Lab RDFS entailment

Veuillez charger nos fichiers human-rdf.ttl et human-rdfs.ttl dans corese car nous y avons effectué des modifications…

Premierement, nous avons essayé de reperer à l'œil ligne par ligne les anomalies qui peuvent être réctifiés par des construct directement dans le fichier. C'est notament comme cela que nous avons su que Sophie n'était pas une ressource qui existe et donc nous l'avons crée. La deuxieme méthode était de lancer des requêtes SPARQL basiques pour identifier les differences entre avec RDFS et sans RDFS. Enfin nous avons véiller à ce que les 13 règles de la recommandation soient utilisées. Pour cela nous avons réécrit en language naturel les 13 régles avant de les implementer ce qui nous a grandement facilité le travail (cf : construct.md).

Dans notre fichier select.md nous avons mis en évidence les query qui pour nous représentent au mieux les régles de la recommandation w3c en précisant bien les résultats obtenus avec rdfs activé et non activé.

Passons au crible le fichier select.md pour expliquer les différences qu'il y a entre rdfs activé et rdfs non activé :

(si certaines regles ne sont pas mentionnées ici c'est que leurs select respectifs ne donnent lieu à des différences de résultats)

RDFS2

On teste si pour la propriété shirsize ayant pour domaine Personne les individus sont bien ce type. Si le rdfs est desactivé on que les individus de type Personne. Sinon on trouve aussi d'autres types qui sont des sous classes de personnes.

RDFS3

On a un test similaire mais cette fois avec range au lieu de domain. Si le rdfs est desactivé nous n'avons que les individus de type Woman. Sinon on en trouve aussi de type Person (car Woman est une sous classe de Person)

RDFS4

On test pour un seul sujet en l'occurrence John. Si le rdfs est désactive rdfs :Resource n'apparaît pas. Sinon il apparaît. Nous aurions put teste tout les sujets et tout les objets avec select * where {

?x ?y ?z

?x a rdfs :Resource

?z a rdfs :Resource

}

Mais les résultats sont trop longs si rdfs est activé et au contraire vide si rdfs est désactivé

RDFS5

On teste avec un exemple concret de transivité de propriété : hasAncestor avec hasMother ou hasFather en passant par hasParent. Si rdfs est desactivé on ne trouve pas ces deux derniers car ils sont sous propriété de hasParent et non de hasAncestor. Sinon on trouve les 4 propriétes citées.

RDFS6

On test avec hasFather abitrairement et on remarque que hasFather réapparait dans les résultats que ce soit avec rdfs activé ou non(étonnament). Une autre façon de faire aurait été :

Select * where {?x rdfs:subPropertyOf ?x}

RDFS7

On teste les propriétés qui relient Mark et John et on remarque qu'avec rdfs desactivé nous on avons une seule : hasFather. Alors qu'avec rdfs activé nous avons cette même propriété mais aussi les propriétés dont elle est la sous propriété.

RDFS8

On test si Man est une sous classe de rdfs :Resource en listant les classes pour lesquels Man est une sous classe. On remarque que rdfs :Resource apparaît seulement quand rdfs est activé.

On aurait put aussi tester avec :

Select * where {?x rdfs:subClassOf rdfs:Resource}

et voir si elle donne des résultats (rdfs desactivé ne donne pas de résultat)

RDFS9

On teste sur les classes de Mark. On remarque qu'avec rdfs desactivé on n'a que sa classe mais qu'avec rdfs activé on a la sous classe (=classe de Mark) et la classe dont elle est la sous classe.

RDFS10

C'est la même régle que la RDFS 6 mais pour les classes au lieu des propriétés. Dans la liste des sous classes on retrouve bien Man quand rdfs est activé contrairement à rdfs desactivé. Nous auriosn put aussi faire :

Select * where { ?x rdfs:subClassOf ?x }

Mais elle aurait donné un résultat nul pour rdfs desactivé

RDFS11

C'est la même régle que rdfs 5 mais pour les classes au lieu des propriétés. On teste pour Man quelles sont les classes dont elle est la sous classe. Pour Rdfs desactivé nous avons seulement la transitivité directe. Alors que pour rdfs desactivé nous avons aussi la classe dont les sous classes sont les classes dont Man est une sous classe.

Passons maintenant aux construct que nous avons crée spécialement pour les données fournies dans humans.rdf et humans.rdfs (cf : construct.txt) avec en parrallèle les query select qui nous ont inspiré (cf : select.txt). Ce travail bien que non nesseçaire nous a permit de mettre en évidence les différences entre lorqu'on active rdfs et lorsqu'on ne l'active pas. Cela correspond à écrire avec un construct les triplets qui différents entre avec et sans rdfs activé :

Le reste de ce travail n'a pas été fait après qu'on ai su qu'il n'était pas nesseçaire (suite à un échange avec Mme Faron)

Concernant la query 16 : Laura n'est pas de type femme(rdfs activé ou desactivé ne change rien). On rectifie avec la construct 13.

Concerant les queries de 17 à 19 Laura est écrite plusieurs fois dans le fichier avec à chaque fois un des types donc activer rdfs ou pas donne le même résultat.

Cela se remarque bien avec les queries de 21 à 24 avec Gaston qui lui si on lui donne un type autre que Researcher on ne trouve aucun résultat (si rdfs desactivé) à part pour Man. On rectifie avec les construct 14 et 15 (comme régles 11 et 9).

C'est aussi confirmé par les queries de 25 à 28 avec David qui sort pour les types researcher et Person. On rectifie avec les construct de 16 et 17 (comme régles 11 et 9).

Enfin nous avons ajoutés des constructs afin de rajouter des informations au graph

Construct n 18 : Toute personne qui a un parent a pour autre parent l'époux du premier.

Si x hasMother ou hasFather y et z hasSpouse x alors x hasFather y.

Construct n 19 : Tout ami ou epoux d'un ami est aussi mon ami.

Si x hasfriend y et y hasfriend ou hasSpouse z alors x hasfriend z (à peu près comme régle 7).

Construct n 20 : Creer type adulte si >18.

Construct n 21 : Pour être un epoux(se) il faut être un ami

(Disclaimer : ceci sont des exemples et ne réprensentent pas forcément la réalité)

Fait par : Gaymard Erwan

Mekouar Ayoub