Vision logique de RDFS

HAI824 – Traitement sémantique des données

ML Mugnier

Contenu

- Rappels de RDF(S)
  - Graphe RDF(S) : comporte à la fois des triplets « factuels » et des triplets « ontologiques »
  - Règles d'inférence RDF (« RDF entailment rules »)
  - Saturation d'un graphe avec les règles d'inférence
  - Interrogation d'un graphe RDF saturé avec une requête SPARQL
    (la sous-classe des Basic Graph Pattern Queries correspond aux CQ)

- Deux traductions RDFS → logique

  qui produisent toutes deux une base de connaissances
  composée d'une base de faits et d'une base de règles Datalog

RDF Graph

An *RDF graph* G is a set of triples $(\mathtt{s}, \mathtt{p}, \mathtt{o})$ where

- $\mathtt{s}$ is the subject (IRI or Blank)
- $\mathtt{p}$ is the property (IRI)
- $\mathtt{o}$ is the object (IRI, Blank or Literal)

Example

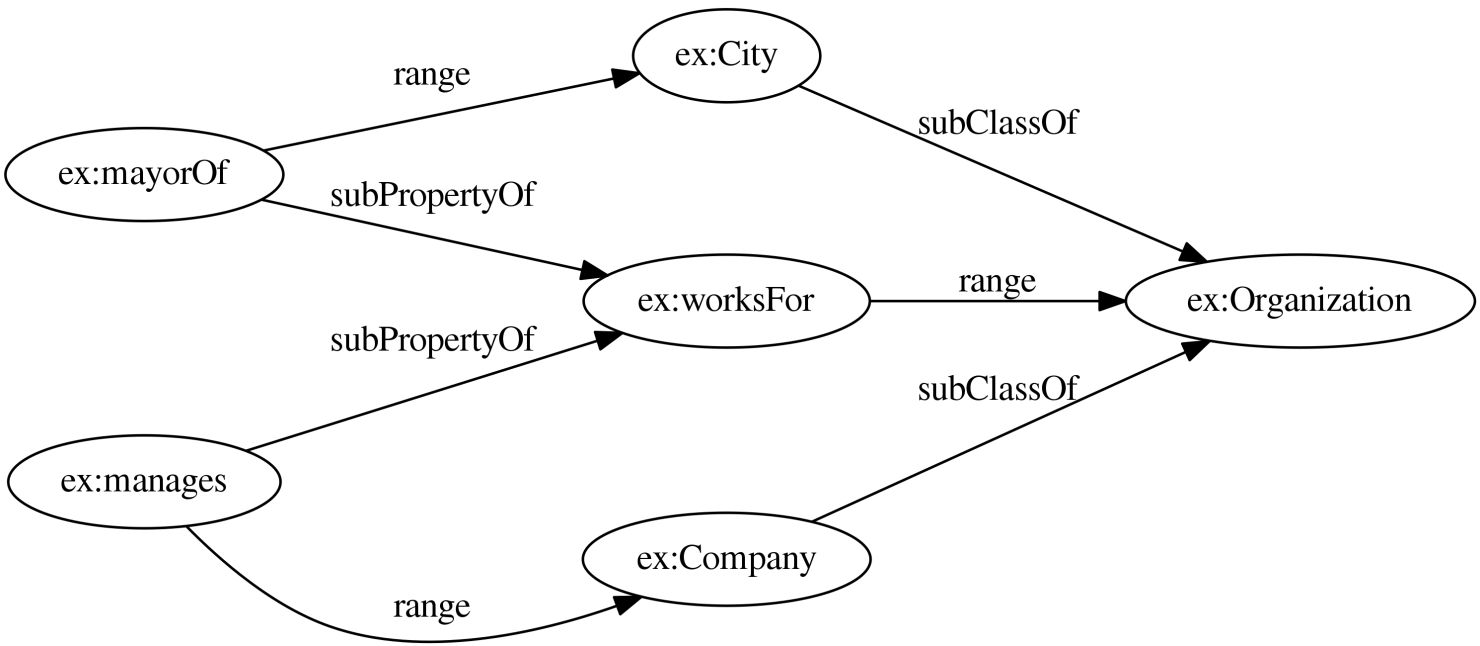
$G =$

RDFS Ontology (RDF Schema)

An *RDFS ontology* declares semantic constraints between classes and properties using built-in properties:

- rdfs:subClassOf inclusion between classes
- rdfs:subPropertyOf inclusion between properties
- rdfs:domain class of the subject of a property
- rdfs:range class of the object of a property

Ontology Example



$O =$

Main RDF Entailment Rules

In the rules below, a rule body defines a "pattern" that has to be found to apply the rule
The letters p, o and s have to be replaced by values
The words subclass, subproperty, domain and range are prefixed by "rdfs:"

$$(\mathrm{p}, \mathit{domain}, \mathrm{o}), (s_1, \mathrm{p}, \mathrm{o}_1) \rightarrow (s_1, \mathit{type}, \mathrm{o})$$
$$(\mathrm{p}, \mathit{range}, \mathrm{o}), (\mathrm{s}_1, \mathrm{p}, \mathrm{o}_1) \rightarrow (\mathrm{o}_1, \mathit{type}, \mathrm{o})$$
$$(\mathrm{p}_1, \mathit{subproperty}, \mathrm{p}_2), (\mathrm{p}_2, \mathit{subproperty}, \mathrm{p}_3) \rightarrow (\mathrm{p}_1, \mathit{subproperty}, \mathrm{p}_3)$$
$$(\mathrm{p}_1, \mathit{subproperty}, \mathrm{p}_2), (\mathrm{s}, \mathrm{p}_1, \mathrm{o}) \rightarrow (\mathrm{s}, \mathrm{p}_2, \mathrm{o})$$
$$(\mathrm{s}, \mathit{subclass}, \mathrm{o}), (\mathrm{s}_1, \mathit{type}, \mathrm{s}) \rightarrow (\mathrm{s}_1, \mathit{type}, \mathrm{o})$$
$$(\mathrm{s}, \mathit{subclass}, \mathrm{o}), (\mathrm{o}, \mathit{subclass}, \mathrm{o}_1) \rightarrow (\mathrm{s}, \mathit{subclass}, \mathrm{o}_1)$$
$$(\mathrm{p}, \mathit{domain}, \mathrm{o}), (\mathrm{o}, \mathit{subclass}, \mathrm{o}_1) \rightarrow (\mathrm{p}, \mathit{domain}, \mathrm{o}_1)$$
$$(\mathrm{p}, \mathit{range}, \mathrm{o}), (\mathrm{o}, \mathit{subclass}, \mathrm{o}_1) \rightarrow (\mathrm{p}, \mathit{range}, \mathrm{o}_1)$$
$$(\mathrm{p}, \mathit{subproperty}, \mathrm{p}_1), (\mathrm{p}_1, \mathit{domain}, \mathrm{o}) \rightarrow (\mathrm{p}, \mathit{domain}, \mathrm{o})$$
$$(\mathrm{p}, \mathit{subproperty}, \mathrm{p}_1), (\mathrm{p}_1, \mathit{range}, \mathrm{o}) \rightarrow (\mathrm{p}, \mathit{range}, \mathrm{o})$$

