Répondre aux requêtes par reformulation dans les bases de données RDF

François Goasdoué, Ioana Manolescu, Alexandra Roatis

Répondre aux requêtes par reformulation dans les bases de données RDF

François Goasdoué
fg@lri.fr

Ioana Manolescu
ioana.manolescu@inria.fr

Alexandra Roatiş
alexandra.roatis@inria.fr

Université Paris-Sud and INRIA Saclay–Île-de-France, bât. 650, Univ. Paris-Sud, 91405 Orsay, France

Abstract

RDF query answering relies either on data saturation or on query reformulation. The idea of both techniques is to decouple RDF entailment – the reasoning mechanism based on which queries' answers are defined – from query evaluation. In this paper, we extend the state of the art by devising a reformulation-based query answering technique for a more significant fragment of RDF and a more expressive query language than those investigated in the literature. We also experimentally compare our technique with standard saturation-based query answering.

Keywords

RDF data management, reasoning, semantic web

Résumé

Dans RDF, répondre aux requêtes repose soit sur la saturation des données, soit sur la reformulation des requêtes. L'idée des deux techniques est de découpler la notion d'entailment – le mécanisme de raisonnement à partir duquel les réponses aux requêtes sont définies – de l'évaluation de requêtes. Dans cet article, nous étendons l'état de l'art en proposant une technique de réponse aux requêtes par reformulation pour un fragment de RDF plus significatif et un langage de requêtes plus expressif que ceux étudiés dans la littérature. Nous comparons ensuite expérimentalement cette nouvelle technique avec la technique standard fondée sur la saturation de données.

Mots-clés

Gestion de données RDF, raisonnement, web sémantique

1 Introduction

Le *Resource Description Framework* (RDF) [9] est un modèle de données de type « graphe » reconnu comme le standard du W3C pour les applications du Web Sémantique. En tant que tel, il est doté d'un langage d'ontologie, *RDF Schema* (RDFS), utilisé pour enrichir les descriptions des *graphes RDF*. Une particularité de RDF est la notion *d'entailment* qui permet de dériver des informations implicites au sein d'un graphe RDF.

Le standard du W3C pour l'interrogation de graphes RDF est *SPARQL Protocol and Query Language* (SPARQL) [10]. La technique usuelle pour répondre aux requêtes sur un graphe RDF repose sur la *saturation du graphe* (ou clôture du graphe), par ex. [1, 2]. Cette technique découle directement de la définition normative des réponses à une requête. Le principe est de pré-calculer la saturation du graphe RDF, dans laquelle les informations implicites sont explicitées. Ainsi, au moment de répondre aux requêtes, les réponses sont obtenues simplement et efficacement en évaluant les requêtes sur le graphe saturé. La saturation est la force de cette technique, mais aussi sa faiblesse car il faut du temps pour la calculer, de l'espace pour la stocker et elle doit être recalculée lors de mises-à-jour.

Dans cet article, nous étudions une technique alternative pour répondre aux requêtes, fondée sur la *reformulation des requêtes*. Cette technique a été transférée des Logiques de Description (LD) [4] vers le fragment LD de RDF [3, 5], càd. modélisant des bases de connaissances en LD. Ici, les réponses sont obtenues en évaluant les *reformulations* des requêtes sur le graphe RDF (original), les reformulations tenant comptent des informations implicites du graphe. L'avantage de cette technique est que les reformulations sont calculées – en fonction du graphe courant – au moment de répondre aux requêtes. Il n'y a donc pas de saturation à (re-)calculer ou à stocker. Toutefois, l'inconvénient est le temps supplémentaire requis pour reformuler les requêtes et évaluer les reformulations, celles-ci résultant généralement en des requêtes plus complexes à évaluer.

Notre contribution est d'étendre l'état de l'art [3, 5], afin que répondre aux requêtes par reformulation s'applique à un fragment de RDF plus significatif que celui des LD et à un langage de requêtes plus expressif que les requêtes conjonctives relationnelles considérées en LD. Notamment, nous étendons cette technique au fragment *Bases de données* (BD) de RDF ainsi qu'aux requêtes *Basic Graph Pattern* (BGP) de SPARQL, ce qui permet de capturer maintenant des propriétés essentielles de RDF comme la *modélisation d'information incomplètes*, la *non-distinction entre constantes et noms de relation*, et l'*utilisation de relations inconnues dans les requêtes* (les relations peuvent être des variables). Enfin, nous fournissons une comparaison expérimentale entre notre technique et la technique standard fondée sur la saturation de graphe RDF.

L'article est organisé comme suit. Les sections 2 et 3 introduisent les graphes RDF et les requêtes BGP. La section 4 définit notre fragment BD de RDF, et la section 5 introduit les techniques de réponse aux requêtes fondées sur la saturation des graphes et la reformulation des requêtes. Celles-ci sont ensuite étudiées en détails, pour le fragment BD de RDF, dans les sections 6 et 7, et sont comparées expérimentalement dans la section 8. Enfin, nous discutons des travaux connexes dans la section 9, avant de conclure.

Constructeur	Triplet
Assertion de classe	s rdf:type o .
Assertion de propriété	s p o .

FIGURE 1 – Assertions RDF

2 Graphes RDF

Un *graphe RDF* est un ensemble de *triplets* de la forme s p o . (le point final précédé d'un espace est la syntaxe normalisée d'un triplet). Un triplet dit que son *sujet* s est décrit par la *propriété* p, et que la valeur de cette propriété est *l'object* o. Étant donnés un ensemble U d'URIs, un ensemble L de littéraux (constantes) et un ensemble B de nœuds blancs (URIs ou littéraux inconnus), tels que U, B et L sont deux-à-deux disjoints, un triplet est *bien formé* si son sujet appartient à $U \cup B$, sa propriété appartient à U et son objet appartient à $U \cup B \cup L$. Dans la suite, nous ne considérons que des triplets bien formés.

Les nœuds blancs sont une particularité de RDF permettant l'utilisation *d'URI/littéraux non spécifiés*. Par exemple, on peut utiliser un nœud blanc _:b_1 pour modéliser que le pays de _:b_1 est *France* et que la ville de ce même _:b_1 est *Lyon*. Plusieurs nœuds blancs peuvent coexister au sein d'un graphe RDF, par ex. on pourrait aussi modéliser que le pays de _:b_2 est *Romania* et que la ville de _:b_2 est *Timişoara* ; tout en ajoutant que la population de *Timişoara* est une valeur non spécifiée _:b_3.

Notations. Nous utilisons s, p, o et _:b dans les triplets (éventuellement avec des indices) comme marqueurs génériques : s représente des valeurs de $U \cup B$, p des valeurs de U, o des valeurs de $U \cup B \cup L$, et _:b des valeurs de B. Nous utilisons des chaînes de caractères entre guillemets, comme dans "$string$", pour représenter les littéraux.

Une propriété prédéfinie dans la spécification RDF [9] est rdf:type. Elle permet d'indiquer à quelles *classes* des ressources appartiennent. Ceci peut être vu comme une forme de typage des ressources. La figure 1 montre comment utiliser des triplets pour décrire des ressources. Nous utilisons les préfixes usuels rdf: et rdfs: pour les *espaces de noms* des classes et propriétés fournies par le standard RDF [9], celles-ci pouvant être vues comme un méta-modèle RDF.

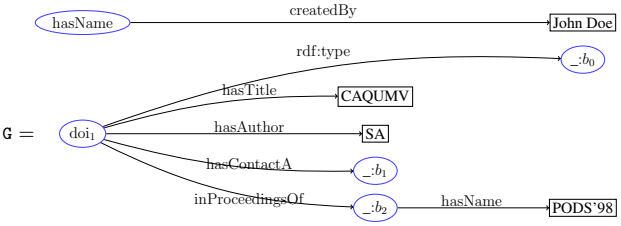
Un graphe RDF est souvent donné sous sa représentation sagittale plus intuitive. Chaque valeur (distincte) de sujet ou objet est représentée par un nœud étiqueté par ce sujet ou objet. Chaque triplet définit un arc étiqueté par sa propriété allant du nœud sujet vers le nœud objet.

