Étude des propriétés de règles de réécriture de requêtes sur le Web sémantique

Galaad Langlois

Étude des propriétés de règles de réécriture de requêtes sur le Web sémantique

Stage L3

Galaad Langlois

ENS de Lyon

Encadré par

Olivier Bruneau `olivier.bruneau@univ-lorraine.fr`
AHP-PReST (Archives Henri-Poincaré)

Nicolas Lasolle `nicolas.lasolle@univ-lorraine.fr`
AHP-PReST (Archives Henri-Poincaré)
LORIA (Université de Lorraine, CNRS, Inria), Nancy

Jean Lieber `jean.lieber@loria.fr`, équipe $\mathcal{K}$
LORIA (Université de Lorraine, CNRS, Inria), Nancy

Table des matières

1 Introduction

Le Web sémantique est un ensemble d'outils et de technologies qui permettent d'organiser, de lier ou encore d'interroger les données du Web, en leur donnant du « sens » (contrairement au Web classique, qui permet de faire des liens entre des pages Web, mais pas de caractériser et de lier des représentations de personnes, événements, objets concrets, etc.). Parmi ces technologies, on retrouve RDF (*Resource Description Framework*), un langage de bases de données, et SPARQL, un langage de requêtes sur des bases de données RDF. Le langage RDF organise les données sous forme de graphes, en liant des « ressources » avec des arcs étiquetés par des propriétés.

Aux Archives Henri-Poincaré, le Web sémantique est utilisé pour l'accès et la gestion du corpus de la correspondance du mathématicien, physicien et philosophe Henri Poincaré (1854-1912), né à Nancy. Des milliers de lettres de cette correspondance ont été transcrites, et les données qu'elles comportent ont été organisées sous forme de graphes RDF. Cela permet, notamment dans un cadre de recherches historiques, d'interroger ces données afin d'obtenir facilement des informations sur les lettres de ce corpus (correspondants, date et lieu de rédaction, thème, etc.), mais aussi sur des éléments de la vie d'Henri Poincaré, les personnes associées à son réseau scientifique, académique et privé, les sujets qui l'ont intéressé, etc. Pour cela, on peut écrire des requêtes SPARQL, qui interrogent la base RDF du corpus (une base RDF est une manière équivalente de désigner un graphe RDF).

$\mathcal{Q}$ = « Sélectionner les lettres envoyées à Henri Poincaré par des astronomes en 1901 »

est un exemple de requête écrite de manière informelle. On présentera plus loin l'écriture en SPARQL d'une telle requête. Ce type de requêtes sur la base de la correspondance d'Henri Poincaré peut être utile par exemple pour des historiens dans le cadre de leurs recherches. Cependant, ces requêtes sont « exactes », dans le sens où le résultat de la requête ne fournira que des données qui correspondent exactement aux conditions énoncées. Dans le cas où une requête ne retourne rien, il serait utile de pouvoir proposer à l'utilisateur des requêtes proches mais légèrement différentes, qui lui fourniraient donc des résultats approchés vis-à-vis de sa requête initiale. Par exemple, si $\mathcal{Q}$ ne retourne aucun résultat, on pourrait réduire nos exigences et rechercher des lettres envoyées par tout type de scientifique, et pas seulement des astronomes. On pourrait aussi inverser dans la requête expéditeur et destinataire. Avec ces deux propositions, on obtiendrait les requêtes suivantes :

$\mathcal{Q}_1$ = « Sélectionner les lettres envoyées à Henri Poincaré par des scientifiques en 1901 »

$\mathcal{Q}_2$ = « Sélectionner les lettres envoyées à des astronomes par Henri Poincaré en 1901 »

Le langage SQTRL (*SPARQL Query Transformation Rule Language*), présenté dans [1], permet de réaliser de telles modifications de requêtes en définissant formellement des règles de transformation. Par exemple, pour obtenir $\mathcal{Q}_2$, on a utilisé une règle qu'on pourrait appeler « échange de l'expéditeur et du destinataire ».

Le langage SQTRL associe à chaque règle un coût : celui-ci détermine à quel point la requête obtenue après application de la règle s'éloigne de la requête initiale. On peut imaginer utiliser un coût variable pour une même règle, mais dans le cadre de ce stage on considérera que chaque règle a un coût constant. Le coût permet notamment de diviser les règles de transformation en deux ensembles : les règles à coût nul et les règles à coûts strictement positifs. Les premières consistent en de simples réécritures en requêtes équivalentes, tandis que les secondes modifient réellement le sens de la requête, comme dans le cas des

exemples précédents.

Le sujet du stage consiste à étudier les propriétés que peuvent posséder des ensembles de règles de transformation. La première piste était de s'intéresser à la propriété de confluence dans le cas des règles à coût positif, et d'éventuellement définir une propriété de confluence avec coût pour intégrer la spécificité du langage SQTRL. La deuxième piste était de chercher à définir des règles à coût nul qui soient des règles de normalisation. Ainsi, si deux requêtes $\mathcal{Q}_1$ et $\mathcal{Q}_2$ sont équivalentes, au sens où sur toute base de données, les exécutions de $\mathcal{Q}_1$ et $\mathcal{Q}_2$ renvoient le même résultat, on voudrait définir une ou plusieurs règles à coût nul qui, appliquées à ces deux requêtes, permettraient d'aboutir à la même requête.

Ainsi, ce rapport commence par présenter et définir plus formellement les outils du Web sémantique que sont RDF et SPARQL, avant d'introduire les systèmes de réécriture abstraits et de définir le langage SQTRL dans ce cadre des systèmes de réécriture. Par la suite sont présentées les recherches menées et les pistes explorées autour des règles à coût positif et de la notion de confluence. En particulier, quelques idées sont apportées pour réduire la complexité d'une recherche exhaustive des requêtes atteignables à partir d'une requête initiale et d'un ensemble de règles SQTRL. Cette partie sur les règles à coût positif est centrée autour du langage SQTRL, elle s'inscrit donc dans la continuité des articles [1] et [2] qui présentent et définissent SQTRL et ses applications au corpus de la correspondance d'Henri Poincaré. Enfin, quelques propositions et conjectures sur la normalisation de requêtes à l'aide de règles à coût nul sont énoncées. Le langage SQTRL n'a pas été utile dans cette partie, et les propositions énoncées couvrent des cas très réduits de requêtes SPARQL (mais ce sont surtout ces requêtes simplifiées qui sont, pour l'instant, prises en charge par SQTRL). Pour un processus plus exhaustif de mise sous forme canonique des requêtes SPARQL, voir par exemple [3].

2 Présentation et définitions

2.1 RDF et SPARQL

2.1.1 Resource Description Framework

D'après [4], « le *Resource Description Framework* (RDF) est un cadre [*framework*] pour exprimer des informations à propos de ressources. Les ressources peuvent être n'importe quoi, notamment des documents, des personnes, des objets physiques et des concepts abstraits. »

En RDF, toute information prend la forme d'une relation entre deux ressources. Elle comporte trois éléments : un sujet, un prédicat et un objet, et on l'appelle donc un triplet. Le prédicat représente une relation qui va du sujet vers l'objet. Par exemple, le triplet `<Henri Poincaré> <type> <Mathématicien>` traduit la relation « type » entre Henri Poincaré et la désignation « mathématicien », et signifie donc « Henri Poincaré est un mathématicien ».

Un ensemble de triplets RDF peut être représenté sous forme d'un graphe orienté, où les sujets et les objets constituent des sommets, et les prédicats des arcs. C'est pourquoi on appelle « graphe RDF » un ensemble de triplets RDF.

Les données qui apparaissent dans les graphes RDF peuvent être de trois types : IRI (*International Resource Identifier*), littéraux ou nœuds anonymes (ou nœuds blancs, ou nœuds vides, *blank nodes* en anglais).

— Les IRI sont utilisés pour désigner des ressources (qui peuvent être, comme dit plus haut, des documents, des personnes, des objets physiques, des concepts abstraits, etc.). Par exemple, les URL qui identifient les sites Web sont un cas particulier d'IRI. Le Web sémantique, justement, permet de faire des liens entre autre chose que les documents que sont les sites Web, grâce aux IRI. Au sein d'un triplet, un IRI peut occuper chacune des positions, c'est-à-dire qu'il peut être un sujet, un prédicat ou un objet.

— Les littéraux sont aussi des ressources, mais ils n'ont pas d'identifiant car ils désignent des valeurs basiques, comme des nombres, des dates ou des chaînes de caractères. Ils sont constitués d'une chaîne de caractères contenant la donnée, ainsi que d'un type de donnée qui permet de l'interpréter correctement. Au sein d'un triplet, ils ne peuvent occuper que la position d'objet.
— Les nœuds anonymes permettent de désigner des ressources qui n'ont pas d'identifiant. Ce sont des sortes de variables quantifiées existentiellement, qui ne désignent rien de particulier en elles-mêmes et qui ne sont définies que par les relations auxquelles elles appartiennent. Dans ce rapport, on les note avec le préfixe `?`. Par exemple, supposons qu'Henri Poincaré a écrit à une personne dont on ne connaît que le nom, on aura les triplets `<Henri Poincaré> <a écrit à> ?x` et `?x <se nomme> "Prénom Nom"`, où `?x` est un nœud anonyme. Au sein d'un triplet, les nœuds anonymes peuvent être sujets où objets.

2.1.2 SPARQL

SPARQL est un langage de requêtes pour RDF. Il existe quatre formes de requêtes SPARQL : `SELECT`, `CONSTRUCT`, `ASK` et `DESCRIBE`. Dans le cadre de ce stage, on ne s'intéressera qu'à la plus classique, `SELECT`. Une telle requête est typiquement composée de deux parties, la clause `SELECT` et la clause `WHERE`. Dans la première se trouvent les variables qui apparaîtront dans les résultats ; dans la seconde se trouve un motif de graphe RDF, c'est-à-dire un ensemble de triplets dans lequel apparaissent les variables de la clause `SELECT`. Les variables sont notées avec des points d'interrogation, comme les nœuds anonymes ; la différence est que les variables apparaissent après le `SELECT`, et pas les nœuds anonymes. Le résultat d'une telle requête sur une base de données est l'ensemble des appariements possibles des variables à des ressources tels que les triplets obtenus soient tous présents dans la base de données.