These rules are not expressed in logic
but a similar notion of homomorphism is defined to apply rules

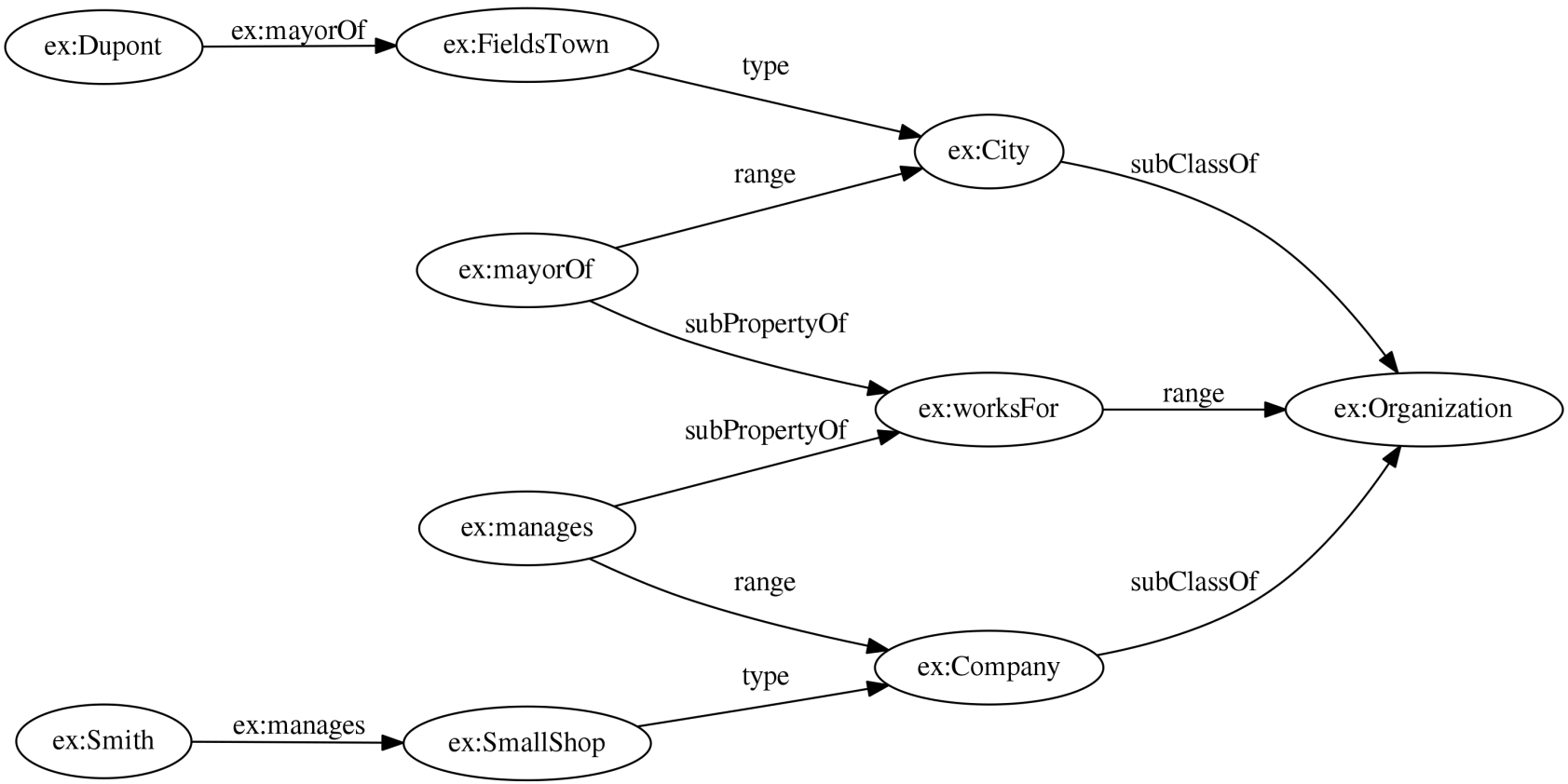
RDF Entailment Rules (Sample)

Let $\mathcal{R}$ contain the following RDF entailment rules:

1. $(\mathtt{p}_1, \text{rdfs:subPropertyOf}, \mathtt{p}_2), (\mathtt{s}, \mathtt{p}_1, \mathtt{o}) \rightarrow (\mathtt{s}, \mathtt{p}_2, \mathtt{o})$
2. $(\mathtt{p}, \text{rdfs:range}, \mathtt{o}), (\mathtt{s}_1, \mathtt{p}, \mathtt{o}_1) \rightarrow (\mathtt{o}_1, \text{rdf:type}, \mathtt{o})$
3. $(\mathtt{p}, \text{rdfs:range}, \mathtt{o}), (\mathtt{o}, \text{rdfs:subClassOf}, \mathtt{o}_1) \rightarrow (\mathtt{p}, \text{rdfs:range}, \mathtt{o}_1)$