Exemple 1. *Les représentations suivantes modélisent le même graphe* G. *Ce graphe décrit la ressource* doi_1 *appartenant à une classe non spécifiée, dont le titre (*hasTitle*) est "Complexity of Answering Queries Using Materialized Views", dont l'auteur (*hasAuthor*) est "Serge Abiteboul", et pour lequel il existe un auteur à contacter non spécifié (*hasContactA*). Cet article est dans les actes (*inProceedingsOf*) d'une ressource non spécifiée dont le nom (*hasName*) est "PODS'98". "John Doe" a créé la propriété associant des noms aux ressources (*hasName*).*

Constructeur	Triplet
Contrainte de sous-classe	s rdfs:subClassOf o .
Contrainte de sous-propriété	s rdfs:subPropertyOf o .
Contrainte de domaine d'une propriété	s rdfs:domain o .
Contrainte de range d'une propriété	s rdfs:range o .

FIGURE 2 – Assertions RDFS (*domaine* et *range* sont resp. les premier et second attributs de chaque propriété)

G= {doi_1 rdf:type _:b_0 ., doi_1 hasTitle *"CAQUMV"* .,
doi_1 hasAuthor *"SA"* ., doi_1 hasContactA _:b_1 .,
doi_1 inProceedingsOf _:b_2 ., _:b_2 hasName *"PODS'98"* .,
hasName createdBy *"John Doe"* .}



RDF Schema (RDFS) est une extension importante de RDF dont le but est d'améliorer les descriptions de ressources dans les graphes. Un schéma RDF permet de déclarer des *contraintes sémantiques* entre les classes et les propriétés utilisées dans ces graphes. La figure 2 montre les contraintes autorisées et comment les déclarer.

Le modèle de données RDF – et ce travail – est fondé sur l'hypothèse du monde ouvert [4], qui implique que certains faits peuvent être vrais, même s'ils ne sont pas explicitement présents dans un graphe RDF. C'est comme cela que sont interprétées les contraintes de la figure 2. Par exemple, si les triplets hasFriend rdfs:domain Person . et Anne hasFriend Marie . sont vrais, alors il en est de même pour le triplet Anne rdf:type Person . Ce dernier est du à la contrainte de domaine de la figure 2.

Exemple 2 (suite)**.** *Considérons en plus du graphe* G *ci-dessus, le schéma modélisant que les articles posters (*posterCP*) et ceux de la classe non spécifiée _:b_0, dont* doi_1 *est une instance, sont des sous-classes des articles de conférence (*confP*), qui sont eux-mêmes des articles scientifiques (*paper*). De plus, les titres (*hasTitle*), auteurs (*hasAuthor*), auteurs à contacter (*hasContactA*) qui sont des auteurs, sont utilisés pour décrire des articles, ainsi que les actes (*inProceedingsOf*) de conférences (*conference*) publiant des articles. Enfin, les noms (*hasName*) décrivent les conférences et les créateurs (*createdBy*) décrivent des ressources. Le graphe* G′ *correspondant à l'extension de* G *par ce schéma est :*

G′=G ∪ {posterCP rdfs:subClassOf confP .,
_:b_0 rdfs:subClassOf confP .,
confP rdfs:subClassOf paper .,
hasTitle rdfs:domain paper .,
hasTitle rdfs:range rdfs:Literal .,
hasAuthor rdfs:domain paper .,
hasAuthor rdfs:range rdfs:Literal .,
hasContactA rdfs:subPropertyOf hasAuthor .,
inProceedingsOf rdfs:domain confP .,
inProceedingsOf rdfs:range conference .,
hasName rdfs:domain conference .,
hasName rdfs:range rdfs:Literal .,
createdBy rdfs:range rdfs:Literal .}

Entailment. Une caractéristique importante de RDF est la modélisation de *triplets implicites* : ils sont considérés comme faisant partie d'un graphe, même s'ils ne sont

pas explicitement présents dans celui-ci. Le W3C nomme *RDF entailment* le mécanisme par lequel les triplets implicites sont dérivés (ou engendrés) à partir des triplets explicites d'un graphe et de *règles d'entailment*. Nous notons $\vdash^i_{\text{RDF}}$ un *entailment immédiat*, càd. la dérivation de triplets par l'application d'une règle d'entailment. Plus généralement, un triplet $\mathtt{s\ p\ o}$. est engendré par un graphe $\mathtt{G}$, noté $\mathtt{G} \vdash_{\text{RDF}} \mathtt{s\ p\ o}$. si et seulement si il existe une séquence d'entailments immédiats menant de $\mathtt{G}$ à $\mathtt{s\ p\ o}$. (où à chaque pas de la séquence, les triplets précédemment engendrés sont aussi pris en compte).

Saturation de graphe. Les règles d'entailment permettent de définir la *saturation* (finie) $\mathtt{G}^\infty$ d'un graphe $\mathtt{G}$, qui est un graphe, comme le point fixe de :

– $\mathtt{G}^0 = \mathtt{G}$

– $\mathtt{G}^\alpha = \mathtt{G}^{\alpha-1} \cup \{\mathtt{s\ p\ o}\ . \mid \mathtt{G}^{\alpha-1} \vdash^i_{\text{RDF}} \mathtt{s\ p\ o}\ .\}$

La saturation d'un graphe RDF est unique (au renommage des nœuds blancs près) et ne contient plus de triplets implicites (tous ont été explicités par saturation). Les triplets engendrés par un graphe $\mathtt{G}$ et la saturation de ce graphe ont un lien évident : $\mathtt{G} \vdash_{\text{RDF}} \mathtt{s\ p\ o}$. si et seulement si $\mathtt{s\ p\ o}\ . \in \mathtt{G}^\infty$.

Il est important de noter que le RDF entailment fait partie de la norme RDF. Par conséquent, en RDF, tout graphe $\mathtt{G}$ est (sémantiquement) équivalent à sa saturation $\mathtt{G}^\infty$.

3 Requêtes RDF

Nous considérons un sous-langage de SPARQL permettant de modéliser des (unions de) *requêtes conjonctive*. Une requête conjonctive est définie par un *Basic Graph Pattern* (BGP), c'est-à-dire un ensemble d'*atomes triplets* ou plus simplement de triplets. Chaque triplet a un sujet, une propriété et un objet. Sujets et propriétés peuvent être des URIs, nœuds blancs ou *variables* ; les objets peuvent aussi être des littéraux.

Une requête conjonctive booléenne est de la forme $\texttt{ASK WHERE}\ \{t_1, \ldots, t_\alpha\}$, alors qu'une requête conjonctive non booléenne est de la forme $\texttt{SELECT}\ \bar{x}\ \texttt{WHERE}\ \{t_1, \ldots, t_\alpha\}$, où $\{t_1, \ldots, t_\alpha\}$ est un BGP ; les variables $\bar{x}$ dans la *tête* de la requête sont nommées *variables distinguées* et forme un sous-ensemble des variables apparaissant dans $t_1, \ldots, t_\alpha$.

Notations. Nous utilisons – sans perte de généralité – la notation $q(\bar{x})$:- $t_1, \ldots, t_\alpha$ pour les requêtes ASK et SELECT (pour les requêtes booléennes, $\bar{x}$ est vide). Nous utilisons x, y et z (éventuellement avec des indices) pour modéliser les variables des requêtes. Nous notons $\texttt{VarBl}(q)$ l'ensemble de variables *et* nœuds blancs apparaissant dans la requête q. L'ensemble des valeurs (URI, nœuds blancs et littéraux) d'un graphe $\mathtt{G}$ est noté $\texttt{Val}(\mathtt{G})$.