Par exemple, la requête informelle

$$\mathcal{Q} = \text{« Sélectionner les lettres envoyées à Henri Poincaré par des astronomes en 1901 »}$$

s'écrit, en SPARQL :

```
Q =
SELECT ?l
WHERE {
    ?l envoyéePar ?x .
    ?x type Astronome .
    ?l envoyéeÀ HenriPoincaré .
    ?l année 1901
}
```

Ici, `?x` est un nœud anonyme, qui désigne une hypothétique personne qui aurait envoyé une lettre à Henri Poincaré et serait un astronome. En revanche, `?l` est une variable, c'est sa valeur qui nous intéresse : on veut trouver dans la base de données des lettres vérifiant les relations fournies. Par exemple, supposons qu'il y ait dans la base de données les triplets suivants (parmi d'autres) :

```
lettre123 envoyéePar AugusteLebeuf
AugusteLebeuf type Astronome
lettre123 envoyéeÀ HenriPoincaré
lettre123 année 1901
```

En assignant, dans le motif de graphe de `Q`, la valeur `lettre123` à `?l` et `AugusteLebeuf` à `?x`, on obtient bien un ensemble de triplets qui est inclus dans la base de données. Puisque la seule variable de la requête est `?l`, le couple (`?l`, `lettre123`) apparaîtra dans le résultat. Quant au nœud anonyme `?x`, il faut lui assigner une ressource mais celle-ci n'apparaîtra pas dans le résultat, puisque ce n'est pas une variable. Bien sûr s'il existe d'autres couples de ressources qu'on peut assigner à `?l` et `?x` tels que les triplets obtenus

soient dans la base de données, les ressources correspondantes associées à `?l` apparaîtront aussi dans le résultat.

Plus précisément, chaque appariement de `?l` apparaîtra dans le résultat autant de fois qu'il y a d'appariements de `?x` qui fonctionnent avec ce même appariement de `?l`. Cependant, il existe un paramètre optionnel, `DISTINCT`, qui permet de ne faire apparaître qu'une seule fois chaque appariement possible de `?l`, même si plusieurs appariements de `?x` lui correspondent.

Remarque. *En réalité, les ressources comme* `HenriPoincaré` *ou* `envoyéePar` *apparaîtraient sous des IRI plus complexes, mais on simplifie ici l'écriture des triplets.*

2.2 Définitions

Les définitions qui suivent sont tirées de [5] et [6]. Certaines sont simplifiées ou adaptées afin de s'inscrire dans le cadre du stage.

Définition 1 (Terme RDF)**.** *Soit $\mathcal{I}$ l'ensemble de tous les IRI, $\mathcal{L}$ l'ensemble de tous les littéraux, et $\mathcal{A}$ un ensemble dénombrable disjoint de $\mathcal{I}$ et de $\mathcal{L}$, qui forme l'ensemble des nœuds anonymes. Un terme RDF est un élément de $\mathcal{I} \cup \mathcal{L} \cup \mathcal{A}$. On note $\mathcal{T}$ l'ensemble des termes RDF.*

Définition 2 (Triplet RDF)**.** *Un triplet RDF est un élément de $(\mathcal{I} \cup \mathcal{A}) \times \mathcal{I} \times (\mathcal{I} \cup \mathcal{L} \cup \mathcal{A})$.*

Définition 3 (Graphe RDF)**.** *Un graphe RDF est un ensemble de triplets RDF.*

Définition 4 (Variable de requête)**.** *Soit $\mathcal{V}$ un ensemble dénombrable disjoint de $\mathcal{T}$. Une variable de requête est un élément de $\mathcal{V}$.*

Définition 5 (Motif de triplet)**.** *Un motif de triplet est un élément de $(\mathcal{T} \cup \mathcal{V}) \times (\mathcal{I} \cup \mathcal{V}) \times (\mathcal{T} \cup \mathcal{V})$.*

Définition 6 (Motif de graphe)**.** *Un motif de graphe est un ensemble de motifs de triplets.*

Définition 7 (Requête)**.** *Une requête est un triplet (t, V, C), où t, le type de la requête, est un élément de l'ensemble $\{\text{SELECT}, \text{SELECT DISTINCT}\}$, V est un sous-ensemble non vide de $\mathcal{V}$ et C, le corps de la requête, est un motif de graphe dans lequel apparaît au moins une fois chacune des variables de V.*

Définition 8 (Fonction d'instanciation RDF)**.** *Une fonction d'instanciation RDF est une fonction partielle $\sigma : \mathcal{A} \rightarrow \mathcal{T}$. Par extension, si t est un triplet RDF (resp. motif de triplet), on note $\sigma(t)$ le triplet (resp. motif de triplet) obtenu en remplaçant dans t les nœuds anonymes n pour lesquels σ est définie par $\sigma(n)$. Si G est un graphe RDF (resp. motif de graphe), on note $\sigma(G) = \bigcup_{t \in G} \sigma(t)$.*

Définition 9 (Fonction solution)**.** *Une fonction solution est une fonction partielle $\mu : \mathcal{V} \rightarrow \mathcal{T}$. Par extension, si t est un motif de triplet, on note $\sigma(t)$ le motif de triplet obtenu en remplaçant dans t les variables v pour lesquelles μ est définie par $\mu(v)$. Si G est un motif de graphe, on note $\mu(G) = \bigcup_{t \in G} \mu(t)$.*

Définition 10 (Fonction d'instanciation de motif)**.** *Une fonction d'instanciation de motif M est la composée d'une fonction d'instanciation RDF σ et d'une fonction solution μ. En particulier, si G est un motif de graphe, $M(G) = \mu(\sigma(G))$.*

Définition 11 (Solution à un motif de graphe)**.** *Soit G un motif de graphe et $\mathcal{B}$ un graphe RDF. Une fonction $\mu : \mathcal{V} \rightarrow \mathcal{T}$ est une solution pour G sur $\mathcal{B}$ s'il existe une fonction d'instanciation de motif M telle que $M(G)$ est un sous-graphe partiel de $\mathcal{B}$ et μ est la restriction de M à $\mathcal{V}$. On appelle multiplicité de μ le nombre de fonctions d'instanciation RDF distinctes σ telles que $\mu(\sigma(G))$ est un sous-graphe partiel de $\mathcal{B}$.*

Définition 12 (Résultat d'une requête)**.** *Soit une requête* $\mathtt{Q} = (t, V, C)$ *et une base* $\mathcal{B}$*.*

Si $t =$ SELECT*, le résultat de* $\mathtt{Q}$ *sur* $\mathcal{B}$ *est le multiensemble constitué des solutions pour* C *sur* $\mathcal{B}$*, chaque solution étant associée à sa multiplicité.*

Si $t =$ SELECT DISTINCT*, le résultat de* $\mathtt{Q}$ *sur* $\mathcal{B}$ *est l'ensemble constitué des solutions pour* C *sur* $\mathcal{B}$*.*

On note $\mathtt{exec}(\mathtt{Q}, \mathcal{B})$ *le résultat de* $\mathtt{Q}$ *sur* $\mathcal{B}$*.*

Définition 13 (Égalité de requêtes)**.** *Deux requêtes* $\mathtt{Q} = (t, V, C)$ *et* $\mathtt{Q}' = (t', V', C')$ *sont égales si* $t = t'$*,* $V = V'$ *et, à renommage près des nœuds anonymes (mais pas des variables),* $C = C'$*. On note alors* $\mathtt{Q} = \mathtt{Q}'$*.*

Notation. *Étant donné une requête* $Q = (t, V, C)$*, on note* $V(Q) = V$*,* $C(Q) = C$*,* $T(Q)$ *l'ensemble des termes de* $C(Q)$ *et* $A(Q)$ *l'ensemble des nœuds anonymes de* $C(Q)$*.*

2.3 RDFS

Définition 14 (RDFS)**.** *Le langage RDFS (RDF Schema) est une extension sémantique de RDF, qui permet d'exprimer des ontologies simples. On considérera ici cinq propriétés issues du langage RDFS :*

- `rdf:type`*, abrégé en* `a` *:* $\langle s \ \ \mathtt{a} \ \ C \rangle$ *indique que* s *est une instance de la classe* C *(ou « s est de type* C *») ;*
- `rdfs:subClassOf`*, abrégé en* `subc` *:* $\langle C \ \ \mathtt{subc} \ \ D \rangle$ *indique que* C *est une sous-classe de* D*, c'est-à-dire que toute instance de* C *est aussi une instance de* D *;*
- `rdfs:subPropertyOf`*, abrégé en* `subp` *:* $\langle p \ \ \mathtt{subp} \ \ q \rangle$ *indique que* p *est une sous-propriété de* q*, c'est-à-dire que tout couple* (s, o) *vérifiant la propriété* p *vérifie aussi la propriété* q *;*
- `rdfs:domain`*, abrégé en* `domain` *:* $\langle p \ \ \mathtt{domain} \ \ C \rangle$ *indique que pour tout couple* (s, o) *vérifiant la propriété* p*,* s *est de type* C *;*
- `rdfs:range`*, abrégé en* `range` *:* $\langle p \ \ \mathtt{range} \ \ C \rangle$ *indique que pour tout couple* (s, o) *vérifiant la propriété* p*,* o *est de type* C*.*

Définition 15 (Clôture RDFS)**.** *Soit* $\mathcal{G}$ *un graphe RDF, on appelle clôture RDFS de* $\mathcal{G}$*, et on note* $\mathcal{G}^{\vdash}$*, la clôture déductive de* $\mathcal{G}$ *par les règles d'inférence suivantes :*

$$\frac{\langle x \ \ \mathtt{a} \ \ C \rangle \ \ \langle C \ \ \mathtt{subc} \ \ D \rangle}{\langle x \ \ \mathtt{a} \ \ D \rangle} r_1 \qquad \frac{\langle C \ \ \mathtt{subc} \ \ D \rangle \ \ \langle D \ \ \mathtt{subc} \ \ E \rangle}{\langle C \ \ \mathtt{subc} \ \ E \rangle} r_2$$

$$\frac{\langle x \ \ p \ \ y \rangle \ \ \langle p \ \ \mathtt{subp} \ \ q \rangle}{\langle x \ \ q \ \ y \rangle} r_3 \qquad \frac{\langle p \ \ \mathtt{subp} \ \ q \rangle \ \ \langle q \ \ \mathtt{subp} \ \ r \rangle}{\langle p \ \ \mathtt{subp} \ \ r \rangle} r_4$$

$$\frac{\langle x \ \ p \ \ y \rangle \ \ \langle p \ \ \mathtt{domain} \ \ D \rangle}{\langle x \ \ \mathtt{a} \ \ D \rangle} r_5 \qquad \frac{\langle x \ \ p \ \ y \rangle \ \ \langle p \ \ \mathtt{range} \ \ R \rangle}{\langle y \ \ \mathtt{a} \ \ R \rangle} r_6$$

Si un triplet RDF t *peut être obtenu à partir de* $\mathcal{G}$ *à partir de ces règles d'inférences, on note* $\mathcal{G} \vdash t$ *ou bien* $t \in \mathcal{G}^{\vdash}$*.*

Définition 16 (Ontologie et données)**.** *On divise couramment l'ensemble des triplets d'un graphe RDF* $\mathcal{G}$ *en deux parties, l'ontologie* $\mathcal{O}$ *et les données* $\mathcal{D}$ *:* $\mathcal{G} = \mathcal{O} \cup \mathcal{D}$*. L'ontologie permet de poser le cadre général, le contexte dans lequel s'inscriront les bases de données. Elle définit pour cela des classes et des propriétés, tout en établissant des relations entre elles. Les triplets faisant partie de l'ontologie sont ici ceux qui utilisent les propriétés* `subc`*,* `subp`*,* `domain` *ou* `range`*.*

Pour désigner l'ensemble des classes ou propriétés liées ensemble par les propriétés `subc` ou `subp`, on peut parler de taxonomies.