Saturation Example

$G \cup O =$



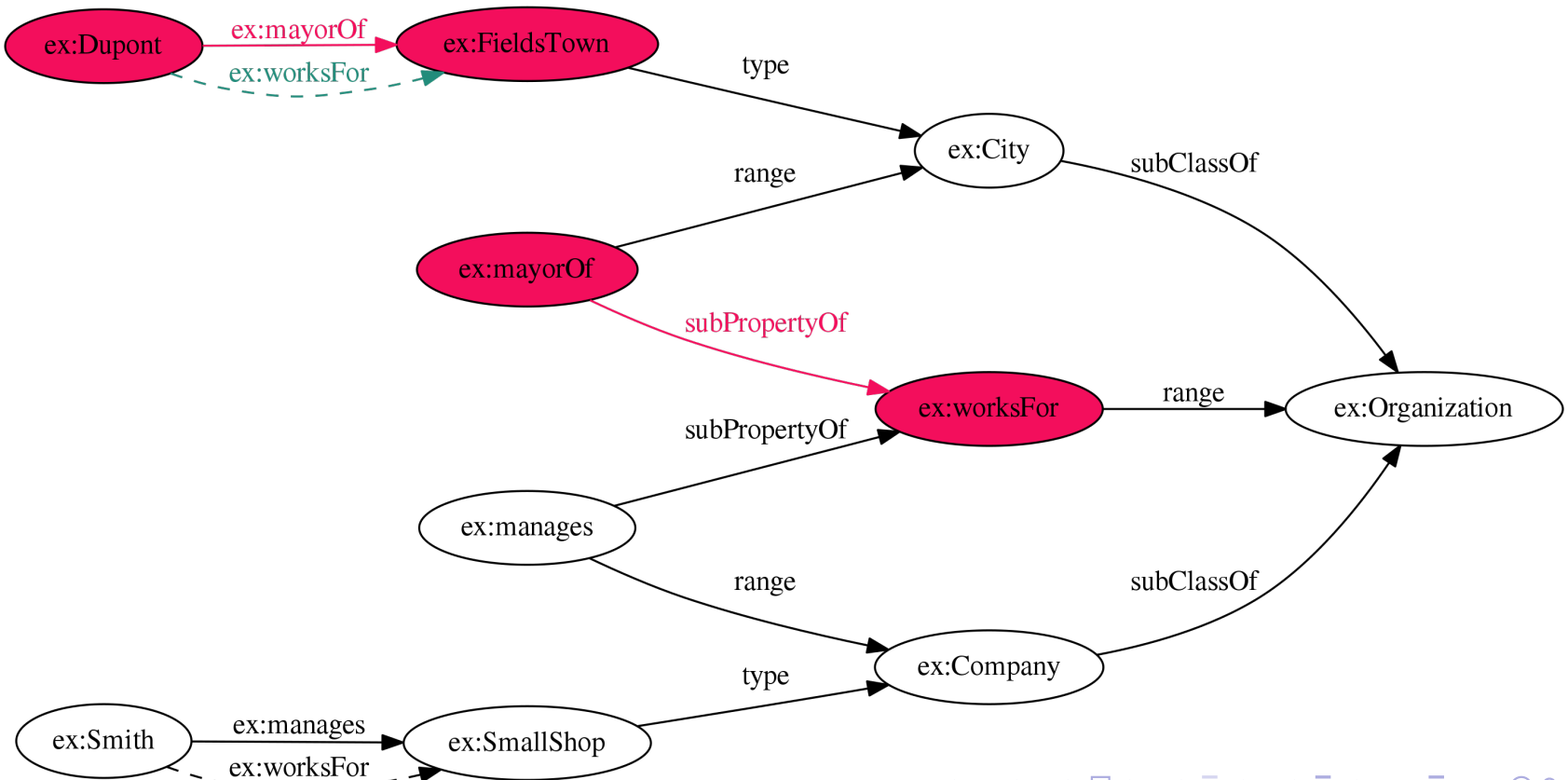
RDF Entailment Rules

$\mathcal{R}$ contains the following RDF entailment rules:

1. $(\mathtt{p}_1, \text{rdfs:subPropertyOf}, \mathtt{p}_2), (\mathtt{s}, \mathtt{p}_1, \mathtt{o}) \rightarrow (\mathtt{s}, \mathtt{p}_2, \mathtt{o})$
2. $(\mathtt{p}, \text{rdfs:range}, \mathtt{o}), (\mathtt{s}_1, \mathtt{p}, \mathtt{o}_1) \rightarrow (\mathtt{o}_1, \text{rdf:type}, \mathtt{o})$
3. $(\mathtt{p}, \text{rdfs:range}, \mathtt{o}), (\mathtt{o}, \text{rdfs:subClassOf}, \mathtt{o}_1) \rightarrow (\mathtt{p}, \text{rdfs:range}, \mathtt{o}_1)$

Saturation Example

$(G \cup O)^{\mathcal{R}} =$



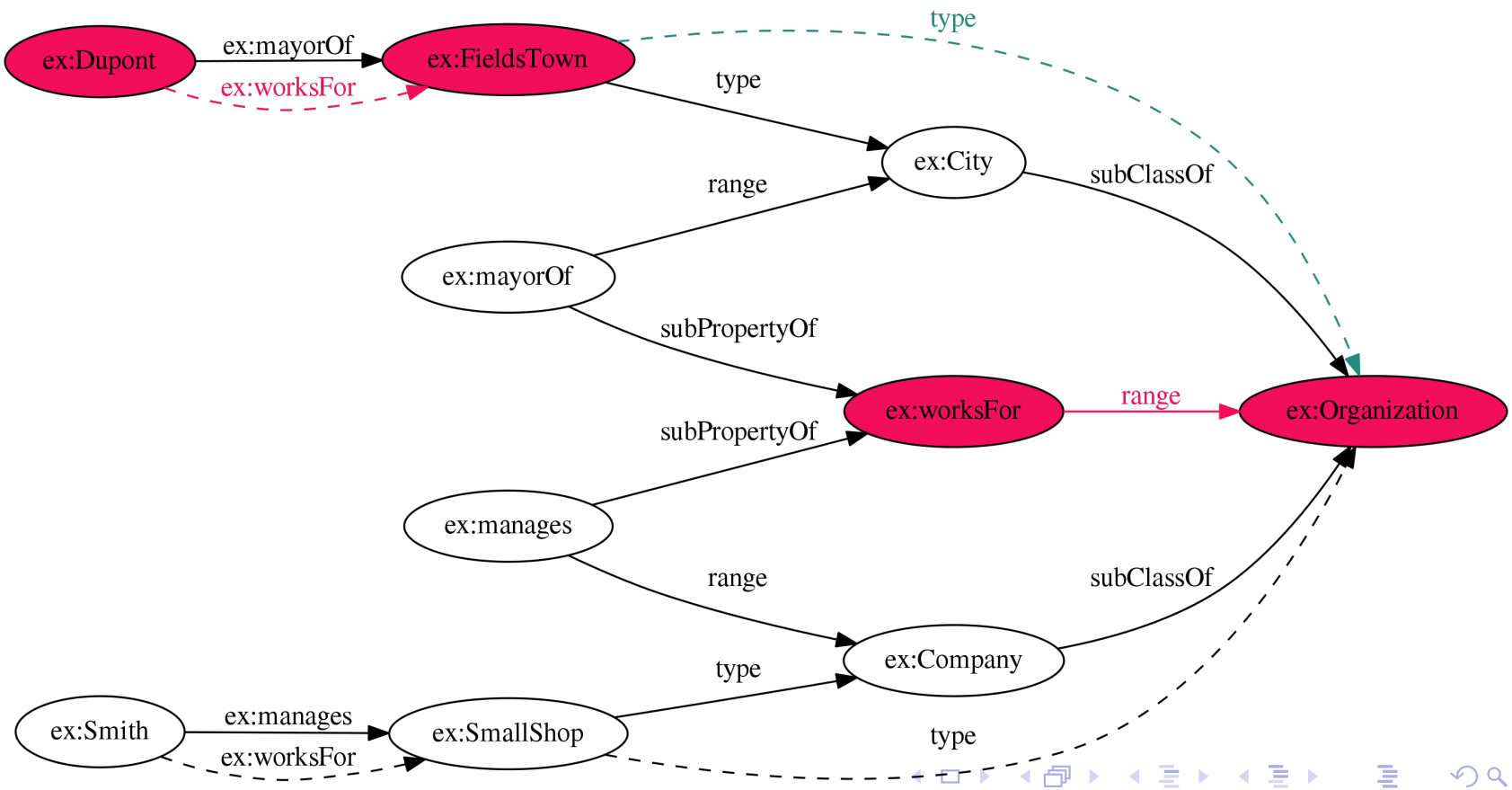
RDF Entailment Rules

$\mathcal{R}$ contains the following RDF entailment rules:

1. $(\mathtt{p}_1, \text{rdfs:subPropertyOf}, \mathtt{p}_2), (\mathtt{s}, \mathtt{p}_1, \mathtt{o}) \rightarrow (\mathtt{s}, \mathtt{p}_2, \mathtt{o})$
2. $(\mathtt{p}, \text{rdfs:range}, \mathtt{o}), (\mathtt{s}_1, \mathtt{p}, \mathtt{o}_1) \rightarrow (\mathtt{o}_1, \text{rdf:type}, \mathtt{o})$
3. $(\mathtt{p}, \text{rdfs:range}, \mathtt{o}), (\mathtt{o}, \text{rdfs:subClassOf}, \mathtt{o}_1) \rightarrow (\mathtt{p}, \text{rdfs:range}, \mathtt{o}_1)$

Saturation Example

$(G \cup O)^{\mathcal{R}} =$



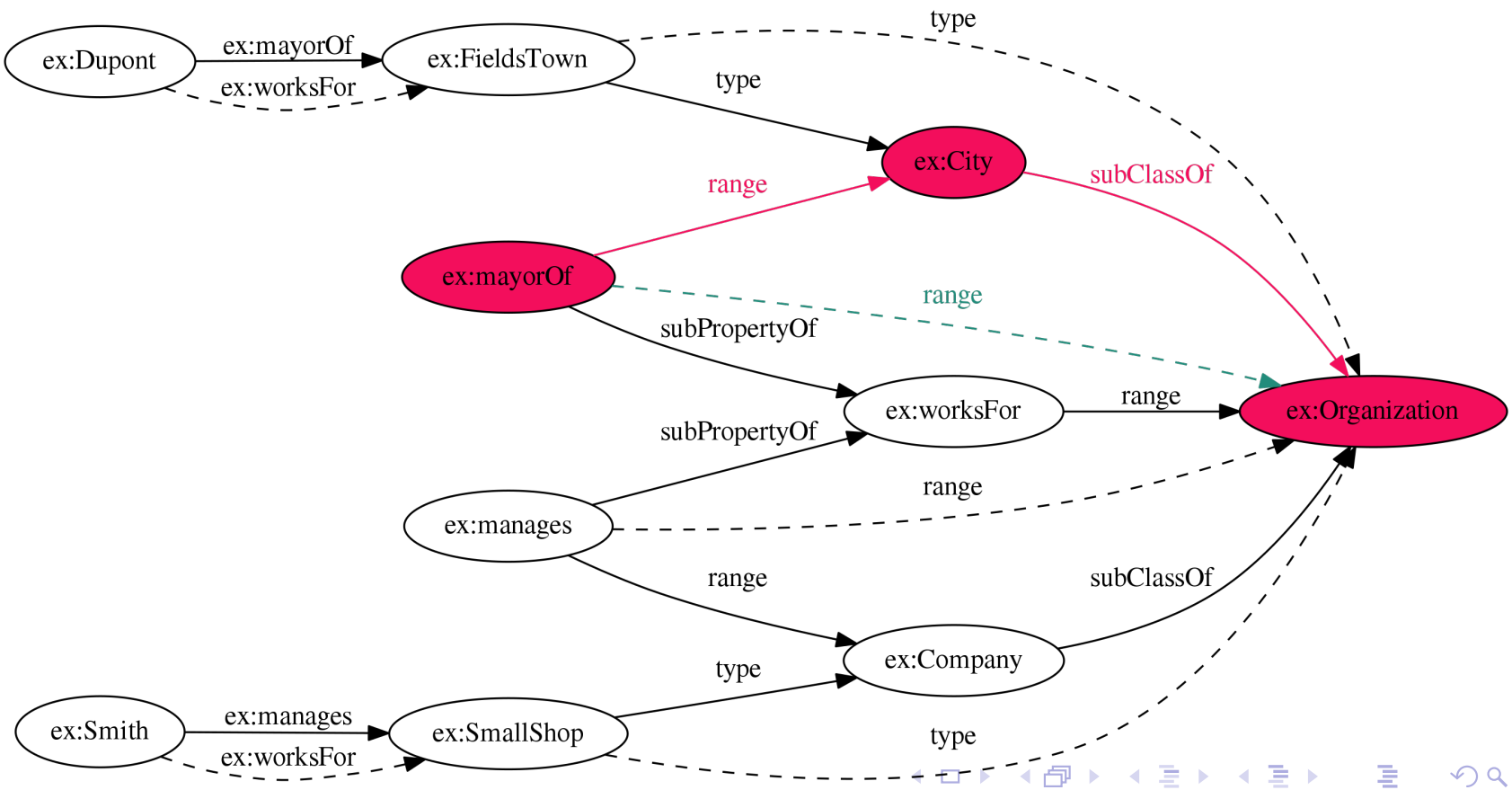
RDF Entailment Rules (we may also have non-standard RDF entailment rules)

$\mathcal{R}$ contains the following RDF entailment rules:

1. $(\mathtt{p}_1, \text{rdfs:subPropertyOf}, \mathtt{p}_2), (\mathtt{s}, \mathtt{p}_1, \mathtt{o}) \rightarrow (\mathtt{s}, \mathtt{p}_2, \mathtt{o})$
2. $(\mathtt{p}, \text{rdfs:range}, \mathtt{o}), (\mathtt{s}_1, \mathtt{p}, \mathtt{o}_1) \rightarrow (\mathtt{o}_1, \text{rdf:type}, \mathtt{o})$
3. $(\mathtt{p}, \text{rdfs:range}, \mathtt{o}), (\mathtt{o}, \text{rdfs:subClassOf}, \mathtt{o}_1) \rightarrow (\mathtt{p}, \text{rdfs:range}, \mathtt{o}_1)$

Saturation Example

$(G \cup O)^{\mathcal{R}} =$



Basic Graph Pattern Query

A *Basic Graph Pattern query* is a conjunctive query on RDF triples.

