OWL2-ER Compatible Prescriptive Constraints to Evaluate Ontology Completeness

Contraintes prescriptives compatibles avec OWL2-ER pour évaluer la complétude d'ontologies

Philippe MARTIN (philippe.martin@univ-reunion.fr) [1] and
Jun JO (j.jo@griffith.edu.au) [2]

[1] MCF HDR, EA2525 LIM, Université de La Réunion, France
(+ chercheur associé de l'Université de Griffith, Australie)

[2] Université de Griffith, Australie

Plan

1. Contraintes considérées : règles pour la vérification *seulement*, pas la génération d'objets ;
   positives / négatives, descriptives / prescriptives

2. Représentation considérée de ces contraintes : règles utilisant
   “rdfs:subclassOf” pour exprimer l'implication (–> "compatibilité avec OWL2-ER")
   (cette représentation est possible avec quasiment n'importe quel langage)

3. Vérification de ces contraintes via *quelques* requêtes génériques en SPARQL

   3.1. Clonages d'objet pour permettre de vérifier les contraintes s-prescriptives

   3.2. Vérification des contraintes s-prescriptives positives

   3.3. Vérification des contraintes négatives

   3.4. Évaluation d'une forme de complétude de la base de connaissances (BC)

4. Évaluation, comparaisons et conclusion

1. Contraintes considérées : ..., positives / négatives, ...

Règle d'inférence : "si A est vrai, B l'est aussi".

Inf1 : "si x est un homme, x a un parent"
"Tom est un homme" => "Tom a un parent"

Contrainte + : "si A est vrai, B doit aussi l'être".

Ctr1 : "si x est un homme, x doit avoir un parent"
"Tom est un homme" (=> "Tom doit avoir un parent")

Contrainte – : "si A est vrai, B ne doit pas l'être"
[Chein & Mugnier, 2008]

Implémentation dans un système à base de règles unidirectionnel : $X \Rightarrow \bot$ E.g. : $A \wedge B \Rightarrow \bot$

Contraintes (Cs ; dans une BC) : phrases permettant d'en vérifier d'autres, pas d'en générer.

$$\forall c \quad c \in Cs \quad \Rightarrow \quad \forall p \ (\ (BC \Rightarrow p) \Rightarrow ((BC - c) \Rightarrow p) \ )$$

1. Contraintes considérées : ..., descriptives / prescriptives

Contrainte **descriptive :** contrainte sur le monde (représenté) –>
vérifie si certains termes utilisés le sont correctement (en tenant compte des inférences).

E.g. : *contrainte exprimant une signature de relations*
(note : rdfs:domain et rdfs:range ne sont pas des contraintes mais des règles d'inférences)

Contrainte **prescriptive :** contrainte sur la présence ou l'absence de certains termes/faits dans la BC –>
vérifie si certains termes sont utilisés ou non dans la BC dans certaines conditions
–> permet de vérifier un modèle prescriptif (schémas XML / BDD, modèle SHACL, ...)

Contrainte **s-prescriptive** : pour vérifier que certains termes/faits ont été entrés ("assertés") dans la BC
(plutôt qu'inférés) –> pour vérifier que des (méta-)modèles (e.g., KADS) ou schémas de conception
d'ontologies sont bien suivis dans une BC
sans utiliser des procédures ou 1 requête différente pour chaque contrainte

E.g. : * *Inf1 : "si x est un homme, x a un parent"*
* *Ctr1_s-prescriptive : "si x est un homme, x doit_s-prescriptif avoir a un parent"*
* *"Tom est un homme"*
–> erreur si aucun parent n'a été associé à Tom ; pas d'erreur avec (au lieu de Ctr1_s-prescriptive)
* *Ctr1_descriptive : "si x est un homme, x doit_descriptif avoir a un parent"*
* *Ctr1_ns-prescriptive : "si x est un homme, x doit_prescriptif_non-s-prescriptif avoir a un parent"*

–> Pour une contrainte *“si A est vrai, B **doit_s-descriptif** l'être”*
le régime d'inférence utilisé pour tester (une spécialisation de) B
ne doit pas permettre “l'héritage”, i.e. l'exploitation des définitions dans les types de l'objet testé, et donc
- ne doit exploiter que les relations sous-type (sous-classe, sous-propriété) ayant pour source un type non défini, ou *de manière équivalente*
- ne doit pas exploiter les définitions associées aux sources des relations sous-type.

- Avec OWL : pas d'exploitation des restrictions, ...
  –> régime d'inférence limité à “RDFS–” : RDFS moins rdfs:domain et rdfs:range.