Définition 17 (Taxonomie des classes)**.** *Soit* $\mathcal{O}$ *une ontologie, et* $\mathcal{C}$ *l'ensemble des classes intervenant dans cette ontologie. Alors la taxonomie des classes de* $\mathcal{O}$ *est le graphe d'ensemble de sommets* $\mathcal{C}$*, et tel qu'il y a un arc de* C *vers* D *si et seulement si* $\mathcal{O} \vdash \langle C \ \ \mathtt{subc} \ \ D \rangle$*.*

Définition 18 (Taxonomie des propriétés)**.** *Soit $\mathcal{O}$ une ontologie, et $\mathcal{P}$ l'ensemble des propriétés intervenant dans cette ontologie. Alors la taxonomie des propriétés de $\mathcal{O}$ est le graphe d'ensemble de sommets $\mathcal{P}$, et tel qu'il y a une arête de p vers q si et seulement si $\langle p \ \texttt{subp} \ q \rangle \in \mathcal{O}$.*

Enfin, le langage RDFS permet d'interroger des bases de données en prenant en compte les triplets qui peuvent être déduits par inférence à partir des triplets de la base. Cela nous permet notamment de définir l'équivalence de deux requêtes.

Définition 19 (Exécution RDFS d'une requête)**.** *L'exécution RDFS d'une requête consiste à interroger la clôture RDFS du graphe fourni. Étant donné une requête $\texttt{Q}$ et une base $\mathcal{B}$, on note donc $\texttt{exec}_{\vdash}(\texttt{Q}, \mathcal{B}) := \texttt{exec}(\texttt{Q}, \mathcal{B}^{\vdash})$.*

Définition 20 (Requête plus spécifique, plus générale)**.** *Soit deux requêtes $\texttt{Q} = (t, V, C)$ et $\texttt{Q}' = (t', V', C')$ telles que $t = t'$ et $V = V'$. Si pour toute base $\mathcal{B}$, $\texttt{exec}_{\vdash}(\texttt{Q}, \mathcal{B}) \subseteq \texttt{exec}_{\vdash}(\texttt{Q}', \mathcal{B})$, alors on dit que $\texttt{Q}$ est plus spécifique que $\texttt{Q}'$, ou encore que $\texttt{Q}'$ est plus générale que $\texttt{Q}$. On note $\texttt{Q} \sqsubseteq \texttt{Q}'$. $\sqsubseteq$ est un préordre sur l'ensemble des requêtes.*

Définition 21 (Équivalence de requêtes)**.** *Deux requêtes $\texttt{Q} = (t, V, C)$ et $\texttt{Q}' = (t', V', C')$ sont équivalentes si $\texttt{Q}$ est plus spécifique que $\texttt{Q}'$ et $\texttt{Q}'$ est plus spécifique que $\texttt{Q}$. Autrement dit, elles sont équivalentes si $t = t'$, $V = V'$ et pour toute base $\mathcal{B}$, $\texttt{exec}_{\vdash}(\texttt{Q}, \mathcal{B}) = \texttt{exec}_{\vdash}(\texttt{Q}', \mathcal{B})$. On note alors $\texttt{Q} \equiv \texttt{Q}'$. Naturellement, $\equiv$ est une relation d'équivalence.*

3 Réécriture de requêtes

3.1 Définitions et propriétés sur les systèmes de réécriture abstraite

Ces définitions sont issues du cours *Théorie de la programmation* donné par Daniel Hirschkoff en L3 d'informatique à l'ENS de Lyon (2020-2021).

Définition 22 (Système de réécriture abstraite)**.** *Un système de réécriture abstraite (SRA) est la donnée d'une relation binaire $\longrightarrow$ sur un ensemble E. On le note $(E, \longrightarrow)$, ou plus simplement $\longrightarrow$.*

Notation. *On désigne par $\longrightarrow^*$ la clôture réflexive et transitive de $\longrightarrow$.*

Définition 23 (Paire joignable)**.** *Une paire $\{a, b\}$ d'éléments de E est dite joignable s'il existe $c \in E$ tel que $a \longrightarrow^* c$ et $b \longrightarrow^* c$.*

Définition 24 (Terminaison)**.** *Un SRA $(E, \longrightarrow)$ termine s'il n'existe pas de suite infinie $(a_i)_{i \in \mathbb{N}}$ de E telle que pour tout $i \in \mathbb{N}$, $a_i \longrightarrow a_{i+1}$.*

Définition 25 (Confluence)**.** *Le SRA $(E, \longrightarrow)$ est confluent en a si pour tous $b, c \in E$, on a :*

$$\textit{si } a \longrightarrow^* b \textit{ et } a \longrightarrow^* c \textit{ alors } b \textit{ et } c \textit{ sont joignables.}$$

$(E, \longrightarrow)$ est confluent s'il est confluent en tout $a \in E$.

Définition 26 (Confluence locale)**.** *Le SRA $(E, \longrightarrow)$ est localement confluent si pour tous $a, b, c \in E$, on a :*

$$\textit{si } a \longrightarrow b \textit{ et } a \longrightarrow c \textit{ alors } b \textit{ et } c \textit{ sont joignables.}$$

Lemme (de Newman)**.** *Si $(E, \longrightarrow)$ termine et est localement confluent, alors $(E, \longrightarrow)$ est confluent.*

Définition 27 (Propriété du losange)**.** *La relation $\longrightarrow$ a la propriété du losange si pour tous $a, b, c \in E$, on a :*

$$\textit{si } a \longrightarrow b \textit{ et } a \longrightarrow c \textit{ alors il existe } d \in E \textit{ tel que } b \longrightarrow d \textit{ et } c \longrightarrow d$$

3.2 Le langage SQTRL

SQTRL (*SPARQL Query Transformation Rule Language*) est un langage conçu pour la transformation de requêtes SPARQL ([1], [2]). Le principe de la réécriture d'une requête est similaire à celui de la réécriture de termes, à ceci près que les règles SQTRL ont, en plus d'un membre gauche et d'un membre droit, un contexte, un coût et une explication. Le contexte est un ensemble de conditions à l'application de la règle, qui portent sur la base de données considérée.

Définition 28 (Règle SQTRL)**.** *Une règle SQTRL est un sextuplet (n, C, L, R, c, e), où C, L et R sont des ensembles de triplets RDF, c est un flottant positif, n et e sont des chaînes de caractères. n est le nom de la règle (et permet de l'identifier), C est le contexte de la règle, L est sa partie gauche, R sa partie droite, c est son coût et e est l'explication de la règle.*

Notation. *On écrit les règles de réécriture au format XML, sous la forme :*

```
<rule name="nom de la règle">
    <context>triplets RDF séparés par des points</context>
    <left>triplets RDF séparés par des points</left>
    <right>triplets RDF séparés par des points</right>
    <cost>coût (un flottant)</cost>
    <explanation>explication de la règle</explanation>
</rule>
```

Définition 29 (Système de réécriture de requêtes)**.** *Un système de réécriture de requêtes sur une base $\mathcal{B}$ est un ensemble $\mathcal{R}$ de règles de réécriture, qui définit une relation binaire $\longrightarrow_{\mathcal{R}}$ sur l'ensemble des requêtes SPARQL de la façon suivante : étant donné deux requêtes $\mathtt{Q}$ et $\mathtt{Q}'$, on a $\mathtt{Q} \longrightarrow_{\mathcal{R}} \mathtt{Q}'$ si et seulement si $V(\mathtt{Q}) = V(\mathtt{Q}')$ et il existe une règle $r = (n, C, L, R, c, e)$ et une fonction d'instanciation RDF σ telles que*

- *$\sigma(C) \subseteq \mathcal{B}$;*
- *$\sigma(L) \subseteq C(\mathtt{Q})$;*
- *$(C(\mathtt{Q}) \setminus \sigma(L)) \cup \sigma(R) = C(\mathtt{Q}')$;*
- *les nœuds anonymes qui apparaissent dans R, mais pas dans L ni dans C n'apparaissent pas dans le corps de $\mathtt{Q}$, et σ vaut l'identité dessus.*

On est ainsi ramenés à un SRA $(\mathfrak{Q}, \longrightarrow_{\mathcal{R}})$, où $\mathfrak{Q}$ est l'ensemble des requêtes SPARQL.

Définition 30 (Application d'une règle)**.** *Étant donné une règle r, une base $\mathcal{B}$ et une requête $\mathtt{Q}$, on note $\mathtt{apply}(r, \mathtt{Q}, \mathcal{B})$ l'ensemble des requêtes pouvant être obtenues en appliquant r à $\mathtt{Q}$ sur la clôture RDFS de la base $\mathcal{B}$ (c'est-à-dire que c'est la clôture RDFS de $\mathcal{B}$ qui est prise en compte pour le contexte de la règle). On dit que r est applicable à $\mathtt{Q}$ si $\mathtt{apply}(r, \mathtt{Q}, \mathcal{B})$ n'est pas vide. Voir [1] pour la description d'un algorithme de calcul de* `apply`*.*

Étant donné un ensemble de règles $\mathcal{R}$, une autre manière de formuler la définition de $\longrightarrow_{\mathcal{R}}$ est alors : $\mathtt{Q} \longrightarrow_{\mathcal{R}} \mathtt{Q}'$ si et seulement si $\mathtt{Q}' \in \bigcup_{r \in \mathcal{R}} \mathtt{apply}(r, \mathtt{Q}, \mathcal{B})$.