Example

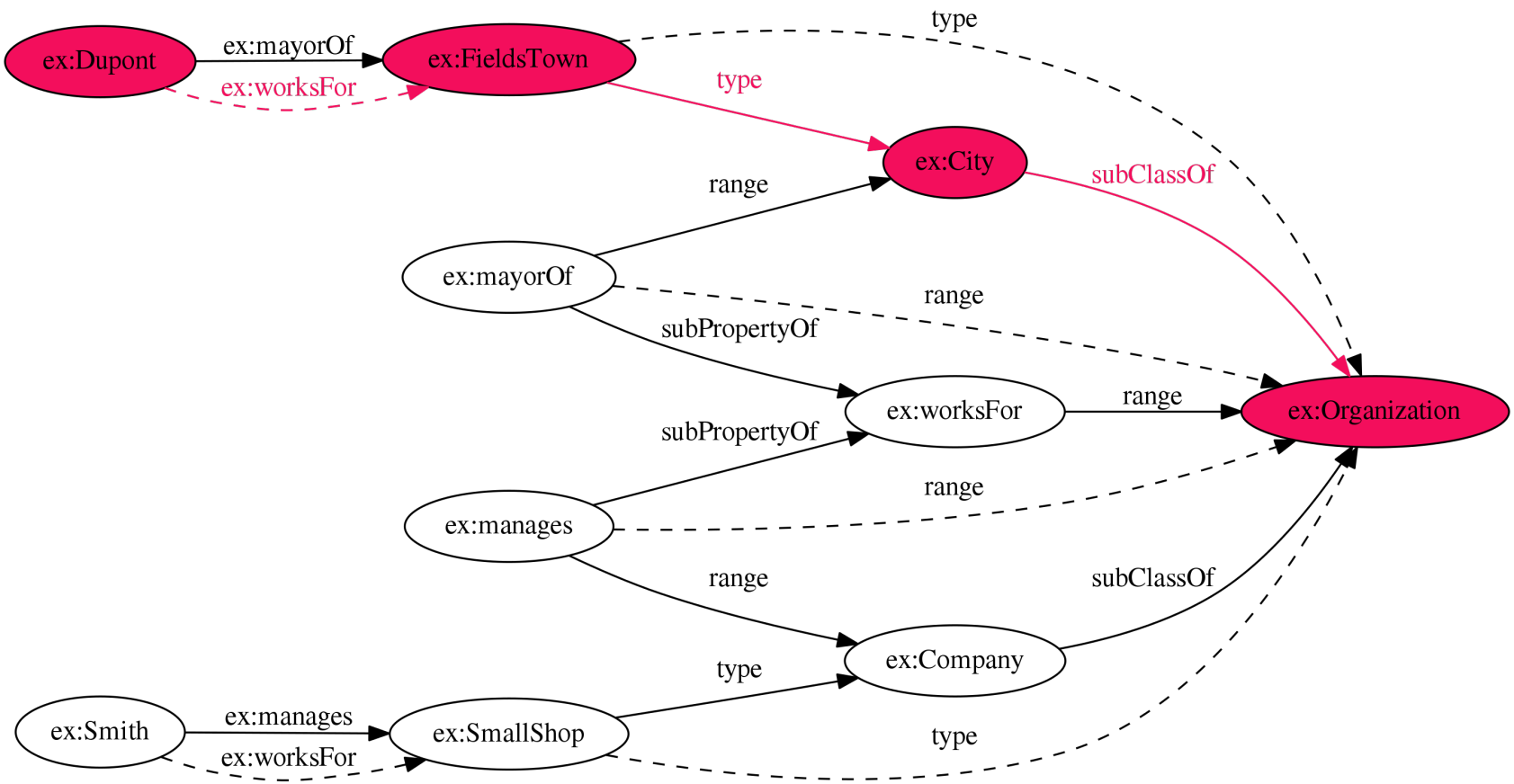
$$q(pers, orgClass) \leftarrow (pers, ex{:}\mathrm{worksFor}, org),\ (org, \mathrm{rdf{:}type}, orgClass),\ (orgClass, \mathrm{rdfs{:}subClassOf}, ex{:}\mathrm{Organization})$$

SPARQL Syntax

```
SELECT ?pers ?orgClass
WHERE {
    ?pers ex:worksFor ?org.
    ?org rdf:type ?orgClass.
    ?orgClass rdfs:subClassOf ex:Organization.
}
```

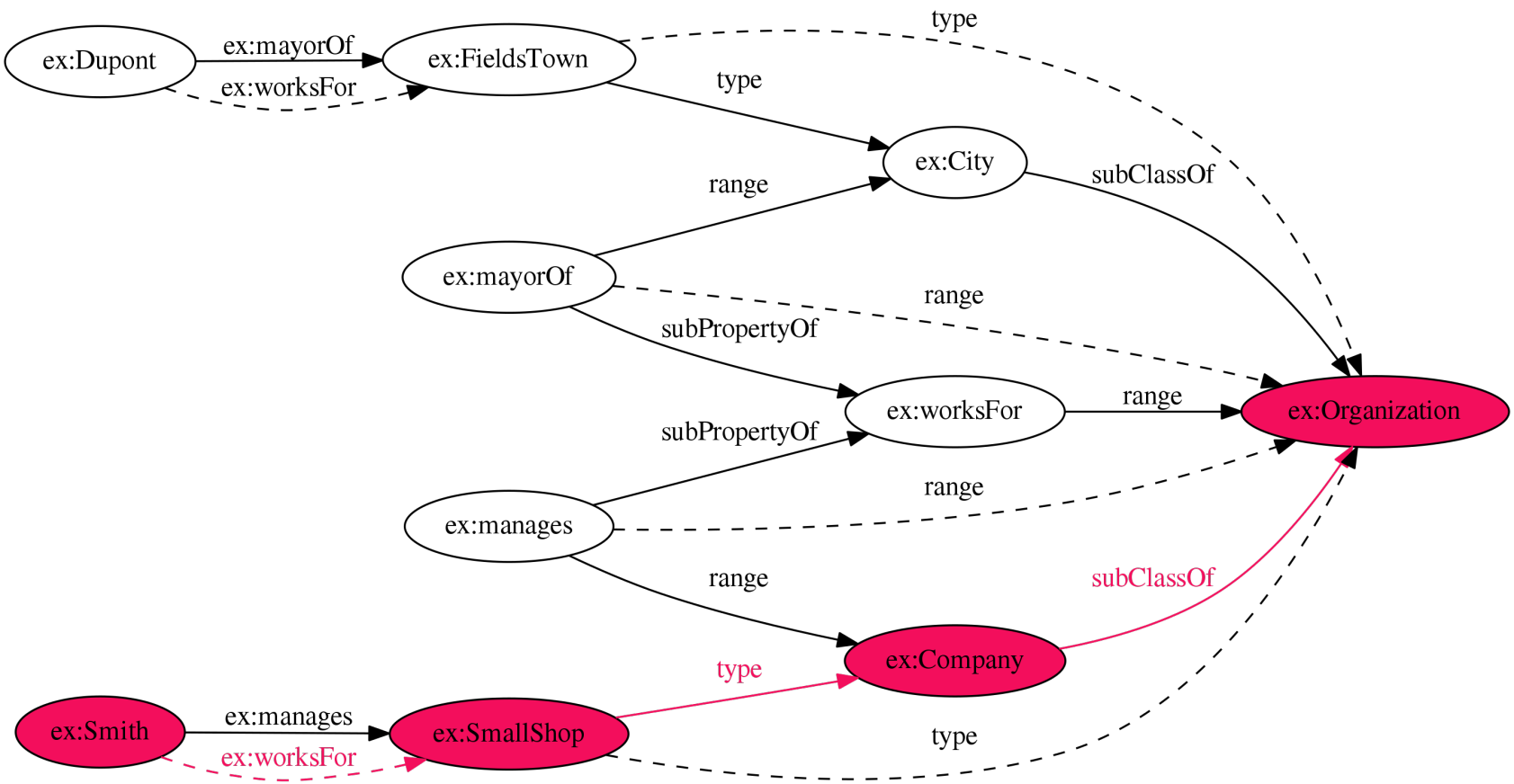
Query Answering Example

$$q(pers, orgClass) \leftarrow (pers, ex{:}\text{worksFor}, org),\ (org, \text{rdf:type}, orgClass),\ (orgClass, \text{rdfs:subClassOf}, ex{:}\text{Organization})$$



