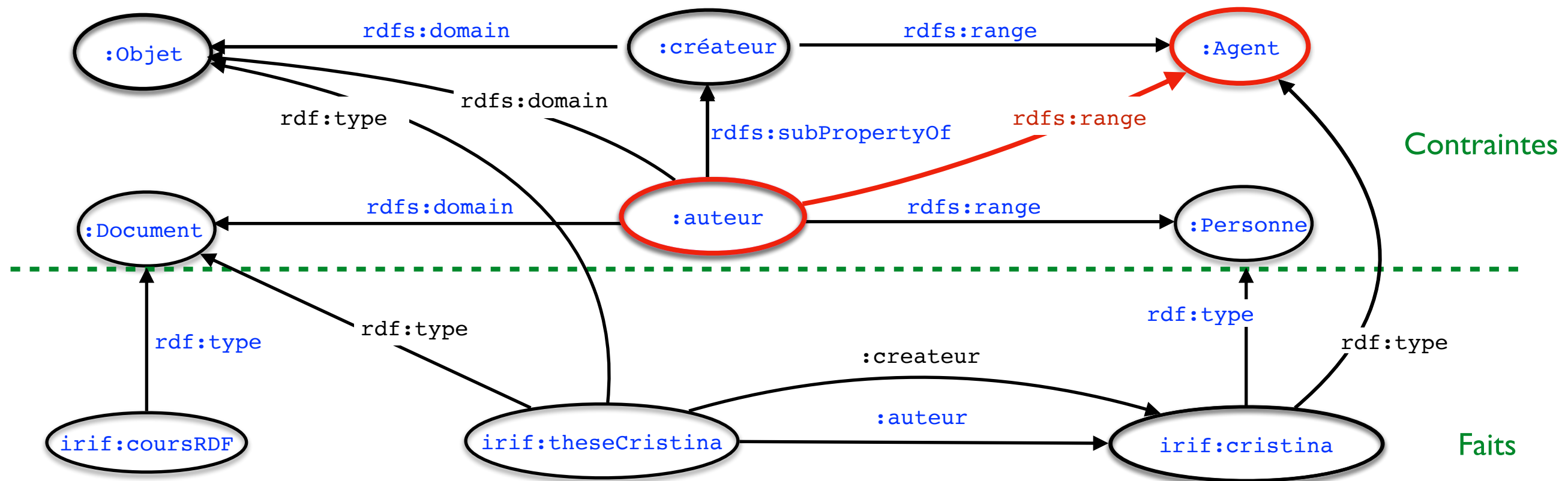
```
<:createur rdfs:range :Agent>
<:auteur rdfs:subPropertyOf :createur >
____
<:auteur rdfs:range :Agent>
```

Exemple : algorithme de saturation pour les faits et les contraintes

- $P = \{ :auteur, :createur \}$ $C = \{ :Objet, :Document, :Agent, :Personne \}$

F

En plus si sémantique extensionnelle



$F := \text{AUT} \cup \{ \text{irif:theseCristina rdfs:type :Document, irif:theseCristina :createur irif:cristina, irif:theseCristina :rdf:type :Objet, irif:theseCristina rdf:type :Agent, :auteur rdfs:domain :Objet, :auteur rdfs:range :Agent} \}$

5.

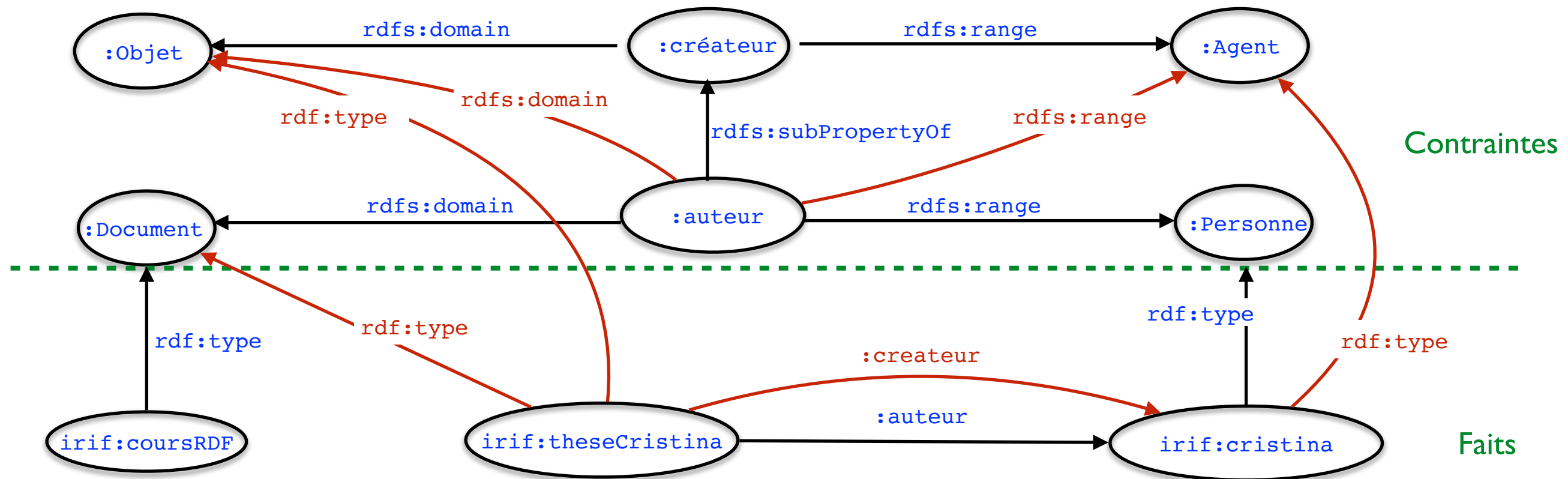
```
<:createur rdfs:range :Agent>
<:auteur rdfs:subPropertyOf :createur >
____
<:auteur rdfs:range :Agent>
```

Exemple : algorithme de saturation pour les faits et les contraintes

- $P = \{ :auteur, :createur \}$ $C = \{ :Objet, :Document, :Agent, :Personne \}$

F

En plus si sémantique extensionnelle



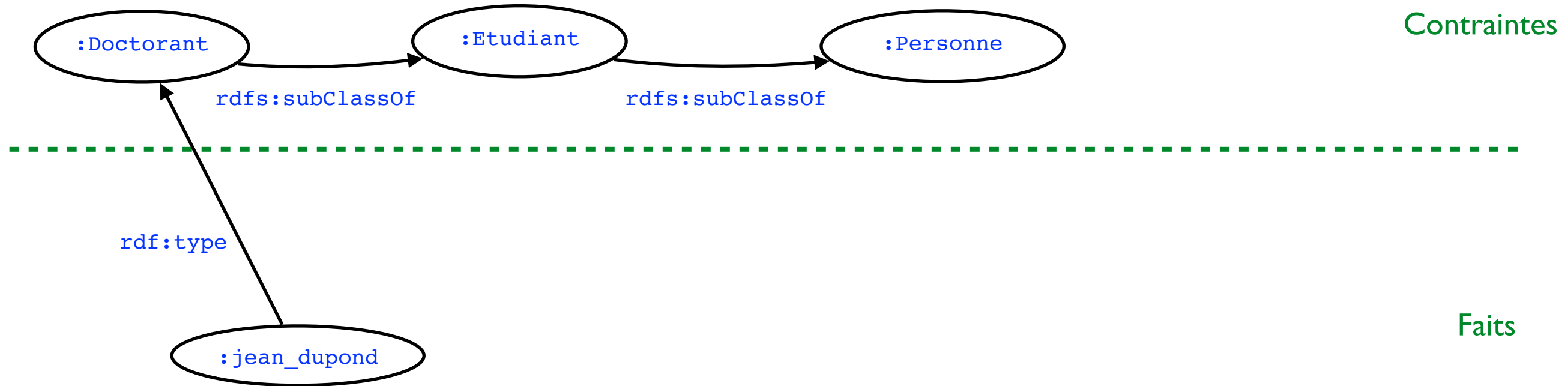
Plus aucune règles n'est applicable. **Saturation terminée. On a dérivé en F toutes les conséquences** (de type fait ou contrainte)

(remarque : à la fois des règle de Γ_A et de Γ_T étaient applicables, donc on a dérivé à la fois de nouveaux faits et de nouvelles contraintes (en rouge))

(on fait omission des triplets réflexifs)

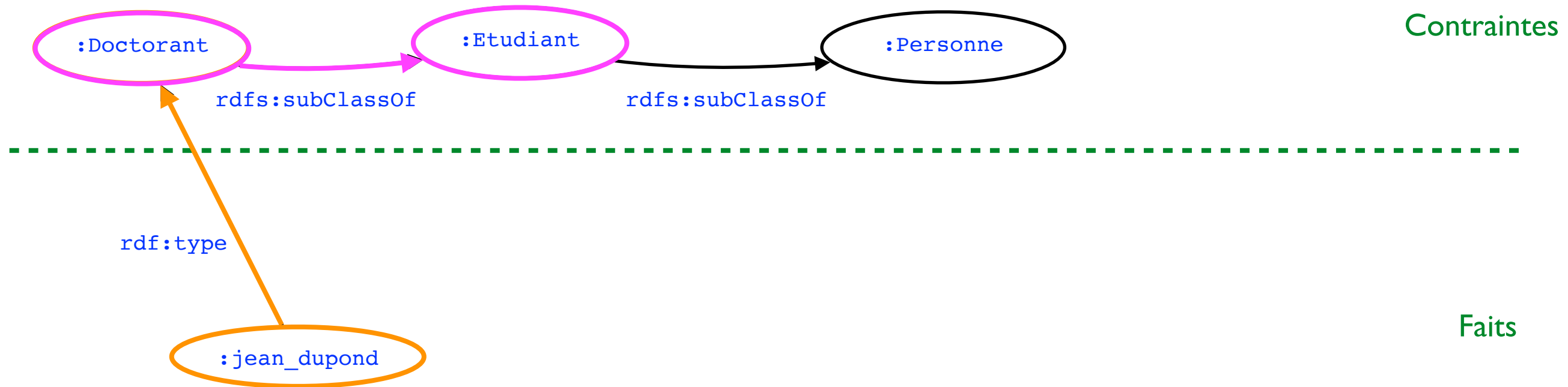
Un autre exemple : saturation des faits et des contraintes