Définition 31 (Coût d'une transformation)**.** *Étant donné un ensemble de règles $\mathcal{R}$, si $\mathtt{Q} \longrightarrow_{\mathcal{R}} \mathtt{Q}'$, alors le coût de la transformation de $\mathtt{Q}$ en $\mathtt{Q}'$ est le coût de la règle qui a été utilisée pour cette transformation.*

De plus, le coût est additif : si on a $\mathtt{Q}_1 \longrightarrow_{\mathcal{R}} \mathtt{Q}_2 \longrightarrow_{\mathcal{R}} \ldots \longrightarrow_{\mathcal{R}} \mathtt{Q}_k$, le coût du chemin $(\mathtt{Q}_1, \mathtt{Q}_2, \ldots, \mathtt{Q}_k)$ est la somme des coûts successifs des règles utilisées.

Dans le cas où l'on a $\mathtt{Q} \longrightarrow^*_{\mathcal{R}} \mathtt{Q}'$ sans information supplémentaire, le coût de la transformation est le coût minimal des chemins possibles pour aller de $\mathtt{Q}$ à $\mathtt{Q}'$.

Intuitivement, plus le coût d'une transformation est grand, plus la requête obtenue est « éloignée » de la requête initiale.

Notation. *On a déjà vu qu'on pouvait noter $\mathtt{Q} \longrightarrow_{\mathcal{R}} \mathtt{Q}'$ si $\mathtt{Q}'$ est obtenue à partir de $\mathtt{Q}$ par l'application d'une règle de l'ensemble $\mathcal{R}$. Si $\mathcal{R}$ contient une unique règle r, on peut noter $\longrightarrow_r$ à la place de $\longrightarrow_{\{r\}}$.*

Si on veut préciser le coût c de la transformation, on peut noter $\mathtt{Q} \xrightarrow{c}_{\mathcal{R}} \mathtt{Q}'$ *(ou* $\mathtt{Q} \xrightarrow{c}_{r} \mathtt{Q}'$*). Pour une transformation à plusieurs étapes dont on veut préciser le coût, on peut écrire* $\mathtt{Q} \xrightarrow{c}{}^{*}_{\mathcal{R}} \mathtt{Q}'$*, ou* $\mathtt{Q} \xrightarrow{c}{}^{*} \mathtt{Q}'$ *si l'ensemble de règles est implicite.*

4 Réécriture à coût non nul : confluence de requêtes et élagage de l'arbre d'exploration

4.1 Exemples de règles SQTRL

On a donné précédemment la définition d'une règle SQTRL. Cette partie est consacrée à l'étude de certaines propriétés caractérisant des ensembles de règles, on commence donc par fournir des exemples concrets de règles SQTRL. Tous sont issus de [1] et [2].

4.1.1 Règles de généralisation

Définition 32 (Règle de généralisation)**.** *On appelle règle de généralisation une règle SQTRL r telle que pour toute base* $\mathcal{B}$ *et toutes requêtes* $\mathtt{Q}$ *et* $\mathtt{Q}'$*, si* $\mathtt{Q}' \in \mathtt{apply}(r, \mathtt{Q}, \mathcal{B})$ *alors* $\mathtt{Q} \sqsubseteq \mathtt{Q}'$*.*

Exemples.

```
<rule name="Généralisation d'une classe en tant que sujet">
  <context>?C subc ?D</context>
  <left>?C ?p ?y</left>
  <right>?D ?p ?y</right>
  <cost>1.0</cost>
  <explanation>Généraliser ?C en ?D</explanation>
</rule>
```

```
<rule name="Généralisation d'une classe en tant qu'objet">
  <context>?C subc ?D</context>
  <left>?x ?p ?C</left>
  <right>?x ?p ?D</right>
  <cost>1.0</cost>
  <explanation>Généraliser ?C en ?D</explanation>
</rule>
```

```
<rule name="Généralisation d'une propriété en tant que prédicat">
  <context>?p subp ?q</context>
  <left>?x ?p ?y</left>
  <right>?x ?q ?y</right>
  <cost>1.0</cost>
  <explanation>Généraliser ?p en ?q</explanation>
</rule>
```

Remarque. *Ces trois règles soulèvent un problème : s'il y a dans une base les triplets* $\langle A \ \mathtt{subc} \ B\rangle$ *et* $\langle B \ \mathtt{subc} \ C\rangle$*, alors on aura aussi* $\langle A \ \mathtt{subc} \ C\rangle$ *dans la clôture RDFS de cette base. Puisque le coût d'une règle est constant et que les coûts s'additionnent, il serait alors plus coûteux de généraliser A en B puis B en C plutôt que directement A en C, ce qui n'est pas souhaitable. Pour pouvoir utiliser ces règles dans les*

faits, il faudrait que les taxonomies soient sous forme de diagramme de Hasse : s'il existe un arc de A à B et de B à C, il ne doit pas y avoir d'arc de A à C. Ainsi les généralisations de classe et de propriétés se feraient étape par étape, et les coûts pourraient s'additionner sans problème.

```
<rule name="Suppression d'un triplet">
  <context></context>
  <left>?x ?p ?y</left>
  <right></right>
  <cost>1.0</cost>
  <explanation>Supprime le triplet <?x ?p ?y></explanation>
</rule>
```

```
<rule name="Généralisation d'une ressource en nœud anonyme">
  <context></context>
  <left>?x ?p ?y</left>
  <right>?z ?p ?y</right>
  <cost>1.0</cost>
  <explanation>Généralise ?x en ?z</explanation>
</rule>
```

Remarque. *Comme précisé dans la définition du système de réécriture de requêtes, un nœud anonyme qui apparaît dans la partie droite mais pas dans la partie gauche ni dans le contexte doit être une variable neuve. Ainsi, si par exemple le* `?x` *est associé à une constante, cette règle permettra de remplacer cette constante par un simple nœud anonyme et d'obtenir un motif de triplet plus général.*

```
<rule name="Généralisation d'un sujet aux éléments de sa classe">
  <context>?s a ?C</context>
  <left>?s ?p ?o</left>
  <right>?x ?p ?o . ?x a ?C</right>
  <cost>1.0</cost>
  <explanation>Généraliser ?s en ?x de même classe</explanation>
</rule>
```

```
<rule name="Généralisation d'un prédicat aux éléments de sa classe">
  <context>?p a ?C</context>
  <left>?s ?p ?o</left>
  <right>?s ?q ?o . ?q a ?C</right>
  <cost>1.0</cost>
  <explanation>Généraliser ?p en ?q de même classe</explanation>
</rule>
```

```
<rule name="Généralisation d'un objet aux éléments de sa classe">
  <context>?o a ?C</context>
  <left>?s ?p ?o</left>
  <right>?s ?p ?x . ?x a ?C</right>
  <cost>1.0</cost>
  <explanation>Généraliser ?o en ?x de même classe</explanation>
</rule>
```

4.1.2 Autres règles à coût non nul

Les exemples précédents sont des règles qui peuvent être utilisées quelle que soit l'ontologie. En revanche, les règles qui ne sont pas des règles de généralisation sont souvent des règles qui ne s'appliquent que dans un contexte particulier. Dans le cadre du corpus d'Henri Poincaré, un exemple en est donné dans [1] :

```
<rule name="Échange de l'expéditeur et du destinataire">
  <context></context>
  <left>?l envoyéePar ?x . ?l envoyéeÀ ?y</left>
  <right>?l envoyéePar ?y . ?l envoyéeÀ ?x</right>
  <cost>1.0</cost>
  <explanation>Échange expéditeur/destinataire : ?x/?y</explanation>
</rule>
```

4.2 Étude de la terminaison et de la confluence sur un exemple

Afin d'entamer l'étude des propriétés que pouvaient présenter des ensembles de règles SQTRL, la première étape a été de déterminer ces propriétés sur un exemple. En l'occurrence, le système de règles étudié dans cette section est l'ensemble $\mathcal{R}_1 = \{g_{suj}, g_{pred}, g_{obj}\}$ où g_{suj} désigne la règle « généralisation d'une classe en tant que sujet », g_{obj} la règle « généralisation d'une classe en tant qu'objet » et g_{pred} la règle « généralisation d'une propriété en tant que prédicat ».

On considérera uniquement des ontologies dont la taxonomie des classes et la taxonomie des propriétés sont sans circuit. Cela se justifie par le fait que s'il existe, par exemple, un circuit de classes, alors toutes ces classes sont équivalentes (au sens où leurs ensembles d'instances sont égaux), et on peut supprimer le circuit et se ramener à une unique classe (et c'est la même chose pour les propriétés).

Proposition 1. *La relation $\longrightarrow_{\mathcal{R}_1}$ termine sur toute ontologie $\mathcal{O}$ dont les taxonomies sont sans circuit.*

Proposition 2. *Sur une ontologie $\mathcal{O}$, $\longrightarrow_{\mathcal{R}_1}$ est confluente si et seulement si les conditions suivantes sont vérifiées :*

1. *pour toute classe C, si C_1 et C_2 sont deux superclasses de C, alors il existe une classe C_3 qui est une superclasse de C_1 et de C_2 ;*
2. *pour toute propriété p, si p_1 et p_2 sont deux superpropriétés de p, alors il existe une propriété p_3 qui est une superpropriété de p_1 et de p_2.*

Les preuves de ces propositions sont données en annexe.

Dans le cadre de la recherche de requêtes approchées par réécriture SQTRL, il est difficile de trouver une application pratique à cette propriété de confluence pour un ensemble de règles à coûts non nuls. Ainsi, il n'a pas été mené de généralisation de ces propositions à d'autres ensembles de règles. Cependant, la notion de confluence a été un point de départ pour une réflexion autour de l'élagage de l'arbre d'exploration d'une requête. Ce processus, présenté dans la section suivante, a des applications pratiques, notamment l'amélioration de la complexité des algorithmes utilisant le langage SQTRL pour la recherche de requêtes approchées.

4.3 Élagage de l'arbre d'exploration

4.3.1 Concept et exemple

Étant donné une requête `Q` et une base $\mathcal{B}$, on aimerait pouvoir accéder avec la meilleure complexité possible aux requêtes qu'il est possible d'obtenir par transformations SQTRL à partir de `Q`, ainsi qu'aux

coûts minimaux de ces transformations. Par exemple, il est utile de pouvoir récupérer l'ensemble des requêtes qui peuvent être obtenues avec un coût inférieur à un coût seuil fixé.

Pour réaliser ce genre de calculs, il faut partir de `Q`, appliquer toutes les règles possibles à tous les endroits possibles du corps `Q`, ce qui donne un ensemble de requêtes obtenues en une étape, puis recommencer sur ces nouvelles requêtes pour obtenir les requêtes nécessitant deux étapes de transformation, et ainsi de suite. On construit ainsi un arbre d'exploration à partir de `Q` (qui peut être en théorie un arbre infini).