Query Answering Example

$$q(pers, orgClass) \leftarrow (pers, ex{:}\text{worksFor}, org),\ (org, \text{rdf:type}, orgClass),\ (orgClass, \text{rdfs:subClassOf}, ex{:}\text{Organization})$$



En RDFS Tout est « Triplet »

- Triplets factuels (data)

  (s rdf:type o) // s instance, o classe

  (s p o) où p est une propriété « utilisateur » // s et o instances

- Triplets ontologiques
  - (s p o) où p ∈ { rdfs:subClassOf, rdfs:subPropertyOf,

    rdfs:domain, rdfs:range }

    // s et o sont des classes ou des propriétés selon le cas

En RDFS Tout est « Triplet »

- Triplets factuels (data)

  (s rdf:type o) // s instance, o classe
  (s p o) où p est une propriété « utilisateur » // s et o instances

- Triplets ontologiques
  - (s p o) où p ∈ { rdfs:subClassOf, rdfs:subPropertyOf,
    rdfs:domain, rdfs:range }
    // s et o sont des classes ou des propriétés selon le cas

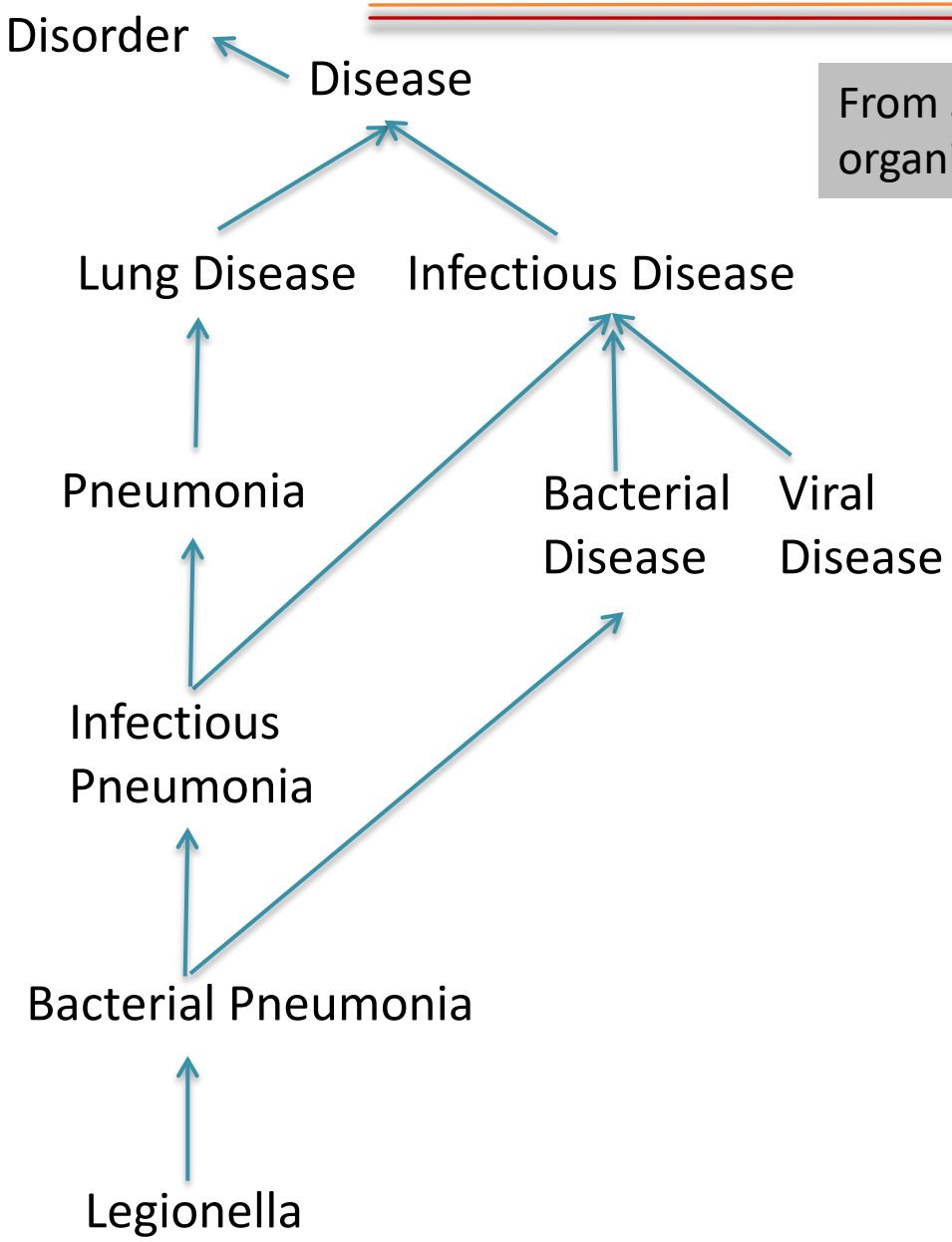
- Une entité peut être à la fois une instance, une classe et une propriété

  (Dupont, mayorOf, FieldsTown), (FieldsTown, rdf:type, City)
  (City, rdf:type, TypeOfHabitation)

  (Dupont, mayorOf, FieldsTown), (Dupont, rdf:type, mayorOf)