Évaluation de requête. Étant donnés une requête q et un graphe $\mathtt{G}$, l'*évaluation de q sur $\mathtt{G}$* est : $q(\mathtt{G}) = \{\bar{x}_\mu \mid \mu : \texttt{VarBl}(q) \to \texttt{Val}(\mathtt{G})$ est une assignation totale tq $(t_1, \ldots, t_\alpha)_\mu \subseteq \mathtt{G}\}$.

Ci-dessus, pour tout triplet ou ensemble de triplets O, nous notons O_μ le résultat du remplacement de chaque occurrence d'une variable ou nœud blanc $e \in \texttt{VarBl}(q)$ dans O par la valeur $\mu(e) \in \texttt{Val}(\mathtt{G})$. Si q est booléenne, l'ensemble de réponses vide encode false alors que l'ensemble de réponses contenant le tuple vide $\langle\rangle_\mu$ encode true.

On remarquera que l'évaluation *traite les nœuds blancs d'une requête comme des variables non distinguées*. Ainsi, on peut considérer de façon équivalente des requêtes sans nœud blanc ou sans variable non distinguée.

Réponses à une requête. L'évaluation de q sur $\mathtt{G}$ utilise seulement les triplets explicites de $\mathtt{G}$, et par conséquent mène à un ensemble de réponses incomplet dans le cas général. L'*ensemble de réponses* (complet) à q sur $\mathtt{G}$ est obtenu par l'évaluation de q sur $\mathtt{G}^\infty$, notée $q(\mathtt{G}^\infty)$.

Exemple 3 (suite)**.** *La requête suivante demande les auteurs d'articles publiés dans les actes de conférences en lien avec "$PODS'98$" :*

$$q(x) \text{:-} \quad y_1\ \text{hasAuthor}\ x\ .,\ y_1\ \text{inProceedingsOf}\ y_2\ .,\ y_2\ y_3\ \text{"}PODS'98\text{"}\ .$$

Cette requête peut être réécrite de façon équivalente en :

$$q(x) \text{:-} \quad \_{:}b_0\ \text{hasAuthor}\ x\ .,\ \_{:}b_0\ \text{inProceedingsOf}\ \_{:}b_1\ .,\ \_{:}b_1\ \_{:}b_2\ \text{"}PODS'98\text{"}\ .$$

L'ensemble de réponses à q sur $\mathtt{G}'$ est : $q(\mathtt{G}'^\infty) = \{\langle\text{"}SA\text{"}\rangle, \langle\_{:}b_1\rangle\}$. La réponse "SA" résulte de l'assignation $\mu = \{y_1 \to \text{doi}_1, x \to \text{"}SA\text{"}, y_2 \to \_{:}b_2, y_3 \to \text{hasName}\}$, tandis que la réponse $\_{:}b_1$ résulte de $\mathtt{G}' \vdash_{\text{RDF}} \text{doi}_1\ \text{hasAuthor}\ \_{:}b_1$. et de l'assignation $\mu = \{y_1 \to \text{doi}_1, x \to \_{:}b_1, y_2 \to \_{:}b_2, y_3 \to \text{hasName}\}$.

On remarquera qu'évaluer q sur $\mathtt{G}'$ mène à un ensemble de réponses incomplet $q(\mathtt{G}') = \{\langle\text{"}SA\text{"}\rangle\} \subset q(\mathtt{G}'^\infty)$.

4 Fragment BD de RDF

Nous définissons le *fragment « Bases de données » (BD) de RDF* en *limitant* le RDF entailment (défini dans les spécifications [9]) *aux seules règles d'entailment dédiées aux schémas RDF*. Notamment, ceci est l'unique restriction imposée à RDF pour obtenir par le fragment BD. Au contraire, le fameux fragment LD – mentionné en introduction – interdit en plus tout ce qui fait la singularité de RDF : la modélisation d'informations incomplètes (grâce aux nœuds blancs) et la non distinction entre les constantes et les classes/propriétés. Notre but est d'avoir une séparation claire entre le *schéma* et les *données*, comme dans la majorité des modèles de données, tout en restant le plus fidèle possible à RDF. Une telle séparation est importante à la fois pour la compréhension du graphe et pour des raisons d'efficacité. Elle permet de montrer le schéma à l'utilisateur afin qu'il comprenne comment les données sont organisées, et donc comment les interroger. Elle permet aussi la transposition de techniques efficaces de gestions de données dans le monde RDF [8].

Nous appelons *base de données (BD) RDF* tout graphe RDF appartenant à notre fragment BD. Une BD $\mathtt{db}$ est un couple $\langle \mathtt{S}, \mathtt{D} \rangle$, où $\mathtt{S}$ et $\mathtt{D}$ sont deux ensembles disjoints de triplets. Les triplets de $\mathtt{S}$ sont uniquement des assertions RDFS, cf. figure 2. Nous appelons ces triplets le *schéma* de $\mathtt{db}$. Les autres triplets sont des assertions RDF appartenant à $\mathtt{D}$, cf. figure 1. Nous les appelons *l'instance* de $\mathtt{db}$.

Triplets	Triplet engendré ($\vdash^i_{\text{RDF}}$)
s rdfs:subClassOf o ., o rdfs:subClassOf o_1 .	s rdfs:subClassOf o_1 .
s rdfs:subPropertyOf o ., o rdfs:subPropertyOf o_1 .	s rdfs:subPropertyOf o_1 .
s rdfs:domain o ., o rdfs:subClassOf o_1 .	s rdfs:domain o_1 .
s rdfs:range o ., o rdfs:subClassOf o_1 .	s rdfs:range o_1 .
s rdfs:domain o ., s_1 rdfs:subPropertyOf s .	s_1 rdfs:domain o .
s rdfs:range o ., s_1 rdfs:subPropertyOf s .	s_1 rdfs:range o .

FIGURE 3 – Entailment schéma vers schéma

Triplets	Triplet engendré ($\vdash^i_{\text{RDF}}$)
s_1 rdfs:subClassOf s_2 ., s rdf:type s_1 .	s rdf:type s_2 .
p_1 rdfs:subPropertyOf p_2 ., s p_1 o .	s p_2 o .
p rdfs:domain s ., s_1 p o_1 .	s_1 rdf:type s .
p rdfs:range s ., s_1 p o_1 .	o_1 rdf:type s .

FIGURE 4 – Entailment schéma et instance vers instance

Observons que S et D forment une partition de tout graphe RDF (tout triplet appartient seulement à l'un d'eux). Une façon équivalente de dire les choses est que notre fragment BD n'impose aucune restriction sur les graphes RDF.

Le fragment BD est délimité du modèle RDF en restreignant les règles d'entailment à celles listées dans les figures 3 et 4, ainsi que celles engendrant des assertions RDFS triviales, par ex. s rdfs:subClassOf o . engendre s rdfs:subClassOf s . et o rdfs:subClassOf o .

La saturation d'une BD db avec cet ensemble restreint de règles est noté $\text{db}^{\propto}$, d'où $\text{db}^{\propto} \subseteq \text{db}^{\infty}$.

L'*évaluation d'une requête q sur une BD* db est exactement l'évaluation de q sur le graphe db, càd. $q(\text{db})$, et l'*ensemble de réponses à q sur* db est $q(\text{db}^{\propto})$, d'où $q(\text{db}^{\propto}) \subseteq q(\text{db}^{\infty})$.