- K



Un autre exemple : saturation des faits et des contraintes

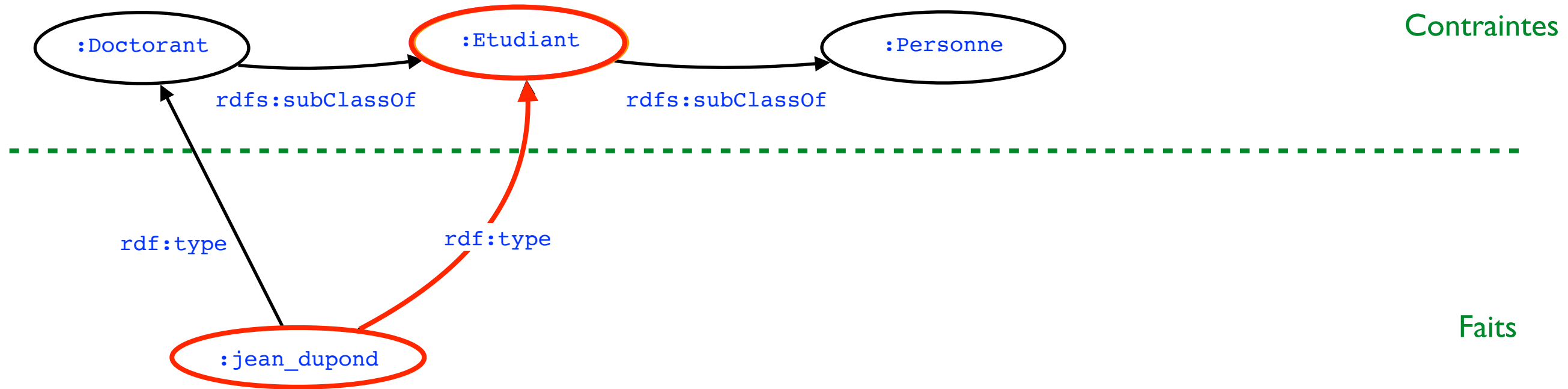
- K



```
<:jean_dupond rdf:type :Doctorant>  
<:Doctorant rdfs:subClassOf :Etudiant >  
_____  
<:jean_dupond rdf:type :Etudiant>
```

Un autre exemple : saturation des faits et des contraintes

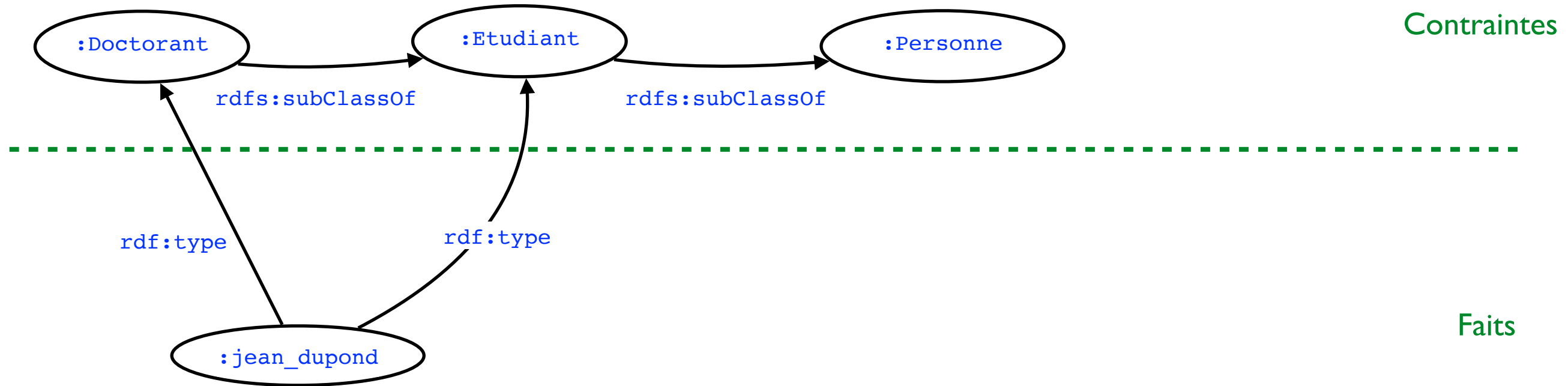
- F



```
<:jean_dupond rdf:type :Doctorant>  
<:Doctorant rdfs:subClassOf :Etudiant >  
_____  
<:jean_dupond rdf:type :Etudiant>
```

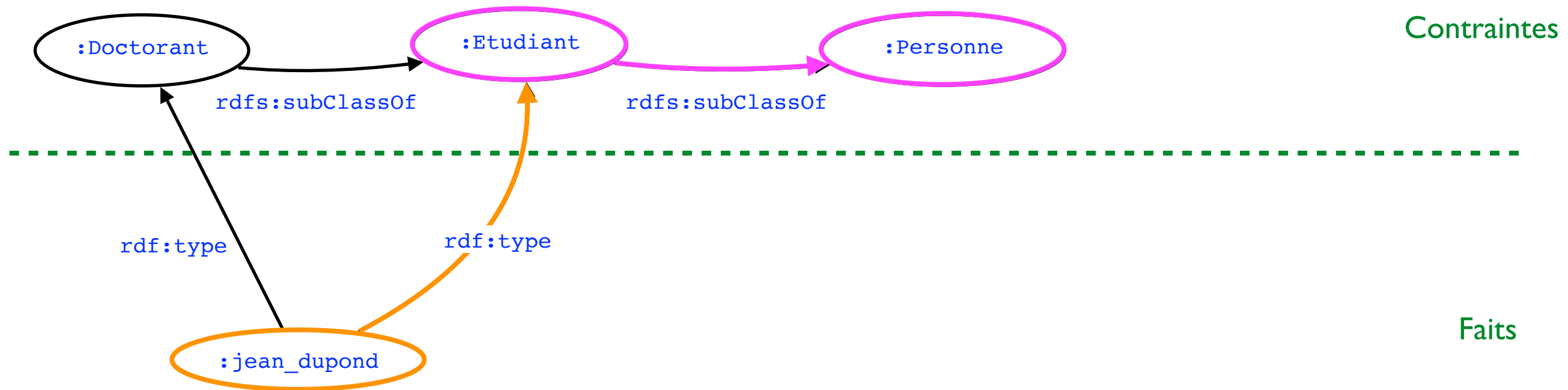

Un autre exemple : saturation des faits et des contraintes

- F



Un autre exemple : saturation des faits et des contraintes

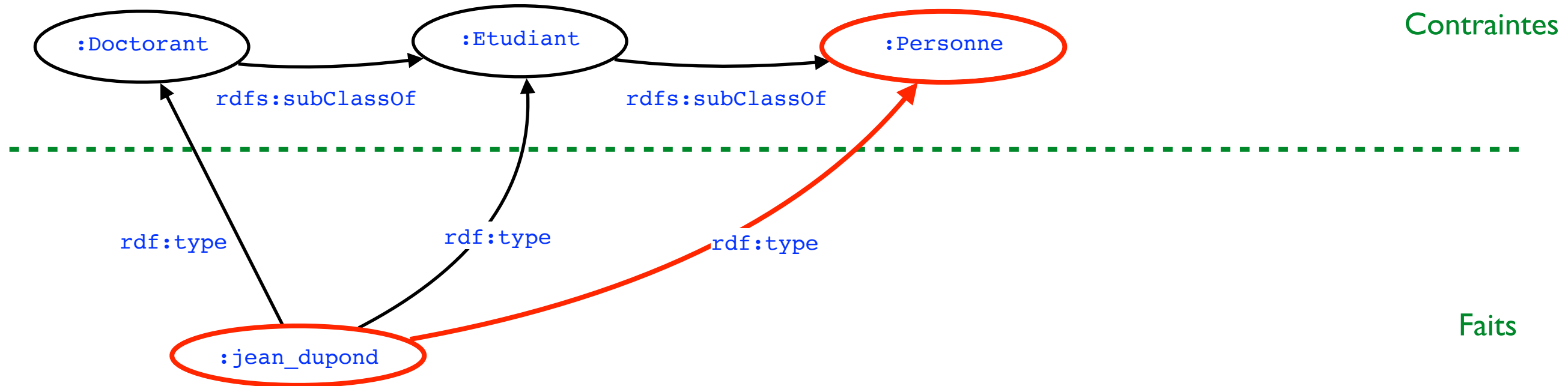
- F



```
<:jean_dupond rdf:type :Etudiant>  
<:Etudiant rdfs:subClassOf :Personne >  
_____  
<:jean_dupond rdf:type :Personne>
```

Un autre exemple : saturation des faits et des contraintes

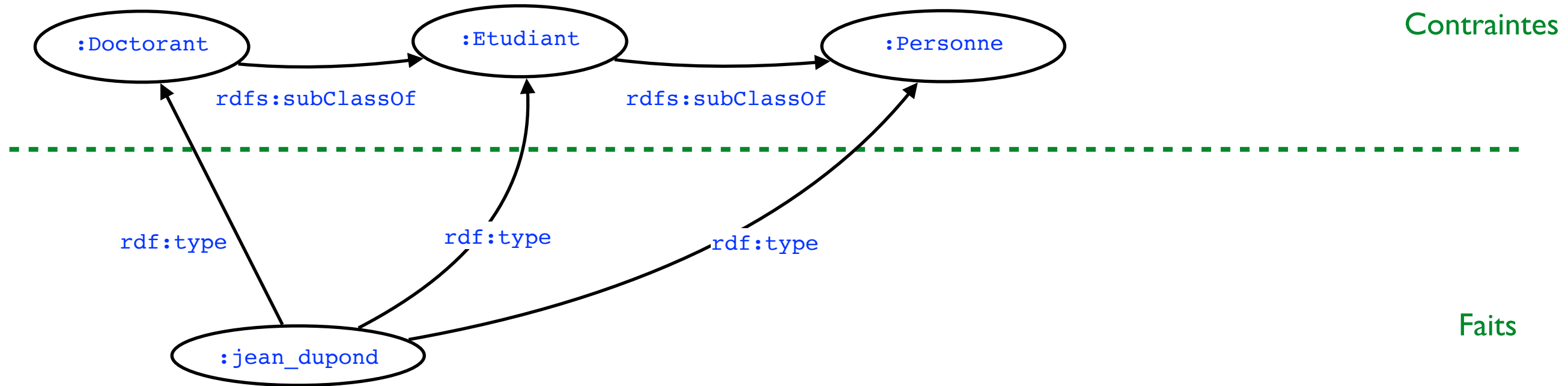
- F



```
<:jean_dupond rdf:type :Etudiant>  
<:Etudiant rdfs:subClassOf :Personne >  
_____  
<:jean_dupond rdf:type :Personne>
```