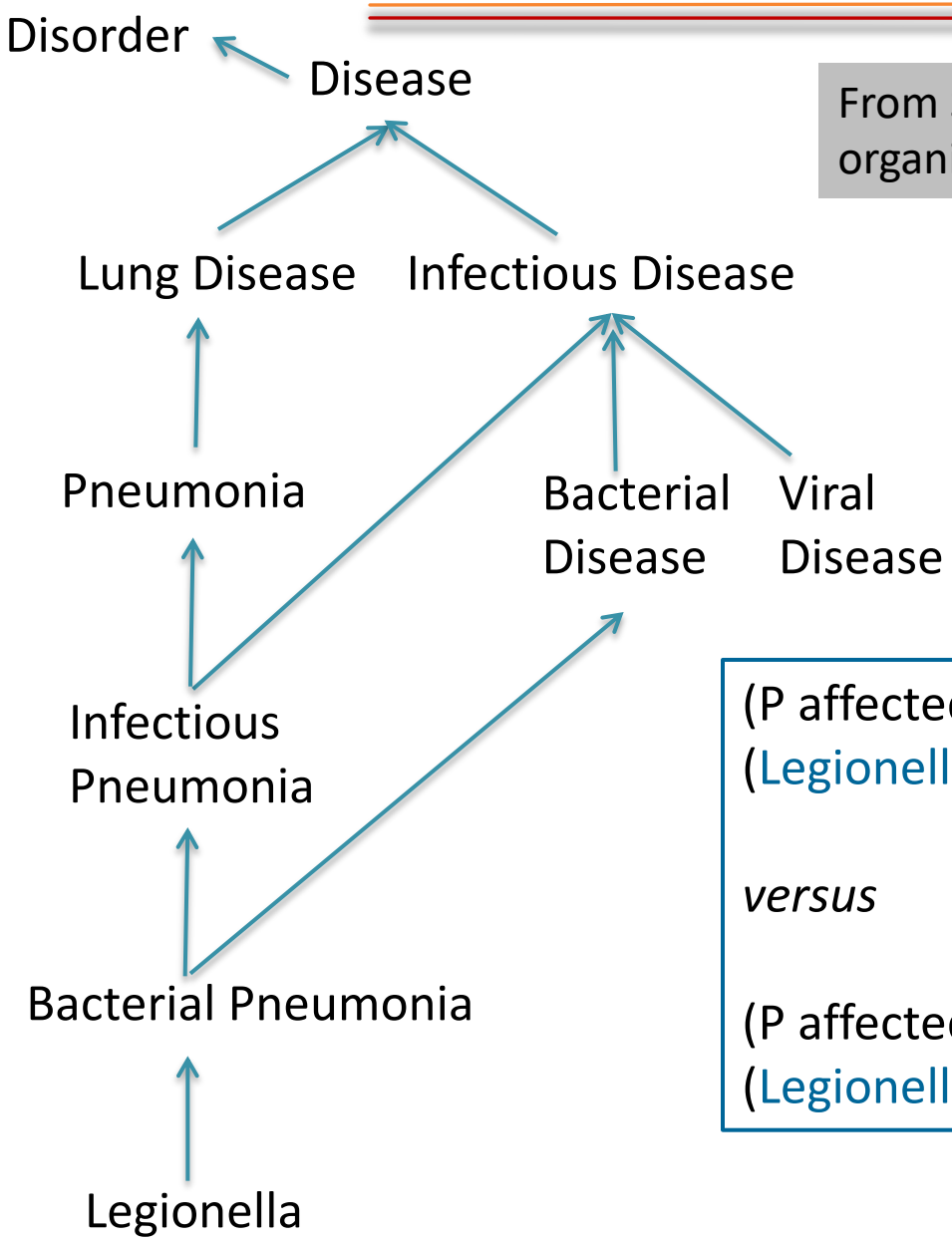
  (rdf:type rdf:type rdf:type) // triplet légal !

EXEMPLE : CLASSE UTILISÉE COMME L'UNE DE SES INSTANCES



From SNOMED CT: hierarchy of concepts organized by specialization (subclass)

EXEMPLE : CLASSE UTILISÉE COMME L'UNE DE SES INSTANCES



From SNOMED CT: hierarchy of concepts organized by specialization (subclass)

(P affectedBy Legionella)
(Legionella subClassOf BacterialPneumonia)

versus

(P affectedBy _:b) (_:b type Legionella)
(Legionella subClassOf BacterialPneumonia)

Première traduction logique de RDFS

On suit une approche classique de représentation des connaissances :

séparation stricte entre classes / propriétés / instances de concept

classe ⇝ prédicat unaire (concept)
propriété utilisateur ⇝ prédicat binaire (relation)
instance (IRI, blank, littéral) ⇝ constante (instance)

On pourrait traduire les blanks par des variables existentielles mais on a supposé qu'on n'en avait pas dans nos bases de faits

(Dupont, mayorOf, FieldsTown)
(FieldsTown, type, City)
(City subClassOf Organization)
(mayorOf, subPropertyOf, worksFor)
(mayorOf, range, City)
(worksFor, range, Organization)

Prédicats :
City/1
Organization/1
mayorOf/2
worksFor/2
Constantes : Dupont, FieldsTown

Première traduction logique de RDFS

On suit une approche classique de représentation des connaissances :

séparation stricte entre classes / propriétés / instances de concept

classe ⇝ prédicat unaire (concept)
propriété utilisateur ⇝ prédicat binaire (relation)
instance (IRI, blank, littéral) ⇝ constante (instance)

On pourrait traduire les blanks par des variables existentielles mais on a supposé qu'on n'en avait pas dans nos bases de faits

(Dupont, mayorOf, FieldsTown)
(FieldsTown, type, City)
(City subClassOf Organization)
(mayorOf, subPropertyOf, worksFor)
(mayorOf, range, City)
(worksFor, range, Organization)

Prédicats :
City/1
Organization/1
mayorOf/2
worksFor/2
Constantes : Dupont, FieldsTown

Remarque : on ne peut pas traduire le fait qu'une même appellation apparaisse à la fois comme classe/propriété/instance.

ex : (Dupont, mayorOf, City), (Dupont, type, mayorOf)

PREMIÈRE TRADUCTION LOGIQUE DE RDFS

Triplet factuel	⇝	**fait**
(s type C)	⇝	C(s)
(s p o) p utilisateur	⇝	p(s,o)
Triplet ontologique	⇝	**règle Datalog**
(A subClassOf B)		$\forall x\ (A(x) \rightarrow B(x))$
(r subPropertyOf s)		$\forall x \forall y\ (r(x,y) \rightarrow s(x,y))$
(r domain A)		$\forall x \forall y\ (r(x,y) \rightarrow A(x)$
(r range B)		$\forall x \forall y\ (r(x,y) \rightarrow B(y))$

Les triplets factuels fournissent une base de faits que l'on sature avec la base de règles provenant des triplets ontologiques => on ne peut donc pas requêter « l'ontologie »

EXEMPLE

(Dupont, mayorOf, FieldsTown)
(FieldsTown, type, City)
(City subClassOf Organization)
(mayorOf, subPropertyOf, worksFor)
(mayorOf, range, City)
(worksFor, range, Organization)

Prédicats :

City/1	concept
Organization/1	concept
mayorOf/2	propriété
worksFor/2	propriété

F = { mayorOf(Dupont, Fieldstown), City(FieldsTown) }
$\mathcal{R}$ = {
City(x) → Organization(x)
mayorOf(x,y) → worksFor(x,y)
mayorOf(x,y) → City(y)
worksFor(x,y) → Organization(y)
}