- Si “si A est vrai, B ***doit_s-prescriptif l'être***” et “a => A”
  alors la BC doit inclure b tel que “b =>[RDFS–] B”

Nécessité de rendre explicite la sémantique spéciale de chaque type de contraintes
via une syntaxe spéciale ou *des types spéciaux (–> l'ontologie CSTR)*
sauf pour les contraintes de la forme syntaxique “X => ⊥” (–> pas de distinction entre descriptif et prescriptif)
lorsque cette forme suffit à traiter la phrase comme une contrainte.

2. Représentation considérée de ces contraintes : règles utilisant “rdfs:subclassOf” pour exprimer l'implication (-> "compatibilité avec OWL2-ER")

OWL2-RL : fragment de OWL2 pouvant se représenter par des règles de Horn avec égalité
et donc pouvant se traduire en Datalog (~ la partie purement déclarative de Prolog).

Datalog+ : extension de Datalog avec des règles existentielles (ER),
i.e. des règles dont la conclusion peut contenir des quantificateurs existentiels.

OWL2-ER [Baget *et al.*, 2015] : fragment de OWL2 pouvant se traduire en Datalog+ (pas en Datalog)
(~ Datalog+ restreint aux relations binaires et basé sur rdfs:subClassOf ;
pas de contraintes positives).

Compatible avec OWL2-ER : juste restreint à l'utilisation de rdfs:subClassOf et donc
impossibilité d'utiliser des variables pour relier des éléments de la condition et de la conclusion d'une règle.

Ctr1_s-prescriptive ("si x est un homme, x doit_s-prescriptif avoir a un parent") représentée
- de manière compatible avec OWL2-ER,
- en utilisant Turtle,
- en utilisant 1 type de l'ontologie CSTR :

```
:Man  #class and constraint condition
   rdfs:subClassOf cstr:SubclassOf-based_prescriptive_constraint_condition ;
   rdfs:subClassOf   #conclusion (types of relations that must be present):
      [ rdf:type owl:Restriction;
        owl:onProperty  :parent ;   owl:someValuesFrom  :Parent ]
```

3. Vérification de contraintes via quelques requêtes génériques en SPARQL

3.1. Via SPARQL ; clonages d'objet pour vérifier les contraintes s-prescriptives

But : permettre, lors de la comparaison/appariement d'un objet avec *la conclusion* d'une contrainte,
de désactiver “l'héritage de relations”,
i.e. la recherche de relations dans les types de cet objet et les superclasses de ces types.
Pour l'appariement de la condition d'une contrainte, plus d'inférences peuvent être utiles.

- **Statiquement :** 1) copie dans la base de faits de chaque objet (et de ses relations / propriétés) par son “clone sans type”, i.e. le même objet sans la relation rdf:type.

```
INSERT { ?o cstr:cloneWithoutType ?o2 . ?o2 ?r ?dest } WHERE
 { ?o ?r ?dest .   FILTER (?r != rdf:type)
   FILTER NOT EXISTS { ?o rdf:type rdfs:Class }
   BIND (uri(concat(str(?o), "_cloneWithoutType")) as ?o2)
 }
```

2) lors de la comparaison d'un objet avec la conclusion d'une règle,
utilisation du “clone sans type” de objet au lieu de l'objet.

- **Dynamiquement**, lors du test de la conclusion d'une règle sur un objet :
création temporaire (par la requête testant une règle) du “clone sans type” de l'objet.

Impossible avec du SPARQL1.1 pur car il ne permet pas d'inclure un `CONSTRUCT` dans un `SELECT`
mais possible avec une extension de SPARQL telle LDScript [Corby, Faron-Zucker & Gandon, 2017].

3.2. Requête SPARQL pour vérifier les contraintes s-prescriptives positives

Avec SPARQL1.1, *pour la recherche d'objets* (pour des relations, remplacer **`rdf:type`** par **`log:implies`**) :

```
SELECT ?objectNotMatchingPosConstr ?posConstr  WHERE
{ ?posConstr rdfs:subClassOf cstr:Subclassof-based_prescriptive_constraint_condition,
                              ?posConstr_conclusion .  #initializes this variable
  FILTER NOT EXISTS { ?posConstr rdfs:subClassOf owl:Nothing }
  ?objectNotMatchingPosConstr rdf:type ?posConstr. #matches condition
  FILTER NOT EXISTS  #objects satisfying the conclusion must not be listed
  { BIND(uri(concat(str(?objectNotMatchingPosConstr),"_cloneWithoutType"))
        as ?cloneWithoutType) ?cloneWithoutType rdf:type ?posConstr_conclusion }
}
```