Par exemple, supposons qu'on parte de la requête initiale

$Q =$
```
SELECT ?x
WHERE {
    ?l envoyéePar ?x .
    ?x type Astronome .
    ?l envoyéeÀ HenriPoincaré
}
```

et de l'ensemble de règles constitué des deux règles « généralisation d'une classe en tant qu'objet » (g_{obj}) et « échange de l'expéditeur et du destinataire » ($r_{echange}$). On suppose que dans l'ontologie, `Astronome` est une sous-classe de `Scientifique` qui est une sous-classe de `ÊtreHumain`. Après une étape de transformation, on obtient les requêtes suivantes :

$Q_1 =$
```
SELECT ?x
WHERE {
    ?l envoyéePar ?x .
    ?x type Scientifique .
    ?l envoyéeÀ HenriPoincaré
}
```

$Q_2 =$
```
SELECT ?x
WHERE {
    ?l envoyéePar HenriPoincaré .
    ?x type Astronome .
    ?l envoyéeÀ ?x
}
```

Après deux étapes de transformation, on obtient les requêtes suivantes :

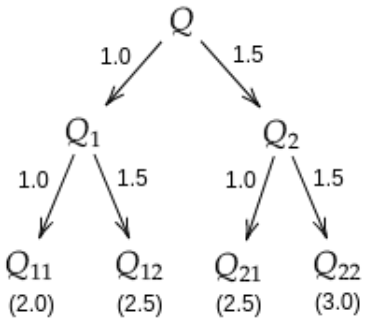
$Q_{11} =$
```
SELECT ?x
WHERE {
    ?l envoyéePar ?x .
    ?x type ÊtreHumain .
    ?l envoyéeÀ HenriPoincaré
}
```

$Q_{12} =$
```
SELECT ?x
WHERE {
    ?l envoyéePar HenriPoincaré .
    ?x type Scientifique .
    ?l envoyéeÀ ?x
}
```

$Q_{21} =$
```
SELECT ?x
WHERE {
    ?l envoyéePar HenriPoincaré .
    ?x type Scientifique .
    ?l envoyéeÀ ?x
}
```

$Q_{22} =$
```
SELECT ?x
WHERE {
    ?l envoyéePar ?x .
    ?x type Astronome .
    ?l envoyéeÀ HenriPoincaré
}
```

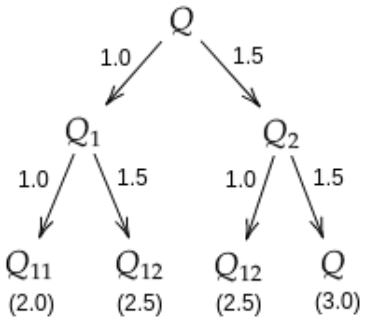
Supposons qu'on attribue à g_{obj} un coût de 1.0 et à $r_{echange}$ un coût de 1.5. L'arbre d'exploration après 2 étapes est alors le suivant :



Cependant, on peut observer au moins deux potentiels problèmes lors d'un tel processus d'exploration :

1. il se peut que la même requête apparaisse à plusieurs reprises dans l'arbre, et que les chemins qui y conduisent aient des coûts différents. Alors, récupérer une requête dans l'arbre et le coût associé ne suffirait pas à connaître le coût minimal pour l'obtenir ; il faudrait vérifier dans tout l'arbre que cette requête n'est obtenue nulle part avec un coût plus faible ;
2. surtout, le fait d'avoir plusieurs occurrences de requêtes identiques augmente inutilement la complexité de calcul et de parcours de l'arbre. En effet, on a non seulement plusieurs fois la même requête, mais en plus, de chacune de ces occurrences découle le même sous-arbre, alors qu'il est inutile de le calculer plusieurs fois.

Dans l'exemple ci-dessus, on remarque que les requêtes `Q` et $\texttt{Q}_{22}$ sont égales, tout comme les requêtes $\texttt{Q}_{12}$ et $\texttt{Q}_{21}$. L'arbre peut donc être présenté ainsi :



Les deux problèmes évoqués apparaissent :

1. la requête `Q` apparaît à la fois au terme d'un chemin de coût 0 (i.e. à la racine) et au terme d'un chemin de coût 3 ;
2. la requête $\texttt{Q}_{12}$ apparaît certes deux fois avec le même coût, mais calculer ces deux occurrences au cours de l'exploration est inutile.

Il faudrait donc, si la même requête apparaît à plusieurs endroits différents dans l'arbre, couper toutes ces branches, sauf une. Plus précisément, il ne faudrait garder que l'une des branches de coût minimal. De plus, le simple fait de supprimer un certain nombre de ces branches pourrait permettre d'améliorer grandement la complexité, même si on ne se ramène pas à une unique occurrence de chaque requête.

On peut formuler les choses ainsi : étant donné un ensemble de règles $\mathcal{R}$, on aimerait définir $\longrightarrow_{\mathcal{R},bis}$ une sous-relation de $\longrightarrow_{\mathcal{R}}$ telle que

1. pour toutes requêtes $\mathtt{Q}$ et $\mathtt{Q}'$, si $\mathtt{Q} \longrightarrow^*_{\mathcal{R}} \mathtt{Q}'$ alors $\mathtt{Q} \longrightarrow^*_{\mathcal{R},bis} \mathtt{Q}'$, avec le même coût minimal ;
2. $\longrightarrow_{\mathcal{R},bis}$ est « le moins possible confluente », c'est-à-dire qu'on cherche à réduire le nombre de $(\mathtt{Q}, \mathtt{Q}_1, \mathtt{Q}_2)$ tels que $\mathtt{Q} \longrightarrow^*_{\mathcal{R},bis} \mathtt{Q}_1$, $\mathtt{Q} \longrightarrow^*_{\mathcal{R},bis} \mathtt{Q}_2$ et $\mathtt{Q}_1$ et $\mathtt{Q}_2$ sont joignables en une requête $\mathtt{Q}'$.

En effet, si ce dernier cas se présente, alors on aura dans l'arbre d'exploration deux occurrences de $\mathtt{Q}'$ (si $\mathtt{Q}_1$ n'est pas sur le chemin de $\mathtt{Q}$ à $\mathtt{Q}_2$ ni $\mathtt{Q}_2$ sur le chemin de $\mathtt{Q}$ à $\mathtt{Q}_1$), ce qui n'est pas souhaitable.

Une idée est de procéder « en aval » : si $\mathtt{Q} \longrightarrow^*_{\mathcal{R}} \mathtt{Q}_1 \longrightarrow^*_{\mathcal{R}} \mathtt{Q}'$ et $\mathtt{Q} \longrightarrow^*_{\mathcal{R}} \mathtt{Q}_2 \longrightarrow^*_{\mathcal{R}} \mathtt{Q}'$, alors on a $\mathtt{Q}'_1$ et $\mathtt{Q}'_2$ telles que $\mathtt{Q} \longrightarrow^*_{\mathcal{R}} \mathtt{Q}'_1 \longrightarrow_{\mathcal{R}} \mathtt{Q}'$ et $\mathtt{Q} \longrightarrow^*_{\mathcal{R}} \mathtt{Q}'_2 \longrightarrow_{\mathcal{R}} \mathtt{Q}'$. Pour $\longrightarrow_{\mathcal{R},bis}$, on ne voudrait garder qu'une seule des deux relations $\mathtt{Q}'_1 \longrightarrow_{\mathcal{R}} \mathtt{Q}'$ et $\mathtt{Q}'_2 \longrightarrow_{\mathcal{R}} \mathtt{Q}'$. Cependant, il paraît difficile de concevoir un tel procédé algorithmiquement, puisque cela impliquerait, au moment où l'on explore $\mathtt{Q}'_1$, de déterminer la localisation dans l'arbre d'exploration d'une éventuelle requête $\mathtt{Q}'_2$ telle que décrite précédemment. On aimerait au contraire pouvoir décider des branches à élaguer sans avoir à consulter d'autres requêtes de l'arbre d'exploration.

Il vaut donc mieux concevoir un procédé d'élagage « en amont ». L'idée est la suivante : on part d'une requête $\mathtt{Q}$, et on suppose qu'il existe une règle $r \in \mathcal{R}$ telle que si $\mathtt{Q} \overset{c}{\longrightarrow}^*_{\mathcal{R}} \mathtt{Q}'$ avec une utilisation de la règle r, il existe un chemin de même coût de $\mathtt{Q}$ à $\mathtt{Q}'$ où l'on commence par appliquer la règle r. Dans ce cas, tous les chemins contenant la règle r mais ne commençant pas par elle sont inutiles, puisqu'on peut s'y ramener **avec le même coût** par un chemin commençant par appliquer r. On peut alors élaguer ces règles inutiles dans l'arbre d'exploration, en stipulant que dès qu'une règle autre que r est utilisée, la règle r ne peut plus être utilisée dans le sous-arbre qui en découle. Algorithmiquement, il suffit de garder en mémoire, au cours de l'exploration, l'ensemble des règles encore utilisables associé à chaque nœud de l'arbre.

Le but de ce qui suit est donc de déterminer sous quelles conditions une telle hypothèse peut être vérifiée, ainsi que le gain de complexité qu'un tel élagage apporterait.

4.3.2 Propriétés et conditions

Étant donné une requête $\mathtt{Q}$ et une règle r, r peut parfois être appliquée de plusieurs manières à $\mathtt{Q}$, et les requêtes obtenues peuvent être différentes. Pour définir formellement une application précise de r à $\mathtt{Q}$, on introduit donc la notion d'application de règle.

Définition 33 (Application de règle)**.** *Soit $\mathcal{B}$ une base. Une application de règle sur $\mathcal{B}$ est un triplet $(r, \mathtt{Q}, \mathtt{Q}')$ où r est une règle et $\mathtt{Q}$ et $\mathtt{Q}'$ sont des requêtes, et telles que $\mathtt{Q}' \in \mathtt{apply}(r, \mathtt{Q}, \mathcal{B})$. Autrement dit, $(r, \mathtt{Q}, \mathtt{Q}')$ est une application de règle si on peut écrire $\mathtt{Q} \longrightarrow_r \mathtt{Q}'$.*

Définition 34 (Applications de règle consécutives)**.** *Deux applications de règle $a_1 = (r_1, \mathtt{Q}_1, \mathtt{Q}'_1)$ et $a_2 = (r_2, \mathtt{Q}_2, \mathtt{Q}'_2)$ sur une même base $\mathcal{B}$ sont dites consécutives si $\mathtt{Q}'_1 = \mathtt{Q}_2$.*

On définit ensuite quelques propriétés sur les applications de règle consécutives, qui ont pour objectif d'être des conditions sous lesquelles on pourra supprimer des branches dans un arbre de transformation en fixant un ordre sur les règles.