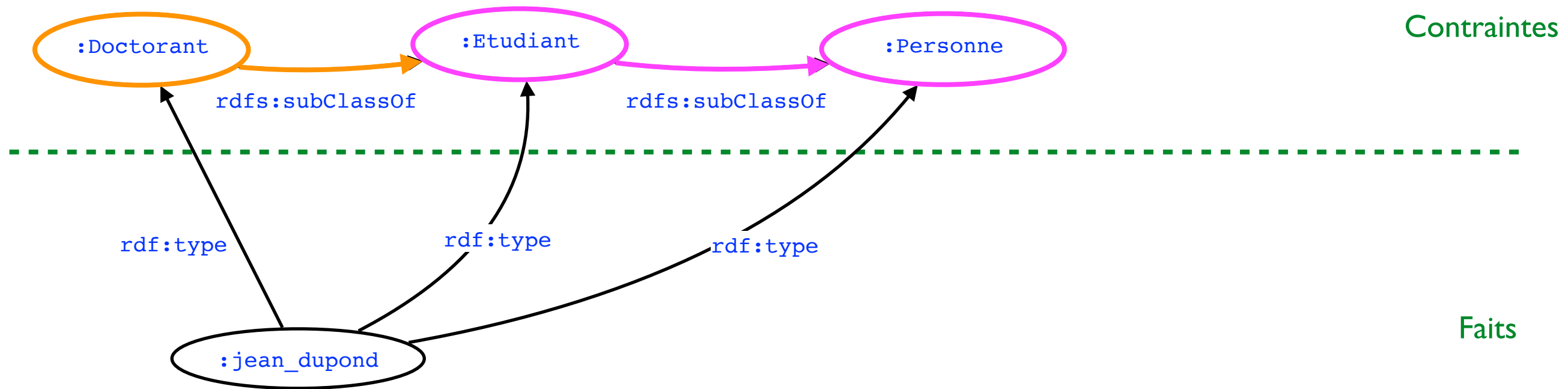
Un autre exemple : saturation des faits et des contraintes

- F



Un autre exemple : saturation des faits et des contraintes

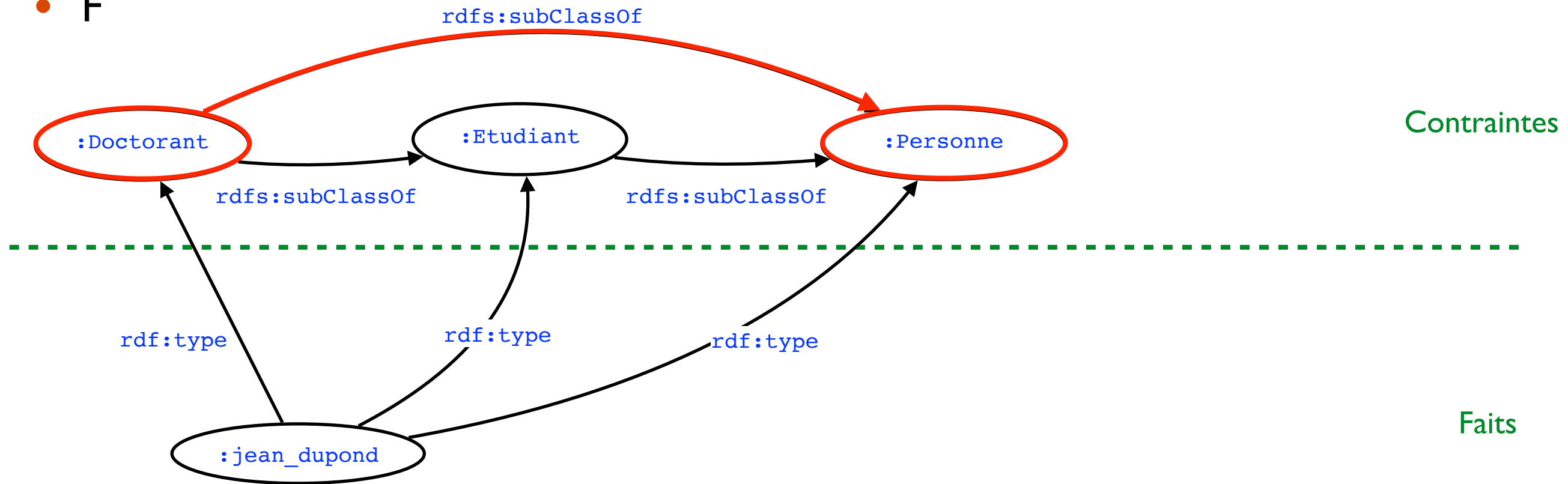
- F



```
<:Doctorant rdfs:subClassOf :Etudiant>
<:Etudiant rdfs:subClassOf :Personne >
_____
<:Doctorant rdfs:subClassOf :Personne>
```

Un autre exemple : saturation des faits et des contraintes

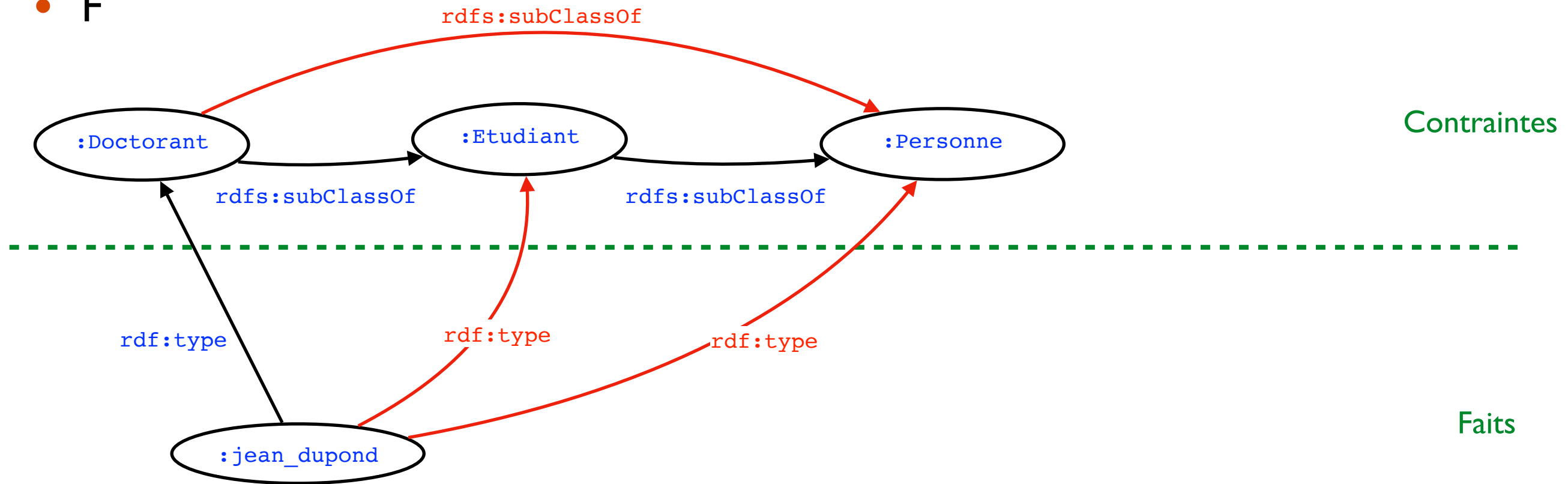
● F



```
<:Doctorant rdfs:subClassOf :Etudiant>
<:Etudiant rdfs:subClassOf :Personne >
_____
<:Doctorant rdfs:subClassOf :Personne>
```

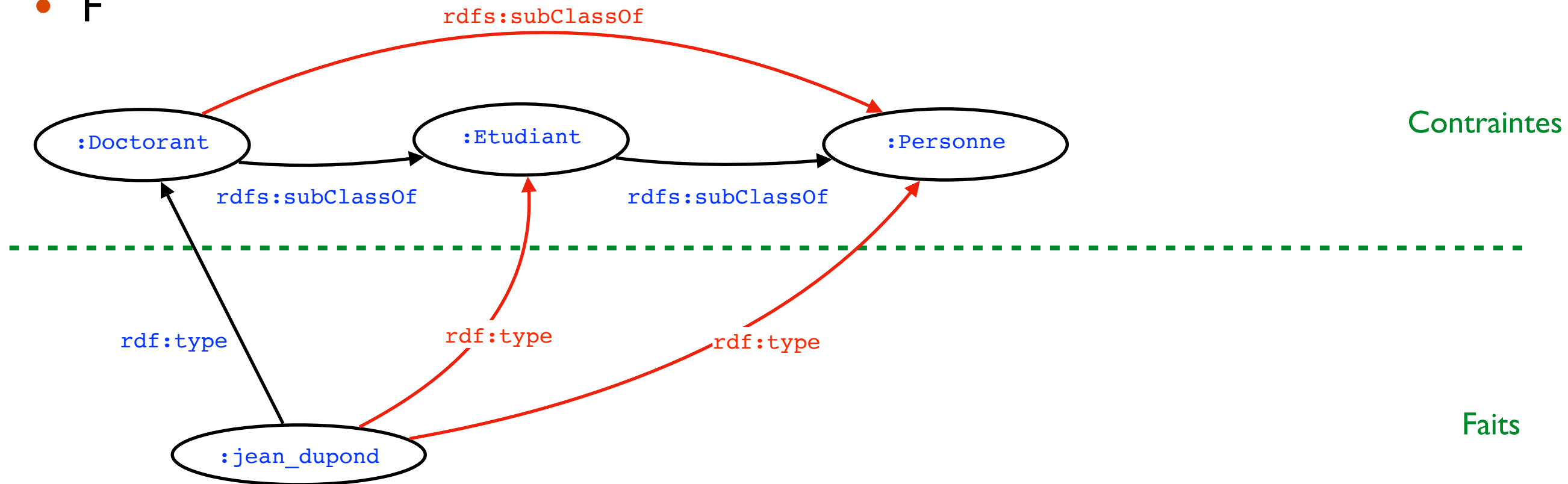
Un autre exemple : saturation des faits et des contraintes