En général, les requêtes « utilisateur » peuvent utiliser les deux parties d'une BD (schéma et instance). Dans notre exemple, on pourrait demander quels sont les ranges des propriétés décrivant les articles de conférence, càd. ClassRelatedToConfPaper(x):- y_1 rdf:type confP ., y_1 y_2 y_3 ., y_2 rdfs:range x . Toutefois, la séparation entre schéma et instance, correspondant à la vision usuelle qu'ont les utilisateurs d'une base de données ou de connaissances, mène souvent à spécifier des requêtes sur le schéma seulement ou sur l'instance seulement. D'un point de vue « bases de données », les *requêtes sur l'instance* sont les plus courantes. Observons toutefois que répondre à ces requêtes nécessite les triplets du schéma, puisque la saturation de la BD (nécessaire pour obtenir un ensemble complet de réponses) repose sur les triplets du schéma (cf. figure 4). Un exemple de requête sur l'instance demandant la ville de RFIA 2012 est : RFIACity(y):- x name "RFIA2012" ., x city y . Les requêtes sur l'instance peuvent aussi retourner des classes et propriétés, par ex. en demandant les classes auxquelles une certaine valeur appartient, càd. ClassFinding(x):- v rdf:type x . D'un point de vue « Représentation des connaissances » (RC), les *requêtes sur le schéma* jouent aussi un rôle important. Ces requêtes offrent un moyen d'explorer les relations entre classes et propriétés du schéma, y compris les relations implicites. Par exemple, on peut demander si une classes est une sous-classe d'une autre, par ex. SubclassChecking():- posterCP rdfs:subClassOf paper ., ou quelles sont les classes typant le domaine d'une certaine propriété, par ex. DomainFinding(x):- inProceedingsOf rdfs:domain x . Une autre requête d'exploration de schéma peut être simplement Schema(x, y, z):- x y z . Cette requête retourne tous les triplets de la BD. En restreignant la requête au schéma de la BD, l'utilisateur récupère toutes les contraintes entre classes et propriétés, quelles soient explicites ou implicites. Nous achevons cette section en observant que notre formalisme est assez général pour intégrer à la fois des requêtes sur l'instance typiques en BD, des requêtes sur le schéma typiques en RC, ainsi que des requêtes sur le schéma et l'instance.

5 Techniques de réponse aux requêtes

Nous étudions deux techniques pour répondre aux requêtes sur des BD RDF : l'une par saturation de la BD, l'autre par reformulation des requêtes.

Répondre par saturation revient à calculer l'ensemble des réponses à une requête exactement comme cet ensemble est formellement défini. La saturation de la BD est calculée (en utilisant les règles d'entailment autorisées), de sorte que l'ensemble de réponses à une requête sur la BD est obtenue par évaluation de la requête sur la BD saturée. L'avantage de cette technique est sa facilité de mise en œuvre. Ses inconvénients sont que la saturation nécessite du temps pour être calculé, de l'espace pour être stockée, et que celle-ci doit être recalculée lors de mises-à-jour.

Répondre par reformulation consiste à reformuler une requête q par rapport à une BD db en une nouvelle requête q', de sorte que l'évaluation de q' sur la BD (originale) db, soit exactement l'ensemble de réponse de q sur db. L'avantage de cette technique est que la saturation n'a pas à être calculée. L'inconvénient est que chaque requête doit être reformulée, et que sa reformulation résulte généralement en une requête plus complexe à évaluer.

Dans la suite, nous nous intéressons à répondre aux requêtes par saturation et par reformulation, dans le cas des requêtes sur l'instance uniquement (les plus courantes). Le théorème suivant montre que pour répondre à de telles requêtes, il suffit de considérer uniquement les règles d'entailment de la figure 4 (alors que notre fragment BD utilise toutes les règles mentionnées en section 4) :

Théorème 1. *Soient* db *une BD RDF, t_1 un triplet de la forme* s rdf:type o . *et t_2 un triplet de la forme* s p o . *$t_1 \in \text{db}^{\propto}$ (resp. $t_2 \in \text{db}^{\propto}$) ssi il existe une séquence d'application des règles de la figure 4 menant de* db *à t_1 (resp. t_2), en supposant que chaque pas d'entailment considère* db *et tous les triplets engendrés précédemment.*

La preuve du théorème 1 et celles des théorèmes suivants sont dans [7].

$$\frac{\{\mathbf{c}_1 \text{ rdfs:subClassOf } \mathbf{c}_2 \text{ .}, \mathtt{s} \text{ rdf:type } \mathbf{c}_1 \text{ .}\} \subseteq \mathtt{db}}{\mathtt{db} = \mathtt{db} \cup \{\mathtt{s} \text{ rdf:type } \mathbf{c}_2 \text{ .}\}} \quad (1)$$

$$\frac{\{\mathbf{p} \text{ rdfs:domain } \mathbf{c} \text{ .}, \mathtt{s}\ \mathbf{p}\ \mathtt{o} \text{ .}\} \subseteq \mathtt{db}}{\mathtt{db} = \mathtt{db} \cup \{\mathtt{s} \text{ rdf:type } \mathbf{c} \text{ .}\}} \quad (2)$$

$$\frac{\{\mathbf{p} \text{ rdfs:range } \mathbf{c} \text{ .}, \mathtt{s}\ \mathbf{p}\ \mathtt{o} \text{ .}\} \subseteq \mathtt{db}}{\mathtt{db} = \mathtt{db} \cup \{\mathtt{o} \text{ rdf:type } \mathbf{c} \text{ .}\}} \quad (3)$$

$$\frac{\{\mathbf{p}_1 \text{ rdfs:subPropertyOf } \mathbf{p}_2 \text{ .}, \mathtt{s}\ \mathbf{p}_1\ \mathtt{o} \text{ .}\} \subseteq \mathtt{db}}{\mathtt{db} = \mathtt{db} \cup \{\mathtt{s}\ \mathbf{p}_2\ \mathtt{o} \text{ .}\}} \quad (4)$$

FIGURE 5 – Règles de saturation pour une BD db

$$\begin{aligned}
&\mathtt{Saturate}^0(\mathtt{db}) = \mathtt{db}\\
&\mathtt{Saturate}^1(\mathtt{db}) = \mathtt{Saturate}^0(\mathtt{db}) \cup\\
&\qquad \{\text{doi}_1 \text{ rdf:type confP .}\\
&\qquad\ \text{doi}_1 \text{ rdf:type paper .}\\
&\qquad\ \text{doi}_1 \text{ hasAuthor } _{:}b_1 \text{ .}\\
&\qquad\ _{:}b_2 \text{ rdf:type conference .}\}\\
&\mathtt{Saturate}^2(\mathtt{db}) = \mathtt{Saturate}^1(\mathtt{db})
\end{aligned}$$

FIGURE 6 – Saturation de la BD db

6 Répondre par saturation

Notre technique de réponse aux requêtes par saturation se fonde sur notre algorithme Saturate, qui ne calcule que la partie pertinente de la saturation d'une BD afin de répondre aux requêtes sur l'instance. Évaluer la requête originale sur cette saturation fournit alors l'ensemble exact de réponses. Saturate repose sur les règles de saturation de la figure 5, découlant directement des règles d'entailment de la figure 4. Dans la figure 5 et dans la suite, les symboles en gras (éventuellement avec indices), **c** pour une classe et **p** pour une propriété, indiquent des marqueurs génériques. Les règles de la figure 5 définissent un ensemble de transformations de BD de la forme $\frac{input}{output}$, où $input$ et $output$ sont des BD. Intuitivement, étant donnée une BD db, Saturate(db) applique exhaustivement les règles de la figure 5, sur db et les triplets engendrés successivement.
La sortie de Saturate(db) est définie comme le point fixe $\mathtt{Saturate}^\infty(\mathtt{db})$, où :

$\mathtt{Saturate}^0(\mathtt{db}) = \mathtt{db}$
$\mathtt{Saturate}^{k+1}(\mathtt{db}) = \mathtt{Saturate}^k(\mathtt{db}) \cup \{t_3 \mid \exists i \in [1,4]$ tq
appliquer la règle (i) sur $\{t_1, t_2\} \subseteq \mathtt{Saturate}^k(\mathtt{db})$ produit $t_3\}$

Exemple 4 (suite)**.** *Considérons dorénavant la BD* db *faite du graphe* G′ *précédent, auquel nous ajoutons le triplet* vldb2012 rdf:type conference . *indiquant que l'URI* vldb2012 *est une conférence, càd.* db = G′ ∪ {vldb2012 rdf:type conference .}.
La saturation de db *est donnée dans la figure 6.*

Le théorème 2 établit une borne supérieure de la taille de la sortie de Saturate.