Définition 35 (Propriétés des applications de règle consécutives)**.** *Soit $\mathcal{B}$ une base, $a_1 = (r_1, \mathtt{Q}, \mathtt{Q}_1)$ et $a_2 = (r_2, \mathtt{Q}_1, \mathtt{Q}')$ deux applications de règle consécutives (i.e. $\mathtt{Q} \longrightarrow_{r_1} \mathtt{Q}_1 \longrightarrow_{r_2} \mathtt{Q}'$). On dit que :*
- *a_2 est échangeable avec a_1 s'il existe une requête $\mathtt{Q}_2$ telle que $\mathtt{Q} \longrightarrow_{r_2} \mathtt{Q}_2 \longrightarrow_{r_1} \mathtt{Q}'$;*
- *a_2 absorbe a_1 si $\mathtt{Q}' \in \mathtt{apply}(r_2, \mathtt{Q}, \mathcal{B})$.*

Définition 36 (Règles commutatives)**.** *Deux règles r_1 et r_2 sont commutatives si pour toutes requêtes $\mathtt{Q}, \mathtt{Q}_1, \mathtt{Q}'$, les applications de règle $(r_1, \mathtt{Q}, \mathtt{Q}_1)$ et $(r_2, \mathtt{Q}_1, \mathtt{Q}')$ sont échangeables l'une avec l'autre.*

Proposition 3. *Soit $\mathcal{R}$ un ensemble de règles deux à deux commutatives et $>$ un ordre total strict arbitraire sur $\mathcal{R}$. Alors pour toutes requêtes $\mathtt{Q}, \mathtt{Q}'$ telles que $\mathtt{Q} \overset{c}{\longrightarrow}^*_{\mathcal{R}} \mathtt{Q}'$, il existe un chemin de $\mathtt{Q}$ à $\mathtt{Q}'$ de coût c et tel que la suite de règles utilisées est décroissante pour $>$.*

Principe de la preuve. On part d'un chemin de $\mathtt{Q}$ à $\mathtt{Q}'$ de coût minimal (i.e. de coût c). On échange petit à petit les règles adjacentes ne respectant pas la décroissance, à la manière d'un tri à bulles par exemple, jusqu'à obtenir une suite de règles décroissante. La commutativité mutuelle de règles assure que cela est possible, et que le coût est conservé. □

Exemple. Les règles g_{suj}, g_{pred} et g_{obj} sont deux à deux commutatives. Pour explorer toutes les possibilités de transformation à partir d'une requête $\mathtt{Q}$, on peut donc fixer un ordre arbitraire sur les règles (disons $g_{suj} > g_{pred} > g_{obj}$) et ne prendre en compte que les chemins pour lesquels la suite de règles décroît.

Définition 37 (Règle échangeable ou absorbante)**.** *Une règle r_2 est dite « échangeable ou absorbante » par rapport à r_1 si pour toutes requêtes $\mathtt{Q}, \mathtt{Q}_1, \mathtt{Q}'$, soit $(r_2, \mathtt{Q}_1, \mathtt{Q}')$ est échangeable avec $(r_1, \mathtt{Q}, \mathtt{Q}_1)$, soit $(r_2, \mathtt{Q}_1, \mathtt{Q}')$ absorbe $(r_1, \mathtt{Q}, \mathtt{Q}_1)$.*

Proposition 4. *Soit $\mathcal{R}$ un ensemble fini de règles $\{r_1, \ldots, r_k\}$, sur lesquelles on fixe l'ordre $r_k > \ldots > r_1$, et telles que pour $1 \leqslant i < j \leqslant k$, r_j est échangeable ou absorbante par rapport à r_i. Alors pour toutes requêtes $\mathtt{Q}, \mathtt{Q}'$ telles que $\mathtt{Q} \overset{c}{\longrightarrow}{}^*_{\mathcal{R}} \mathtt{Q}'$, il existe un chemin de $\mathtt{Q}$ à $\mathtt{Q}'$ de coût c et tel que la suite de règles utilisées est décroissante pour $>$.*

Principe de la preuve. On part d'un chemin de $\mathtt{Q}$ à $\mathtt{Q}'$ de coût minimal. Si deux applications de règle consécutives de ce chemin $a_1 = (r_i, \mathtt{Q}, \mathtt{Q}_1)$ et $a_2 = (r_j, \mathtt{Q}_1, \mathtt{Q}')$ sont telles que r_i et r_j sont telles que $r_i < r_j$, alors soit a_2 absorbe a_1, soit a_2 est échangeable avec a_1. Dans le premier cas, on peut remplacer les deux applications de règles par l'unique application $(r_j, \mathtt{Q}, \mathtt{Q}')$, ce qui réduit le coût du chemin puisque le coût de r_i est strictement positif ; c'est donc absurde, puisqu'on a supposé le chemin de coût minimal. Ainsi, chaque fois que cette situation est rencontrée, r_j est échangeable avec r_i, et on peut procéder comme pour la proposition précédente, en échangeant petit à petit les règles pour obtenir une suite décroissante. □

Exemple. La règle « suppression d'un triplet » (r_{sup}) est échangeable ou absorbable avec g_{suj}, g_{pred} et g_{obj}. En effet, si on applique une de ces règles de généralisation, puis r_{sup} :

- soit le triplet supprimé est celui sur lequel on a fait la généralisation, et dans ce cas l'application de g_{suj}, g_{pred} ou g_{obj} peut être absorbée par l'application de r_{sup} ;
- soit le triplet supprimé est différent de celui sur lequel on a fait la généralisation, et dans ce cas on peut sans problème échanger l'ordre d'application des règles.

On en déduit que l'ensemble ordonné de règles $r_{sup} > g_{suj} > g_{pred} > g_{obj}$ respecte les conditions de la proposition 4, et qu'on peut donc ne prendre en compte que les chemins de transformations où l'ordre des règles décroît.

4.3.3 Perspectives

Pour poursuivre ces recherches sur l'élagage de l'arbre d'exploration, il faudrait implémenter les propositions précédentes et réaliser des calculs expérimentaux de complexité. De plus, on a considéré ici des ordres sur les règles, mais on pourrait aller plus loin et considérer, dans une certaine mesure, des ordres sur les applications de règles. En effet, dans le cadre des définitions de ce rapport, le corps d'une requête est un ensemble non ordonné de motifs de triplet, mais d'un point de vue algorithmique, on peut supposer que cet ensemble de motifs est ordonné et numéroté ; pour une même règle, on pourrait alors mettre un ordre sur les différentes applications de règles suivant le numéro du motif auquel elle s'applique. Cela réduirait encore davantage la taille de l'arbre d'exploration.

5 Réécriture à coût nul : équivalence de requêtes

Problématique. On se demande sous quelles hypothèses il est vrai que pour toutes requêtes Q et Q', $Q \equiv Q'$ si et seulement si $Q = Q'$, et par quelles règles se ramener à des formes normales de requêtes vérifiant cette propriété.

5.1 Cas du SELECT

Cadre. On se restreint aux requêtes qui ne possèdent pas de nœuds anonymes. Autrement dit, tous les termes de la forme `?x` qui apparaissent dans le corps de la requête font partie de ses variables.

Proposition 5. *Soit deux requêtes* $\mathtt{Q}$ *et* $\mathtt{Q}'$ *de type* SELECT. *Si* $V(\mathtt{Q}) = V(\mathtt{Q}')$ *et* $A(\mathtt{Q}) = A(\mathtt{Q}') = \emptyset$, *alors :*

$$\mathtt{Q} = \mathtt{Q}' \text{ si et seulement si } \mathtt{Q} \equiv \mathtt{Q}'$$

Démonstration. On note $C = C(\mathtt{Q})$, $C' = C(\mathtt{Q}')$ et $V = V(\mathtt{Q}) = V(\mathtt{Q}')$.

On suppose $\mathtt{Q} = \mathtt{Q}'$. Soit $\mathcal{B}$ une base quelconque. On veut montrer $\mathtt{exec}_{\vdash}(\mathtt{Q}, \mathcal{B}) = \mathtt{exec}_{\vdash}(\mathtt{Q}', \mathcal{B})$. Puisque $\mathtt{Q} = \mathtt{Q}'$, on a $C = C'$ à renommage des nœuds anonymes près, or $A(\mathtt{Q}) = A(\mathtt{Q}') = \emptyset$, donc $C = C'$. On en déduit que C et C' ont les mêmes solutions avec même multiplicité, d'où $\mathtt{exec}_{\vdash}(\mathtt{Q}, \mathcal{B}) = \mathtt{exec}_{\vdash}(\mathtt{Q}', \mathcal{B})$.

Ceci étant vrai pour toute base $\mathcal{B}$, on a bien $\mathtt{Q} \equiv \mathtt{Q}'$.

Réciproquement, on suppose donc que pour toute base $\mathcal{B}$, $\mathtt{exec}_{\vdash}(\mathtt{Q}, \mathcal{B}) = \mathtt{exec}_{\vdash}(\mathtt{Q}', \mathcal{B})$. Soit NI un ensemble de nouveaux IRI (non présents dans C et C') tel que $|NI| = |V|$, et $\mu : V \to NI$ une bijection. En particulier, μ est une fonction partielle de $\mathcal{V}$ vers $\mathcal{T}$, donc c'est par définition une fonction solution. On définit $\mathcal{B}_{C'} := \mu(C')$. Autrement dit, $\mathcal{B}_{C'}$ est un graphe RDF égal à C' dans lequel on a remplacé chaque variable par une ressource neuve.

Soit alors t un motif de triplet de $\mathtt{Q}$. On veut montrer que $t \in \mathtt{Q}'$.

Étant donné la définition de $\mathcal{B}_{C'}$, on a $\mu \in \mathtt{exec}_{\vdash}(\mathtt{Q}', \mathcal{B}_{C'})$. Par hypothèse, on en déduit que $\mu \in \mathtt{exec}_{\vdash}(\mathtt{Q}, \mathcal{B}_{C'})$. Donc $\mu(C)$ est un sous-ensemble de $\mathcal{B}_{C'}$. Or $t \in C$, donc $\mu(t) \in \mathcal{B}_{C'}$. μ étant une bijection, on en déduit que $t \in \mu^{-1}(\mathcal{B}_{C'}) = C'$.