• F



Un autre exemple : saturation des faits et des contraintes

• F



Plus aucune règles n'est applicable. **Saturation terminée. On a dérivé en F toutes les conséquences** (de type fait ou contrainte)

(remarque : à la fois des règles de Γ_A et de Γ_T étaient applicables donc on a dérivé à la fois de nouveaux faits et de nouvelles contraintes (en rouge))

(on fait omission des triplets réflexifs)

Graphe saturé (ou cloture)

Le graphe **F** obtenu par saturation du graphe K d'origine est appelé **graphe saturé** ou **cloture de K**

- dénoté K^∞
- ou K^{Γ^∞} pour rendre explicite l'ensemble des règles de dérivation Γ utilisées pour saturer
- Quand Γ est l'ensemble de toutes les règles RDFS on écrira K^{RDFS^∞}

Algorithme de saturation RDFS : complexité

- Dans toutes les contraintes RDFS la tête (conclusion) contient uniquement des variables qui apparaissent dans le corps (condition).
- Exemple :

```
<r rdf:type A>  
<A rdfs:subClassOf B>  
-----  
<r rdf:type B>
```

(appelées “full TGDS”)

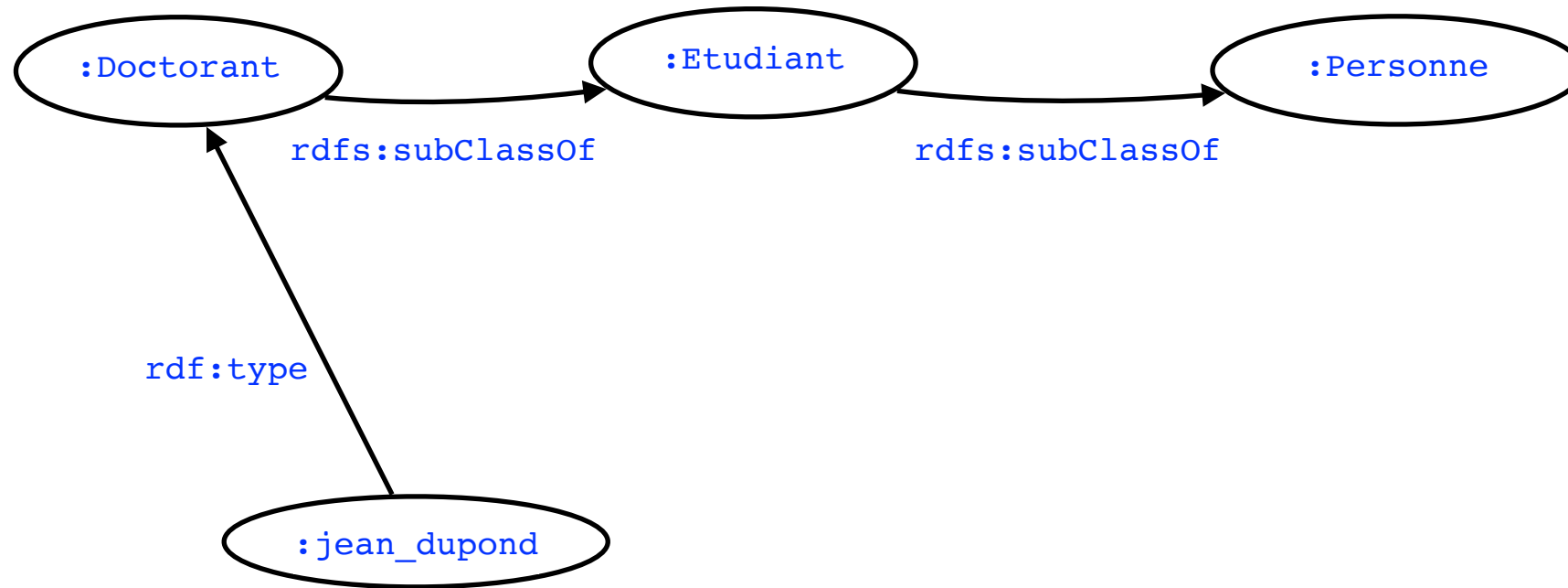
- Ce qui explique que les nouveaux triplets inférés peuvent utiliser seulement des ressources déjà présentes dans l'ontologie ($A \cup T$)
- Cela garantit que l'algorithme de saturation termine toujours
- De plus seulement un nombre polynomial de nouveau triplets peut être dérivé (polynomial en le nombre de ressources de $A \cup T$)
- \Rightarrow L'algorithme de saturation est PTIME (en la taille $|A \cup T|$ de l'ontologie)

Interrogation de bases de connaissances RDFS

Interrogation de données RDF : hypothèse de monde fermée

- Sémantique par défaut des langages d'interrogation RDF (par ex. SPARQL) :
 - ▶ Spécifie quel est le résultat d'une requête évaluée sur un graphe RDF, sans aucune inférence : uniquement les triplets explicites sont interrogés
 - ▶ Le graphe RDF peut contenir des faits et des contraintes (RDFS), mais il n'est pas vu comme une ontologie, simplement comme un ensemble de triplets RDF
 - ▶ Appelée hypothèse de monde fermé (ou “*simple entailment regime*” dans le standard W3C) - la base de triplets est considérée “complete”

Interrogation de données RDF : hypothèse de monde fermée



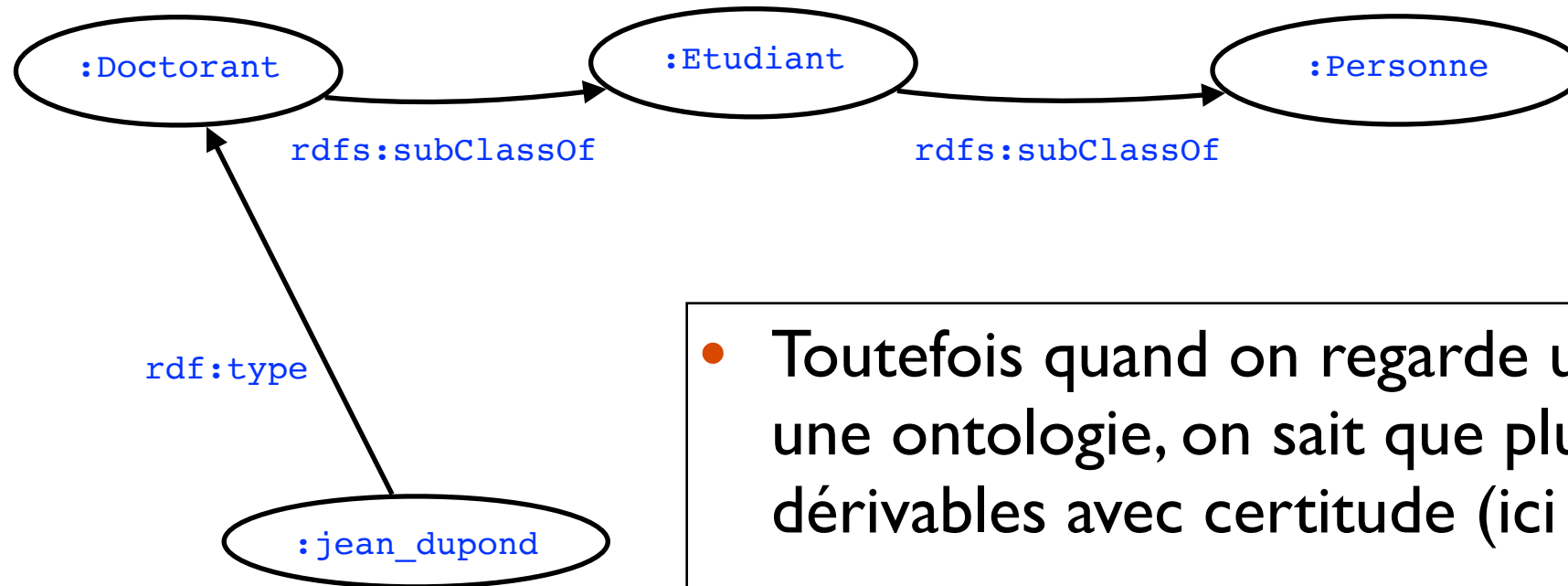
Select ?x

Where (:jean_dupond rdf:type ?x)

Retourne uniquement

x
:Doctorant

Interrogation de données RDF : hypothèse de monde fermée



- Toutefois quand on regarde un graphe RDF/S comme une ontologie, on sait que plus de réponses sont dérivables avec certitude (ici `:Etudiant`, `:Personne`)
 - ▶ On souhaiterait que les réponses aux requêtes tiennent compte de ces nouvelles connaissances

Select ?x

Where (:jean_dupond rdf:type ?x)