F* = { mayorOf(Dupont, Fieldstown), City(FieldsTown),
Organization(FieldsTown),
worksFor(Dupont, Fieldstown)
}

Deuxième traduction logique de RDFS

On suit l'approche RDF : on n'a que des triplets, la sémantique des triplets avec une propriété rdfs est donnée par les règles d'inférence RDFS

IRI, literal, blank $\rightsquigarrow$ constante
1 seul prédicat (ternaire) : triple

(là aussi, on pourrait vouloir traduire un blank par une variable)

Traduction des triplets : (s, p, o) $\rightsquigarrow$ triple(s,p,o)

Le raisonnement est basé sur les règles d'inférence RDF(S)

Ex: (s, type, C1), (C1, subClassOf, C2) → (s, type, C2)

$\forall s \forall C1 \forall C2$ (triple(s,type,C1) $\wedge$ triple(C1,subclass,C2) → triple(s,type,C2))

Là aussi, on obtient une base de faits et un ensemble de règles Datalog

EXEMPLE

(Dupont, mayorOf, FieldsTown)
(FieldsTown, type, City)
(City subClassOf Organization)
(mayorOf, subPropertyOf, worksFor)
(mayorOf, range, City)
(worksFor, range, Organization)

Prédicats : {triple/3}

F = { triple (Dupont, mayorOf, FieldsTown), ..., triple(worksFor, range, Organization) }

$\mathcal{R}$ = ensemble des règles d'inférence RDFS

F* = F U { triple(Dupont, worksFor, FieldsTown), triple(FieldsTown,type,Organization)
triple(mayorOf,range,Organization) }

Le dernier triplet est obtenu en appliquant la règle RDFS
(p, range, o), (o, subclass, o_1) → (p, range, o_1)

COMPARAISON SUR UN EXEMPLE

C1 sous-classe de C2 ... sous-classe de Cn

A a pour type C1

COMPARAISON SUR UN EXEMPLE

C1 sous-classe de C2 ... sous-classe de Cn

A a pour type C1

Traduction 1

F = { C1(A) }

$\mathcal{R}$ = { C1(x) → C2(x),

...

Cn-1(x) → Cn(x)

COMPARAISON SUR UN EXEMPLE

C1 sous-classe de C2 ... sous-classe de Cn

A a pour type C1

Traduction 1

F = { C1(A) }

$\mathcal{R}$ = { C1(x) → C2(x),

...

Cn-1(x) → Cn(x)

|F] = 1

|$\mathcal{R}$| = n-1

|F*| = n

COMPARAISON SUR UN EXEMPLE

C1 sous-classe de C2 ... sous-classe de Cn

A a pour type C1

Traduction 1

F = { C1(A) }

$\mathcal{R}$ = { C1(x) → C2(x),

...

Cn-1(x) → Cn(x)

|F] = 1

|$\mathcal{R}$| = n-1

|F*| = n

Traduction 2

F = { triple(A, type, C1), triple(C1, subClassOf, C2), ... (Cn-1 subClassOf Cn)}

R = { triple(s, subClass, o), triple(s1, type, s) → triple(s1, type, o)

triple(s, subClass, o), triple(o, subClass, o1) → triple(s, subClass, o1)

}

en ne prenant que les 2 règles RDFS utiles

COMPARAISON SUR UN EXEMPLE

C1 sous-classe de C2 ... sous-classe de Cn

A a pour type C1

Traduction 1

F = { C1(A) }

$\mathcal{R}$ = { C1(x) → C2(x),

...

Cn-1(x) → Cn(x)

|F] = 1

|$\mathcal{R}$| = n-1

|F*| = n

Traduction 2

F = { triple(A, type, C1), triple(C1, subClassOf, C2), ... (Cn-1 subClassOf Cn)}

R = { triple(s, subClass, o), triple(s1, type, s) → triple(s1, type, o)

triple(s, subClass, o), triple(o, subClass, o1) → triple(s, subClass, o1)

}

en ne prenant que les 2 règles RDFS utiles

|F| = n

|$\mathcal{R}$| = 2

|F*| = n + n x (n-1)/2 même nombre d'atomes « factuels »

explosion (qui reste polynomiale) des atomes « ontologiques »

INTÉRÊT DE LA TRADUCTION 2 POUR LE REQUÊTAGE

q1(rel) ← (Dupont, rel, FieldsTown)

q2(pers, orgClass) ← (pers, worksFor, org), (org, type, orgClass),
(orgClass, subClassOf, Organization)

INTÉRÊT DE LA TRADUCTION 2 POUR LE REQUÊTAGE

q1(rel) ← (Dupont, rel, FieldsTown)

q2(pers, orgClass) ← (pers, worksFor, org), (org, type, orgClass),
(orgClass, subClassOf, Organization)

Traduction 1 : on ne peut pas traduire vraiment ces requêtes

q11() ← mayorOf(Dupont,FieldsTown) // on peut juste demander si Dupont a
// une certaine relation avec FiedsTown

q12() ← worksFor(Dupont,FieldsTown)

q2(pers) ← worksFor(pers,org) // Organization(org) sera inféré par une règle

INTÉRÊT DE LA TRADUCTION 2 POUR LE REQUÊTAGE

q1(rel) ← (Dupont, rel, FieldsTown)

q2(pers, orgClass) ← (pers, worksFor, org), (org, type, orgClass),
(orgClass, subClassOf, Organization)

Traduction 1 : on ne peut pas traduire vraiment ces requêtes
q11() ← mayorOf(Dupont,FieldsTown) // on peut juste demander si Dupont a
// une certaine relation avec FiedsTown
q12() ← worksFor(Dupont,FieldsTown)

q2(pers) ← worksFor(pers,org) // Organization(org) sera inféré par une règle

Traduction 2 :
q2(rel) ← triple(Dupont,rel,FieldsTown)

q2(pers, orgClass) ← triple(pers, worksFor, org),
triple(org, type, orgClass),
triple(orgClass, subClassOf, ex:Organization)

CONCLUSION

Les deux traductions produisent une base de faits et une base de règles Datalog

Traduction 1 :

- (+) On suit une approche « représentation de connaissances » qui distingue clairement la partie ontologique de la partie factuelle
- (-) On ne peut donc pas traduire n'importe quel ensemble de triplets en gardant sa sémantique
- (-) On ne peut interroger que les faits provenant des triplets factuels et de leur saturation

Traduction 2 :

- Tous les triplets deviennent des faits
- La base de règles est indépendante d'une base RDFS particulière
- (-) Il y a un grand nombre de triplets ontologiques inférés
- (+) On peut faire du « méta-raisonnement » : raisonner sur les classes et les propriétés
- (+) On peut « tout » interroger, y compris avec des requêtes portant à la fois sur les « vrais » faits et sur l'ontologie