Théorème 2. *Pour une BD RDF* db*, la taille (nombre de triplets) de la sortie de* Saturate(db) *est en* $O(\#\mathtt{db}^2)$*, où* #db *est la taille (nombre of triplets) de* db*.*

À partir de la notion de saturation définie ci-dessus, nous pouvons définir notre technique de réponse aux requêtes par saturation de BD :

Théorème 3. *Étant données une requête* q *et une BD RDF* db*, l'égalité suivante est vérifiée :*

$$q(\mathtt{db}^\infty) = q(\mathtt{saturate}(\mathtt{db})).$$

La preuve découle trivialement du théorème 1 et de la définition de notre algorithme Saturate.

Exemple 5 (suite)**.** *Soit la requête* $q(x, y)$*:-* x rdf:type y . *demandant les ressources et les classes auxquelles elles appartiennent. L'ensemble des réponses à* q *sur* db *est :*

$q(\mathtt{Saturate}(\mathtt{db})) = \{\langle \text{doi}_1, \_{:}b_0\rangle, \langle \text{vldb2012, conference}\rangle, \langle \text{doi}_1, \text{confP}\rangle,$
$\langle \text{doi}_1, \text{paper}\rangle, \langle \_{:}b_2, \text{conference}\rangle\}.$

7 Répondre par reformulation

Notre technique de réponse aux requêtes par reformulation se fonde sur notre algorithme Reformulate. Étant données une requête q et une BD db, Reformulate$(q, \mathtt{db})$ reformule q en un ensemble de requêtes, tel que l'union des *évaluations non standard* (voir ci-dessous) de ces requêtes sur db produit $q(\mathtt{db}^\infty)$, l'ensemble exact de réponses de la requête originale sur la BD.
Reformulate applique exhaustivement l'ensemble de règles de la figure 7, en partant d'une requête q et d'une BD db. Chaque règle définit une transformation de la forme $\frac{input}{output}$, où $input$ est de la forme ⟨ condition sur db, condition sur q ⟩ et $output$ est une requête q'. Chacune des conditions de l'input, mais pas les deux, peut être vide. Intuitivement, chaque règle produit une nouvelle requête lorsque les conditions de son input sont satisfaite, l'une par la BD db, et l'autre par la requête (soit la requête originale q, soit une requête q' produite par une application précédente d'une règle). L'ensemble de toutes les requêtes générées par application des règles est le résultat de la reformulation de q par rapport à db.
Requêtes partiellement instanciées. Soit q une requête. Une *requête partiellement instanciée*, notée q_σ, est une requête $q_\sigma(\bar{x}_\sigma)$:- $(t_1, \ldots, t_\alpha)_\sigma$ où σ assigne un sous-ensemble des variables et nœuds blancs de q à des valeurs (URIs, nœuds blancs, et littéraux). D'une façon non standard, certaines variables distinguées de q_σ peuvent être instanciées. Si $\sigma = \emptyset$ aors q_σ coïncide avec la requête q.
Les notions d'évaluation et d'ensemble de réponses de requêtes, fournies en section 3 pour des graphes RDF et en section 4 pour les BD RDF, s'étendent aux requêtes partiellement instanciées comme suit. Étant données une BD db dont l'ensemble de valeurs (URIs, nœuds blancs et littéraux) est Val(db), et une requête $q_\sigma(\bar{x}_\sigma)$:- $(t_1, \ldots, t_\alpha)_\sigma$ dont l'ensemble des variables et nœuds blancs est VarBl(q_σ), l'*évaluation* de q_σ sur db est : $q_\sigma(\mathtt{db}) = \{(\bar{x}_\sigma)_\mu \mid \mu : \mathtt{VarBl}(q_\sigma) \to \mathtt{Val}(\mathtt{db})$ est une assignation totale tq $((t_1, \ldots, t_\alpha)_\sigma)_\mu \subseteq \mathtt{db}\}$.
L'*ensemble des réponses* à q_σ sur db est l'évaluation de q_σ sur $\mathtt{db}^\infty$, notée $q_\sigma(\mathtt{db}^\infty)$.
Règles de reformulation. Les règles (5)–(13) reformulent les requêtes en instanciant des variables. Les autres règles (14)–(17) remplace des triplets d'une requête par d'autres.
Considérons par exemple la règle (5). Elle dit que si une requête q_σ a un triplet de la forme s y o ., càd. peu importe son sujet et son object pourvu qu'il ait une variable en lieu

$$\frac{\langle \mathtt{s}\ y\ \mathtt{o}\ .\ \in q_\sigma\rangle}{q_{\sigma\cup\nu=\{y\to \text{rdf:type}\}}} \quad (5)$$

$$\frac{\langle \mathtt{s}_1\ \mathbf{p}\ \mathtt{o}_1\ .\ \in \mathtt{db}, \mathtt{s}\ y\ \mathtt{o}\ .\ \in q_\sigma\rangle}{q_{\sigma\cup\nu=\{y\to \mathbf{p}\}}} \quad (6)$$

$$\frac{\langle \mathtt{s}_1 \text{ rdfs:subPropertyOf } \mathbf{p}\ .\ \in \mathtt{db}, \mathtt{s}\ y\ \mathtt{o}\ .\ \in q_\sigma\rangle}{q_{\sigma\cup\nu=\{y\to \mathbf{p}\}}} \quad (7)$$

$$\frac{\langle \mathbf{p} \text{ rdfs:subPropertyOf } \mathtt{o}_1\ .\ \in \mathtt{db}, \mathtt{s}\ y\ \mathtt{o}\ .\ \in q_\sigma\rangle}{q_{\sigma\cup\nu=\{y\to \mathbf{p}\}}} \quad (8)$$

$$\frac{\langle \mathtt{s}_1 \text{ rdf:type } \mathbf{c}\ .\ \in \mathtt{db}, \mathtt{s} \text{ rdf:type } z\ .\ \in q_\sigma\rangle}{q_{\sigma\cup\nu=\{z\to \mathbf{c}\}}} \quad (9)$$

$$\frac{\langle \mathtt{s}_1 \text{ rdfs:subClassOf } \mathbf{c}\ .\ \in \mathtt{db}, \mathtt{s} \text{ rdf:type } z\ .\ \in q_\sigma\rangle}{q_{\sigma\cup\nu=\{z\to \mathbf{c}\}}} \quad (10)$$

$$\frac{\langle \mathbf{c} \text{ rdfs:subClassOf } \mathtt{o}\ .\ \in \mathtt{db}, \mathtt{s} \text{ rdf:type } z\ .\ \in q_\sigma\rangle}{q_{\sigma\cup\nu=\{z\to \mathbf{c}\}}} \quad (11)$$

$$\frac{\langle \mathtt{s}_1 \text{ rdfs:domain } \mathbf{c}\ .\ \in \mathtt{db}, \mathtt{s} \text{ rdf:type } z\ .\ \in q_\sigma\rangle}{q_{\sigma\cup\nu=\{z\to \mathbf{c}\}}} \quad (12)$$

$$\frac{\langle \mathtt{s}_1 \text{ rdfs:range } \mathbf{c}\ .\ \in \mathtt{db}, \mathtt{s} \text{ rdf:type } z\ .\ \in q_\sigma\rangle}{q_{\sigma\cup\nu=\{z\to \mathbf{c}\}}} \quad (13)$$