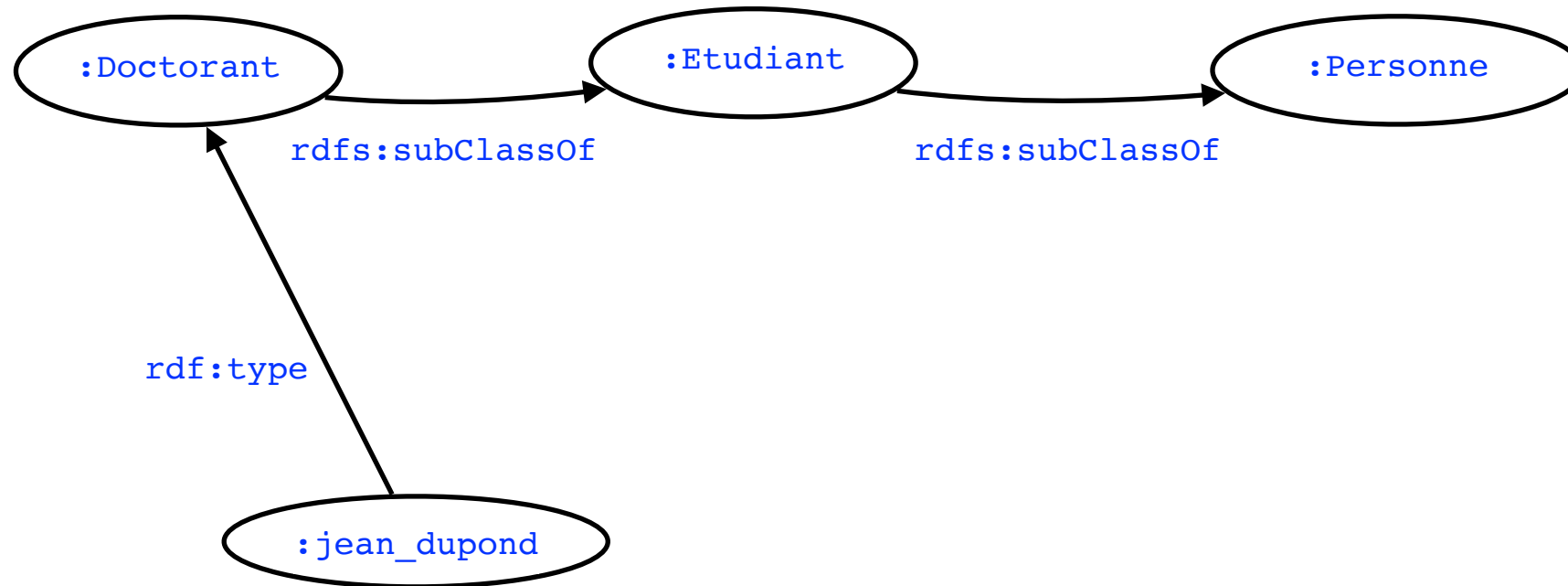
Retourne uniquement

x
:Doctorant

Interrogation d'ontologies : hypothèse de monde ouvert

- Interroger une ontologie est une tâche plus complexe qu'interroger une base de faits considérée "complète"
 - ▶ Intuitivement : toute réponse à une requête derivable implicitement par les faits et contraintes du graphe interrogé doit être prise en compte
 - ▶ Appelée hypothèse de **monde ouvert** : la base de faits à interroger ne se limite pas aux faits explicitement représentés
 - ▶ Dans le standard W3C SPARQL 1.1 on parle de **entailment regimes** : un *entailment regime* spécifie quelle est la sémantique d'une requête SPARQL quand on autorise une forme de raisonnement
 - Plusieurs *entailment regimes* ont été spécifiés, selon l'ensemble de règles d'inférence Γ qu'on autorise pour raisonner (RDF, RDFS, OWL etc)
 - Introduits uniquement dans la version 1.1 !!

Interrogation de données RDF : hypothèse de monde fermée



Select ?x

Where (:jean_dupond rdf:type ?x)

Response attendue **sous RDFS entaillement regime**

x
:Doctorant
:Etudiant
:Personne

Remarques

- Pas tous les systèmes commerciaux implémentent les *entailment regimes* (cela n'est pas rendu obligatoire par le W3C)
- Activer un mode d'évaluation SPARQL par *entailment regime* n'est pas toujours immédiat
 - ▶ En Jena par exemple :
 - La commande `sparql` implémente le *simple entailment regime* (pas d'inférence)
 - pas d'outils en ligne de commande équivalents à la commande `sparql` pour adopter un autre *entailment regime*
 - Mais facilement implémentable dans un programme Java par la bibliothèque ARQ de Jena

La théorie et la pratique

- Les entailment regimes sont définis dans le standard W3C uniquement pour les BGP (forme simple de requêtes positives) !!
 - ▶ Bien que SPARQL possède des mécanismes complexe de negation!
- Cela veut dire qu'il n'y a pas accord sur ce qu'une requête avec negation devrait retourner quand on veut tenir compte des inférences
 - ▶ chaque système commercial implémente sa propre sémantique
- Sujet de recherche très actif !
- Nous nous limiterons aux BGP dans ce cours...

Sémantique des BGP en presence d'inférence

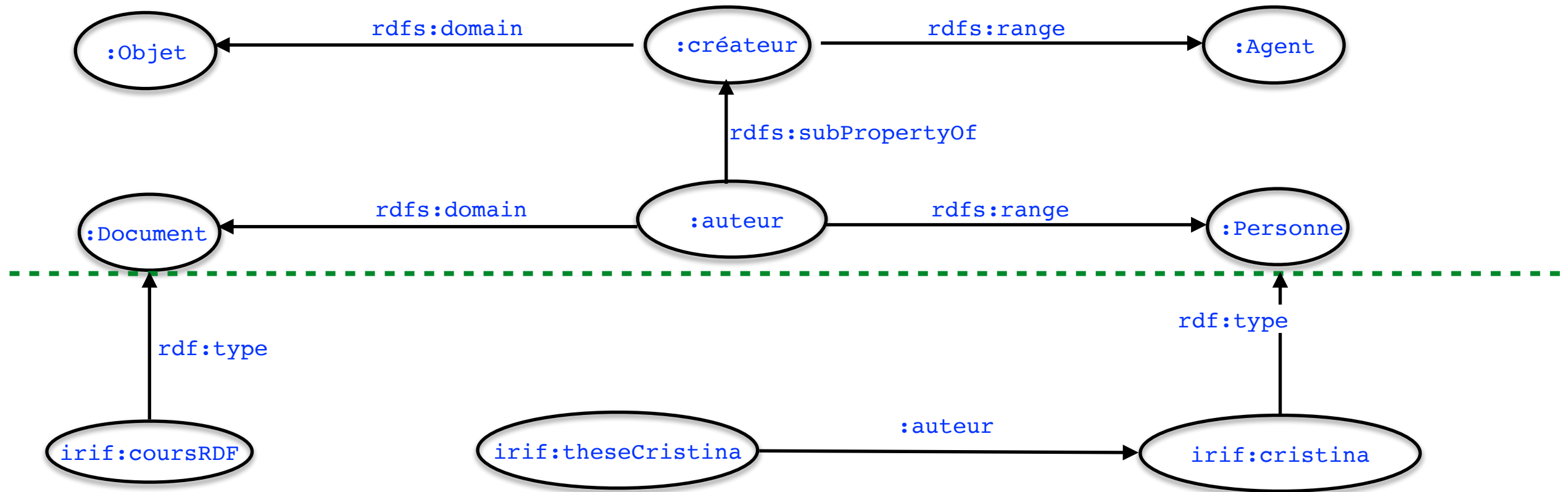
- On denote $Q(K)$ l'évaluation directe de Q sur un graphe RDF K (pas d'inférence)
- Sous un ensemble Γ de règles d'inférence autorisées (i.e. l'*entailment regime*) on denote $Q^\Gamma(K)$ la réponse attendue de Q en presence de l'inférence Γ
 - ▶ Si Γ est l'ensemble des règles RDFS on écrira $Q^{RDFS}(K)$
- Rappel : on denote K^{Γ_∞} la saturation de K par les règles d'inférences Γ
- Pour les BGP le standard W3C en gros* définit $Q^\Gamma(K)$ comme :

$$Q^\Gamma(K) := Q(K^{\Gamma_\infty})$$

* plus compliqué que cela en presence de blank nodes, extensions des BGP avec OPTIONAL,...

Sémantique des BGP en presence d'inférence

K



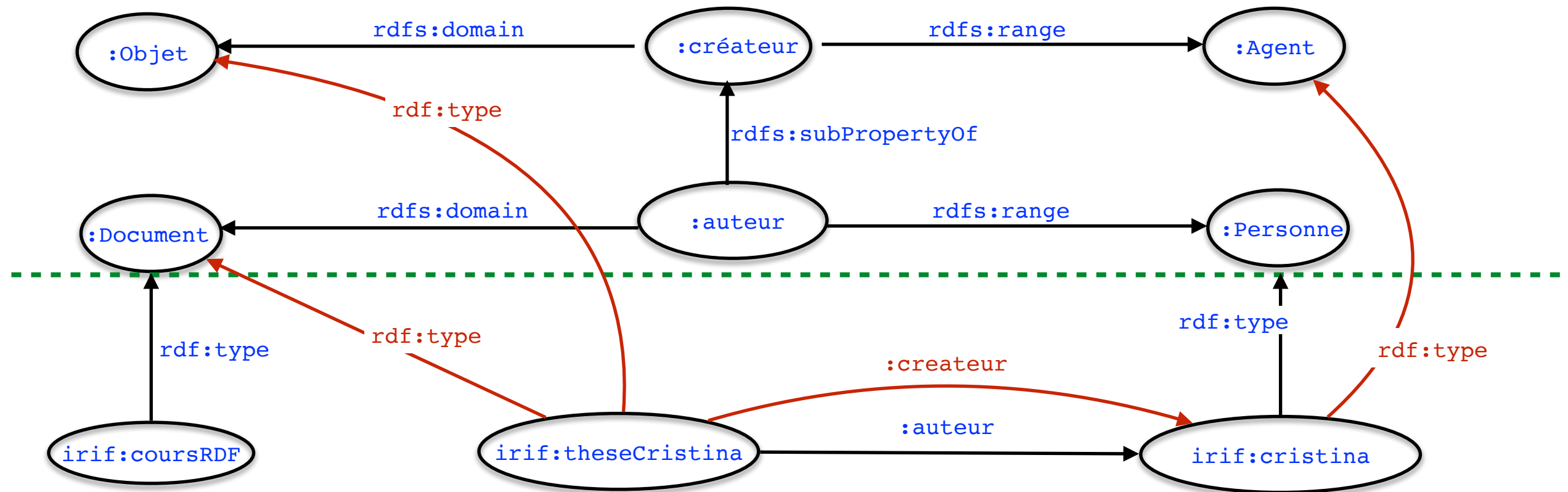
Q

```
Select ?x
Where {
  ?x rdf:type :Object .
  ?x :createur irif:cristina .
}
```

Q(K) vide
(résultat de la commande sparql)

Sémantique des BGP en presence d'inférence

$F = \mathcal{K}^{\text{RDFS}^\infty}$ (restraints aux faits)



Q

```

Select ?x
Where {
    ?x rdf:type :Object .
    ?x :createur irif:cristina .
}
    
```

$$Q^{\text{RDFS}}(\mathcal{K}) = Q(F)$$

x
:irifTheseCristina

Calcul des réponses en présence d'inférence

- **Approche saturation :**
 - ▶ Le graphe saturé est calculé plus ou moins entièrement pour y évaluer les requêtes
- **Approche réécriture**
 - ▶ Pas de calcul d'inférences : la requête es re-écrite pour “simuler” la présence des faits dérivés