Rappel – *Ctr1_s-prescriptive ("si x est un homme, x doit_s-prescriptif avoir a un parent")*
représentée de manière compatible avec OWL2-ER et en utilisant Turtle et 1 type de l'ontologie CSTR :

```
:Man  #class and constraint condition
   rdfs:subClassOf cstr:Subclassof-based_prescriptive_constraint_condition ;
   rdfs:subClassOf   #conclusion (types of relations that must be present):
      [ rdf:type owl:Restriction;
        owl:onProperty  :parent ;   owl:someValuesFrom  :Parent ]
```

Avec LDScript, *pour la recherche d'objets* :

```
SELECT ?objectNotMatchingPosConstr ?posConstr WHERE
{ ?posConstr rdfs:subClassOf cstr:SubclassOf-based_prescriptive_constraint_condition;
             rdfs:subClassOf ?posConstr_conclusion .
  FILTER NOT EXISTS { ?posConstr rdfs:subClassOf owl:Nothing }
  ?objectNotMatchingPosConstr rdf:type ?posConstr. #matches condition
  BIND( uri(concat(str(?object),"_cloneWithoutType")) as ?cloneWithoutType )
  BIND( us:copyKbButWithCloneWithoutType(?objectNotMatchingPosConstr,
                                         ?cloneWithoutType) as ?g )
  FILTER NOT EXISTS { GRAPH ?g { ?cloneWithoutType rdf:type ?posConstr_conclusion } }
}


function us:copyKbButWithCloneWithoutType
         (?objectNotMatchingPosConstr, ?cloneWithoutType)
{ let (?g = CONSTRUCT { ?cloneWithoutType ?r ?dest .  ?x ?r2 ?y } WHERE
            { values ?cloneWithoutType { UNDEF }
              ?objectNotMatchingPosConstr ?r ?dest .  FILTER (?r != rdf:type)
              ?x ?r2 ?y .  FILTER (?x != ?objectNotMatchingPosConstr)
            })
  { xt:entailment(?g) }  #triggers inferences on ?g
}
```

3.3. Requête SPARQL pour vérifier les contraintes négatives

```
SELECT ?objectMatchingNegConstr ?negConstr WHERE
{ ?negConstr rdfs:subClassOf cstr:SubclassOf-based_constraint_condition,
                             owl:Nothing .
  ?objectMatchingNegConstr rdf:type ?negConstr .
}
```

3.4. Requête SPARQL pour évaluer une forme de complétude de la BC

```
SELECT ( ((?nbObjs - ?nbAgainstPosCs - ?nbMatchingNegCs) / ?nbObjs)
         AS ?completeness)
{ {SELECT (COUNT(DISTINCT ?o) AS ?nbObjs)
    WHERE { ?o ?r ?o2 } } #any object related to another
  {SELECT(COUNT(DISTINCT ?objectNotMatchingPosConstr) AS ?nbAgainstPosCs)
    WHERE { ... #body of the query for positive prescriptive constraints
          } }
  {SELECT (COUNT(DISTINCT ?objectMatchingNegConstr) AS ?nbMatchingNegCs)
    WHERE { ... #body of the query for negative constraints
          } }
}
```

4. Évaluation, comparaisons et conclusion

Originalité : représentation de contraintes prescriptives avec tout LRC dont l'expressivité est au moins égale à RDFS
–> possibilité d'exploiter n'importe quel moteur d'inférences
y compris via *quelques requêtes génériques*, comme ici montré avec SPARQL.

Évaluation théorique : cf. études théoriques des LRCs, langages de requêtes et moteurs d'inférences utilisés.

Évaluation pratique : tests de quelques schémas de conception d'ontologies (cf. article Web associé).

Comparaisons :

- **SHACL** (SHApes Constraint Language) : similarités avec OWL2 sans le réutiliser ;
  contraintes prescriptives mais pas s-prescriptives ; réutilisation possible de SPARQL `SELECT`
  mais pas de `INSERT` ou de `CONSTRUCT` ni donc de requêtes génériques ici données.
- **SPIN** (SParql Inferencing Notation) : stockage en RDF et attachement procédural de
  fonctions Javascript ou de requêtes SPARQL à des objets RDF.
- **Langages/systèmes de "transformation de RCs"** – tels PatOMat [Zamazal & Svátek, 2015]
  et STTL [Corby & Faron-Zucker, 2015] – et donc exploitant une fonction d'appariement de motifs de RCs **:**
  adaptables pour gérer les requêtes génériques ici illustrées et donc des contraintes prescriptives).