$$\frac{\langle \mathbf{c}_1 \text{ rdfs:subClassOf } \mathbf{c}_2\ .\ \in \mathtt{db}, \mathtt{s} \text{ rdf:type } \mathbf{c}_2\ .\ \in q_\sigma\rangle}{q_{\sigma[\mathtt{s} \text{ rdf:type } \mathbf{c}_2\ ./\mathtt{s} \text{ rdf:type } \mathbf{c}_1\ .]}} \quad (14)$$

$$\frac{\langle \mathbf{p} \text{ rdfs:domain } \mathbf{c}\ .\ \in \mathtt{db}, \mathtt{s} \text{ rdf:type } \mathbf{c}\ .\ \in q_\sigma\rangle}{q_{\sigma[\mathtt{s} \text{ rdf:type } \mathbf{c}\ ./\mathtt{s}\ \mathbf{p}\ y\ .]}} \quad (15)$$

$$\frac{\langle \mathbf{p} \text{ rdfs:range } \mathbf{c}\ .\ \in \mathtt{db}, \mathtt{s} \text{ rdf:type } \mathbf{c}\ .\ \in q_\sigma\rangle}{q_{\sigma[\mathtt{s} \text{ rdf:type } \mathbf{c}\ ./y\ \mathbf{p}\ \mathtt{s}\ .]}} \quad (16)$$

$$\frac{\langle \mathbf{p}_1 \text{ rdfs:subPropertyOf } \mathbf{p}_2\ .\ \in \mathtt{db}, \mathtt{s}\ \mathbf{p}_2\ \mathtt{o}\ .\ \in q_\sigma\rangle}{q_{\sigma[\mathtt{s}\ \mathbf{p}_1\ \mathtt{o}\ ./\mathtt{s}\ \mathbf{p}_2\ \mathtt{o}\ .]}} \quad (17)$$

FIGURE 7 – Règles de reformulation pour une requête partiellement instanciée q_σ par rapport à une BD db

et place de la propriété, alors créer une nouvelle requête $q_{\sigma\cup\nu}$ en instanciant y à la propriété prédéfinie rdf:type. On observera que si y était une variable distinguées de q_σ, alors la tête de $q_{\sigma\cup\nu}$ contient la constante rdf:type en lieu et place de la variable y. Considérons maintenant la règle (6) appliquée à une requête q_σ. Si q_σ contient un triplet de la même forme $\mathtt{s}\ y\ \mathtt{o}\ .$ et que la db contient un triplet avec une propriété $\mathbf{p}$, alors la règle crée la nouvelle requête $q_{\sigma\cup\nu}$, où y est instanciée à $\mathbf{p}$. Les règles (7) et (8) instancient des variables apparaissant comme propriétés de triplets, à des valeurs apparaissant dans des contraintes de sous-propriété. L'intuition est que le sujet comme l'objet de telles assertions RDFS sont des propriétés, par conséquent ils peuvent être utilisés pour instancier la variable y.

Les règles (9)–(13) instancient la variable z de triplets de la forme $\mathtt{s}$ rdf:type z . Le méta-modèle RDF spécifie que, dans ce cas, les valeurs possibles de z sont des classes. Ainsi, les règles instancient z à des valeurs de db pouvant être inférées comme étant des classes, càd. celles apparaissant à des positions particulières dans les assertions de la BD. Par exemple, si $\mathtt{s}_1$ rdf:type $\mathbf{c}$. $\in$ db, alors $\mathbf{c}$ est une classe et z dans la règle (9) peut être instanciée à $\mathbf{c}$. De façon similaire, le sujet et l'objet des contraintes de sous-classe sont utilisés dans les règles (10) et (11), et l'objet des contraintes de domaine et de range est utilisé dans les règles (12) et (13). Enfin, les règles (14)–(17) utilisent des assertions RDFS pour remplacer (noté *[ancien triplet / nouveau triplet]*) un triplet dans la requête de l'input par un nouveau triplet. Par exemple, la règle (14) exploite les contraintes de sous-classe : si la requête q_σ demande les instances de la classe $\mathbf{c}_2$ et que $\mathbf{c}_1$ est une sous-classe de $\mathbf{c}_2$ dans la BD, alors les instances de $\mathbf{c}_1$ doivent aussi être retournées, c'est alors ce que fait la requête de l'output. Les trois dernières règles sont similaires, et utilisent resp. les contraintes de domaine, de range, et de sous-propriété.

$\texttt{Reformulate}^0(q, \mathtt{db}) = \{q(x, y)\text{:- } x \text{ rdf:type } y\ .\}$
$\texttt{Reformulate}^1(q, \mathtt{db}) = \texttt{Reformulate}^0(q, \mathtt{db}) \cup$
$\{q(x, \text{confP})\text{:- } x \text{ rdf:type confP}\ .$
$q(x, \text{posterCP})\text{:- } x \text{ rdf:type posterCP}\ .$
$q(x, \_{:}b_0)\text{:- } x \text{ rdf:type } \_{:}b_0\ .$
$q(x, \text{paper})\text{:- } x \text{ rdf:type paper}\ .$
$q(x, \text{conference})\text{:- } x \text{ rdf:type conference}\ .\}$
$\texttt{Reformulate}^2(q, \mathtt{db}) = \texttt{Reformulate}^1(q, \mathtt{db}) \cup$
$\{q(x, \text{confP})\text{:- } x \text{ rdf:type posterCP}\ .$
$q(x, \text{confP})\text{:- } x \text{ rdf:type } \_{:}b_0\ .$
$q(x, \text{confP})\text{:- } x \text{ inProceedingsOf } z\ .$
$q(x, \text{paper})\text{:- } x \text{ rdf:type confP}\ .$
$q(x, \text{paper})\text{:- } x \text{ hasTitle } z\ .$
$q(x, \text{paper})\text{:- } x \text{ hasAuthor } z\ .$
$q(x, \text{conference})\text{:- } z \text{ inProceedingsOf } x\ .$
$q(x, \text{conference})\text{:- } x \text{ hasName } z\ .\}$
$\texttt{Reformulate}^3(q, \mathtt{db}) = \texttt{Reformulate}^2(q, \mathtt{db}) \cup$
$\{q(x, \text{paper})\text{:- } x \text{ rdf:type posterCP}\ .$
$q(x, \text{paper})\text{:- } x \text{ rdf:type } \_{:}b_0\ .$
$q(x, \text{paper})\text{:- } x \text{ inProceedingsOf } z\ .$
$q(x, \text{paper})\text{:- } x \text{ hasContactA } z\ .\}$
$\texttt{Reformulate}^4(q, \mathtt{db}) = \texttt{Reformulate}^3(q, \mathtt{db})$

FIGURE 8 – Reformulation de q par rapport à db

Exemple 6 (suite)**.** *Considérons maintenant la requête* $q(x, y, z)$*:-* $x\ y\ z$ *. demandant les triplets de* db *(dont les implicites). Nous montrons comment certaines des règles ci-dessus sont utilisées pour reformuler* q *par rapport à* db*. Utiliser* q *comme input de la règle (5) produit la requête :* $q_{\{y\to \text{rdf:type}\}}$*, càd.* $q(x, y, z)$*:-* x rdf:type z *.*
Utiliser $q_{\{y\to \text{rdf:type}\}}$ *comme input de la règle (11) mène à :* $q_{\{y\to \text{rdf:type}, z\to \text{confP}\}}$*, càd.* $q(x, \text{rdf:type}, \text{confP})$*:-* x rdf:type confP *.*
Enfin, prendre $q_{\{y\to \text{rdf:type}, z\to \text{confP}\}}$ *comme input de la règle (14) mène à :* $q(x, \text{rdf:type}, \text{confP})$*:-* x rdf:type $\_{:}b_0$ *.*

Algorithme de reformulation de requête. Pour une requête q et une BD db, la sortie de $\texttt{Reformulate}(q, \mathtt{db})$ est définie comme le point fixe $\texttt{Reformulate}^\infty(q, \mathtt{db})$, où :

$\texttt{Reformulate}^0(q, \mathtt{db}) = \{q\}$
$\texttt{Reformulate}^{k+1}(q, \mathtt{db}) = \texttt{Reformulate}^k(q, \mathtt{db}) \cup \{q''_{\sigma''} \mid \exists i \in [5, \ldots, 17]$
tq appliquer la règle (i) sur db et une requête
$q'_{\sigma'} \in \texttt{Reformulate}^k(q, \mathtt{db})$ produit la requête $q''_{\sigma''}\}$

Le théorème 4 fournit une borne supérieure pour la taille de la sortie de Reformulate.