Calcul des réponses en presence d'inférence : approche saturation

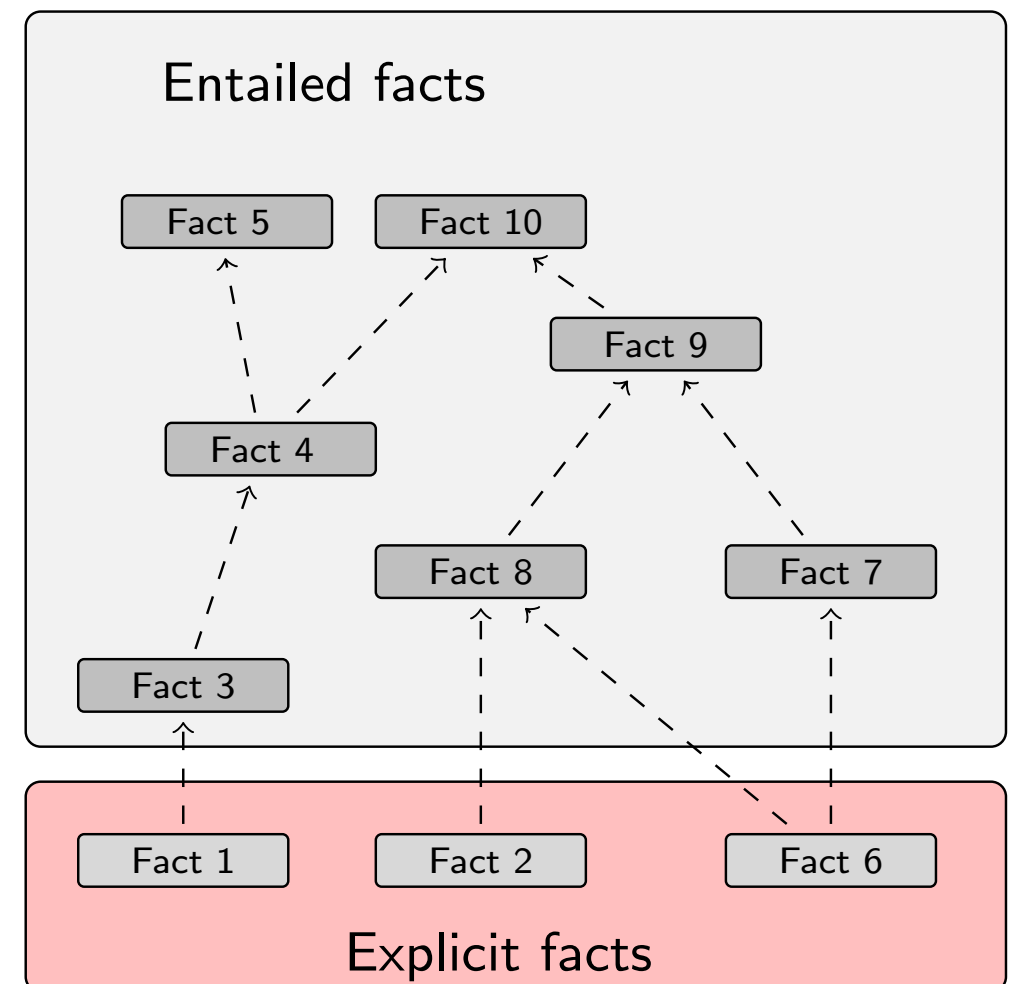
- Approche saturation : évaluer la requête sur le graphe saturé $\mathbf{K}^{\Gamma_{\infty}}$
- Demande un moteur de deduction entre le système de stockage et le système d'évaluation de requête
- Deux approches :
 - ▶ **Forward chaining**
 - ▶ **Backward chaining**

Forward chaining

I) Forward chaining

- Appliquer l'algorithme de saturation pour dériver les conséquences (le graphe saturé $\mathbf{K}^{\Gamma\infty}$) progressivement à partir des faits connus et matérialiser les conséquences au fur et à mesure
 - Evaluer la requête sur $\mathbf{K}^{\Gamma\infty}$ avec des techniques standard
- Le raisonnement se fait donc *des prémisses vers les conclusions* des règles : les faits ajoutés correspondent aux conclusions : ils sont stockés et réutilisés par la suite

- La matérialisation peut être faite avant l'exécution de la requête (phase statique)
- Solution adoptée par la plupart des moteurs SPARQL



Forward chaining

Avantages

- Le pré-calcul et le stockage des inférence est avantageux pour les données qui sont fréquemment interrogées
 - ▶ À condition que les données soient peu dynamiques
 - ▶ Et que les inférences ne soient pas trop volumineuses : elles doivent être stockées efficacement
- Materialisation pre-calculée : optimise la phase d'évaluation de la requête, aucune inférence additionnelle n'est nécessaire au moment de l'évaluation

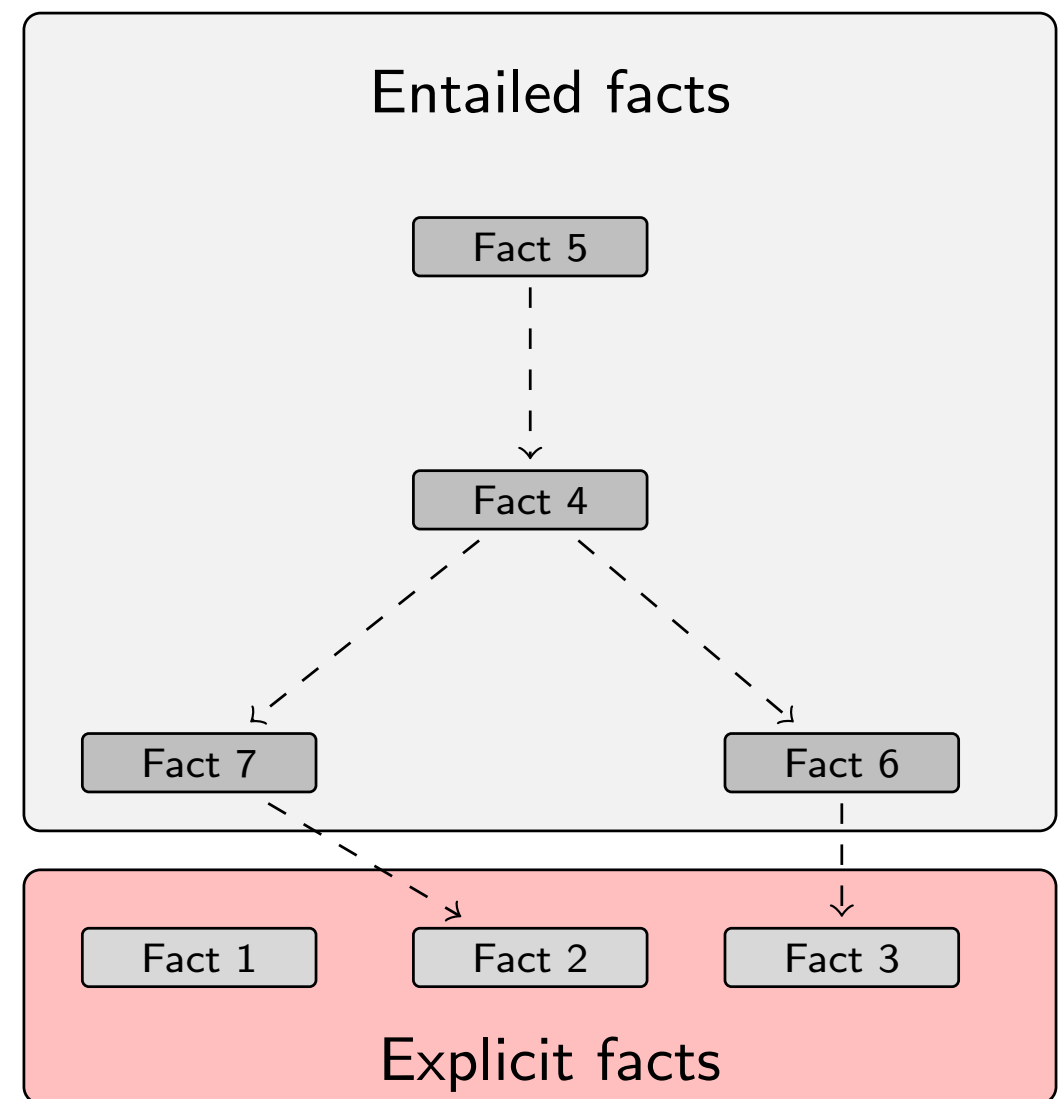
Inconvénients

- L'espace de stockage utilisé
 - ▶ Il peut y avoir beaucoup d'inférences, génère de très grands graphes
- Les surcoût de la modification des données (il faut mettre à jour les inférences)
 - ▶ pas adapté aux données très dynamiques

Backward chaining

2) Backward chaining

- Le raisonnement se fait *des conclusions vers les premisses* des règles
 - ▶ On part des faits nécessaires pour satisfaire la requête (les faits de la requête)
 - ▶ Quels autres faits doivent être vrais pour obtenir ces conclusions ?
- Les règles sont donc utilisées pour étendre la requête
- Raisonnement à la demande : seulement les faits dont on a besoin pour calculer une réponse sont calculés
- Ne nécessite pas de matérialisation persistante



Backward chaining : exemple

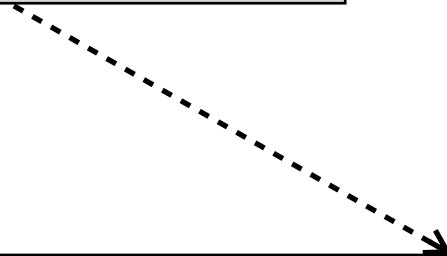
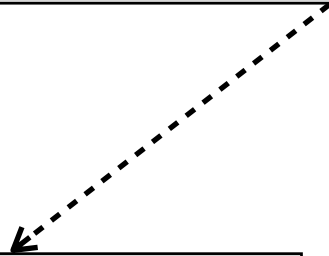
Base de connaissance

```
ex:Mammal rdfs:subClassOf ex:Vertebrate .  
ex:KillerWhale rdfs:subClassOf ex:Mammal .  
ex:Lion rdfs:subClassOf ex:Mammal .  
ex:Keiko rdf:type ex:KillerWhale .  
ex:Simba rdf:type ex:Lion .
```

Requête

```
SELECT ?x WHERE  
{ ?x rdf:type ex:Vertebrate . }
```

?x rdf:type ex:Vertebrate



?x rdf:type ex:Mammal

ex:Mammal rdfs:subClassOf ex:Vertebrate

Règle utilisée



```
<r rdf:type A>  
<A rdfs:subClassOf B>  
-----  
<r rdf:type B>
```