Ainsi C est inclus dans C', et par symétrie du raisonnement $C = C'$, d'où $Q = Q'$. □

Pour aller plus loin. Est-ce qu'on a le même résultat si $\mathtt{Q}$ et $\mathtt{Q}'$ possèdent des nœuds anonymes ?

Conjecture 1. *Soit deux requêtes* $\mathtt{Q}$ *et* $\mathtt{Q}'$ *de type* SELECT. *Si* $V(\mathtt{Q}) = V(\mathtt{Q}')$, *alors :*

$$\mathtt{Q} = \mathtt{Q}' \text{ si et seulement si } \mathtt{Q} \equiv \mathtt{Q}'$$

5.2 Cas du SELECT DISTINCT

Voir des solutions apparaître plusieurs fois dans le résultat d'une requête de type SELECT n'est pas très utile dans un contexte de recherches simples dans une base de données. Il est plus pratique d'utiliser des requêtes SELECT DISTINCT. Celles-ci sont également plus simples à manipuler d'un point de vue théorique, puisqu'il n'y a pas à se soucier de la multiplicité des solutions dans le résultat de la requête.

En revanche, la conjecture précédente n'est pas vérifiée si les requêtes sont de type SELECT DISTINCT ; il va falloir trouver des règles pour la rétablir.

Voici par exemple deux requêtes équivalentes pour un SELECT DISTINCT mais pas pour un SELECT :

$\mathtt{Q} =$
```
SELECT [DISTINCT] ?x
WHERE {
    ?x p v .
    ?x p ?y
}
```

$\mathtt{Q}' =$
```
SELECT [DISTINCT] ?x
WHERE {
    ?x p v
}
```

En effet, si on prend par exemple $\mathcal{B} = \{(x\ p\ v), (x\ p\ w)\}$, alors l'unique solution à Q comme à Q' est $\{(\mathtt{?x}, x)\}$, mais elle est de multiplicté 2 pour Q (`?y` pouvant être appairé à v ou à w) et de multiplicité 1 pour Q'.

Définition 38 (Élimination des redondances)**.** *Soit* $\mathtt{Q}$ *une requête. Soit* α *le plus grand sous-ensemble non vide de* $A(\mathtt{Q})$ *vérifiant la condition suivante :* il existe une fonction d'instanciation RDF σ_α telle que σ_α vaut l'identité sur $A \setminus \alpha$ et $\sigma_\alpha(T_\alpha) \subseteq C(\mathtt{Q}) \setminus T_\alpha$*, où* T_α *est l'ensemble des triplets de* $C(\mathtt{Q})$ *où apparaissent au moins un élément de* α*. Si* α *existe, alors appliquer l'élimination des redondances à* $\mathtt{Q}$ *signifie supprimer* T_α *de* $C(\mathtt{Q})$*.*

Proposition 6. *Soit* Q *une requête simple de type* SELECT DISTINCT *à laquelle on peut appliquer l'élimination des redondances. Alors, si on nomme* Q' *la requête résultante, on a* $Q \equiv Q'$*.*

Démonstration. Soit une requête $\mathtt{Q} = (t, V, C)$ à laquelle on peut appliquer l'élimination des redondances, et $\mathtt{Q}' = (t', V', C')$ la requête obtenue en appliquant l'élimination. Celle-ci ne modifie ni le type ni les variables, donc on a $t = t'$ et $V = V'$. Pour montrer que $\mathtt{Q} \equiv \mathtt{Q}'$, il reste donc à montrer que pour toute base $\mathcal{B}$, $\mathtt{exec}_\vdash(\mathtt{Q}, \mathcal{B}) = \mathtt{exec}_\vdash(\mathtt{Q}', \mathcal{B})$. Soit donc $\mathcal{B}$ une base quelconque.

Soit μ une solution à C sur $\mathcal{B}$, et σ une fonction d'instanciation RDF telle que $\mu(\sigma(C)) \subseteq \mathcal{B}$. Puisque C' est un sous-ensemble de C, on a en particulier $\mu(\sigma(C')) \subseteq \mathcal{B}$. Donc μ est une solution pour C'.

Réciproquement, soit μ' une solution à C' sur $\mathcal{B}$, et σ' une fonction d'instanciation RDF telle que $\mu'(\sigma'(C')) \subseteq \mathcal{B}$. Soit α l'ensemble utilisé pour l'élimination des redondances. Soit $\sigma = \sigma' \circ \sigma_\alpha$.

On a $\sigma(C \setminus T_\alpha) = \sigma(C') = \sigma'(C')$ (car σ_α vaut l'identité sur $A \setminus \alpha$, donc sur les nœuds anonymes de $\mathtt{Q}'$), et donc $\mu'(\sigma(C \setminus T_\alpha)) \subseteq \mathcal{B}$.

On a de plus $\sigma(T_\alpha) = \sigma'(\sigma_\alpha(T_\alpha))$. Or, par définition de σ_α, $\sigma_\alpha(T_\alpha) \subseteq C'$. On en déduit $\mu'(\sigma'(\sigma_\alpha(T_\alpha))) \subseteq \mu'(\sigma'(C')) \subseteq \mathcal{B}$, et donc finalement $\mu'(\sigma(T_\alpha)) \subseteq \mathcal{B}$.

Finalement, on a $\mu'(\sigma(C)) = \mu'(\sigma(C \setminus T_\alpha)) \cup \mu'(\sigma(T_\alpha)) \subseteq \mathcal{B}$, et donc μ' est une solution pour C.

On en conclut que pour toute base, $\mathcal{B}$ $\mathtt{exec}_\vdash(\mathtt{Q}, \mathcal{B}) = \mathtt{exec}_\vdash(\mathtt{Q}', \mathcal{B})$. □

Exemple 1.

$\mathtt{Q}_1 =$
```
SELECT DISTINCT ?x
WHERE {
    ?x p ?y .
    ?x p v
}
```

$\mathtt{Q}_2 =$
```
SELECT DISTINCT ?x
WHERE {
    ?x p v .
    ?z p v
}
```

On ne peut espérer appliquer l'élimination des redondances à $\mathtt{Q}_1$ qu'avec $\alpha = \{\mathtt{?y}\}$ (puisque `?x` est une variable), et donc avec $T_\alpha = \{\langle \mathtt{?x}\ \ p\ \ \mathtt{?y} \rangle\}$. Il suffit alors de prendre $\sigma_\alpha(\mathtt{?y}) = v$, et on obtient :

$\mathtt{Q}' =$
```
SELECT DISTINCT ?x
WHERE {
    ?x p v
}
```

En effet, le triplet $\langle \mathtt{?x}\ \ p\ \ \mathtt{?y} \rangle$ apparaît deux fois après la transformation, mais étant donné que le corps d'une requête est un ensemble, il est réduit à un seul triplet.

De même, on applique l'élimination des redondances à $\mathtt{Q}_2$ avec $\alpha = \{\mathtt{?z}\}$ et $\sigma_\alpha(\mathtt{?z}) =$ `?x`, et on obtient la même requête $\mathtt{Q}'$. Par la proposition précédente et par transitivité de $\equiv$, on en déduit que $\mathtt{Q}_1 \equiv \mathtt{Q}_2$.

Exemple 2.

$$\mathtt{Q}_3 = \left|\begin{array}{l}\texttt{SELECT DISTINCT ?x}\\ \texttt{WHERE \{}\\ \quad\texttt{?x p v .}\\ \quad\texttt{?x p ?y .}\\ \quad\texttt{?y q w}\\ \texttt{\}}\end{array}\right.$$

Ici, il est impossible d'appliquer l'élimination des redondances à $\mathtt{Q}_3$. En effet, il faudrait prendre, comme pour $\mathtt{Q}_1$, $\sigma_\alpha(\mathtt{?y}) = v$, afin de faire coïncider le deuxième triplet avec le premier. Mais le troisième triplet deviendrait $\langle v \quad q \quad w\rangle$, or il n'est pas présent dans $\mathtt{Q}_3$. On ne peut donc pas simplifier cette requête avec l'élimination des redondances.

Conjecture 2. *Soit* $\mathtt{Q}$ *et* $\mathtt{Q}'$ *deux requêtes simples de la forme* SELECT DISTINCT *auxquelles il est impossible d'appliquer l'application des redondances (soit parce que c'était impossible dès le départ, soit parce qu'on la leur a déjà appliquée). Alors* $\mathtt{Q} = \mathtt{Q}'$ *si et seulement si* $\mathtt{Q} \equiv \mathtt{Q}'$.

5.3 Perspectives

La notion de requête plus spécifique (resp. plus générale), et donc d'équivalence de requêtes, peut se généraliser en la faisant dépendre d'une ontologie : étant donné une ontologie $\mathcal{O}$ et deux requêtes $\mathtt{Q}$ et $\mathtt{Q}'$, on note $\mathtt{Q} \sqsubseteq_\mathcal{O} \mathtt{Q}'$ si pour toute base $\mathcal{B}$, $\mathtt{exec}_\vdash(\mathtt{Q}, \mathcal{O} \cup \mathcal{B}) \subseteq \mathtt{exec}_\vdash(\mathtt{Q}', \mathcal{O} \cup \mathcal{B})$, et donc $\mathtt{Q} \equiv_\mathcal{O} \mathtt{Q}'$ si pour toute base $\mathcal{B}$, $\mathtt{exec}_\vdash(\mathtt{Q}, \mathcal{O} \cup \mathcal{B}) = \mathtt{exec}_\vdash(\mathtt{Q}', \mathcal{O} \cup \mathcal{B})$. La notion d'équivalence utilisée jusque là est donc un cas particulier de cette définition, qui consiste à prendre $\mathcal{O} = \emptyset$.

6 Conclusion

Le sujet du stage était assez ouvert, il s'agissait principalement de découvrir le Web sémantique et ses possibilités, et de fournir quelques apports théoriques au langage SQTRL conçu par l'équipe. Une grande partie du travail a consisté à sélectionner les parties de la documentation de RDF et SPARQL utiles pour la suite, et parfois à les reformuler ou modifier pour qu'elles entrent dans le cadre du stage.

La partie sur les règles à coût nul établit quelques résultats sur l'équivalence de requêtes et énonce une règle de réécriture permettant de simplifier l'écriture de certaines requêtes, dans l'objectif d'aboutir à une forme canonique. Cependant, cette règle ne peut pas être formulée dans le langage SQTRL. Or, indépendamment de SQTRL, la problématique de la mise sous forme canonique de requêtes a déjà été traitée [3].