Théorème 4. *Étant données une requête* q *et une BD* db*, la taille (nombre de requêtes) de la sortie de* $\texttt{Reformulate}(q, \mathtt{db})$ *est en* $O((6 * \#\mathtt{db}^2)^{\#q})$*, avec* $\#\mathtt{db}$ *et* $\#q$ *les tailles (nombre de triplets) de* db *et de* q *resp.*

Exemple 7 (suite)**.** *La reformulation de la requête* $q(x, y)$*:-* x rdf:type y *. par rapport à* db*, demandant les ressources et les classes auxquelles elles appartiennent, est fournit en figure 8.*

Répondre par reformulation de requête. Un prérequis pour toute technique de réponse aux requêtes par reformulation est que les requêtes obtenues par reformulation soient équivalentes ou incluses dans la requête originale (par rapport à aux contraintes de la BD), sinon leur évaluation produirait des réponses erronées. Il apparaît que notre technique de reformulation ne respecte pas ce prérequis, si l'on considère les définitions précédemment fournies d'évaluation et d'ensembles de réponses à une requête sur une BD, comme l'illustre l'exemple suivant.

Exemple 8 (suite)**.** *Considérons encore la BD* db. *Pour la requête précédente* $q(x, y)$*:-* x rdf:type y *., l'évaluation des requêtes de* $\texttt{Reformulate}(q, \texttt{db})$*, illustrées en figure 8, fournit des réponses erronées. Par ex. le tuple* $\langle \text{vldb2012}, \text{confP} \rangle$ *est une réponse erronée (voir l'exemple 5 pour les réponses exactes) résultant de l'évaluation sur* db *de* $q(x, \text{confP})$*:-* x rdf:type $\_{:}b_0$. *dans* $\texttt{Reformulate}^2(q, \texttt{db})$*, avec l'assignation* $\mu = \{x \rightarrow \text{vldb2012}, \_{:}b_0 \rightarrow \text{conference}\}$.

Comme le suggère l'exemple ci-dessus, le problème vient des nœuds blancs. Il y a un décalage entre la sémantique des nœuds blancs dans les requêtes et les raisons pour lesquelles ils sont introduits dans les requêtes produites par notre algorithme Reformulate. Pour mémoire, la sémantique d'un nœud blanc dans une requête sur une BD est celle d'une variable non distinguées. Toutefois, lorsque notre algorithme Reformulate introduit un nœud blanc dans une requête à partir d'une instanciation de variable ou un remplacement de triplet, il réfère précisément à ce nœud blanc particulier de la BD. Dans l'exemple ci-dessus, lorsque la requête $q(x, \text{confP})$:- x rdf:type $\_{:}b_0$. dans $\texttt{Reformulate}^2(q, \texttt{db})$ est produite par la règle (14) à partir de $\_{:}b_0$ rdfs:subClassOf confP . $\in$ db et $q(x, \text{confP})$:- x rdf:type confP . dans $\texttt{Reformulate}^1(q, \texttt{db})$, l'objectif est précisément de trouver des articles de conférence pour x à partir de la sous-classe $\_{:}b_0$ de confP.

Évaluation et ensemble de réponses non standard d'une requête sur une BD. Pour résoudre le problème identifié ci-dessus, nous introduisons les *notions alternatives d'évaluation et d'ensemble de réponses d'une requête partiellement instanciée sur une BD*. La différence fondamentale entre les définitions alternatives et les définitions standards introduites en section 7 concerne les nœuds blancs. L'évaluation d'une requête précédemment définie se fonde sur des assignations de $\texttt{VarBl}(q)$, tous les variables et nœuds blancs de la requête, à des valeurs dans la BD. En revanche, la définition alternative cherche seulement à assigner les variables de la requête ; les nœuds blancs ne sont pas touchés, comme les URIs et littéraux.

Étant données une BD db, dont l'ensemble de valeurs (URIs, nœuds blancs et littéraux) est $\texttt{Val}(\texttt{db})$, et une requête $q_\sigma(\bar{x}_\sigma)$:- $(t_1, \ldots, t_\alpha)_\sigma$, dont l'ensemble de variables (uniquement) est $\texttt{Var}(q_\sigma)$, l'*évaluation non standard* de q_σ sur db est : $\tilde{q_\sigma}(\texttt{db}) = \{(\bar{x}_\sigma)_\mu \mid \mu : \texttt{Var}(q_\sigma) \rightarrow \texttt{Val}(\texttt{db})$ est une assignation totale tq $((t_1, \ldots, t_\alpha)_\sigma)_\mu \subseteq \texttt{db}\}$.

L'*ensemble non standard des réponses* à q_σ sur db est l'évaluation non standard de q_σ sur $\texttt{db}^\infty$, càd. $\tilde{q_\sigma}(\texttt{db}^\infty)$.

Théorème 5. *Étant données une requête* q *sans nœud blanc et une BD* db*, l'égalité suivante est vérifiée :*

$$q(\texttt{db}^\infty) = \bigcup_{q'_{\sigma'} \in \texttt{Reformulate}(q, \texttt{db})} \tilde{q}'_{\sigma'}(\texttt{db}).$$

Remarquons que le théorème 5 considère des requêtes *sans* nœud blanc. Cette hypothèse est faite *sans perte de généralité*, puisque les nœuds blancs jouent le rôle (donc peuvent être remplacés de façon équivalente par) des variables non distinguées dans les requêtes (originales). Nous faisons cette hypothèse afin qu'à l'issue de l'étape de reformulation, nous n'ayons pas deux types (donc deux traitements différents lors de l'évaluation de requêtes produites par reformulation) des nœuds blancs : ceux provenant de la requête originale (càd. des variables non distinguées) et ceux venant de l'étape de reformulation (càd. des valeurs référant à elles-mêmes dans la BD).

De façon plus pragmatique, le théorème 5 dit que pour répondre à une requête q sur une BD db, il suffit de (i) reformuler q par rapport à db et (ii) évaluer chacune des requêtes produites par reformulation sur la BD *originale*, en utilisant l'évaluation *non standard*. En d'autres termes, la reformulation et l'évaluation non standard de requête permettent de calculer l'ensemble *standard* des réponses à une requête, *sans* saturer la BD.

Exemple 9 (suite)**.** *Considérons de nouveau la requête* $q(x, y)$*:-* x rdf:type y *. L'ensemble des réponses à* q *sur* db *est :*

$$\bigcup_{q_\sigma \in \texttt{Reformulate}(q, \texttt{db})} \tilde{q}_\sigma(\texttt{db}) = \{\langle \text{doi}_1, \_{:}b_0 \rangle, \langle \text{vldb2012}, \text{conference} \rangle, \langle \text{doi}_1, \text{confP} \rangle, \langle \text{doi}_1, \text{paper} \rangle, \langle \_{:}b_2, \text{conference} \rangle\}.$$

Remarquons que cet ensemble coïncide avec celui obtenu par saturation dans l'exemple 5.