Backward chaining : exemple

Base de connaissance

```
ex:Mammal rdfs:subClassOf ex:Vertebrate .  
ex:KillerWhale rdfs:subClassOf ex:Mammal .  
ex:Lion rdfs:subClassOf ex:Mammal .  
ex:Keiko rdf:type ex:KillerWhale .  
ex:Simba rdf:type ex:Lion .
```

Requête

```
SELECT ?x WHERE  
{ ?x rdf:type ex:Vertebrate . }
```

Règle utilisée

↑
⋮
<r rdf:type A>
<A rdfs:subClassOf B>
———
<r rdf:type B>

↑
⋮
<r rdf:type A>
<A rdfs:subClassOf B>
———
<r rdf:type B>

?x rdf:type ex:Vertebrate

?x rdf:type ex:Mammal

ex:Mammal rdfs:subClassOf ex:Vertebrate

?x rdf:type ex:KillerWhale

?KillerWhale rdfs:subClassOf ex:Mammal

Backward chaining : exemple

Base de connaissance

```
ex:Mammal rdfs:subClassOf ex:Vertebrate .  
ex:KillerWhale rdfs:subClassOf ex:Mammal .  
ex:Lion rdfs:subClassOf ex:Mammal .  
ex:Keiko rdf:type ex:KillerWhale .  
ex:Simba rdf:type ex:Lion .
```

Requête

```
SELECT ?x WHERE  
{ ?x rdf:type ex:Vertebrate . }
```

Règle utilisée

```
<r rdf:type A>  
<A rdfs:subClassOf B>  
-----  
<r rdf:type B>
```

```
<r rdf:type A>  
<A rdfs:subClassOf B>  
-----  
<r rdf:type B>
```

?x rdf:type ex:Vertebrate

?x rdf:type ex:Mammal

ex:Mammal rdfs:subClassOf ex:Vertebrate

?x rdf:type ex:KillerWhale

?KillerWhale rdfs:subClassOf ex:Mammal

?x = ex:Keiko

ex:Keiko rdf:type ex:KillerWhale

Backward chaining : exemple

Base de connaissance

```
ex:Mammal rdfs:subClassOf ex:Vertebrate .  
ex:KillerWhale rdfs:subClassOf ex:Mammal .  
ex:Lion rdfs:subClassOf ex:Mammal .  
ex:Keiko rdf:type ex:KillerWhale .  
ex:Simba rdf:type ex:Lion .
```

Requête

```
SELECT ?x WHERE  
{ ?x rdf:type ex:Vertebrate . }
```

Règle utilisée

```
<r rdf:type A>  
<A rdfs:subClassOf B>  
-----  
<r rdf:type B>
```

```
<r rdf:type A>  
<A rdfs:subClassOf B>  
-----  
<r rdf:type B>
```

?x rdf:type ex:Vertebrate

?x rdf:type ex:Mammal

ex:Mammal rdfs:subClassOf ex:Vertebrate

?x rdf:type ex:Lion

?Lion rdfs:subClassOf ex:Mammal

?x = ex:Simba

ex:Simba rdf:type ex:Lion

Backward chaining

Avantages

- Seulement les inférences utiles sont faites, pas de mémorisation persistante
- Adapté à des sources dynamiques (pas de re-calcul des inférences nécessaire suite aux modifications des données)
- Utile quand il n'y a pas (ou il y a peu) besoin de réutiliser les réponses déjà calculées

Inconvénients

- Temps de calculs plus lents : les inférences doivent être faites à chaque requête
 - ▶ Caching possible pour améliorer les performances

Calcul des réponses en présence d'inférence

- Approche saturation :
 - ▶ Le graphe saturé est calculé plus ou moins entièrement pour y évaluer les requêtes
- **Approche réécriture**
 - ▶ Pas de calcul d'inférences : la requête est re-écrite pour “simuler” la présence des faits dérivés

Approche réécriture

- Rappel : pour calculer les réponses à une requête sur une base de connaissance K en présence d'inférence il faudrait calculer

$$Q(K^{\Gamma_{\infty}})$$

- L'approche réécriture en revanche construit une nouvelle requête Q' telle que

$$Q(K^{\Gamma_{\infty}}) = Q'(K) \quad \text{pour tout } K$$

(Q' dépendant de Γ)

- Cela évite de calculer les inférences ! On interroge seulement les faits explicites

Approche réécriture : exemple

Q
SELECT ?x WHERE { ?x rdf:type ex:Vertebrate . }

Q'
SELECT ?x WHERE {
{?x rdf:type ?z . ?z rdfs:subClassOf* ex:Vertebrate .}
UNION
{?x ?p ?y . ?p rdfs:subPropertyOf*/rdfs:domain/rdfs:subClassOf* ex:Vertebrate}
UNION
{?y ?p ?x . ?p rdfs:subPropertyOf*/rdfs:range/rdfs:subClassOf* ex:Vertebrate }
}

- Sous sémantique RDFS :

$$Q(K^{\text{RDFS}\infty}) = Q'(K) \quad \text{pour tout } K$$

- Intuitivement les règles de dérivation RDFS sont “intégrées” dans la réécriture Q'

Approche réécriture : exemple

- La réécriture n'est pas toujours expressible en SPARQL même
- En effet pour des requêtes SPARQL Q très simples Q' demande un langage plus puissant
- Par exemple :

Q_0

```
SELECT ?x ?y WHERE { ?x :prop ?y }
```

a une réécriture en SPARQL

Q_0'

```
SELECT ?x ?y WHERE { ?x ?z ?y . ?z rdfs:subPropertyOf* :prop }
```

- Mais comment réécrire en SPARQL

Q_1

```
SELECT ?x ?y WHERE { ?x :prop+ ?y }
```

?

Approche réécriture : exemple

- Comment réécrire en SPARQL

QI

```
SELECT ?x ?y WHERE { ?x :prop+ ?y }
```

?

- Cela n'est pas possible.
- Il faudrait pouvoir avoir l'étoile (+) au dessus d'un motif de graphe :

```
?x ( ?start ?z ?end . ?z rdfs:subPropertyOf* :prop )+ ?y
```
- Mais cela n'est pas autorisé en SPARQL
- nSPARQL* : une extension de SPARQL proposée pour inclure ces formes expressions régulières “imbriquées”
 - ▶ Pourrait être utilisé internement aux moteurs de réécriture

* J. Pérez, M. Arenas, and C. Gutierrez. nSPARQL: A navigational language for RDF J. Web Sem., 8(4):255–270, 2010.

Approche réécriture

Avantages

- Pas de stockage d'inférences (ni en disque ni en mémoire)
- pas besoin de moteur d'inférence sur les données, tout est délégué au moteur d'évaluation de requête

Inconvénients

- Algorithmes de réécritures et réécritures complexes (surtout au delà de RDFS)

Back to theory : les réponses certaines

- Rappel :
Interrogation d'une ontologie : intuitivement toute réponse derivable implicitement par les faits et contraintes du graphe interrogé doit être prise en compte
- Pour les bases de connaissance RDFS/OWL le W3C a adopté une sémantique opérationnelle pour obtenir cela :
 - ▶ Répondre à la requête comme si le graphe était saturé
- La recherche théorique a proposé en revanche une notion d'inférence logique pour définir la sémantique d'interrogation d'ontologies :

Soit K une ontologie

On dit qu'une réponse à Q est **impliquée** par K (ou est une **réponse certaine** à Q sur K) si a est une réponse à $Q(M)$ pour tout modèle M de K

Rappel : modèle

- Soit K une ontologie RDFS avec faits A et contraintes T

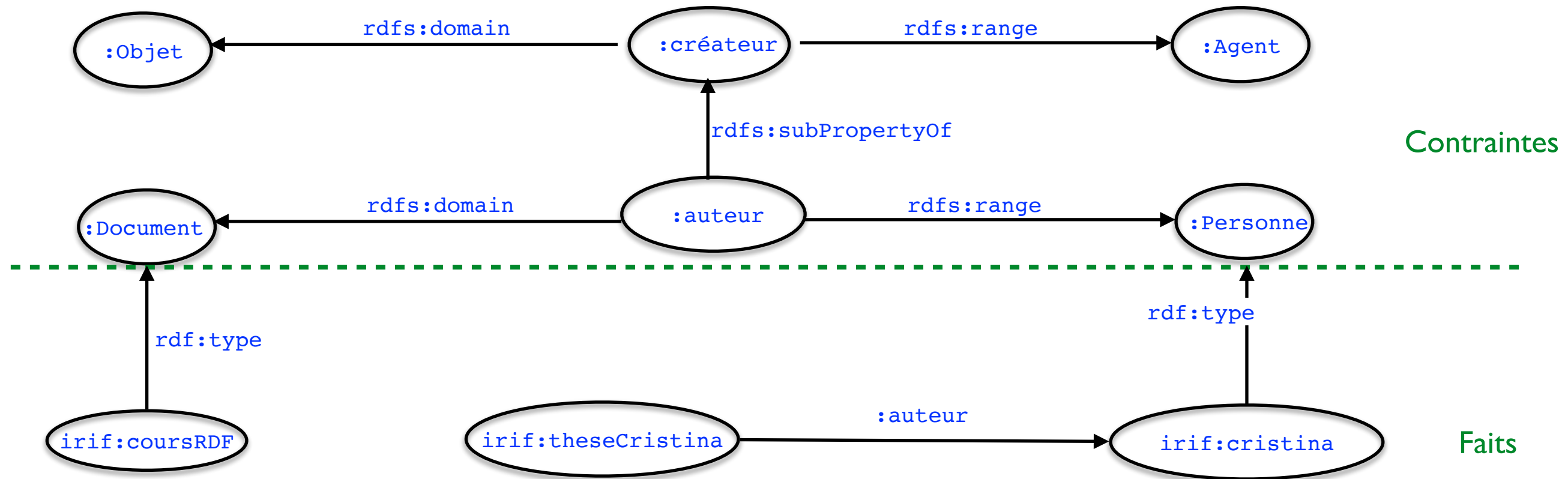
un **modèle RDFS** de K est un graphe RDF dont les propriétés sont soit dans P soit des propriétés prédéfinies `rdf/rdfs`, qui :