La partie la plus originale du stage est celle sur les règles SQTRL à coût non nul. Dans un premier temps, les recherches autour de la confluence de requêtes n'ont pas permis d'aboutir à des résultats utiles en pratique. Dans un second temps, la question de l'élagage de l'arbre d'exploration a permis d'aboutir à quelques résultats qui pourraient améliorer la complexité temporelle d'applications concrètes de SQTRL. Par manque de temps, de véritables calculs de complexité n'ont pas pu être menés.

Références

[1] Olivier Bruneau, Emmanuelle Gaillard, Nicolas Lasolle, Jean Lieber, Emmanuel Nauer, and Justine Reynaud. A SPARQL Query Transformation Rule Language - Application to Retrieval and Adaptation in Case-Based Reasoning. In David Aha and Jean Lieber, editors, *ICCBR 2017 - Case-Based Reasoning Research and Development, 25th International Conference on Case-Based Reasoning*, pages 76–91, Trondheim, Norway, June 2017. Springer.

[2] Olivier Bruneau, Nicolas Lasolle, Jean Lieber, Emmanuel Nauer, Siyana Pavlova, and Laurent Rollet. Applying and Developing Semantic Web Technologies for Exploiting a Corpus in History of Science : the Case Study of the Henri Poincaré Correspondence. *Semantic Web – Interoperability, Usability, Applicability*, 12(2) :359–378, 2021.

[3] Jaime Salas and Aidan Hogan. Semantics and Canonicalisation of SPARQL 1.1. *Semantic Web*, 2021. à paraître.

[4] Guus Schreiber and Yves Raimond. RDF 1.1 Primer, 2014. Dernière consultation : juillet 2021.

[5] Richard Cyganiak, David Wood, and Markus Lanthaler. RDF 1.1 Concepts and Abstract Syntax, 2014. Dernière consultation : juillet 2021.

[6] Steve Harris and Andy Seaborne. SPARQL 1.1 Query Language, 2013. Dernière consultation : juillet 2021.

A Annexe : Contexte institutionnel

Parmi mes encadrants, Jean Lieber et Nicolas Lasolle (en doctorat) font partie de l'équipe $\mathcal{K}$ du LORIA, appartenant au département *Traitement automatique des langues et des connaissances*. Olivier Bruneau fait partie de l'unité de recherche *Archives Henri-Poincaré*, à laquelle Nicolas Lasolle est également rattaché. Cette unité de recherche s'intéresse à la philosophie et l'histoire des sciences, et en particulier à l'œuvre d'Henri Poincaré.

J'ai effectué la majeure partie de mon stage à distance, j'étais donc principalement en contact avec mes encadrants Jean Lieber et Nicolas Lasolle. J'ai aussi pu assister à quelques réunions de leur équipe au sein du LORIA (équipe $\mathcal{K}$), ainsi qu'à des journées d'étude *Humanités Numériques et Web sémantique*. J'ai également pu passer deux jours à Nancy, et j'ai travaillé dans le bâtiment des Archives Henri Poincaré, où travaillent en général mes encadrants. Là-bas, la proximité entre chercheurs en informatique et chercheurs en philosophie des sciences illustre bien l'interdisciplinarité des humanités numériques.

B Annexe : preuves de la section 4.2

Démonstration de la Proposition 1. Un résultat de la théorie des graphes est qu'un graphe orienté sans circuit (DAG, pour *directed acyclic graph*, en anglais) possède un tri topologique, c'est-à-dire un ordre total sur l'ensemble des sommets tel que pour toute arc (u, v), u se trouve avant v. On assigne alors à chaque propriété un nombre correspondant à sa position dans l'ordre topologique (1 pour le plus petit, puis on incrémente).

Soit `Q` une requête quelconque. On veut montrer qu'on ne pourra la transformer suivant $\mathcal{R}$ qu'un nombre fini de fois. On note n son nombre initial de motifs de triplets. On fixe un ordre arbitraire sur ces motifs de triplets, et on définit K, qui varie au cours des transformations, le $(n+1)$-uplet constitué de l'opposé du nombre de triplets de `Q` (donc initialement $-n$), suivi de la position du prédicat de chaque triplet dans l'ordre topologique. Le nombre de triplets peut baisser si deux triplets deviennent identiques après une transformation, et dans ce cas on pourra attribuer la valeur 0 à celui des triplets qu'on choisit de ne plus considérer.

On va alors prouver que :

1. à chaque application de g_{pred}, la valeur de K augmente strictement selon l'ordre lexicographique ;
2. à chaque application de g_{suj} ou de g_{obj}, la valeur de K reste égale ou augmente selon l'ordre lexicographique.

En effet :

1. À chaque application de g_{pred}, tous les triplets restent inchangés sauf un, qu'on appelle t. S'il devient identique à un autre triplet, alors le nombre de triplets de `Q` va diminuer, donc le premier élément

de K va augmenter et K aura bien strictement augmenté selon l'ordre lexicographique. Sinon, on va transformer le prédicat p de t en un prédicat q tel que p est une sous-propriété de q, donc tel que q est après p dans l'ordre topologique. La valeur correspondant à ce triplet étant le seul élément de K à changer, on a bien augmenté strictement K suivant l'ordre lexicographique.

2. À chaque application de g_{suj} ou de g_{obj}, tous les prédicats des triplets restent inchangés. Si un triplet se retrouve en double, alors comme précédemment, K augmente suivant l'ordre lexicographique. Sinon, K reste inchangé.

De manière très similaire, on peut définir un $(m+1)$-uplet L, où m est le nombre de triplets de `Q` ayant une classe pour sujet, tel que la valeur de L augmente strictement à chaque application de g_{suj}, et augmente ou reste inchangée à chaque application de g_{pred} ou de g_{obj}. On peut faire de même avec g_{obj}, et avoir un $(k+1)$-uplet M. En concaténant K, L et M, on obtient un $(n+m+k+3)$-uplet U dont la valeur augmente strictement à chaque transition du système de transformation. Or U ne peut prendre qu'un nombre fini de valeurs : chaque élément est limité soit par le nombre de triplets de `Q`, soit par son nombre de classes, soit par son nombre de propriétés. On en déduit donc que la suite de transformations ne peut qu'être finie. Puisqu'on a choisi une requête `Q` quelconque, on obtient bien que la relation $\longrightarrow$ termine. □

Démonstration de la Proposition 2. Commençons par prouver le sens direct par contraposée : supposons qu'il existe une classe C et deux superclasses C_1 et C_2 telles qu'il n'existe pas de classe C_3 qui est une superclasse de C_1 et de C_2. Soit les requêtes

`Q` =
```
SELECT ?x
WHERE {
    ?x a C
}
```
`Q1` =
```
SELECT ?x
WHERE {
    ?x a C1
}
```
`Q2` =
```
SELECT ?x
WHERE {
    ?x a C2
}
```

On a, par la règle g_{obj}, que `Q` $\longrightarrow$ `Q1` et `Q` $\longrightarrow$ `Q2`. En revanche, il n'existe pas de requête `Q3` telle que `Q1` $\longrightarrow^*$ `Q3` et `Q2` $\longrightarrow^*$ `Q3`. En effet, on ne peut appliquer à `Q1`, `Q2` et leurs transformations par $\mathcal{R}$ que la règle g_{obj} (puisque `a` n'est pas une propriété apparaissant dans l'ontologie). Pour obtenir une telle `Q3`, il faudrait donc une classe qui soit une superclasse à la fois de C_1 et de C_2, ce qui est impossible par hypothèse. On en déduit ainsi que $\longrightarrow$ n'est pas confluente.

Par un raisonnement similaire, on déduit aussi que si la seconde propriété de la proposition n'est pas vérifiée, alors $\longrightarrow$ n'est pas confluente.

Considérons maintenant le sens réciproque : on suppose 1. et 2. vraies, et on veut montrer que $\longrightarrow$ est confluente. Puisque $\longrightarrow$ est terminante (d'après l'hypothèse faite sur les taxonomies), par le lemme de Newman, il suffit de montrer que $\longrightarrow$ est localement confluente pour obtenir que $\longrightarrow$ est confluente. On va en réalité même montrer que $\longrightarrow$ vérifie la propriété du losange (qui est plus forte que la confluence locale).

Soit donc `Q` une requête, et `Q1` et `Q2` telles que `Q` $\longrightarrow$ `Q1` et `Q` $\longrightarrow$ `Q2`. On note t_1 et t_2 les triplets de `Q` qui ont été modifiés, t'_1 et t'_2 les triplets par lesquels ils ont été remplacés respectivement dans `Q1` et `Q2`, et r_1 et r_2 les règles qui ont été utilisées pour ce faire. Il y a trois possibilités :

— si $t_1 \neq t_2$, alors $t_1 \in$ `Q2` et $t_2 \in$ `Q1`, donc on peut appliquer r_1 à `Q2` et r_2 à `Q2`. On obtient la même requête `Q3`, qui correspond à `Q` dans laquelle on a substitué t_1 par t'_1 et t_2 par t'_2. On a bien la propriété du losange.

— si $t_1 = t_2$ et $r_1 \neq r_2$, alors r_1 reste applicable à `Q2` (sur t'_2) et r_2 reste applicable à `Q1` (sur t'_1). En les appliquant effectivement, on obtient le même triplet t_3, et donc la même requête `Q3` à partir de

$\mathtt{Q}_1$ et $\mathtt{Q}_2$. (Par exemple, si $r_1 = g_{pred}$ et $r_2 = g_{obj}$ avec $t_1 = t_2 = \langle x \;\; p \;\; C\rangle$, on a t'_1 de la forme $\langle x \;\; q \;\; C\rangle$, t'_2 de la forme $\langle x \;\; p \;\; D\rangle$, et t_3 de la forme $\langle x \;\; q \;\; D\rangle$.)

— le dernier cas est $t_1 = t_2$ et $r_1 = r_2$, on va prendre l'exemple $r_1 = r_2 = g_{suj}$, les deux autres cas sont similaires. Si $t_1 = t_2 = \langle C \;\; p \;\; y\rangle$, alors t'_1 est de la forme $\langle C_1 \;\; p \;\; y\rangle$ et t'_2 de la forme $\langle C_2 \;\; p \;\; y\rangle$, où C_1 et C_2 sont deux superclasses de C. Par l'hypothèse 1., on a une classe C_3 dont C_1 et C_2 sont des sous-classes. On peut donc appliquer g_{suj} à $\mathtt{Q}_1$ et $\mathtt{Q}_2$ pour obtenir la même requête avec le triplet $\langle C_3 \;\; p \;\; y\rangle$.

Ainsi $\longrightarrow$ vérifie la propriété du losange, et donc, puisqu'elle termine, elle est confluente par le lemme de Newman, ce qui achève la preuve. □