8 Évaluation expérimentale

Nous évaluons maintenant nos techniques de réponse aux requêtes sur une version RDF de DBLP [6], constituée de 41 assertions RDFS et 8.4 millions d'assertions RDF.

Nous avons implémenté nos algorithmes Saturate et Reformulate en Java 1.6 « au-dessus » d'un serveur PostgreSQL. L'instance de DBLP et de sa saturation sont stockées respectivement dans les tables $\texttt{Triple}(\texttt{s}, \texttt{p}, \texttt{o})$ et $\texttt{Sat}(\texttt{s}, \texttt{p}, \texttt{o})$, tandis que le schéma est en mémoire. Nous procédons de la sorte afin de déléguer l'évaluation de nos requêtes au serveur relationnel. En effet, l'évaluation relationnelle de requête coïncide soit avec l'évaluation standard de requête RDF quand il n'y a pas de nœud blanc (c'est le cas de nos requêtes originales, cf. théorème 5) ou avec l'évaluation non standard lorsque la requête a des nœuds blancs (c'est le cas de nos reformulations, cf. théorème 5). La table Sat a été peuplée par notre algorithme Saturate en 950 secondes, avec 11.9 millions de triplets.

Nous avons défini un ensemble de 34 requêtes, afin d'illustrer une variété de comportements lors de la reformulation ; une description détaillée de ces requêtes est fournie dans [7].

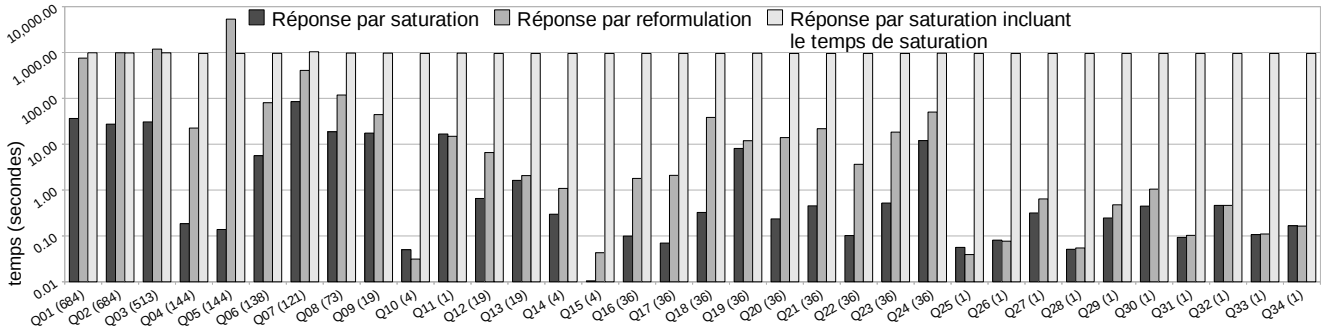


FIGURE 9 – Temps de réponse aux requêtes sur $DBLP$.

La figure 9 montre, pour chaque requête, (i) la taille de la reformulation (entre parenthèses après le nom de la requête), (ii) le temps pour répondre par saturation en utilisant la table `Sat` ; (iii) le temps pour répondre par reformulation en utilisant l'algorithme `Reformulate` et la table `Triple` ; (iv) le temps pour peupler la table `Sat` et répondre à la requête sur celle-ci. Chaque valeur représente une moyenne de 10 essais. Comme prévu, répondre à une requête est significativement plus rapide par saturation que par reformulation, pour les requêtes avec des reformulations de grandes tailles. La saturation est plus rapide puisqu'une seule requête est évaluée sur la table `Sat` (déjà saturée). Toutefois, ceci a un prix, car la saturation nécessite d'être calculée initialement (ainsi que lors des mises à jour). Partant de la figure 9, nous avons calculé combien de fois chaque requête doit être exécutée pour amortir le coût de la saturation. Nous montrons dans [7] – sur Barton, DBLP et DBpedia – que pour des requêtes avec des reformulations de petites tailles, ceci peut dépasser les 10 millions de fois.

9 Travaux connexes

À notre connaissance, répondre à des requêtes par reformulation n'a été étudié que dans le fragment de LD de RDF et pour les requêtes conjonctives relationnelles [3, 5].

Le fragment LD de RDF permet de modéliser des bases de connaissances (BC) qui sont des théories de la logique du premier ordre. Das ces BC, la notion de conséquence logique joue un rôle similaire à celui de l'entailment dans RDF : des assertions implicites de schéma ou de données peuvent être exhibées. En particulier, les fragments LD et BD de RDF partagent les mêmes règles d'entailment. Toutefois, le fragment LD est *strictement moins* expressif que le fragment BD. Tout d'abord, les nœuds blancs ne sont pas autorisés dans les BC de LD, empêchant ainsi d'exprimer des informations incomplètes. Ensuite, il n'est pas possible d'utiliser une classe ou une propriété comme une constante dans l'instance d'une BC de LD (une BC étant une théorie de la logique du premier ordre, l'ensemble des relations et celui des constantes sont disjoints).

Le langage de requête considéré dans le fragment LD de RDF est celui des (unions de) requêtes conjonctives relationnelles. Ce language est *strictement moins* expressif que celui des requêtes RDF, car il correspond à des (unions de) requêtes RDF dont les triplets sont uniquement de la forme `s` $\mathrm{rdf{:}type}\ c$. ou `s` p `o` ., empêchant de mettre des variables en lieux et places des noms de classes et propriétés. Enfin, les algorithmes de reformulation de requêtes pour le fragment LD de RDF sont des restrictions de notre algorithme `Reformulate` aux seules règles (14)–(17). La reformulation dans le langage plus expressif des requêtes RDF requiert en plus les règles (5)–(13). De plus, répondre aux requêtes RDF par reformulation sur des BD RDF requiert une évaluation *non standard* des requêtes reformulées.

10 Conclusion

Dans cet article, nous avons étendu l'état de l'art sur la réponse aux requêtes par reformulation dans RDF en considérant un fragment plus significatif de RDF et un langage de requête plus expressif que ceux précédemment étudiés. Nous avons aussi comparé les performances de répondre aux requêtes par reformulation ou par saturation, en fournissant une première estimation à gros grain de quelle technique est plus appropriée en fonction du type et de la fréquence des requêtes.

Nous envisageons d'étudier la maintenance de saturation lors de mise-à-jour, dans le but de l'optimiser. Nous souhaitons aussi comparer plus précisément l'efficacité de nos techniques de réponse aux requêtes en prenant en compte plus de facteurs comme la taille et la fréquence des mises à jour. Cette analyse sera théorique avec des BD et requêtes synthétiques, et pratique avec des BD bien connues (par ex. Barton, DBpedia et DBLP) et leurs logs de requêtes.

Références

[1] Jena. http ://jena.sourceforge.net/.

[2] Owlim. http ://owlim.ontotext.com.

[3] P. Adjiman, F. Goasdoué, and M.-C. Rousset. SomeRDFS in the semantic web. *Journal On Data Semantics*, 8, 2007.

[4] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook*. Cambridge University Press, 2003.

[5] D. Calvanese, G. D. Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Tractable reasoning and efficient query answering in description logics : The DL-Lite family. *Journal of Automated Reasoning (JAR)*, 39(3), 2007.

[6] http ://kdl.cs.umass.edu/data/dblp/dblp-info.html.

[7] http ://www.lri.fr/~roatis/RFIA2012-TR.pdf.

[8] G. Gottlob, G. Orsi, and A. Pieris. Ontological queries : Rewriting and optimization. In *ICDE*, 2011.

[9] W3C. Resource description framework. http ://www.w3.org/RDF/.

[10] W3C. SPARQL protocol and RDF query language. http ://www.w3.org/TR/rdf-sparql-query/.