- ▶ contient les triplets A et T et les triplets axiomatiques,
- ▶ satisfait les contraintes prédéfinies RDFS
(cf slide “Ontologie RDFS prédéfinie - contraintes”)

- Un modèle peut être défini de façon analogue quel que soit le langage d'ontologie choisi

Responses certaines : exemple

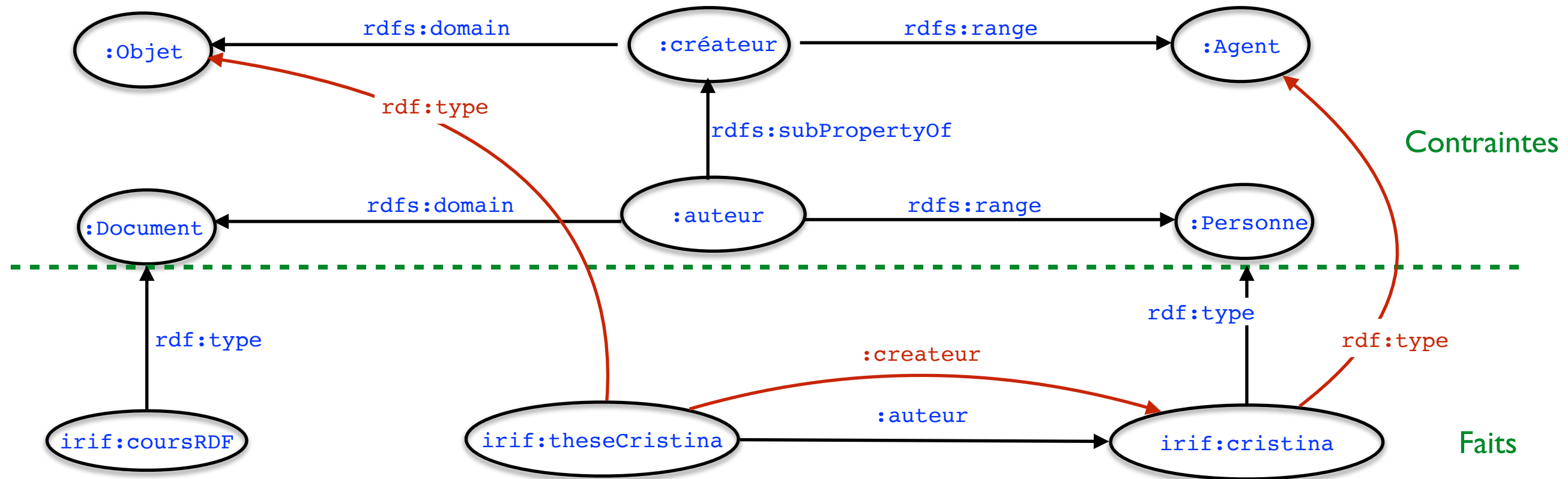
- Exemple : K requête : “quels sont les objets créés par un agent”



- `irif:theseCristina` est une **réponse certaine**

Responses certaines : exemple

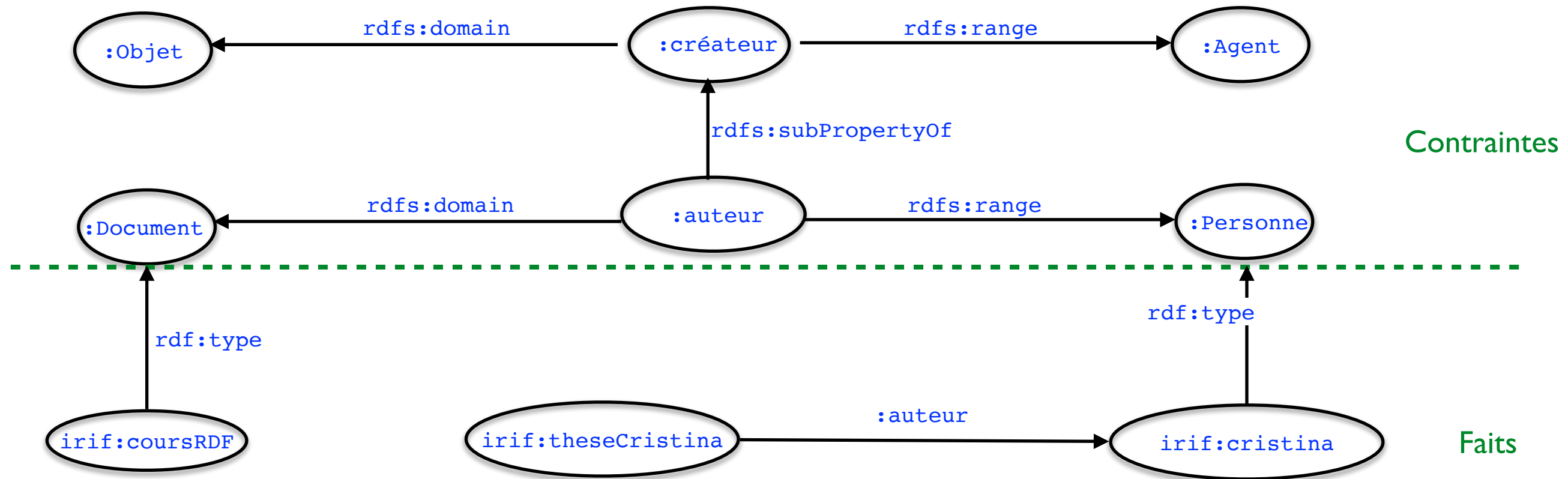
- Exemple : K requête : “quels sont les objets créés par un agent”



- `irif:theseCristina` est une **réponse certaine**
 - Parce que les faits en rouge ci-dessus sont présents dans tous les modèles (RDFS) de K

Responses certaines : exemple

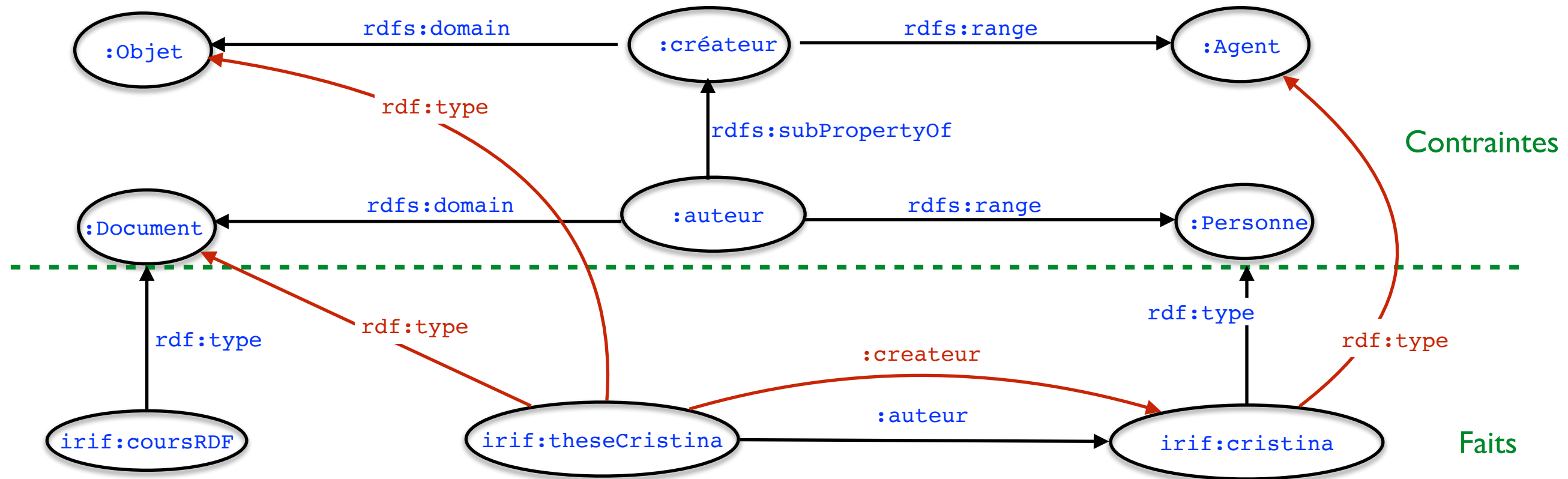
- Exemple : K requête : “quels sont les objets créés par un agent”



- irif:coursRDF** n'est **pas une réponse certaine**

Responses certaines : exemple

- Exemple : K



- `irif:coursRDF` n'est **pas une réponse certaine**

- En effet il y a bien des modèles où `irif:coursRDF` n'a pas de créateur (rappel : seuls les faits en rouge sont présents dans tous les modèles RDFS)

Réponses certaines et interrogation du graphe saturé

- Ils sont bien sûr liés !
- Le graphe saturé \mathbf{K}^∞ est contenu dans tous les modèles de K (par définition)
- Mais il est également un modèle (satisfait les contraintes)
- **Requêtes monotones**
 - ▶ Une requête Q est monotone si $Q(M') \subseteq Q(M)$ pour tout $M' \subseteq M$
 - ▶ \Rightarrow pour toute requête Q monotone $Q(\mathbf{K}^\infty) \subseteq Q(M)$ pour tout modèle M
 - ▶ \Rightarrow si $t \in Q(\mathbf{K}^\infty)$ alors t est une réponse certaine
 - ▶ Mais si t est certaine alors $t \in Q(\mathbf{K}^\infty)$ (car \mathbf{K}^∞ est un modèle)
 - ▶ \Rightarrow

Pour toute requête monotone (incluant les BGP)
 $Q(\mathbf{K}^\infty)$ coïncide avec les réponses certaines

La théorie rencontre la pratique, mais jusqu'où?

Pour toute requête monotone (incluant les BGP)
 $Q(K^\infty)$ coïncide avec les réponses certaines

La sémantique d'interrogation recommandée par le W3C ($Q(K^\infty)$) coïncide avec l'approche implication logique

- Mais cela n'est pas garanti au delà des requêtes monotones !
- Fragment SPARQL monotone
 - ▶ BGP plus UNION, . , FILTER positif, expressions de chemin, et dans une certaine mesure OPTIONAL
- En presence de negation $Q(K^\infty)$ ne calcule pas les réponses certaines
 - ▶ A-t-il encore du sens d'adopter $Q(K^\infty)$ comme sémantique d'interrogation ?
 - ▶ Le W3C ne s'est pas encore officiellement posé la question...
 - ▶ ...attendons SPARQL 1.2 !